Universidad Nacional Autónoma de México

Facultad de Ciencias

ALGORITMO DE RECONOCIMIENTO LINEAL PARA COGRÁFICAS

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

GARCÍA ARGUETA JAIME DANIEL

DIRECTOR DE TESIS:
CÉSAR HERNÁNDEZCRUZ

CD. MX. 2024

1.Datos del alumno

Apellido paterno	García
Apellido materno	Argueta
Nombre(s)	Jaime Daniel
Teléfono	55 21 81 55 65
Universidad	Universidad Nacional Autónoma de México
Facultad o escuela	Facultad de Ciencias
Carrera	Ciencias de la Computación
Número de cuenta	312104739

2. Datos del tutor

Grado	Dr.
Nombre(s)	César
Apellido paterno	Hernández
Apellido materno	Cruz

3. Datos del sinodal 1

Grado	Dr.
Nombre(s)	Fernando Esteban
Apellido paterno	Contreras
Apellido materno	Mendoza

4. Datos del sinodal 2

Grado	Dr.
Nombre(s)	Juan José
Apellido paterno	Montellano
Apellido materno	Ballesteros

5. Datos del sinodal 3

Grado	Dra. en C.I.C.
Nombre(s)	Karla Rocío
Apellido paterno	Vargas
Apellido materno	Godoy

6. Datos del sinodal 4

Grado	Dr.
Nombre(s)	Germán
Apellido paterno	Benítez
Apellido materno	Bobadilla

7.Datos del trabajo escrito

Título	Algoritmo de reconocimiento lineal para cográficas
Número de páginas	178 p.
Año	2024

Agradecimientos

A Gloria Argueta Cruz (Madre): Agradezco todo. Sin querer aparentar pretensión, debo a ella desarrollar más. Una persona de magnífico corazón que me inculcó muchas cosas positivas en mí. Agradezco todo su amor incondicional, no solo en la carrera, sino a lo largo de mi vida. El fastidioso ente que soy, es gracias a su apoyo económico en mis necesidades; doy gracias a ese apoyo incondicional, tanto moral como material; y por último, doy gracias a que ella es quién ella es... aunque no fue mi elección, no podría desear una mejor madre y tutora que ella.

A César Hernández Cruz (Asesor): Un ser humano, y matemático, maravilloso desde mi perspectiva. Una persona que desde que lo conozco me ha apoyado. Tanta es mi admiración por él, que no puedo concebir, en nuestra cultura normalizada, como puede apoyar a alguien sin ninguna promesa, y con muchos problemas como yo. Para ser más conciso, agradezco su apoyo, tanto en mi trabajo como en mi trayectoría escolar y docente; agradezco su habilidad y pericia en las matemáticas; agradezco que tenga la visión docente y mentorial que posee, pues creo que esto hace una diferencia en el mundo del "pensar", con todo lo loable que posee este término informal.

A los demás (Amigos, profesores, conocidos y personas admirables): No quiero abrir la puerta a tener que poner por escrito en este trabajo a las perosnas que les agradezco algo, pues esto desbodaría el escrito y seguramente me faltaría por mencionar a alguien. Entonces, aquí agradezco a todas las personas admirables que se han cruzado en mi camino y han sido una baliza para convertirme en lo que soy, en particular en el aspecto matemático. Debo hacer una mención especial a los que me han introducido en el ámbito filosófico, ya que sin la filosofía mi vida no es nada, al igual que sin las matemáticas mi vida es nada.

Lo que he dicho de los referidos se queda corto, sin embargo, así logro que mis agradecimientos quepan en una cuartilla. Esto no es probelma, pues una cuartilla les resta la misma justicia a sus virtudes que miles.

Índice general

Introducción

La familia de cográficas se define como la clase más pequeña de gráficas simples que satisface las siguientes propiedades: contiene a todas las gráficas triviales, es cerrada bajo la unión ajena, y es cerrada bajo la unión completa. En este trabajo vamos a estudiar la clase de cográficas en dos aspectos principalmente. El primero de estos aspectos es la caracterización de la propiedad de ser cográfica, mientras el segundo es el reconocimiento algorítmico de una cográfica en tiempo lineal.

Este trabajo está dividido en cuatro capítulos, sin contar la introducción ni las conclusiones. Los dos primeros capítulos, el de preliminares y árboles, conciernen a conceptos de teoría de gráficas y complejidad temporal de algoritmos que vamos a necesitar para el desarrollo de los otros dos capítulos subsecuentes, estos dos primeros capítulos se pueden omitir si ya se posee cierto dominio de la teoría; quizás la sección menos trillada de estos capítulos es la que estudia los árboles enraizados y algunas de sus propiedades.

En los otros dos capítulos se encuentran los resultados principales del presente escrito. Por su parte, el capítulo de cográficas se encarga de introducir formalmente la clase de cográficas junto con algunas de sus propiedades, para después demostrar un teorema que caracteriza a las cográficas, ligando la propiedad de ser cográfica a otros múltiples conceptos. Estas caracterizaciones se mencionan en [3, 5, 6]. Sin embargo, las demostraciones del teorema de caracterización que presentamos aquí, fueron desarrolladas por el autor de este trabajo, aunque, como era de esperarse, seguramente no resultaron ser originales en su mayoría. También en este capítulo dedicamos una sección a la noción de módulos, una noción importante, no ya para este trabajo, sino para el estudio de las cográficas en general.

El capítulo de BFS-Lexicográfico y algoritmo de reconocimiento es el más extenso, pues ahí se discute ampliamente todo el bagaje necesario para construir el algoritmo de reconocimiento. Este algoritmo de reconocimiento está descrito en [2], pero, en el presente trabajo se proporcionan a detalle todas las demostraciones, subrutinas y análisis de tiempo que fundamentan rigurosamente el algoritmo. Entre otras cosas, se estudia el algoritmo BFS-Lexicográfico y las propiedades principales de los orde-

namientos LexBFS, se verifica la linealidad de LexBFS, y se generan varias rutinas que luego el algoritmo de reconocimiento principal utiliza como componentes.

Por último, hay un apéndice dedicado a explorar dos ejecuciones con el algoritmo final. Una de esas ejecuciones documenta lo que ocurre cuando la entrada es una cográfica, en ese caso se analiza el proceso de LexBFS y de LexBFS$^-$, además de construir su coárbol. La otra ejecución es con una gráfica que no es cográfica, en ese caso también se analiza el proceso de LexBFS y de LexBFS$^-$, pero ahí se ve el proceso para hallar un P_4 inducido en la gráfica.

Capítulo 1

Preliminares

1.1. Gráficas

Una **gráfica** es una terna ordenada $G = \langle V, E, \psi \rangle$. En donde, V y E son conjuntos arbitrarios con la condición de que $V \neq \varnothing$, $V \cap E = \varnothing$ y

$$\psi \colon E \to \{\{x, y\} : x, y \in V\}.$$

Dada una gráfica $G = \langle V, E, \psi \rangle$, nos referimos a V como el conjunto de **vértices** de G, a E como el conjunto de **aristas** de G y a ψ como la **función de adyacencia** de G. Los elementos de V son llamados vértices y los elementos de E son llamados aristas. Cuando hablemos de una gráfica G, se entenderá que V_G, E_G y ψ_G referencian a los conjuntos de vértices, aristas y a la función de adyacencia de G, respectivamente. En una gráfica G, se conoce como **orden** de G al número $|V_G|$ y se conoce como **tamaño** de G al número $|E_G|$.

Podemos representar a una gráfica con un diagrama como sigue: si tenemos la gráfica G, dibujamos un punto en el plano con etiqueta u, por cada elemento $u \in V_G$. Una vez representados todos los vértices de G, unimos con una línea de etiqueta e, a los puntos que son símil de u y v, si y solo si, e es una arista que satisface $\psi(e) = \{u, v\}$. Gracias a lo anteior, muchas veces nos permitiremos definir una gráfica dando simplemente su diagrama, en lugar de escribir sus componentes explicitamente. Para entender mejor estos diagramas, veamos unos cuantos ejemplos.

Consideremos la gráfica que tiene un único vértice y no tiene aristas. Esta gráfica es conocida como la**gráfica trivial** y la podemos definir como

$$G = \langle \{v_1\}, \varnothing, \varnothing \rangle.$$

El diagrama de esta gráfica se puede ver en la Figura 1.1.

Figura 1.1: El diagrama de una gráfica trivial.

Sea $G_1 = \langle V, E, \psi \rangle$ la gráfica que tiene las siguientes componentes:

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$E = \{a, b, c, d, e, f, g, h, i, j, k, l\}, \text{ y con función de adyacencia:}$$

$$a \xrightarrow{\psi} \{v_2\}, b \xrightarrow{\psi} \{v_4\}, c \xrightarrow{\psi} \{v_1, v_4\}, d \xrightarrow{\psi} \{v_1, v_4\}, e \xrightarrow{\psi} \{v_5, v_6\}, f \xrightarrow{\psi} \{v_5, v_6\},$$

$$g \xrightarrow{\psi} \{v_5, v_6\}, h \xrightarrow{\psi} \{v_1, v_6\}, i \xrightarrow{\psi} \{v_3, v_6\}, j \xrightarrow{\psi} \{v_2, v_5\}, k \xrightarrow{\psi} \{v_4, v_5\}, l \xrightarrow{\psi} \{v_2, v_3\}.$$

El diagrama de esta gráfica lo podemos ver en la Figura 1.2.

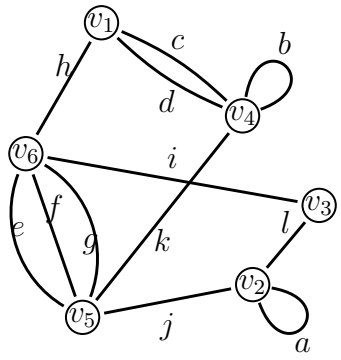


Figura 1.2: Diagrama de la gráfica G_1.

Demos un último ejemplo. Sea $G_2 = \langle V, E, \psi \rangle$ la gráfica que tiene las siguientes componentes:

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\},$$

$$E = \{a, b, c, d, e, f, g, h, i, j, k\}, \text{ y con función de adyacencia:}$$

$$a \xrightarrow{\psi} \{v_1, v_7\}, b \xrightarrow{\psi} \{v_1, v_6\}, c \xrightarrow{\psi} \{v_1, v_4\}, d \xrightarrow{\psi} \{v_1, v_2\}, e \xrightarrow{\psi} \{v_2, v_5\}, f \xrightarrow{\psi} \{v_3, v_7\},$$

$$g \xrightarrow{\psi} \{v_3, v_4\}, h \xrightarrow{\psi} \{v_4, v_6\}, i \xrightarrow{\psi} \{v_4, v_7\}, j \xrightarrow{\psi} \{v_5, v_6\}, k \xrightarrow{\psi} \{v_3, v_8\}.$$

El diagrama de esta gráfica lo podemos ver en la Figura 1.3.

Consideremos una gráfica G. Supongamos que tenemos dos vértices u, v de G y una arista e que satisface $\psi_G(e) = \{u, v\}$. Podemos decir entonces que los vértices u

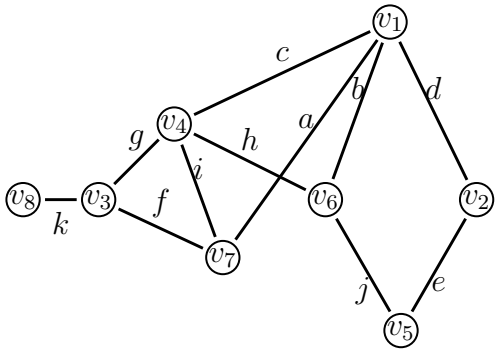


Figura 1.3: Diagrama de la gráfica G_2.

y v son **vértices adyacentes**, los vértices u y v son **extremos** de la arista e, y que la arista e **incide** en los vértices u y v. Si resulta que $u = v$, entonces llamaremos a e un **lazo** en u. Por ejemplo, en la Figura 1.2 la arista a es un **lazo** en el vértice v_2. Además, si a es una arista distinta de e que también incide en u y en v, decimos que e y a son aristas **paralelas** ; podemos ver que en la Figura 1.2 las aristas e y g son paralelas.

El concepto de grado es la forma de contar el número de extremos de aristas que inciden sobre un vértice. Sean G una gráfica y v un vértice de G. El **grado** de v en G, es el número

$$d_G(v) = |\{e \in E_G \colon v \in \psi_G(e), \psi_G(e) \neq \{v\}\}| + 2|\{e \in E_G \colon \psi_G(e) = \{v\}\}|.$$

Esto es, $d_G(v)$ es el número de aristas incidentes en v, pero contando dos veces los lazos. Si el contexto dispersa la ambigüedad, escribiremos el grado de v en G simplemente como $d(v)$. Utilizando el grado de los vértices, podemos definir para G los parámetros $\delta_G = \text{mín}\{d(u) \colon u \in V_G\}$ y $\Delta_G = \text{máx}\{d(u) \colon u \in V_G\}$, estos parámetros se llaman **grado mínimo** y **grado máximo** respectivamente. Por otro lado, el concepto de vecinos nos permite referenciar a los vértices adyacentes a un vértice dado. La **vecindad** de v en G, es el conjunto

$$N_G(v) = \{u \in V_G \colon \exists e \in E_G[\psi_G(e) = \{u, v\}]\}.$$

Del mismo modo que en el grado, escribiremos simplemente $N(v)$ si es clara la gráfica sobre la cual se toma la vecindad. Adicionalmente tenemos la notación $N[v]$ para el conjunto $N(v) \cup \{v\}$, este conjunto $N[v]$ es la **vecindad cerrada** de v. A modo de ejemplo, podemos ver en la Figura 1.2 que

$$d_{G_1}(v_4) = 5, \text{ y}$$

$$N_{G_1}(v_3) = \{v_2, v_6\},$$

mientras que de acuerdo con la Figura 1.3 se cumple

$$d_{G_2}(v_4) = 4, \text{ y}$$

$$N_{G_2}(v_3) = \{v_4, v_7, v_8\}.$$

Tenemos un primer resultado sobre las gráficas que relaciona la suma de los grados con el tamaño.

Proposición 1.1.1. *Sea G una gráfica. Si m es el tamaño de G, se satisface que $\sum_{v \in V_G} d(v) = 2m$.*

Demostración. Para cada $(v, e) \in V_G \times E_G$, definamos

$$\nu_{v,e} = \begin{cases} 0 & \text{si} \quad e \text{ no incide en } v \\ 1 & \text{si} \quad e \text{ incide en } v, \text{ pero no es lazo} \\ 2 & \text{si} \quad e \text{ incide en } v, \text{ y es lazo} \end{cases}$$

Observemos que si $e \in E_G$, se cumple que $\sum_{v \in V_G} \nu_{v,e} = 2$. Para ver lo anterior, basta notar que si e es un lazo en un vértice u, se cumple $\nu_{u,e} = 2$ y si $v \neq u$, entonces $\nu_{v,e} = 0$. Pero, si e tiene extremos distintos u y w, entonces $\nu_{w,e} = 1 = \nu_{u,e}$ y para cualquier $v \notin \{w, u\}$, se va a tener $\nu_{v,e} = 0$. Entonces, se satisface que

$$\sum_{v \in V_G} d(v) = \sum_{v \in V_G} (\sum_{e \in E_G} \nu_{v,e}) = \sum_{e \in E_G} (\sum_{v \in V_G} \nu_{v,e}) = \sum_{e \in E_G} 2 = 2m.$$

■

1.2. Subgráficas

Sean G y H gráficas. Diremos que H es una **subgráfica** de G, escrito $H \subseteq G$, si y solo si $V_H \subseteq V_G$, $E_H \subseteq E_G$ y $\psi_H \subseteq \psi_G$. Aquí $\psi_H \subseteq \psi_G$ es otra forma de escribir $\psi_H = \psi_G|_{E_H}$. Adicionalmente, H será una **subgráfica generadora** de G, si $H \subseteq G$ y $V_H = V_G$. Consideremos las gráficas H_1 y H_2, representadas en las Figuras 1.4 y 1.5, respectivamente. Tomemos en cuenta a G_1 y G_2 como se muestran en las Figuras 1.2 y 1.3. Podemos observar que H_1 es una subgráfica de G_1, más aún, es una subgráfica generadora de G_1, pero no es siquiera subgráfica de G_2. Por otro lado, H_2 es una subgráfica de G_2, aunque no generadora, pero no es subgráfica de G_1.

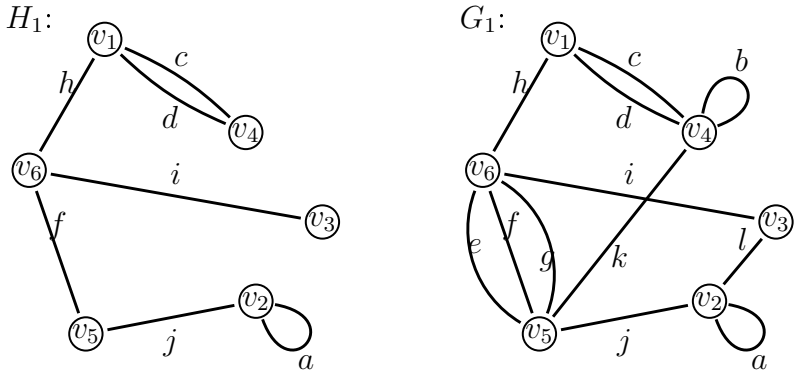


Figura 1.4: Gráfica H_1 como subgráfica de G_1.

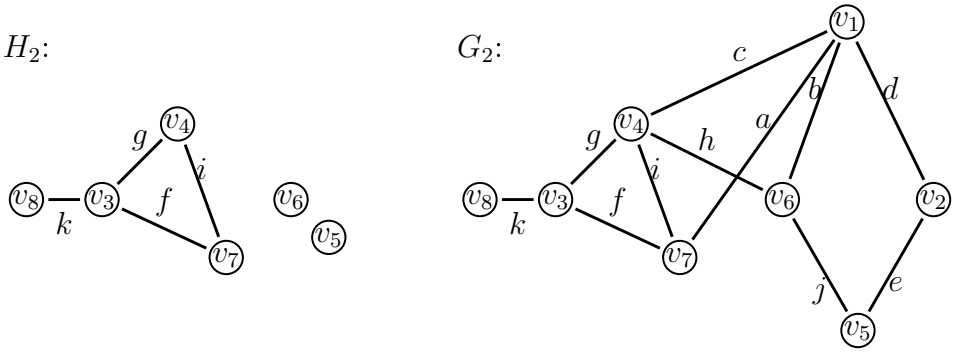


Figura 1.5: Gráfica H_2 como subgráfica de G_2.

Para definir a las **subgráficas inducidas**, fijemos una G gráfica, $X \subseteq V_G$ no vacío, y $F \subseteq E_G$ no vacío. Definimos la **subgráfica inducida por vértices** en G por el conjunto de vértices X, como la gráfica

$$G[X] = \langle X, A, \psi_G|_A \rangle$$

en donde

$$A = \{e \in E_G \colon \psi_G(e) \subseteq X\}.$$

También definimos la **subgráfica inducida por aristas** en G por el conjunto de aristas F, como la gráfica

$$G[F] = \langle Y, F, \psi_G|_F \rangle$$

en donde

$$Y = \{u \in V_G \colon \exists e \in F[u \in \psi_G(e)]\}.$$

Para ilustrar las definiciones anteriores, utilizaremos de nueva cuenta a las gráficas en las Figuras 1.2 y 1.3. Sean $X_1 = \{v_2, v_3, v_5, v_6\} \subseteq V_{G_1}$, $X_2 = \{v_1, v_7, v_4, v_5\} \subseteq V_{G_2}$, $F_1 = \{b, f, h, i\} \subseteq E_{G_1}$ y $F_2 = \{a, b, d, g, j\} \subseteq E_{G_2}$. En la Figura 1.6 se muestran las gráficas $G_1[X_1], G_2[X_2]$, mientras que en la Figura 1.7 están los diagramas de $G_1[F_1]$ y $G_2[F_2]$.

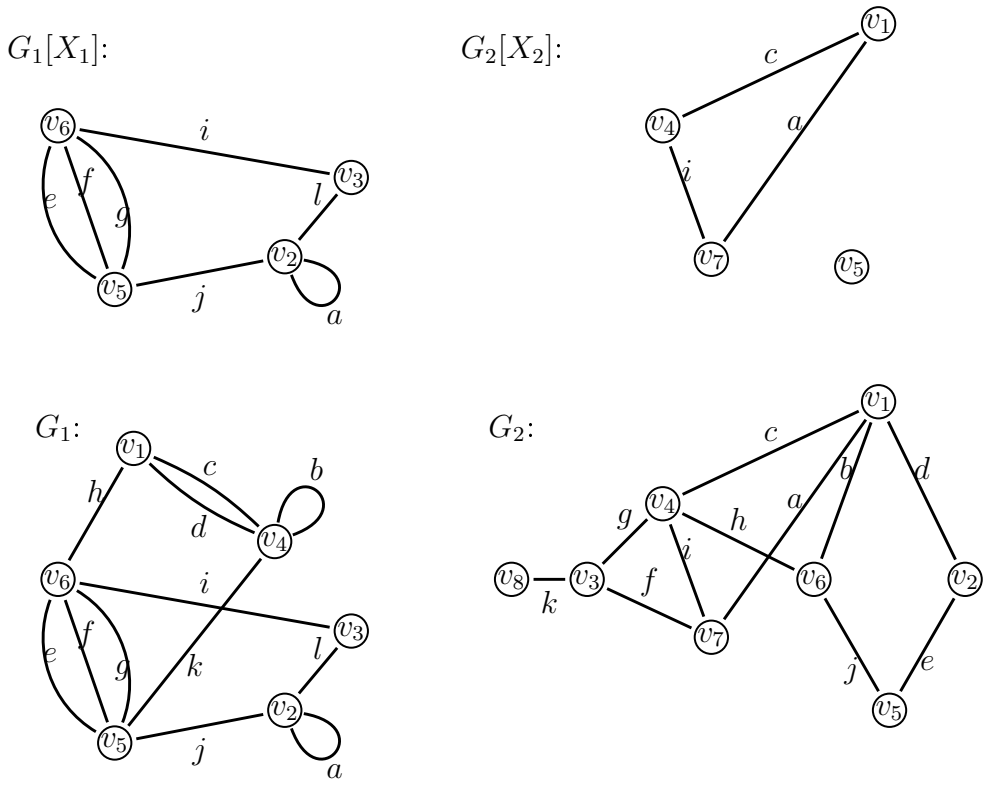


Figura 1.6: Gráficas inducidas por los conjuntos X_1 y X_2.

1.3. Isomorfismo

La noción de isomorfismo plasma la idea de igualdad estructural entre gráficas. Notemos que las gráficas $G = \langle\{v_1\}, \varnothing, \varnothing\rangle$ y $H = \langle\{\{v_1\}\}, \varnothing, \varnothing\rangle$ son ambas representantes de gráficas triviales, sin embargo G es distinta de H vistas como conjuntos. La noción de isomorfismo homologa las gráficas que en principio pueden ser distintas, pero tienen la misma estructura.

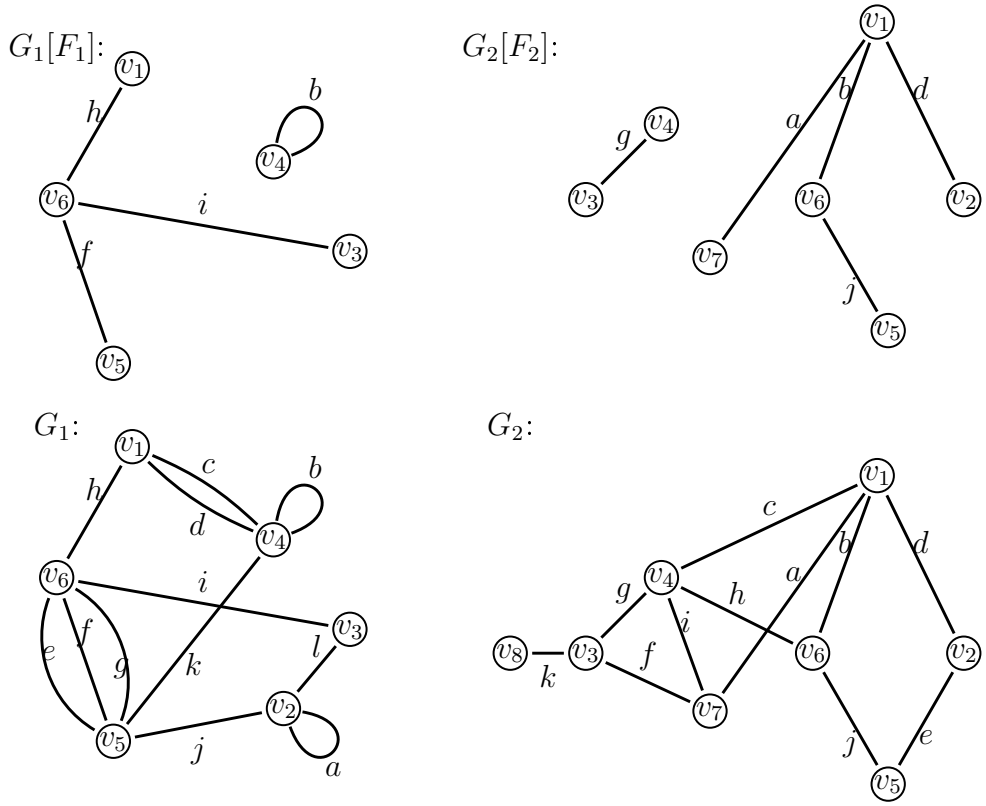


Figura 1.7: Gráficas inducidas por los conjuntos F_1 y F_2.

Si G y H son gráficas, un **isomorfismo** entre G y H es un par de funciones (ϕ, φ), donde $\phi\colon V_G \to V_H$ y $\varphi\colon E_G \to E_H$, que satisfacen lo siguiente:

i) Tanto ϕ como φ son biyectivas.

ii) Para cualquier $e \in E_G$ con $\psi_G(e) = \{u, v\}$, se satisface $\psi_H(\varphi(e)) = \{\phi(u), \phi(v)\}$.

Diremos que dos gráficas G y H son **gráficas isomorfas** cuando exista un isomorfismo entre G y H, este hecho lo escribimos como $G \cong H$. En la Figura 1.8 podemos ver dos gráficas que son isomorfas.

Consideremos G y H dos gráficas. Podemos abusar del lenguaje y decir que, H es subgráfica de G en caso de que exista una subgráfica $H' \subseteq G$ que sea isomorfa a H. Observemos que en este caso estaríamos diciendo que H es subgráfica de G por el simple hecho de que hay una subgráfica de G que es estructuralmente la misma que H, esto a pesar de que no se cumpla cabalmente la definición para que $H \subseteq G$. También tenemos el concepto de unicidad salvo isomorfismo, este concepto es útil

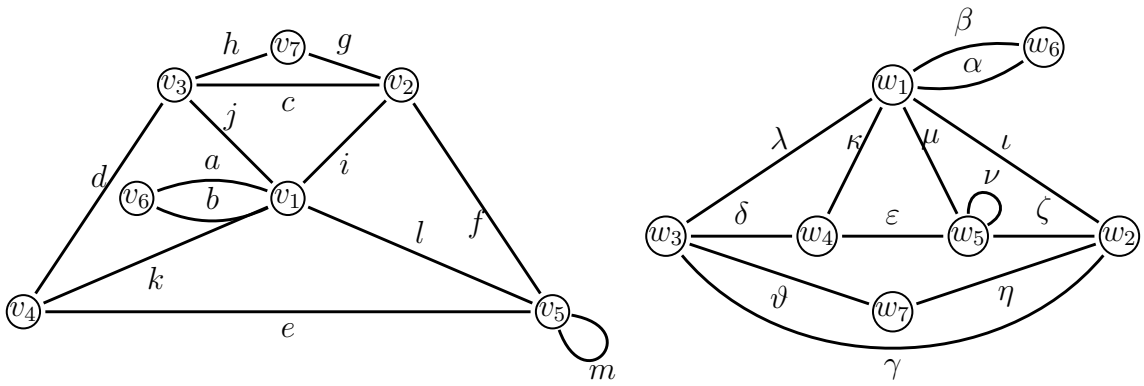


Figura 1.8: Dos gráficas isomorfas.

para describir gráficas mediante propiedades y tener la certeza de que cualesquiera gráficas que satisfagan esa propiedad deben ser isomorfas entre sí. Formalicemos lo anterior, para esto, tomemos una propiedad P acerca de gráficas que describe a una gráfica G, es decir que G cumple P. Decimos que la gráfica G es **única salvo isomorfismo** con la propiedad P, si para cualquier gráfica H que satisfaga P, se cumple $G \cong H$. Por ejemplo, en el primer párrafo de esta sección discutimos que las gráficas $G = \langle \{v_1\}, \varnothing, \varnothing \rangle$ y $H = \langle \{\{v_1\}\}, \varnothing, \varnothing \rangle$ deben ser ambas representantes de la gráfica trivial, o sea que son estructuralmente la misma gráfica y por lo tanto son isomorfas. Entonces, usando la propiedad de "ser una gráfica trivial" que es la propiedad de "tener un único vértice", podemos decir que G es única salvo isomorfismo, lo mismo para H.

1.4. Recorridos

Sea G una gráfica. Un camino en G es una sucesión no vacía, alternada de vértices y aristas, que comienza y termina en vértices. O sea, un **camino** es una sucesión

$$(v_1, e_1, v_2, e_2 \ldots, e_{k-1}, v_k),$$

donde para cada $i \in \{1, \ldots, k-1\}$ se cumple que $v_i, v_{i+1} \in V_G$, $e_i \in E_G$, y además se satisface que $\psi_G(e_i) = \{v_i, v_{i+1}\}$. Consideremos a $W = (v_1, e_1, v_2, e_2 \ldots, e_{k-1}, v_k)$ un camino en G. Cuando el primer vértice de un camino W es u, y el último vértice de W es v, decimos que W es un uv-camino. Un atributo importante del camino W es su **longitud**, denotada como $\ell(W)$, esta longitud es el número de aristas de

la sucesión, o sea $k-1$. Llamamos a W **camino cerrado** si es un camino donde el primer y último vértice de W son el mismo, es decir, si $v_1 = v_k$. Si un camino no es cerrado, diremos que es un **camino abierto**. Llamamos a W **trayectoria** si la sucesión W es un camino que no repite vértices. Llamaremos a W **paseo** si la sucesión W es un camino que no repite aristas. Por otro lado, podemos decir que W es un **ciclo**, si la sucesión W es un paseo de longitud al menos 1, donde los únicos vértices de la sucesión que son iguales son v_1 y v_k. También decimos que W es un **circuito**, si W es un camino cerrado que además es un paseo.

Consideremos la gráfica G_1 en la Figura 1.2. Podemos ver en la Figura 1.9 que $W_1 = (v_6, e, v_5, j, v_2, j, v_5, f, v_6, i, v_3)$ es un v_6v_3-camino, mientras que $W_2 = (v_6, e, v_5, j, v_2, j, v_5, f, v_6)$ es un camino cerrado, $W_3 = (v_6, e, v_5, j, v_2, l, v_3)$ es una v_6v_2-trayectoria, $W_4 = (v_6, e, v_5, j, v_2, l, v_3, i, v_6)$ es un ciclo, por su parte $W_5 = (v_6, e, v_5, f, v_6, g, v_5, j, v_2, a, v_2)$ es un v_6v_2 paseo, y para terminar tenemos que $W_6 = (v_1, c, v_4, b, v_4, d, v_4)$ es un circuito. El orden y la dirección de las aristas en la sucesión están marcadas con flechas.

Consideremos los siguientes caminos en una gráfica G: $W = (x_1, e_1, \ldots, e_{k-1}, x_k)$, $W_1 = (y_1, a_1, \ldots, a_{l-1}, y_l)$ y $W_2 = (z_1, f_1, \ldots, a_{m-1}, z_m)$. Cuando W_1 termina en el mismo vértice en el que comienza W_2, es decir, cuando $y_l = z_1$, podemos definir al camino en G que es la **concatenación** de W_1 con W_2 como el camino $W_1W_2 = (y_1, a_1, \ldots, y_l = z_1, f_1, \ldots, f_{m-1}, z_m)$. Observemos que la longitud de W_1W_2 es $\ell(W_1) + \ell(W_2)$. También es posible considerar segmentos de un camino. Tomemos $i, j \in \{1, \ldots, k\}$ tales que $i \leq j$, las subsucesiones de W dadas por $(x_i, e_i, \ldots e_{j-1}, x_j)$, $(x_1, e_1, \ldots e_{j-1}, x_j)$ y $(x_i, e_{i+1} \ldots, e_{k-1}, x_k)$ las denotamos por x_iWx_j, Wx_j y x_iW respectivamente. Resulta que x_iWx_j, x_iW y Wx_j son también caminos en G que llamamos **segmentos** de W. Dado un vértice u que aparece en W, o sea que para algún i se tiene que $u = x_i$, podemos escribir uW en lugar de x_iW siempre que no sea ambiguo, esto es, satisfaciéndose que no haya otro índice $j \neq i$ tal que $u = x_j$. Por último, definamos a la **reversa** de W, denotado por W^{-1}, esto no es otra cosa que la sucesión que define a W en reversa, o sea $W^{-1} = (x_k, e_{k-1}, \ldots, e_1, x_1)$; resulta que por definición W^{-1} es un vu-camino. Mencionemos también que con frecuencia escribimos V_W y E_W para referirnos a los vértices y a las aristas del camino, o sea a los conjuntos $\{x_1, \ldots, x_k\}$ y $\{e_1, \ldots, e_{k-1}\}$ respectivamente. Veamos ahora un resultado que establece una suerte de equivalencia entre la existencia de trayectorias y caminos.

Proposición 1.4.1. *Sean G una gráfica y $u, v \in V_G$ distintos. Si $W = (u = x_1, e_1, \ldots, e_k, v = x_{k+1})$ es un uv-camino en G, existe una uv-trayectoria contenida en W. Esto es, una trayectoria T tal que todos los vértices y aristas que aparecen en T, también aparecen en W y en el mismo orden.*

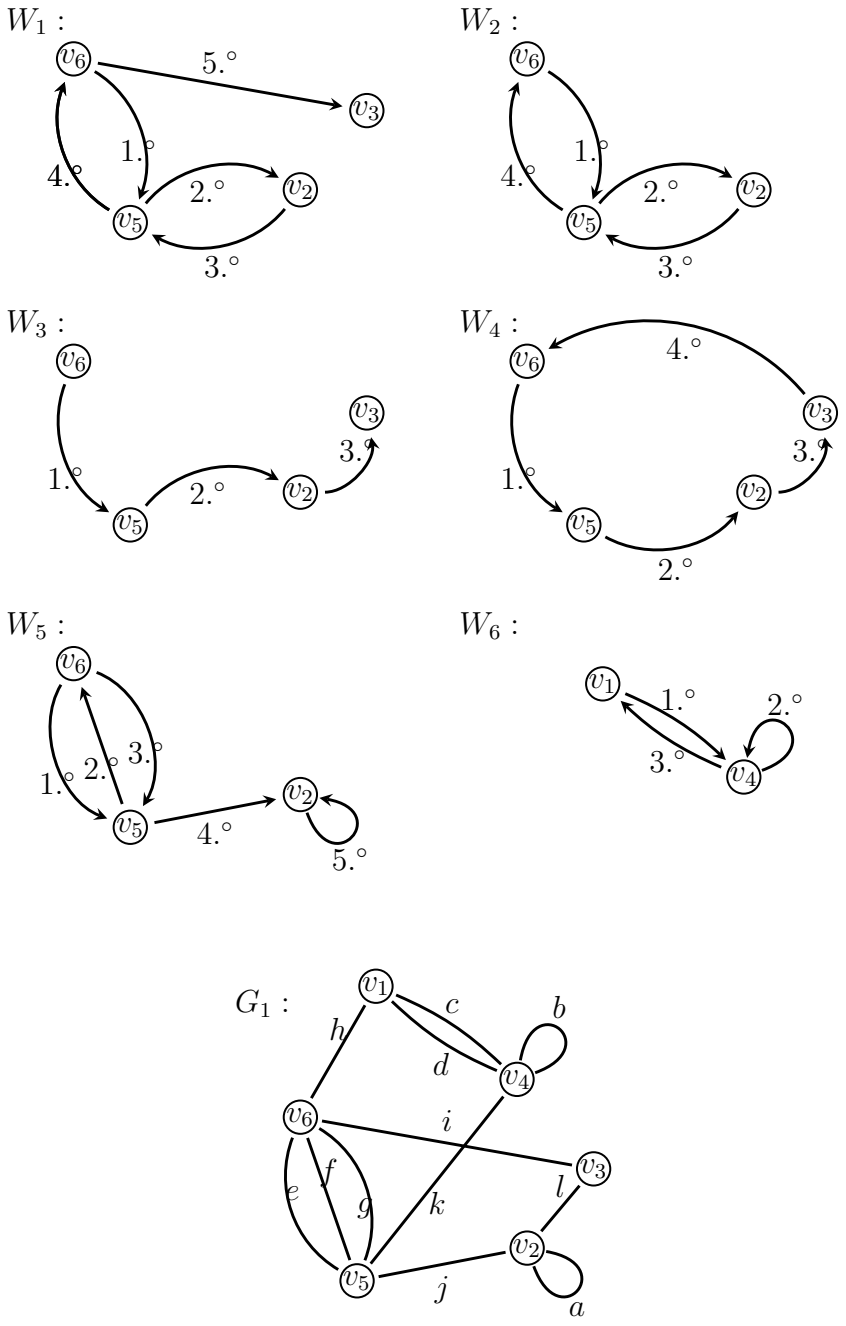


Figura 1.9: Recorridos en la gráfica G_1.

Demostración. Procedamos por inducción fuerte sobre la longitud de W, o sea k. Como $u \neq v$, la longitud de W debe ser al menos 1, así que tomaremos como caso base $k = 1$. En caso de que $k = 1$ tenemos por definición que los únicos dos vértices de la sucesión que define a W son u y v, entonces W mismo es una trayectoria. Supongamos que todo uv-camino de longitud menor que $k > 1$ fijo contiene una uv-trayectoria. Consideremos a W como un uv-camino arbitrario de longitud k como se presenta en el enunciado de la proposición. El único caso que hay que desarrollar es en el que W no es una trayectoria, en esta situación, deben existir $i, j \in \{1, \ldots, k\}$ distintos, tales que $i < j$, pero $x_i = x_j$. Tomando $W' = (u = x_1, e_1, \ldots, x_i = x_j, e_j, \ldots, x_{k+1} = v)$, se verifica que $\ell(W') = (i-1) + ((k+1) - j) = k - (j - i) < k$, así que podemos aplicar la hipótesis inductiva a W' para garantizar la existencia de una uv-trayectoria T contenida en W'. Como W' es un camino contenido en W, la trayectoria T también está contenida en W. ■

Observemos que el resultado anterior nos garantiza que, en una gráfica cualquiera, existe un camino entre un par de vértices $u \neq v$, si y solamente si, existe una trayectoria entre ese mismo par de vértices.

1.5. Conexidad y distancia

Consideremos una gráfica G y dos de sus vértices, u y v. Decimos que u está conectado con v, si existe un uv-camino en G. Podemos también definir lo anterior como una relación $R \subseteq V^2$, donde $R = \{(u, v) \colon u \text{ está conectado con } v\}$, esta relación recibe el nombre de **relación de conectividad**. En el siguiente resultado vemos que R es, de hecho, una relación de equivalencia.

Proposición 1.5.1. *Para cualquier gráfica G, la relación en $V_G \times v_G$ definida como $R = \{(u, v) \colon u \text{ está conectado con } v\}$, es una relación de equivalencia en V.*

Demostración. Para todo $u \in V_G$, el camino (u) atestigua que u está conectado con u, esto muestra que R es reflexiva. Supongamos que $u, v \in V_G$ son tales que u está conectado con v, entonces existe un uv-camino W en G. Como W^{-1} es un vu-camino en G, concluimos que v está conectado con u, de esto se sigue que R es simétrica. Sean $u, v, z \in V_G$ tales que $(u, v), (v, z) \in R$. Deben existir W_1 y W_2 un uv-camino en G y un vz-camino en G respectivamente. Como W_1W_2 es un uz-camino en G tenemos que v está conectado con z y por lo tanto $(v, z) \in R$, o sea que R es transitiva. ■

Consideremos una gráfica G para definir los conceptos de **conexidad**. En virtud del resultado anterior, dados $u, v \in V_G$, es equivalente que u esté conectado con v a que v esté conectado con u, por lo que, de satisfacerse lo anterior diremos simplemente que u y v están **conectados** en G. Decimos que la gráfica tiene la propiedad de ser **conexa**, solo cuando todo par de vértices en V_G están conectados entre sí. En caso de que G no sea conexa decimos que es **inconexa**, en este caso tendremos subgráficas de G que son conexas, como las inducidas por los unitarios de los vértices, y también tendremos subgráficas de G inconexas, como G misma. Podemos ver en la Figura 1.7 que las gráficas G_1 y G_2 son conexas, pero las gráficas $G_1[F_1]$ y $G_2[F_2]$ son inconexas. Si H es una subgráfica de G que es conexa y máxima por contención con esta propiedad, decimos que H es una **componente conexa** de G. Podemos caracterizar a las componentes conexas de una gráfica con la relación de conectividad como dice el siguiente resultado.

Proposición 1.5.2. *Sean G una gráfica, y $\sim = \{(u, v)\colon u$ está conectado con $v\} \subseteq V^2$. Las componentes conexas de G son las gráficas inducidas por los elementos del conjunto cociente $V_G/\sim$, es decir, las subgráficas $G[[x]_\sim]$ con $x \in V_G$.*

Demostración. Sea $x \in V_G$ arbitrario. Mostremos que $G[[x]_\sim]$ es una componente conexa de G. Para empezar recordemos que $V_{G[[x]_\sim]} = [x]_\sim$ y probemos que $G[[x]_\sim]$ es conexa. Para cualesquiera $y, z \in [x]_\sim$ se tiene $x \sim z$ y $x \sim y$, esto implica por la simetría y transitividad de $\sim$ que $y \sim z$. Sea $H \subseteq G$ conexa tal que $G[[x]_\sim] \subseteq H$ y notemos que $x \in [x]_\sim \subseteq V_H$. Así que, para cualquier $y \in V_H$ se tiene que x está conectado con y por la conexidad de H, lo anterior implica que $y \in [x]_\sim$. Como en lo anterior y fue arbitrario, podemos concluir que $V_H \subseteq [x]_\sim$ y por la definición de subgráfica inducida se sigue que $H \subseteq G[[x]_\sim]$. Esto prueba que la gráfica inducida por la clase de equivalencia de x es, en efecto una componente conexa de G. Para terminar probemos que las gráficas inducidas por las clases de equivalencia son las únicas componentes conexas. Tomemos una componente conexa de G, digamos H. Para cualquier $x \in V_H$ se cumple por la conexidad de H que $V_H \subseteq [x]_\sim$ y por lo tanto $H \subseteq G[[x]_\sim]$, pero H es componente conexa, por lo que $H = G[[x]_\sim]$. ■

Sean G una gráfica y $u, v \in V_G$. Si u y v están conectados, existe un uv-camino en G, pero sabemos por la Proposición 1.4.1 que entonces debe haber también alguna uv-trayectoria en G. Lo anterior nos permite definir la **distancia** de u hacia v en G, como el número mín$\{\ell(T)\colon T$ es una uv-trayectoria en $G\}$; esta distancia de u hacia v la denotamos por $d_G(u, v)$. Si u y v no están conectados, podemos pensar en la distancia de u a v como "infinita", así que justamente definimos $d(u, v)$ como el símbolo ∞. Lo anterior requiere aclarar la interacción de los números con el nuevo

símbolo con el que aumentamos el codominio de la función distancia. Simplemente tendremos que para cualquier $n \in \mathbb{N} \setminus \{0\}$ se satisface $n < \infty$, $n + \infty = \infty$ y $n \cdot \infty = \infty$; mientras que para 0 tendremos que $0 < \infty$, $0 + \infty = \infty$ y $0 \cdot \infty = 0$. Siempre que sea clara la gráfica en la cual se esta trabajando la distancia, omitiremos el subíndice de la gráfica.

Así definida la distancia, satisface las propiedades de una métrica, veamos esto rápidamente. La trayectoria trivial en el vértice u, o sea (u), muestra que $d(u,u) = 0$, además no existe una trayectoria de longitud 0 entre un par de vértices distintos. Dado un camino W, se cumple que $\ell(W) = \ell(W^{-1})$, así que para un par de vértices u y v, se va a cumplir que $d(u,v) = d(v,u)$. Por último, tomemos $u, v, w \in V_G$, y en caso de que existan, también T_1 y T_2 trayectorias mínimas en longitud de u hacia v y de v hacia w respectivamente. Si elegimos T una uw-trayectoria contenida en T_1T_2, se cumple que $d(u,w) \leq \ell(T) \leq \ell(T_1T_2) = \ell(T_1) + \ell(T_2) = d(u,v) + d(v,w)$. En el caso $d(u,v) = \infty$ o $d(v,w) = \infty$, es claro que se cumple la desigualdad.

Por lo anterior, para cada vértice u, podemos definir $e_G(u) = \text{máx}\{d_G(u,v) \colon v \in V_G\}$, este número se llama **excentricidad** de u en G. Ahora, tenemos también unos parámetros en G que marcan, de cierta forma, la distancia mínima y la distancia máxima, estos son el **radio** y el **diámetro** de G, definidos como $rad_G = \text{mín}\{e(u) \colon u \in V_G\}$ y $diam_G = \text{máx}\{e(u) \colon u \in V_G\}$ respectivamente. Ejemplificando estos conceptos, podemos ver en la Figura 1.3 que, $d(v_8, v_5) = 4$, $e(v_6) = 3$, $rad_{G_2} = 2$ y $diam_{G_2} = 4$.

1.6. Gráficas simples

Una clase particular de gráficas son las **gráficas simples** . Si tenemos una gráfica G, decimos que G es una gráfica simple, o simple a secas, si G no tiene aristas paralelas ni tampoco lazos. Podemos ver que la gráfica en la Figura 1.2 no es simple, mientras que la gráfica en la Figura 1.3 sí lo es. El considerar gráficas simples en lugar de gráficas en general, nos permite simplificar varios conceptos. Dado que las gráficas simples no tienen aristas paralelas ni lazos, si G es una gráfica simple, entonces ψ_G es inyectiva y su codominio es $[V]^2 = \{X \subseteq V \colon |X| = 2\}$. Entonces, para cualquier arista $e \in E_G$, podemos determinar a e solo por sus extremos, o sea por su conjunto correspondiente en $[V]^2$, ya que ninguna otra arista puede tener los mismos extremos. Por lo anterior, cuando se tratan solo gráficas simples, se utiliza la definición de **gráfica** como un par $\langle V, E\rangle$, donde $V \neq \varnothing$ y $E \subseteq [V]^2$, en lugar de la definición más extensa para una gráfica en general. Como dijimos que una arista de una gráfica simple está determinada por sus extremos, así mismo denotaremos a la arista $\{u,v\}$ simplemente por uv o por vu. También el concepto de grado se simplifica hablando de gráficas simples. Si G es una gráfica simple y $u \in V_G$, el **grado** de u en G es

simplemente $d_G(u) = |N_G(u)|$.

Consideremos dos gráficas simples G y H. Como la función de adyacencia no tiene relevancia ya, podemos reformular la definición de subgráfica para decir que H es una **subgráfica** de G, si y solo si, $V_H \subseteq V_G$ y $E_H \subseteq E_G$. Revisando la definición de isomorfismo, notemos que podemos omitir esa definición general y dar una nueva para gráficas simples. Así, diremos que un **isomorfismo** de G en H es simplemente una función $\varphi\colon V_G \to V_H$ biyectiva satisfaciendo que, para cualquier par de vértices $u, v \in V_G$, los vértices u, v son adyacentes en G, si y solo si, los vértices $\varphi(v), \varphi(u)$ son adyacentes en H. Ahora fijemonos en la definición de camino y recordemos de nuevo que, si G es una gráfica simple, para cualquier par de vértices $u, v \in V_G$, la arista que conecta a u con v, si la hay, es única. Podemos definir entonces un **camino** en G, como una sucesión no vacía de vértices en G, digamos $(v_1, \ldots, v_k)$, de tal forma que para todo $i \in \{1, \ldots, k-1\}$ se satisfaga que v_i y v_{i+1} son adyacentes.

Damos, a manera de ejemplo, algunas gráficas simples que son usadas frecuentemente. Para $n \in \mathbb{Z}^+$, la gráfica simple con n vértices y con todas las aristas posibles es conocida como **gráfica completa de orden** n. De la definición se desprende que esta gráfica es única salvo isomorfismo y la denotaremos por K_n. Las gráficas K_4 y K_7 las podemos ver en la Figura 1.10.

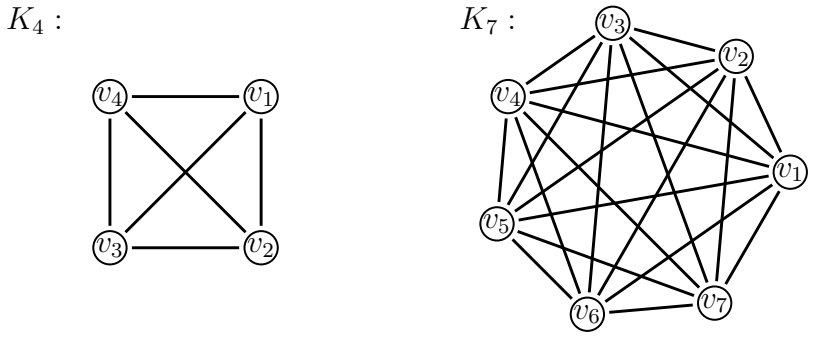


Figura 1.10: Gráficas K_4 y K_7.

Para $n \in \mathbb{Z}^+$, la gráfica simple con n vértices, $v_1, \ldots, v_n$, y conjunto de aristas $\{\{v_i, v_{i+1}\}\colon 1 \leq i < n\}$ es conocida como **trayectoria de orden** n. De la definición se desprende que esta gráfica es única salvo isomorfismo y la denotaremos por P_n. Las gráficas P_3 y P_6 las podemos ver en la Figura 1.11.

Para $n \in \mathbb{Z}^+$ con $n > 2$, la gráfica simple con n vértices, $v_1, \ldots, v_n$, y conjunto de aristas $\{\{v_i, v_{i+1}\}\colon 1 \leq i < n\} \cup \{\{v_1, v_n\}\}$ es conocida como **ciclo de orden** n o n**-ciclo**. De la definición se desprende que esta gráfica es única salvo isomorfismo y la denotaremos por C_n. Las gráficas C_4 y C_8 las podemos ver en la Figura 1.12.

P_3 : P_6 :

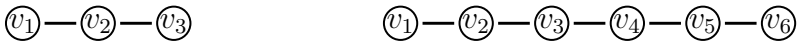


Figura 1.11: Gráficas P_3 y P_6.

C_4 : C_8 :

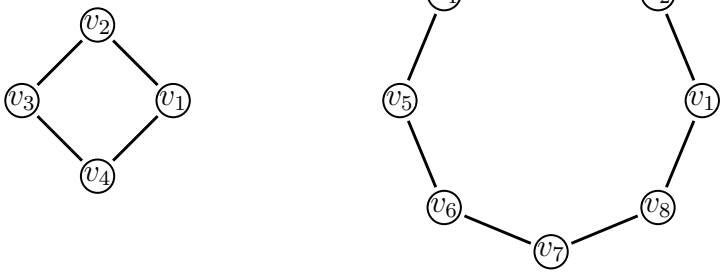


Figura 1.12: Gráficas C_4 y C_8.

1.7. Operaciones de gráficas

Es posible definir **operaciones entre gráficas**, es decir, operaciones que toman una o más gráficas para generar una gráfica nueva. Sean G una gráfica, $X \subsetneq V_G$ y $F \subseteq E_G$. Definimos la gráfica resultante de **eliminar el conjunto de vértices** X de G, como la subgráfica de G dada por $G-X = G[V_G \setminus X]$. También definimos la gráfica resultante de **eliminar el conjunto de aristas** F de G, como la subgráfica de G dada por $G - F = \langle V_G, E', \psi_G|_{E'} \rangle$ en donde E' es el conjunto $E_G \setminus F$. Si $X = \{x\}$ o $F = \{f\}$, escribimos simplemente $G-x$ o $G-f$ respectivamente, en lugar de escribir $G - \{x\}$ o $G - \{f\}$, según sea el caso. Una ventaja de trabajar con gráficas simples es que la cantidad de aristas posibles en una gráfica de cierto orden es limitada. Supongamos que tenemos G una gráfica simple, podemos definir el **complemento** de G, denotado como $\overline{G}$, como la gráfica $\langle V_G, [V_G]^2 \setminus E_G \rangle$. Esto es, la gráfica con los mismos vértices que G y teniendo de entre todas las aristas posibles exactamente las aristas que no tiene G. Podemos observar por ejemplo que, el complemento de P_4 es isomorfo a P_4 misma. Si fijamos $n \in \mathbb{Z}^+$, la gráfica $\overline{K_n}$ recibe un nombre especial que es **gráfica vacía de orden** n; porque como K_n posee todas las aristas posibles, es esperado que $\overline{K_n}$ no tenga aristas. En la Figura 1.13 se muestran los diagramas $\overline{P_4}$ y

$\overline{K_6}$.

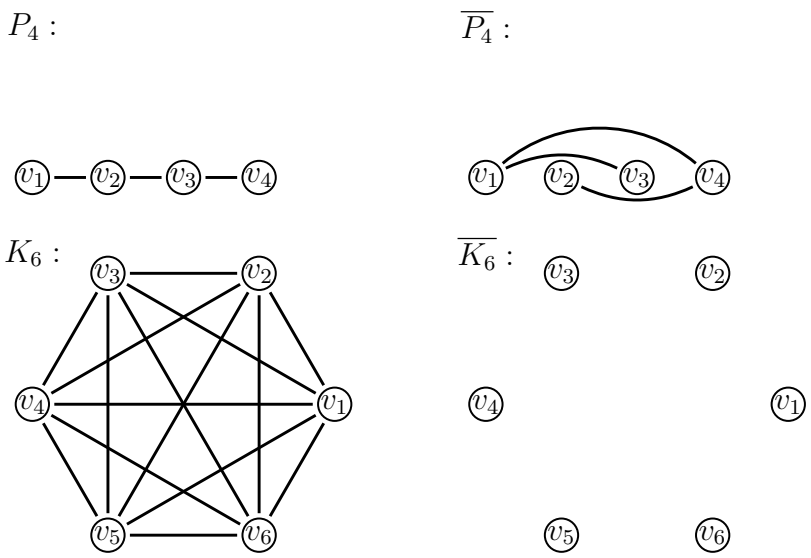


Figura 1.13: Gráficas $\overline{P_4}$ y $\overline{K_6}$.

En algunos problemas, es irrelevante que una gráfica tenga lazos, o más de una arista conectando un mismo par de vértices, o sea aristas paralelas. Motivados por lo anterior, para una gráfica arbitraria G, definimos la **gráfica subyacente** de G, denotada por $U(G)$, como sigue. La gráfica $U(G)$ es la gráfica simple con el mismo conjunto de vértices que G y donde la arista uv está en $E_{U(G)}$ solo cuando existe una arista de G que tiene extremos u y v. Notemos que, si G ya es una gráfica simple, se tiene que $G = U(G)$ con la definición de gráficas simples. En la Figura 1.14 podemos ver la gráfica subyacente de la gráfica G_1 de la Figura 1.2.

Consideremos ahora dos gráficas G y H. La **unión ajena** de G y H es la gráfica que consiste en considerar a las gráficas G y H como si fueran una sola, formalmente es $G + H = \langle (V_G \times \{0\}) \cup (V_H \times \{1\}), (E_G \times \{0\}) \cup (E_H \times \{1\}), \varphi \rangle$ donde φ es la función de adyacencia de $G \cup H$ y está definida como

$$\varphi(e,i) = \begin{cases} \{(u,0),(v,0)\} & \text{si} \quad i = 0 \text{ y } \psi_G(e) = \{u,v\}. \\ \{(u,1),(v,1)\} & \text{si} \quad i = 1 \text{ y } \psi_H(e) = \{u,v\}. \end{cases}$$

Si G y H son simples, podemos considerar otra unión en donde los vértices y aristas que compartan G y H no se duplican. Definimos esta operación simplemente como la **unión** de G y H, y es la gráfica dada por $G \cup H = \langle V_G \cup V_H, E_G \cup E_H \rangle$. También podemos definir la **unión completa**, **join** o **amalgamiento** de G y H como la

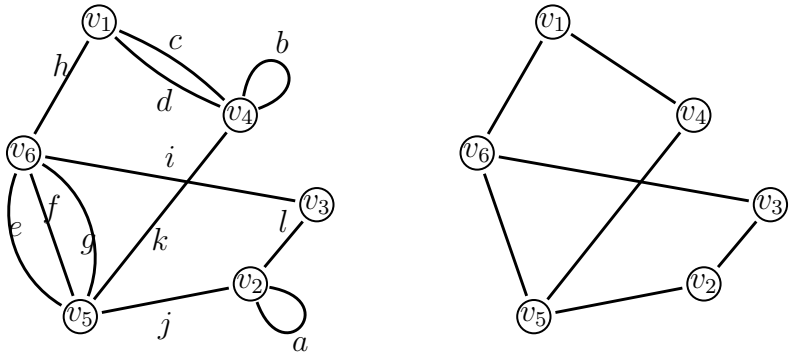


Figura 1.14: Gráficas $U(G_1)$ y $U(G_2)$.

gráfica que resulta de conectar en $G + H$ a cada vértice de G con cada vértice de H usando una arista nueva, formalmente es $G \oplus H = \langle (V_G \times \{0\}) \cup (V_H \times \{1\}), (E_G \times \{0\}) \cup (E_H \times \{1\}) \cup E', \varphi \rangle$ donde E' es el conjunto $\{(z, 2) \colon z \in V_G \times V_H\}$ y φ es la función de adyacencia de $G \oplus H$ que está definida como

$$\varphi(e,i) = \begin{cases} \{(u,0),(v,0)\} & \text{si} \quad i = 0 \text{ y } \psi_G(e) = \{u,v\}. \\ \{(u,1),(v,1)\} & \text{si} \quad i = 1 \text{ y } \psi_H(e) = \{u,v\}. \\ \{(u,0),(v,1)\} & \text{si} \quad i = 2 \text{ y } e = (u,v). \end{cases}$$

Supongamos que G y H son las gráficas isomorfas a K_4, con conjuntos de vértices $\{u_1, u_2, y, z\}$ y $\{v_1, v_2, y, z\}$ respectivamente, cumpliendo que $\{u_1, u_2\} \cap \{v_1, v_2\} = \varnothing$. En la Figura 1.15 podemos ver las gráficas $K_4 + P_3$, $C_5 \oplus K_2$ y $(G - \{zu_1, zu_2\}) \cup (H - \{yv_1, yv_2\})$.

1.8. Representación digital de gráficas

Para obtener la información de una gráfica como sus vértices, aristas, y qué vértices conectan las aristas, necesitamos explícitamente sus tres componentes (o dos, si es simple). Sin embargo, para poder manejar una gráfica con un algoritmo, hay que darle una entidad sintáctica a cada elemento de sus componentes. Para alcanzar este fin, utilizamos usualmente matrices y listas como las que a continuación definimos.

Sea G una gráfica de orden n y tamaño m, tomemos $\{u_1, \ldots, u_n\}$ y $\{e_1, \ldots, e_m\}$ sendas indizaciones de los conjuntos V_G y E_G respectivamente. Definimos la **matriz de adyacencia** de G como la matriz A de $n \times n$, de tal modo que, para $i, j \in \{1, \ldots, n\}$, si $i \neq j$, la entrada $A_{i,j}$ es el número de aristas en G cuyos extremos son u_i y u_j, pero si $i = j$, la entrada $A_{i,j}$ es 2 veces el número de lazos que inciden en u_i.

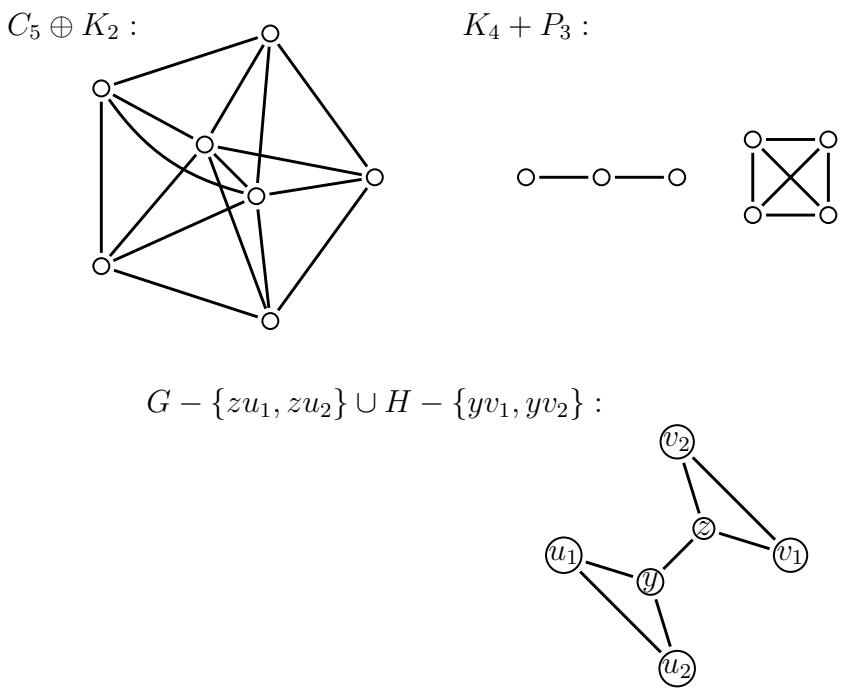


Figura 1.15: Gráficas $K_4 + P_3$, $C_5 \oplus K_2$ y $(G - \{zu_1, zu_2\}) \cup (H - \{yv_1, yv_2\})$.

La **matriz de incidencia** es la matriz I de $n \times m$, tal que para cada $i \in \{1, \ldots, n\}$ y $j \in \{1, \ldots, m\}$

$$I_{i,j} = \begin{cases} 0 & \text{si} \quad e_j \text{ no incide en } u_i \\ 1 & \text{si} \quad e_j \text{ incide en } u_i, \text{ pero no es lazo} \\ 2 & \text{si} \quad e_j \text{ incide en } u_i, \text{ y es lazo} \end{cases}$$

Para ejemplificar los conceptos anteriores, vamos a utilizar como ya es costumbre a la gráfica definida por la Figura 1.2, utilizando como indizaciones las enumeraciones ya dadas. Las matrices de adyacencia y de incidencia de G_1 las podemos ver en la Figura 1.16.

Como se puede ver, si tenemos la matriz de adyacencia o de incidencia de una gráfica, podemos reconstruir la estructura de la gráfica, pues tenemos toda la información de cuántos vértices y aristas tiene, además de cuantas aristas hay entre un par de vértices. En este trabajo no utilizaremos éstas representaciones de una gráfica, pues, como solo trabajaremos con gráficas simples, utilizaremos otra que consiste en tomar para cada vértice, una lista conteniendo los vértices a los cuales es adyacente. Dada una gráfica simple G de orden n, podemos tomar una indización de

Adyacencia:

$V\|V$	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	0	0	2	0	1
v_2	0	2	1	0	1	0
v_3	0	1	0	0	0	1
v_4	2	0	0	2	1	0
v_5	0	1	0	0	0	3
v_6	1	0	1	0	3	0

Incidencia:

$V\|E$	a	b	c	d	e	f	g	h	i	j	k	l
v_1	0	0	1	1	0	0	0	1	0	0	0	0
v_2	2	0	0	0	0	0	0	0	0	1	0	1
v_3	0	0	0	0	0	0	0	0	1	0	0	1
v_4	0	2	1	1	0	0	0	0	0	0	1	0
v_5	0	0	0	0	1	1	1	0	0	1	1	0
v_6	0	0	0	0	1	1	1	1	1	0	0	0

Figura 1.16: Matrices de adyacencia e incidencia de G_1.

$V_G = \{u_0, \ldots, u_{n-1}\}$. La **lista de adyacencias** de G es un arreglo $L[0, \ldots, n-1]$ con n localidades, de tal modo que para cualquier $i \in \{0, \ldots, n-1\}$, el elemento i-ésimo $L[i]$ de L es una lista cuyos elementos son exactamente los índices de los vértices en $N(u_i)$. Como G es simple, la lista de adyacencias contiene toda la información de la estructura de G, que son: el número de vértices de G y las aristas; esto porque para verificar si u_i y u_j son adyacentes, solo hay que verificar que i esté en la lista $L[j]$, o equivalentemente que j esté en la lista $L[i]$. Mencionemos algunas propiedades de la lista de adyacencia L que se desprenden directamente de la definición. Para cada $i \in \{0, \ldots, n-1\}$, la longitud de $L[i]$ es a lo más $n-1$, también tenemos que para $i, j \in \{0, \ldots, n-1\}$, se satisface, $i \in L[j]$ si y solo si $j \in L[i]$. Por último, hagamos la observación de que las listas $L[i]$ pueden estar ordenadas de varias formas, no necesariamente con el orden natural numérico. La lista de adyacencias de la gráfica G_2 definida en la Figura 1.3, se muestra en la Figura 1.17, tomando a v_i como el $(i-1)$-ésimo vértice.

1.9. Algoritmos

En el presente trabajo, entenderemos como un **algoritmo** a una sucesión finita de instrucciones que tiene una entrada de datos antes de comenzar a ejecutar

$$
\begin{aligned}
L[0] &= [1, 5, 3, 6] \\
L[1] &= [4, 0] \\
L[2] &= [7, 3, 6] \\
L[3] &= [5, 0, 6, 2] \\
L[4] &= [1, 5] \\
L[5] &= [4, 0, 3] \\
L[6] &= [0, 2, 3] \\
L[7] &= [2]
\end{aligned}
$$

Figura 1.17: Lista de adyacencia de la gráfica G_2.

las instrucciones, y una salida de datos que se devuelve al terminar de ejecutar las instrucciones. Bastará con esta definición intuitiva, aunque informal, pero muy popularizada. En el Algoritmo 1 presentamos una rutina, sumamente sencilla, que recibe como entrada un conjunto no vacío de naturales D y otro natural d, para decidir si d es divisor de algún elemento de D. El algoritmo devuelve *SI* en caso de que d divida a algún miembro de D, y devuelve *NO* de lo contrario. Este algoritmo nos será de utilidad para explicar parte de la notación y las llamadas "estructuras de control" que vamos a utilizar.

En el Algoritmo 1, lo primero que se hace es declarar una variable entera i, la estructura de control `for` del algoritmo indica que, para cada uno de los elementos $x \in D$, las instrucciones que se encuentran en su alcance se van a realizar. Si tenemos la variable u y el valor v, la instrucción $u \leftarrow v$ indica que u toma el valor v, en otras palabras, que a u se le asigna el valor v. Otras estructuras de control presentes son `while` e `if-then`. El bloque `while` consiste en aplicar las instrucciones que están en su alcance de forma iterada, mientras la condición que se encuentra frente a la palabra `while` se evalue como verdadera. Por otro lado, el bloque `if-then` consiste en ejecutar las instrucciones en su alcance, solo si resulta que la condición enfrente de la palabra `if` se evalua como verdadera. Sin embargo, en el Algoritmo 1 tenemos el bloque `if-then` seguido de un bloque `else`, esto significa que de evaluarse como falsa la condición que determina al bloque `if-then`, se realizaran las instrucciones en el alcance del bloque `else`. La expresión `return` significa que el algoritmo regresa como salida en este punto, lo que sea que se encuentre frente a la palabra `return`. Con esto basta para abordar la terminología de un algoritmo, esta forma de escribir

Algoritmo 1: Divisor

Input: Un conjunto de números naturales $D \neq \varnothing$ y un número natural d.
Output: SI en caso de que d divida a algún elemento de D, NO en caso contrario.

```
1  integer i;
2  for x ∈ D do
3  |  i ← 0;
4  |  while i ≤ x do
5  |  |  if i * d = x then
6  |  |  |  return SI;
7  |  |  else
8  |  |  |  i ← i + 1;
9  |  |  end
10 |  end
11 end
12 return NO;
```

un algoritmo es llamada un pseudocódigo para el algoritmo. Ahora introducimos las nociones de correctud y de complejidad temporal de un algoritmo.

La **correctud** es una propiedad de los algoritmos y consiste en que el algoritmo realice lo que se supone que debe hacer. Es decir, decimos que un algoritmo A es correcto respecto a la propiedad R y la colección de entradas I, si para cualquier entrada $w \in I$, podemos garantizar que el algoritmo siempre devuelve una salida $A(w)$ satisfaciendo $R(w, A(w))$. Por ejemplo, para el Algoritmo 1, afirmamos que el algoritmo es correcto respecto a la propiedad de devolver *SI* solo cuando d divide a algún elemento de D. En este caso, podemos pensar a I como la colección de parejas (D, d) con $D \subseteq \mathbb{N}$ no vacío, y $d \in \mathbb{N}$; mientras que R puede ser interpretada como $R((D, d), t)$: Existe $x \in D$ tal que $d \mid x$, si y solo si $t = SI$.

Proposición 1.9.1. *El Algoritmo 1 es correcto, es decir, devuelve SI solo cuando d divide a algún elemento de D.*

Demostración. Consideremos una entrada para el algoritmo, digamos (D, d) con $D \subseteq \mathbb{N}$ y $d \in \mathbb{N}$. Supongamos que dentro de la ejecución del `for` correspondiente a x se regresa *SI*. Entonces, se verifica para algún valor menor o igual a x, de la variable i, que $id = x$, pero esto significa que $d \mid x$. Por otro lado, supongamos que $y \in D$ es un múltiplo de d, o sea que $y = dk$ para algún $k \in \mathbb{N}$. Si no se devuelve *SI* es

una iteración del `for` anterior a la de y, eventualmente se llevará a cabo el bloque correspondiente a y, por lo que en este bloque se devolverá SI, pues, como $k \leq x$, la variable i eventualmente toma el valor de k y por lo tanto, se va a verificar la condición $y = id$ correspondiente al `if-then`. ■

Algunas veces se vuelve un poco complicado desarrollar la demostración de correctud para algunos algoritmos, pues, dentro de la prueba se hace referencia a distintas entidades y variables. Pero, estas entidades y variables pueden mutar su significado o valor, dependiendo del instante de tiempo, o bien, de la instrucción actual en la que se encuentra el procesamiento del algoritmo. Para esquivar estas dificultades, vamos a intentar ser lo más precisos posible para evitar ambigüedades, pero, sin incurrir en un formalismo ilegible.

Entremos ahora al concepto de **complejidad temporal**, para esto necesitamos primero algunas definiciones precisas. Nos referiremos con el mote de **operaciones aritméticas básicas**, a las siguientes operaciones entre números: suma, resta, multiplicación, división, módulo y exponenciación. También tenemos el nombre de **operaciones lógicas básicas** a las siguientes operaciones entre valores booleanos: NOT, AND y OR; además de sus derivados ya conocidos, como el XOR, la implicación y el bicondicional. Al conglomerado compuesto por las operaciones aritméticas básicas, las operaciones lógicas básicas, la asignación de un valor a una variable y el cálculo de satisfacibilidad de una desigualdad entre un par de números, se le conoce como **instrucciones básicas**. Las instrucciones básicas son las acciones más elementales que hace un algoritmo y en consonancia postulamos que el realizar cualquiera de estas instrucciones toma solo una unidad de tiempo. No es de sorprender que postulemos también en este trabajo, el hecho de que toda acción posible que pueda ser llevada a cabo por un algoritmo, es un conjunto de instrucciones básicas que se ejecutan de cierta forma. Entonces, podemos decir que una instrucción en un algoritmo toma tiempo t, con t el número de operaciones básicas constitutivas de dicha instrucción. Siguiendo la intención anterior, dado un algoritmo A que no se cicla con la entrada w, definiremos el **tiempo de ejecución** de A con entrada w, como la suma de los tiempos de cada una de las instrucciones que se llevan a cabo al ejecutar A con w, dicho tiempo lo denotamos como $T(A, w)$.

Tenemos interés en ver el tiempo de ejecución en función del "**tamaño de entrada**", más que en función de cada entrada en particular. Sin embargo, para esto necesitamos esclarecer a lo que nos referimos con tamaño, lo cual resulta ser otro problema complejo más y también optaremos por no sumergirnos demasiado en la teoría. El tamaño de una entrada lo entenderemos como una especie de consenso previo, por ejemplo, como en este trabajo veremos algoritmos sobre gráficas, las entradas para nuestros algoritmos van a ser justamente gráficas, además de uno que otro elemento

adicional. Por lo que, si tenemos una gráfica $\langle V, E\rangle$ como entrada para un algoritmo, no es descabellado pensar que el tamaño de nuestra entrada está determinado solamente por $n = |V_G|$ y $m = |E_G|$. Para dar variedad a la explicación, consideremos el Algoritmo 1. Una entrada de dicho algoritmo es de la forma (D, d) con $D \subseteq \mathbb{N}$ y $d \in \mathbb{N}$, así que podemos consensuar que el tamaño de tal entrada es $\sum_{x\in D} x$, ya que el número de instrucciones que hace el algoritmo solo está determinado por "qué tan grandes" son los elementos de D.

Sea A un algoritmo que toma entradas en la colección I. Supongamos que los elementos de I ya tienen cierto tamaño definido y que A no se cicla con ninguna de sus entradas. Podemos definir la **función de tiempo** del algoritmo A, como la función $T_A\colon \mathbb{N} \to \mathbb{R}^+$, dada por

$$T_A(n) = \begin{cases} 0, \text{ si no hay elementos de } I \text{ con tamaño } n. \\ \text{máx}\{T(A, w)\colon w \in I \text{ tiene tamaño } n\}, \text{ en otro caso.} \end{cases}$$

Veamos la definición de las clases $\mathcal{O}$ (O grande), que forma parte de la llamada **notación asintótica**. Si X y Y son conjuntos, denotaremos al conjunto de funciones con dominio X y codominio Y por ${}^XY$. Consideremos dos funciones $f, g\colon \mathbb{N} \to \mathbb{R}$. Decimos que f está asintóticamente acotada por g, escrito $f \preceq g$, cuando existan $c \in \mathbb{R}^+$ y $N \in \mathbb{N}$, tales que, para cualquier $n \geq N$ se satisface que $f(n) \leq cg(n)$. Con lo anterior, teniendo una función específica $g \in {}^{\mathbb{N}}\mathbb{R}$, podemos definir el conjunto **O grande** de g como $\mathcal{O}(g) = \{f \in {}^{\mathbb{N}}\mathbb{R}\colon f \preceq g\}$.

Para desvelar la noción intuitiva de la definición anterior, supongamos que tenemos una función $g \in {}^{\mathbb{N}}\mathbb{R}$ y un algoritmo A con su función de tiempo T_A, de tal forma que $T_A \in \mathcal{O}(g)$. Por definición, T_A está asintóticamente acotada por la función g, así, existen $c \in \mathbb{R}^+$ y $N \in \mathbb{N}$ tales que, para cualquier $n \geq N$, se cumple $T_A(n) \leq cg(n)$. De esta manera, la relación que hay entre $T_A(n)$ y $g(n)$, para $n < N$ no la sabemos, lo que de cierta forma es irrelevante porque solo es un conjunto finito. En cambio, para $n \geq N$ sabemos que $T_A(n) \leq cg(n)$, o sea que suponiendo sin pérdida de generalidad que c es un natural, tendremos la certeza de que para cualquier entrada w de A que tenga tamaño n, el agoritmo A en su ejecución con w, va a tomar a lo sumo $cg(n)$ unidades de tiempo, pues $T(A, w) \leq T_A(n) \leq cg(n)$. Dado que c es constante, podemos pensar que las ejecuciones del algoritnmo A para entradas de tamaño al menos N, son tan eficientes como c veces la función g. De cierto modo, los números mayores o iguales a N son casi la totalidad de $\mathbb{N}$, así que abusando un poco de la interpretación conceptual, podemos decir que el algoritmo A es tan eficiente como la función g, a excepción de un factor constante quizás. Es pertinente observar que el hecho de que $T_A \in \mathcal{O}(g)$, no implica de ningún modo la imposibilidad de que $T_A \in \mathcal{O}(f)$ para alguna otra función $f \in {}^{\mathbb{N}}\mathbb{R}$, o sea, no implica que g sea la cota

asintótica más pequeña que hay para T_A.

En un **análisis de complejidad temporal**, lo que buscamos es acotar asintóticamente la función de tiempo T_A de un algoritmo A por otra función, digamos $g \in {}^{\mathbb{N}}\mathbb{R}$. Después, se busca otra función $f \in {}^{\mathbb{N}}\mathbb{R}$ que sea simple y lo más pequeña posible en sus valores, de tal suerte que $g \preceq f$. Lo anterior se hace para poder concluir que $T_A \in \mathcal{O}(f)$ en virtud de la Proposición 1.9.2, siendo f una función relativamente simple y pequeña. Para familiarizarnos un poco con las clases $\mathcal{O}$, vamos a demostrar un par de resultados.

Proposición 1.9.2. *Sean f, g, u, v funciones de $\mathbb{N}$ en $\mathbb{R}$, tales que todas acotan asintóticamente a la función constante 0, esto es, $0 \preceq h$ para cualquier $h \in \{f, g, u, v\}$. Se satisface lo siguiente:*

1. *Si $f \preceq g$ y $g \preceq u$, entonces $f \preceq u$.*
2. *Si $f \preceq g$ y $k \in \mathbb{R}^+$, entonces $kf \preceq kg$.*
3. *Si $f \preceq g$ y $u \preceq v$, entonces $f + u \preceq g + v$.*
4. *Si $f \preceq g$ y $u \preceq v$, entonces $fu \preceq gv$.*

Demostración. Comencemos observando que para $h \in \{f, g, u, v\}$, existen $r \in \mathbb{R}^+$ y $M \in \mathbb{N}$ tales que para toda $n \geq M$, se cumple que $0 \leq rh(n)$, o sea que $0 \leq h(n)$. Puede que r y M varien conforme a la elección de h, sin embargo, tomando al máximo de tales M, podemos concluir que para $M \in \mathbb{N}$ fijo, sucede que todo $n \geq M$ satisface que $0 \leq h(n)$, sin importar quién sea h en $\{f, g, u, v\}$.

1. Por la definición, existen $c_1, c_2 \in \mathbb{R}^+$ y $N_1, N_2 \in \mathbb{N}$, tales que, para cualquier $n \geq N_1$ se tiene $f(n) \leq c_1 g(n)$, mientras que para todo $n \geq N_2$ se cumple $g(n) \leq c_2 u(n)$. Por lo que, si $n \geq N_1 + N_2$, $f(n) \leq c_1 g(n) \leq c_1(c_2 u(n)) = (c_1 c_2)u(n)$. Como $c_1 c_2 \in \mathbb{R}^+$, lo anterior prueba que $f \preceq u$.

2. Sean $c \in \mathbb{R}^+$ y $N \in \mathbb{N}$ tales que, para cualquier $n \geq N$, se cumpla $f(n) \leq cg(n)$. Tenemos que, si $n \geq N$, se cumple $kf(n) \leq k(cg(n)) = (kc)g(n)$. Como $kc \in \mathbb{R}^+$, podemos concluir que $kf \preceq kg$.

3. Sean $c_1, c_2 \in \mathbb{R}^+$ y $N_1, N_2 \in \mathbb{N}$, de tal suerte que para todo $n \geq N_1$ y todo $m \geq N_2$, se cumpla $f(n) \leq c_1 g(n)$ y $u(m) \leq c_2 v(m)$. Tomando $c = c_1 + c_2$, que claramente está en $\mathbb{R}$, tenemos que para $n \geq M + N_1 + N_2$ se va satisfacer que $f(n) + u(n) \leq c_1 g(n) + c_2 v(n) \leq cg(n) + cv(n) = c(g(n) + v(n))$. Así, podemos concluir que $f + u \preceq g + v$.

4. Sean $c_1, c_2 \in \mathbb{R}^+$ y $N_1, N_2 \in \mathbb{N}$, de tal suerte que, para todo $n \geq N_1$ y todo $m \geq N_2$, se cumpla $f(n) \leq c_1 g(n)$ y $u(m) \leq c_2 v(m)$. Así que, para cualquier $n \geq M + N_1 + N_2$, se tiene que $f(n)u(n) \leq c_1 g(n) c_2 v(n) = (c_1 c_2)(g(n)v(n))$. Como $c_1 c_2 \in \mathbb{R}^+$, podemos concluir que $fu \preceq gv$.

■

Proposición 1.9.3. *Sea $f\colon \mathbb{N} \to \mathbb{R}$ una función polinomial de grado $m \in \mathbb{N} \setminus \{0\}$, dada por $f(n) = \sum_{i=0}^{m} a_i x^i$, con $\{a_0, \ldots, a_m\} \subseteq \mathbb{R}$ y $a_m > 0$. Para $k \in \mathbb{N}$, denotemos por p_k a la función x^k con dominio $\mathbb{N}$. Se satisface que $f \in \mathcal{O}(p_m)$, pero $f \notin \mathcal{O}(p_{m-1})$.*

Demostración. Primero veamos que $f \in \mathcal{O}(p_m)$. Definamos $a = \text{máx}\{|a_i|\colon i \in \{0, \ldots, m\}\}$. Para cualquier $n \in \mathbb{N}$, tenemos que $f(n) = \sum_{i=0}^{m} a_i n^i \leq \sum_{i=0}^{m} a n^m = (m+1)a n^m = [(m+1)a]p_m(n)$. De esta forma, la constante $(m+1)a \in \mathbb{R}^+$ atestigua que $f \preceq p_m$.

Ahora probaremos que $f \notin \mathcal{O}(p_{m-1})$, para esto hay que verificar que $f \not\preceq p_{m-1}$, así que tomemos $c \in \mathbb{R}^+$ y $N \in \mathbb{N}$ arbitrarios. Definamos $a = \text{máx}\{|a_i|\colon i \in \{0, \ldots, m\}\}$. Para $n > 0$ se tiene que $f(n) = \sum_{i=0}^{i=m} a_i n^i = a_m n^m + \sum_{i=0}^{i=m-1} a_i n^i \geq a_m n^m - m a n^{m-1}$. Así, para cualquier $n > \frac{1}{a_m}(c + am) + 1$, se satisface que $f(n) \geq a_m n^m - m a n^{m-1} > a_m n^{m-1} \frac{1}{a_m}(c + am) - m a n^{m-1} = n^{m-1} c + n^{m-1} a m - m a n^{m-1} = c p_{m-1}(n)$. Como c y N fueron arbitrarios, se sigue de la definición que $f \not\preceq p_{m-1}$, y por lo tanto $f \notin \mathcal{O}(p_{m-1})$. ■

Utilizaremos los siguientes principios para acotar el número de instrucciones que hace un algoritmo en su ejecución, su veracidad la fundamentamos en intuición combinatoria.

Principio de la suma: Si tenemos dos instrucciones que se realizan una detrás de la otra, el tiempo que toma realizar ambas instrucciones es la suma de los tiempos que tomaron cada una de las instrucciones.

Principio del producto: Supongamos que tenemos un bloque `for` o `while` que se itera k veces, y tal que la totalidad de instrucciones dentro del bloque toma t unidades de tiempo en ejecutarse. El tiempo que ocupa el bloque en ejecutarse, desde antes de la primera iteración hasta terminar la última, es kt unidades de tiempo.

Para aplicar un análisis de tiempo volvamos con el Algoritmo 1, y mostremos que es un algoritmo $\mathcal{O}(n)$, o sea que su función de tienpo T satisface $T \in \mathcal{O}(n)$. Para esto, supongamos que (D, d) es cualquier entrada para el algoritmo, es decir, D es un subconjunto no vacío de $\mathbb{N}$ y $d \in \mathbb{N}$. Recordemos también que estamos bajo el acuerdo de que el tamaño de la entrada (D, d), el cual denotaremos por n, es $\sum_{x \in D} x$.

Proposición 1.9.4. *El Algoritmo 1 tiene complejidad temporal $\mathcal{O}(n)$.*

Demostración. Estas demostraciones se despachan usualmente de adentro hacia afuera, es decir, de los bloques que están anidados en lo más profundo hasta los más superficiales. En el bloque `if-then-else` se hacen solo dos operaciones, la primera es la verificación de la condición, mientras que la segunda es, ya sea el `return` en caso de entrar al `if-then`, o bien la asignación en caso de pasar al `else`. Para $x \in D$ fijo, el `while` realiza a lo más $x+1$ iteraciones, una por cada valor que toma i en el conjunto $\{0, \ldots, x\}$, así que, suponiendo $x \in D$ fijo, todo el `while` toma $2(x+1)$ unidades de tiempo. Por último, hagamos notar que el `for` hace una iteración por cada elemento de D, entonces, el `for` hace $\sum_{x \in D} 2(x+1) = 2(|D| + \sum_{x \in D} x) = 2(|D| + n)$ operaciones. Aparte del `for`, el algoritmo hace a lo sumo una operación más, la cual es devolver *NO* en caso de no haber devuelto *SI* antes. Por lo tanto, podemos concluir que el algoritmo ejecuta a lo más $1+2(|D|+n)$ instrucciones en su ejecución con (D, d). Como $|D|$ está acotado superiormente por $1+(\sum_{x \in D} x) = 1+n$, se tiene que $1+2(|D|+n) \leq 1+2(2n+1) = 4n+3$. Toda la argumentación anterior muestra que el Algoritmo 1 se ejecuta en a lo más $4n+3$ unidades de tiempo, con cualquier entrada que tenga tamaño n. Acabamos de mostrar que, de hecho, la función de tiempo está acotada superiormente por la función $4n+3$, en particular está acotada asintóticamente. Por la Proposición 1.9.2 podemos concluir que el algoritmo es $\mathcal{O}(n)$, porque $4n+3 \in \mathcal{O}(n)$. ■

Para finalizar la sección, exploraremos un algoritmo más, el cual nos será de utilidad para ordenar listas de adyacencias en tiempo lineal. Probaremos su correctud respecto a cierta afirmación, y desplegaremos un análisis de complejidad temporal para verificar que en efecto es lineal.

Un **tipo de dato abstracto** (*TDA*) es un esquema para definir objetos, el cual puede tener definidos algunos atributos y operaciones inherentes. Es importante observar que el tipo de dato abstracto es únicamente el esquema, o el molde en analogía, no es un objeto o entidad particular en sí. Una **estructura de datos** (*EDD*) es un tipo de dato abstracto que se emplea para almacenar u organizar datos, o sea, para almacenar de cierta manera específica ciertos objetos. Como es obvio, al ser cualquier EDD una especie de TDA, puede tener tener definidas operaciones y atributos. Las estructuras de datos utilizadas por nuestro siguiente algoritmo son los arreglos y las listas, así que vamos a definirlas brevemente.

Un **arreglo** es una estructura de datos A que se caracteriza por tener un número fijo y ordenado de localidades de memoria, desde 0 hasta un número antes que el total de localidades. Dicho número que define la "capacidad", se determina desde el momento en el que se crea la instancia, en este caso la instancia A de arreglo. No cabe duda de que tener una cantidad fija de memoria es una limitante fuerte de los arreglos, sin embargo, también tienen la ventaja de que el acceso a cualquier dato

almacenado toma solo una unidad de tiempo. Veamos las operaciones definidas en un arreglo. Primero tenemos la longitud, denotada por $|A|$, la cual devuelve el número que representa la capacidad de almacenamiento de A. Consideremos un índice i en el rango de A, esto es $0 \leq i < |A|$. Para almacenar el dato x en la localidad i-ésima de A, usamos $A[i] \leftarrow x$, mientras que para acceder al dato de la localidad i-ésima de A, utilizamos $A[i]$. Cabe mencionar que consensuamos el valor de $A[i]$ como 0, cuando no se haya almacenado ningún dato en $A[i]$ desde la creación de A.

Una **lista doblemente ligada** es una estructura de datos que también tiene localidades de memoria ordenadas, sin embargo, una lista no tiene almacenamiento fijo, pero esto resulta en cierto detrimento al tiempo usado para accesar sus localidades. Las localidades de memoria de una lista doblemente ligada, también llamadas **celdas**, guardan dos referencias, la primera apunta a la celda inmediata anterior (si hay una anterior), mientras que la segunda apunta a la celda inmediata siguiente (si hay una posterior). Consideremos una lista doblemente ligada L para explicar su funcionamiento y definir sus operaciones. Dada cualquier celda c de L, como ya anticipamos, hay tres cosas que guarda c. La primera es su valor actual, el cual lo denotamos por $valor_L(c)$; la segunda es un apuntador $siguiente_L(c)$ que hace referencia a la celda siguiente si la hay, o bien, es un apuntador nulo en caso de que c sea la última celda de L; la última es un apuntador $anterior_L(c)$ que hace referencia a la celda anterior si la hay, o bien, es un apuntador nulo en caso de que c sea la primera celda de L. Una propiedad primordial que asumiremos obvia, es que no se permite la autoreferencia, esto es, ninguno de los apuntadores de ninguna celda en L hace referencia a tal celda misma. Otro requisito para L es que tenga exactamente una celda cuyo apuntador $siguiente_L$ sea nulo, esta celda tiene el nombre de "última celda"; también requerimos que L tenga exactamente una celda cuyo apuntador $anterior_L$ sea nulo, esta celda recibe el nombre de "primera celda". La propiedad inherente que le da a L la categoría de lista doblemente ligada es la siguiente: si c_1 y c_2 son celdas de L, se satisface que, $siguiente_L(c_1) = c_2$ si y solo si $c_1 = anterior_L(c_2)$.

Tenemos una operación con un resultado booleano para saber si L es no vacía, esta operación la escribimos como $L \neq \varnothing$. También, en caso de que L no sea vacía, tenemos operaciones para recuperar, sin eliminarlas de L, a las celdas primera y última, estas operaciones las denotamos respectivamente como $primero(L)$ y $ultimo(L)$. Si L es vacía, claramente no podemos quitar celdas de L, pues no hay, sin embargo, podemos agregar una celda que tenga el valor x a L con la operación $addL(L, x)$ o $addR(L, x)$, estas operaciones hacen lo mismo cuando L es vacía, solo insertan un elemento en L. Para eliminar celdas de L o insetar nuevas celdas en caso de que L no sea vacía, requerimos tener la referencia a una celda c que forme parte de L. Para quitar la celda c de L sin alterar el orden de sus localidades restantes, utilizamos la

instrucción $del(L, c)$, esta instrucción quita a c de L reacondicionando, los apuntadores necesarios para no alterar el orden. Por otro lado, la operación $addR(L, c, x)$ almacena el valor x en una celda nueva, la cual es colocada inmediatamente después de c en L; mientras que $addL(L, c, x)$ almacena el valor x en una nueva celda que se posiciona justo a la izquierda de c en L. De nuevo, estas operaciones para insertar datos emplean un reordenamiento de las referencias necesarias para que el orden de L se mantenga íntegro. Cuando sea claro sobre la lista en la que hacemos las operaciones, omitiremos el subíndice de la lista. Un pequeño comentario es que, para insertar celdas al principio y al final de L, con $L \neq \varnothing$, no es necesario tener la referencia de ninguna celda, pues estas acciones equivalen respectivamente a $addL(L, primero(L), x)$ y $addR(L, ultimo(L), x)$. Es inmediato de las definiciones, que todas las operaciones definidas se realizan en una unidad de tiempo, claro está, teniendo ya la referencia a la celda necesaria; encontrar tal referencia es justamente el meollo de la complejidad temporal.

Para el resto de la sección, dado cualquier $k \in \mathbb{N}$, denotaremos al conjunto $\{0, \ldots, k-1\}$ por X_k. Si A es un conjunto arbitrario, un **ordenamiento** de A es una función biyectiva de A en $X_{|A|}$, en particular, si $A = X_{|A|}$, los ordenamientos de A son simplemente las permutaciones de A. Fijemos un conjunto de números A y un ordenamiento de A que llamaremos τ. Lo que nos interesa es el orden lineal sobre A inducido por τ, denotado por $<_\tau$, que es el orden de $X_{|A|}$ proyectado bajo τ^{-1}, esto es, $a_1 <_\tau a_2$ si y solo si $\tau(a_1) < \tau(a_2)$. La rutina $ordena(arr, \tau)$ hace lo siguiente: Toma como entrada un arreglo arr de listas doblemente ligadas con valores dentro de $X_{|arr|}$, sin repeticiones, de tal modo que; si el valor i está en $arr[j]$, entonces el valor j está en $arr[i]$. También toma como entrada un ordenamiento τ del conjunto $X_{|arr|}$. Procesa la entrada y devuelve un arreglo arr' que tiene la propiedad de tener las mismas listas que arr y en la misma posición, pero ordenadas crecientemente con el orden $<_\tau$.

Proposición 1.9.5. *Sean τ un ordenamiento de X_k y arr un arreglo de k listas doblemente ligadas sin repeticiones, de tal modo que cada lista de arr tiene valores en X_k, y satisface que si i es un valor en $arr[j]$, entonces el valor j es un valor en $arr[i]$. Supongamos que arr' es el arreglo devuelto por el Algoritmo 2 con τ y arr como entrada, resulta que arr' tiene las mismas listas y en la misma posición que arr, pero $<_\tau$-ordenadas de manera creciente.*

Demostración. Sea $0 \leq i < k$ fijo. Si x está en $arr[i]$, hay una celda y en $arr[valor(x)]$ con valor i, así que, en algún punto de la iteración del `while` correspondiente a $\tau(valor(x))$, se asigna el valor de x a una celda que se añade a $arr'[valor(y)] = arr'[i]$. Por otro lado, si x es una celda en $arr'[i]$, entonces x fue

Algoritmo 2: *ordena*(arr, τ)

```
1  /* k es la longitud de arr y no es parte de la entrada.        */
   Input: Un ordenamiento τ de X_k y arr un arreglo de k listas doblemente
          ligadas sin repeticiones, de tal modo que cada lista de arr tiene
          valores en X_k, y satisface que si i es un valor en arr[j], entonces el
          valor j es un valor en arr[i].
   Output: Un arreglo arr' de longitud k, tal que, para cada i ∈ X_n, arr'[i] es
           la misma lista que arr[i] pero <_τ-ordenada de forma creciente.
2  i ← 0, k ← |arr|;
3  arr' ← arreglo con k listas doblemente ligadas y vacías;
4  while i < k do
5  |  for x en la lista arr[τ^{-1}(i)] do
6  |  |  agregar una nueva celda al final de arr'[valor(x)] con el valor τ^{-1}(i);
7  |  end
8  |  /* La exploración de la lista arr[τ^{-1}(i)] en el for se lleva a
   |     cabo desde la primera celda hasta la última.              */
9  |  i ← i + 1;
10 end
11 return arr'
```

agregada en alguna iteración del `while`. Observemos que según el Algoritmo 2, la iteración donde x se añade es justamente $\tau(valor(x))$. La inserción de x ocurrió porque alguna celda y de $arr[\tau^{-1}(\tau(valor(x)))] = arr[valor(x)]$, satisfacía que $valor(y) = i$. Por lo tanto, $i = valor(y)$ estaba presente en $arr[valor(x)]$, de donde se sigue que $valor(x)$ debe estar presente en $arr[i]$.

Para ver la cuestión del orden, tomemos $0 \leq i < n$ arbitrario y x, y dos elementos de la lista $arr'[i]$, de tal modo que x está antes que y en la lista $arr'[i]$. Observemos que x se inserta en la iteración del `while` correspondiente a $\tau(valor(x))$, por otro lado, la inserción de y se hace en la iteración del `while` correspondiente a $\tau(valor(y))$. Como todas las inserciones se hacen al final de las listas, podemos concluir que x fue insertado en una itearción previa respecto en la que se insertó y, esto significa que $\tau(valor(x)) \leq \tau(valor(y))$. Por lo tanto $valor(x) \leq_\tau valor(y)$. ■

Para ver la complejidad temporal del Algoritmo 2, necesitamos establecer una métrica para el tamanõ de sus entradas. Dado que la entrada son, un arreglo arr y un ordenamiento de $X_{|arr|}$, el tamaño de la entrada solo dependerá de arr, pues el tamaño del ordenamiento está de cierto modo en función de arr. Sin embargo, el establecer el

tamaño de la entrada simplemente como $|arr|$ no tendría mucho sentido, ya que los elemento de arr son listas doblemente ligadas, listas cuyas longitudes no conocemos, pero afectan al número de pasos que hace el algoritmo. Dicho lo anterior, definiremos al tamaño de la entrada (arr, τ) para el Algoritmo 2 como $\sum_{i=0}^{|arr|-1} |arr[i]|$, donde $|arr[i]|$ denota a la longitud de la lista (el número de celdas que posee actualmente) almacenada en la localidad i de arr.

Proposición 1.9.6. *El Algoritmo 2 tiene complejidad temporal $\mathcal{O}(n+|arr|)$, donde $n = \sum_{i=0}^{|arr|-1} |arr[i]|$ para cualquier entrada válida arr del algoritmo.*

Demostración. Dentro del `for` solo se realiza una inserción al final de una lista doblemente ligada, esto nos lleva solamente una unidad de tiempo. Como el `for` hace una iteración por cada una de las celdas de $arr[\tau^{-1}(i)]$, el número de acciones que se ejecutan en el procesamiento de todo el bloque `for` es $|\tau^{-1}(i)|+1$, esto en la iteración del `while` correspondiente a $i \in \{0, \ldots, |arr|-1\}$. Por lo tanto, el algoritmo hace un total de $2+\sum_{i=0}^{i=|arr|-1}(|arr[\tau^{-1}(i)]|+1)$ operaciones, que claramente es $\mathcal{O}(n+|arr|)$. ■

Es menester mencionar aquí, que dada cualquier gráfica simple G con lista de adyacencias L_g, la lista L_g satisface todos los requisitos para ser tomada como entrada del Algoritmo 2. Por lo que al procesar L_g, y cualquier ordenamiento τ con el Algoritmo 2, obtenemos una lista L'_g que es una lista de adyacencias para G. Pero, tal que en L'_g, la lista correspondiente a la vecindad de cualquier vértice, está ordenada de manera creciente respecto a τ.

Capítulo 2

Árboles y BFS

2.1. Árboles

Diremos que una gráfica es **acíclica** o libre de ciclos cuando no exista un recorrido que forma un ciclo en dicha gráfica. Una clase particular de gráficas son los árboles, gráficas útiles para representar ciertas estructuras que tienen buenas propiedades y facilidad para ejecutar algoritmos sobre ellos. Una gráfica T es un **árbol**, si y solo si, es una gráfica conexa y acíclica. Como en una gráfica los lazos y dos aristas paralelas tomadas en sucesión forman un ciclo, los árboles por definición son gráficas sin lazos ni aristas paralelas, o sea simples. Un tipo de vértices destacado en un árbol son los que tienen grado a lo más 1, estos vértices se llaman **hojas**. Al conjunto de los vértices que son hojas en un árbol T, lo denotamos por $H(T)$. En la Figura 2.1 podemos ver dos árboles, uno con 5 hojas y otro con 6 hojas. Al respecto de las hojas tenemos el siguiente resultado.

Proposición 2.1.1. *Si T es un árbol de orden al menos 2, se satisface lo siguiente:*

1. *El árbol T tiene al menos dos hojas.*
2. *Si $h \in V_T$ es una hoja, la gráfica $T - h$ es un árbol.*

Demostración.

1. Sea $P = (u_0, \ldots, u_k)$ una trayectoria en T que es máxima por longitud. Observemos que al ser T una gráfica conexa con $|V_T| \geq 2$, podemos afirmar que $\ell(T) \geq 1$, o sea que $k > 0$. Si existiera $x \in N(u_0) \setminus V_P$, se tendría que xu_0P es una trayectoria de longitud mayor que P, entonces se debe cumplir que $N(u_0) \subseteq V_P$. Un argumento exactamente análogo al anterior muestra que

$N(u_k) \subseteq V_P$. Notemos que, de existir $i \in \{2, \ldots, k\}$ tal que $u_i \in N(u_0)$, tendriamos el ciclo $(u_0, \ldots, u_i, u_0)$ en T. Como T es acíclica, se sigue que $N(u_0) \subseteq \{u_0, u_1\}$ y por lo tanto $N(u_0) = \{u_1\}$. Analogamente se debe cumplir que $N(u_k) = \{u_{k-1}\}$. Así, concluimos que u_0 y u_k son hojas de T, distintas además, porque $k > 0$.

2. Sean $x, y \in V_{T-h}$ arbitrarios y tomemos $P = (x = u_0, \ldots, y = u_k)$ una xy-trayectoria en T. Para todo $i \in \{1, \ldots, k-1\}$ se tiene $u_{i-1}, u_{i+1} \in N(u_i)$, o sea que $d(u_i) \geq 2$. Entonces, como h es una hoja, no es posible que h sea u_i para algun $i \in \{1, \ldots, k-1\}$. Pero, también sabemos que $x, y \in V_{T-h} = V_T \setminus \{h\}$, así que h no es un vértice de P y por lo tanto P es una trayectoria en $T - h$. Al ser x y y arbitrarios, concluimos que $T - h$ es conexa. Ahora, como $T - h \subseteq T$, todo ciclo en $T - h$ es un ciclo en T, eso significa que $T - h$ es acíclica porque T es acíclica.

■

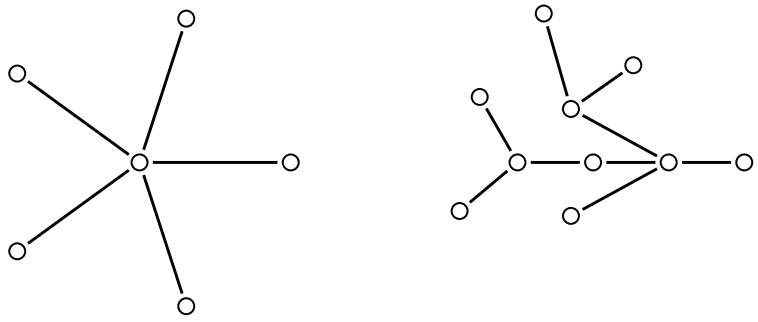


Figura 2.1: Dos árboles.

El siguiente resultado es una caracterización muy importante de los árboles.

Proposición 2.1.2. *Sea G una gráfica simple de orden n y tamaño m. Los siguientes enunciados son equivalentes:*

1. *La gráfica G es un árbol.*
2. *Para cualesquiera $x, y \in V_T$ distintos, existe una única xy-trayectoria en G.*
3. *Se cumple $m = n - 1$ y G es acíclica.*
4. *Se cumple $m = n - 1$ y G es conexa.*

Demostración.

1) ⇒ 2): Veamos la prueba por contrapositiva. Si hay $x, y \in V_G$ distintos, tales que no existe xy-trayectoria en G, entonces G no sería conexa. Supongamos ahora que, para algunos $u, v \in V_G$ diferentes, existen $P_1 = (u = x_0, \ldots, v = x_k)$ y $P_2 = (u = y_0, \ldots, v = y_l)$ dos uv-trayectorias distintas en G, consideremos sin pérdida de generalidad que $k \leq l$. Si $k = l$, como P_1 y P_2 son diferentes, debe haber $i \in \{1, \ldots, k-1\}$ de tal forma que $x_i \neq y_i$. Si $k < l$, entonces $y_k \neq x_k$, pues de lo contrario $x_k = v = y_k$ implicando $y_k = v = y_l$, lo cual es imposible por ser P_2 trayectoria. Lo anterior muestra que siempre podemos considerar $j = \text{mín}\{i \in \{0, \ldots, k\} \colon x_i \neq y_i\}$, notemos que como $x_0 = u = y_0$ y $x_k = v = y_l$ se cumple $0 < j < l$ ya sea que $k = l$ o $k < l$. Como $x_k = v \in \{y_j, \ldots, y_l\}$, podemos definir también $t = \text{mín}\{i \in \{0, \ldots, k\} \colon x_i \in \{y_j, \ldots, y_l\}\}$. Tenemos por la minimalidad de j que, para cada $i < j$, se satisface $x_i = y_i$ y por lo tanto no es posible que $x_i \in \{y_j, \ldots y_l\}$ porque P_2 es trayectyoria. De lo anterior tenemos que $j \leq t$, por lo cual $x_{j-1}P_1x_t = (x_{j-1}, \ldots, x_t)$ es una trayectoria, llamemos a esta última trayectoria Q_1. Definamos $s = \text{mín}\{i \in \{j, \ldots, l\} \colon x_t = y_s\}$ y dado s definamos también a la trayectoria $Q_2 = y_{j-1}P_2y_s = (y_{j-1}, \ldots, y_s)$. Si $i \in \{j, \ldots, t-1\}$, no es posible que $x_i \in \{y_j, \ldots, y_l\}$ por la minimalidad de t, en particular $x_i \notin \{y_j, \ldots, y_s\}$, tampoco es posible que $x_i = y_{j-1}$ porque $y_{j-1} = x_{j-1}$. Recordando que $x_t = y_s$ junto con lo anterior, nos permite concluir que $Q_1(Q_2)^{-1}$ es un ciclo en G.

2) ⇒ 3): Observemos primero que la hipótesis garantiza que G es acíclica, pues de existir un ciclo $C = (u_1, \ldots, u_k, u_1)$ en G, tendriamos las dos u_1u_k-trayectorias distintas en G dadas por la arista u_1u_k y por u_1Cu_k. Ahora solo resta que mostremos $m = n - 1$, esto lo haremos por inducción fuerte sobre $n > 0$. Si $n = 1$, la única gráfica simple de orden 1 es la gráfica trivial, que claramente es acíclica y satisface que $m = 0$ coincide con $n - 1 = 1 - 1$. Supongamos válido el enunciado para cualquier k, con $0 < k < n$. Sean $x \in V_G$ y $G_1, \ldots, G_l$ las componentes conexas de $G - x$. Fijemos $i \in \{1, \ldots, l\}$ y, para $y, z \in V_{G_i}$ distintos, definamos $A = \{P \colon P \text{ es una } yz\text{-trayectoria en } G_i\}$. Como G_i es conexa, se cumple $|A| \geq 1$, además $G_i \subseteq G$ por lo que $|A| \leq |\{P \colon P \text{ es una } yz\text{-trayectoria en } G\}| = 1$, o sea que $|A| = 1$. De lo anterior deducimos por hipótesis inductiva que, para cada $i \in \{1, \ldots, l\}$, la gráfica G_i es acíclica y satisface $|E_{G_i}| = |V_{G_i}| - 1$. Tomemos $i \in \{1, \ldots, l\}$ fijo y notemos que se satisface $|N(x) \cap V_{G_i}| = 1$. Comencemos viendo que $|N(x) \cap V_{G_i}| \geq 1$, para esto, consideremos para $y \in V_{G_i}$, una xy-trayectoria $P = (x = u_0, \ldots, y = u_k)$ en G. Observemos que $u_1 \in V_{G_i}$, porque $(u_1, \ldots, y = u_k)$ es una trayectoria en

$G - x$ y y es un vértice de la componente G_i, entonces $u_1 \in N(x) \cap V_{G_i}$. Lo anterior muestra que x tiene al menos un vecino en G_i, solo queda por mostrar que no puede tener más. Si hubiera $y, z \in N(x) \cap V_{G_i}$ distintos, podriamos tomar P una yz-trayectoria en G_i y tener las dos xz-trayectorias distintas en G dadas por xz y xyP, lo cual es imposible. Entonces, se debe satisfacer que para cada $i \in \{1, \ldots, l\}$, $|N(x) \cap V_{G_i}| = 1$. Concluimos observando que, por todo lo anterior $m = |N(x)| + \sum_{i=1}^{l} |E_{G_i}| = l + \sum_{i=1}^{l} (|V_{G_i}| - 1) = l - l + \sum_{i=1}^{l} |V_{G_i}| = |V_{G-x}| = n - 1$.

3) $\Rightarrow$ 4): Primero observemos que, por hipótesis se cumple que $m = n - 1$, así que solo falta verificar la conexidad de G, para esto procedemos por inducción sobre n. Para $n = 1$, como $m = n - 1$, se tiene forzosamente que G es la gráfica trivial, la cual es conexa. La hipótesis inductiva es que el enunciado es válido para gráficas de orden n, con $n > 0$ fijo. Supongamos que G tiene orden $n + 1$ y tamaño m tales que $m = (n + 1) - 1$, además también suponemos que G es acíclica. Sea $P = (u_0, \ldots, u_k)$ una trayectoria de longitud máxima en G, que debe satisfacer $k > 0$ porque $m = n > 1$, o sea, porque G no es vacía. Por la propiedad de máximo que tiene la longitud de P, se debe tener $N(u_k) \setminus V_P = \varnothing$, pero al ser G acíclica, también se cumple que $N(u_k) \subseteq \{u_{k-1}\}$, por lo tanto $N(u_k) = \{u_{k-1}\}$. De lo anterior, deducimos que $d(u_k) = 1$, esto nos permite garantizar que $|E_{G-u_k}| = m - 1 = n - 1 = |V_{G-u_k}| - 1$. Como G es acíclica y $G - u_k \subseteq G$, también $G - u_k$ es acíclica. Entonces, por hipótesis inductiva aplicada a la gráfica $G - u_k$ de orden n, podemos afirmar que $G - u_k$ es conexa. Observemos que, para todo $x \in V_G \setminus \{u_k\}$, los vértices x y u_{k-1} están conectados en $G - u_k$, así que también están conectados en G. Como en G claramente u_k y u_{k-1} están conectados, concluimos que G es conexa.

4) $\Rightarrow$ 1): También en esta implicación, usaremos inducción sobre el orden de la gráfica. Para $n = 1$, como $m = n - 1$, se tiene forzosamente que G es la gráfica trivial, la cual es acíclica. Consideremos la implicación válida para las gráficas de orden n, con $n > 0$ fijo. Supongamos que G tiene orden $n + 1$ y tamaño m tales que $m = (n+1) - 1$, además también supongamos que G es conexa. No es posible que $\delta_G \geq 2$, ya que de suceder esto, tendríamos, por la Proposición 1.1.1, que $2m = \sum_{u \in V_G} d(u) \geq 2|V_G| = (n+1)2 = 2n+2 = 2m+2$, lo cual es absurdo. Entonces, como G es conexa y de orden al menos 2, podemos tomar $v \in V_G$ que satisfaga $d(v) = 1$. Esto implica que, si v es un vértice de alguna trayectoria, debe ser por fuerza uno de sus extremos. Entonces, para cualesquiera $x, y \in V_G \setminus \{v\}$ diferentes, se cumple que $\{P \colon P$ es una xy-trayectoria en $G\}$ coincide con $\{P \colon P$ es una xy-trayectoria en $G - v\}$. De lo anterior, concluimos que, al

ser G conexa, también lo es $G - v$. Podemos aplicar la hipótesis inductiva a $G - v$ de orden n, porque $|E_{G-v}| = m - 1 = n - 1 = |V_{G-v}| - 1$ y $G - v$ es conexa, con esto tenemos que $G - v$ es un árbol, en particular es acíclica. Observemos que, si C fuera un ciclo en G, para todo $x \in V_G$ pasa que $d(x) \geq 2$, o sea que $v \notin V_C$ y por lo tanto C sería un ciclo en $G - v$. Gracias a lo anterior y a que $G - v$ es acíclica, podemos afirmar que G también es acíclica.

■

Corolario 2.1.3. *Todo árbol de orden n, tiene tamaño $n - 1$.*

Proposición 2.1.4. *Si G es una gráfica conexa, existe una subgráfica generadora $T \subseteq G$ que es un árbol, es decir, un* ***árbol generador*** *de G.*

Demostración. Como G misma es conexa y G solo tiene una cantidad finita de subgráficas, podemos tomar una subgráfica H de G, que sea mínima por contención con la propiedad de ser generadora y conexa. Primeramente notemos que H es simple, porque si no lo fuera, se tendría que $U(H)$ es conexa y $U(H) \subsetneq H$, lo cual no puede pasar. Supongamos que H tiene un ciclo $C = (x_1, \ldots, x_k, x_1)$ para ver que esto resulta absurdo, notemos que $k \geq 3$ por ser H simple. Resulta que $H - x_{k-1}x_k$ es también conexa, para verificar lo anterior, tomemos $u \in V_G \setminus \{x_k\}$ arbitrario y $P = (u = y_0, \ldots, x_k = y_l)$ una ux_k-trayectoria en H, la cual existe por ser H conexa. En caso de que $x_{k-1}x_k \notin E_P$, P es una ux_k-trayectoria en $H - x_{k-1}x_k$, de ocurrir $x_{k-1}x_k \in E_P$, se debe tener $x_{k-1} = y_{l-1}$, porque P es trayectoria y $x_k = y_l$. Sean $t = \text{mín}\{i \in \{0, \ldots, l\} \colon y_i \in V_C\}$ y C' es el segmento de C que conecta a y_t con $y_l = x_k$ y no tiene a la arista $x_{k-1}x_k$. Entonces, u y x_k están conectados en $H - x_{k-1}x_k$ mediante el camino $(u = y_0, \ldots, y_t)C'x_k$, así, resulta que hay una subgráfica propia de H que es conexa, lo que contradice la elección de H. De esta forma, H es acíclica además de ser conexa, o sea, H es un árbol. ■

Veamos en el siguiente resultado cómo se relacionan el tamaño y el orden, en gráficas acíclicas, y en gráficas conexas.

Proposición 2.1.5. *Sea G una gráfica arbitraria de orden n y tamaño m.*

1. *Si G es conexa, entonces $m \geq n - 1$.*

2. *Si G es acíclica, entonces $m \leq n - 1$.*

Demostración.

1. Como G es conexa, por la Proposición 2.1.4 existe un árbol generador T de G, entonces por el Corolario 2.1.3, $m \geq |E_T| = |V_T| - 1 = n - 1$.

2. Sean $G_1, \ldots, G_l$ las componentes conexas de G, como G es acíclica, todas sus componentes también lo son, o sea que las componentes de G son árboles. Por el Corolario 2.1.3, para cada $i \in \{1 \ldots, l\}$, $|E_{G_i}| = |V_{G_i}| - 1$. De lo anterior se deduce que, $m = \sum_{i=1}^{l} |E_{G_i}| = \sum_{i=1}^{l} (|V_{G_i}| - 1) = [\sum_{i=1}^{l} |V_{G_i}|] - l = n - l \leq n - 1$. ■

2.2. Árboles enraizados

Veamos ahora una clase aún más particular de gráficas, que también son manejables en cuanto a la ejecución de algoritmos, estos son los árboles enraizados que en escencia son lo mismo que los árboles, pero con un distintivo. Un **árbol enraizado** es un par $\langle T, r \rangle$ donde T es un árbol y r es un vértice específico de T, este vértice r se llama la **raíz** de T. Para hacer la redacción más natural, escribiremos la mayoría de las veces que, T es un árbol o árbol enraizado con raíz r, en lugar de escribir que $\langle T, r \rangle$ es un árbol enraizado. Nótese que un árbol T puede estar enraizado en cualquiera de sus vértices. Para desarrollar las definiciones pertinentes, vamos a fijar a T un árbol y $r \in V_T$ su raíz. En el árbol T enraizado en r, tenemos la costumbre de darle el nombre de **nodos** a los vértices, por otro lado, le seguimos llamando **hojas** a los vértices que son hojas en T, mientras que los vértices que no son hojas, se llaman **nodos internos**. Si x un nodo arbitrario en $\langle T, r \rangle$, por las propiedades de árboles, garantizamos la existencia de una única xr-trayectoria en T, esta trayectoria es la **rama** de x y la denotamos por $P_{x,T,r}$, omitiendo los subíndices T y r cuando son claros el árbol T y la raíz r a la que nos referimos. Usando las ramas de los vértices en el árbol enraizado, podemos definir algunas operaciones con respecto a la raíz, operaciones que definimos a continuación. Como utilizaremos al vértice x y su rama P_x, denotemos a P_x como $(x = s_0, \ldots, r = s_k)$. El **nivel** de x, está dado por $niv_{T,r}(x) = \ell(P_x) = d_T(x, r)$, omitiremos como siempre el subíndice, cuando sea claro el árbol en el que trabajamos el nivel. Si $x \neq r$, podemos definir el **padre** de x, denotado por $p_{T,r}(x)$, como el único vértice adyacente a x en la trayectoria P_x, con la notación que dimos a P_x resulta ser s_1 en este caso. Consideremos y otro nodo cualquiera del árbol enraizado, decimos que y es **ancestro** de x, cuando y es un vértice de P_x, por otro lado, y es **descendiente** de x, cuando x es un ancestro de y. Decimos que x es un **acenstro propio** o un **descendiente propio** de y, si x es un ancestro o un descendiente de y, según sea el caso, y además $x \neq y$. Algo intuitivo, pero que vale la pena definir concretamente, son los nodos hermanos. Si x

y y son dos nodos diferentes que tienen el mismo padre, decimos que x y y son nodos **hermanos**.

A manera de ejemplo, consideremos al árbol con raíz r representado en el diagrama de la Figura 2.2. Este árbol tiene hojas $a, b, c, d, e, f, g, h, i, j, k$ y l, por otro lado, tiene como nodos internos a r, u, v, w, x, y, z y t. También notemos que para el nodo y, su rama P_y es la trayectoria (y, x, w, r), además, podemos notar que x es padre de f, decendiente de r y ancestro de e, mientras que x no guarda ninguna de estas relaciones con el nodo z.

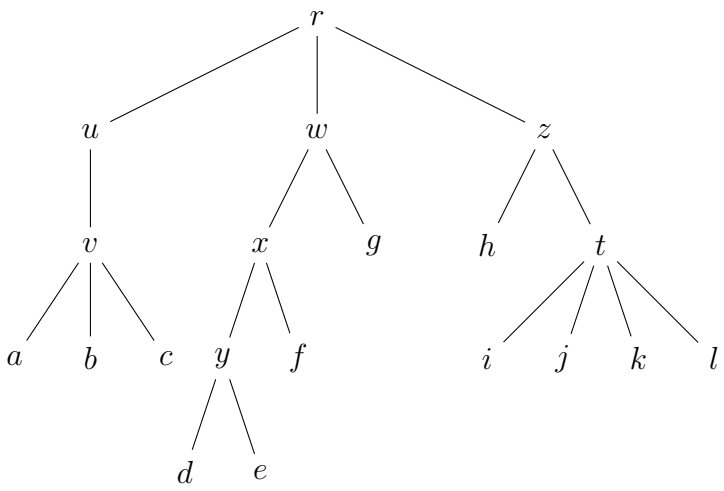


Figura 2.2: Árbol con raíz r.

Para cualquier nodo x en un árbol T con raíz r, con $x \neq r$, podemos escribir explícitamente P_x como $(x = s_0, \ldots, r = s_k)$, teniendo $k > 0$. Una observación pertinente es que, por definición, para cualquier $i \in \{0, \ldots, k-1\}$, $p(s_i) = s_{i+1}$, entonces, es fácil ver que, aplicando iteradamente k veces la función p a x obtenemos la raíz, esto es, $p^k(x) = r$, alternativamente $p^{niv(x)}(x) = r$ usando la notación del nivel. También, tomando otro nodo y, podemos considerar $m = \text{mín}\{i \in \{0, \ldots, k\} \colon s_i \in V_{P_y}\}$, y definimos al **mínimo común ancestro** de x con y como $mca(x, y) = s_m$. Si $P_y = (y = t_0, \ldots, r = t_l)$, observamos que $s_m = t_n$, donde $n = \text{mín}\{i \in \{0, \ldots, l\} \colon t_i \in V_{P_x}\}$, se sigue entonces que $mca(x, y) = mca(y, x)$. Una observación obvia que podemos hacer es que, si v es un nodo ancestro de u, entonces, $v \in V_{P_u}$ y por lo tanto $mca(v, u) = v$. En la Figura 2.2, podemos ver que $mca(h, l) = z$, mientras que $mca(v, t) = r$. Otra forma de definir el mínimo común ancestro, es dotar a V_T del orden $<_T = \{(u, v) \in V_T^2 \colon v \text{ es ancestro propio de } u\}$, y definir $mca(x, y)$ como el $<_T$-mínimo de los ancestros comunes de x y de y, o sea, de los nodos que son tanto

ancestros de u como ancestros de v. Respecto a esta última observación tenemos el siguiente resultado, pero para esto, conviene primero definir los segmentos izquierdo y derecho de una relación dada. Si X es cualquier conjunto, R es una relación binaria sobre X y x es cualquier elemento de X, definimos el **segmento izquierdo** de x respecto a R como $(\leftarrow, x)_R = \{y \in X : (y, x) \in R\}$, también definimos el **segmento derecho** de x respecto a R como $(x, \rightarrow)_R = \{y \in X : (x, y) \in R\}$, omitiendo el índice si no hay duda alguna de la relación a la que son relativos los segmentos. Por último, tenemos que $(\leftarrow, x] = (\leftarrow, x) \cup \{x\}$ y $[x, \rightarrow) = (x, \rightarrow) \cup \{x\}$.

Proposición 2.2.1. *Sea T un árbol con raíz r. Consideremos la relación $<_T = \{(u, v) \in V_T^2 : v$ es ancestro propio de $u\}$. La relación $<_T$ es un orden parcial en V_T y además, para cualesquiera nodos $x, y \in V_T$, se cumple que $mca(x, y) = \min([x, \rightarrow) \cap [y, \rightarrow))$.*

Demostración. Por definición, la relación $<_T$ es irreflexiva, así que solo hay que verificar la transitividad. Dados $x, y, z \in V_T$ tales que $x <_T y$ y $y <_T z$, se sigue por la definición que y es un acestro propio de x y z es un ancestro propio de y. Por la unicidad de las trayectorias en los árboles, el que y sea un ancestro propio de x, implica que P_y está contenida en la trayectoria $p(x)P_x r$, así que por tenerse $z \in P_y$, se sigue que $z \in V_{P_x} \setminus \{x\}$, o sea que z es un ancestro propio de x. Por lo tanto $x <_T z$, lo que muestra que $<_T$ es un orden parcial en V_T^2. Para lo siguiente, observemos que para cualquier nodo x, se tiene que $[x, \rightarrow) = V_{P_x}$.

Ahora, fijemos x y y dos nodos de T. Primero definamos el conjunto $A = [x, \rightarrow) \cap [y, \rightarrow)$ y veamos que A es un subconjunto linealmente ordenado y no vacío de V_T para que tenga sentido tomar el mínimo. Claramente A no es vacío, pues r está en A, por otro lado, el hecho de que A es linealmente ordenado, se sigue de que $A \subseteq [x, \rightarrow)$, siendo $[x, \rightarrow) = V_{P_x}$ un subconjunto linealmente ordenado de V_T^2. Ya que sabemos que A tiene mínimo, denotemos tal mínimo por z. Escribamos P_x como $(x = s_0, \ldots, r = s_k)$ y recordemos que $mca(x, y) = s_m$, donde $m = \min\{i \in \{0, \ldots, k\} : s_i \in P_y\}$. Para empezar notemos que $z \in A = V_{P_x} \cap V_{P_y}$, así que $z = s_j$ para algún $j \in \{0, \ldots k\}$. Como $s_m \in V_{P_x} \cap V_{P_y} = A$, se cumple que $s_m = z$ o $z <_T s_m$. No puede ocurrir que $z <_T s_m$, pues, esto implicaría que s_m es un vértice de la trayectoria $(p(z) = s_{j+1}, \ldots, r = s_k)$, o sea que $j + 1 \leq m \leq k$, lo cual es absurdo porque $z = s_j \in P_y$ con $j < m$. Por lo tanto, $z = s_m = mca(x, y)$. ■

Lo anterior nos garantiza que en un árbol enraízado $\langle T, r\rangle$, la relación $<_T$ es un orden parcial. Así, podemos considerar el orden parcial reflexivo que induce $<_T$, o sea, el orden $<_T \cup \{(x, x) : x \in V_T\}$, el cual denotaremos por $\leq_T$. Aunque ya se menciono antes, sin demostración por la simpleza del enunciado, que $mca(x, y) = mca(y, x)$, del resultado previo se desprende directamente una prueba de este hecho.

Teniendo un árbol enraizado $\langle T, r\rangle$, podemos considerar subestructuras de T que son árboles enraizados en algún nodo de T, los cuales van a heredar el orden $<_T$. Si x es un nodo de T, podemos considerar a T' como la subgráfica de T, inducida por el conjunto de vértices $(\leftarrow, x]$, o sea, por los vértices que son descendientes de x en T. Podemos ver que T' es una subgráfica conexa de T, pues, dado cualquier $y \in (\leftarrow, x]$, tenemos que el segmento de $P_{y,T,r}$ que va de y a x, está completamente contenido en $(\leftarrow, x]$, entonces T' es un árbol también. Al subárbol enraizado $\langle T', x\rangle$ de T, lo denotamos por T_x y se llama el **subárbol enraizado** de T en x. Resulta que como habiamos dicho, T' hereda el orden de T, esto se muestra en la siguiente proposición.

Proposición 2.2.2. *Sean $\langle T, r\rangle$ un árbol enraizado y $x \in V_T$. Dados z, y nodos del árbol enraizado T_x, se cumple que y es ancestro de z en T_x, si y solamente si, y es ancestro de z en T.*

Demostración. Las ramas de z en T y en T_x las denotaremos por P_z y Q_z respectivamente. Veamos la suficiencia del primer enunciado. Al ser x ancestro de z, se tiene que $x \in V_{P_z}$. Como todos los vértices en el segmento zP_zx pertenecen a $(\leftarrow, x] = V_{T_x}$, tenemos que zP_zx es una zx-trayectoria en T_x. Por la unicidad de trayectorias en los árboles, se tiene que $Q_z = zP_zx$. Como y es ancestro de z en T_x, se tiene que $y \in V_{Q_z} \subseteq V_{P_z}$. Por lo tanto y es ancestro de z en T. Prosigamos con la suficiencia del segundo enunciado. Como ya observamos en la implicación anterior, se tiene $Q_z = zP_zx$. Como y es ancestro de z en T, se cumple $y \in V_{P_z}$, pero como $y <_T x$, se debe tener que y es un vértice del segmento zP_zx, pues de lo contrario tendriamos que $x <_T y$, lo cual es imposible. Por lo tanto y es un vértice del segmento zP_zx, o sea $y \in V_{Q_z}$, implicando que y es un ancestro de z en T_x.

■

En la Figura 2.3 podemos ver al árbol enraizado de la Figura 2.2 y a su subárbol T_x. A continuación, vamos a demostrar algunos resultados que nos serán de utilidad en el desarrollo ulterior de este trabajo.

Proposición 2.2.3. *Sea T un árbol enraizado en r. Dados $x, y, z \in V_T$ tales que $mca(x, y) <_T mca(x, z)$, se satisface que $mca(y, z) = mca(x, z)$.*

Demostración. Claramente $mca(x, z)$ es un ancestro de z, además, $mca(x, z)$ es ancestro de y porque $y \leq_T mca(x, y) <_T mca(x, z)$. De lo anterior se sigue que $mca(x, z)$ es un ancestro común de y y de z, lo cual implica que $mca(y, z) \leq_T mca(x, z)$. Busquemos una contradicción, suponiendo que $mca(y, z) <_T mca(x, z)$. De ocurrir lo anterior, no puede ser que $mca(x, y) \leq_T mca(y, z)$, ya que esto implicaría que $mca(y, z)$ es un ancestro común de x y de z con $mca(y, z) <_T mca(x, z)$, lo cual es

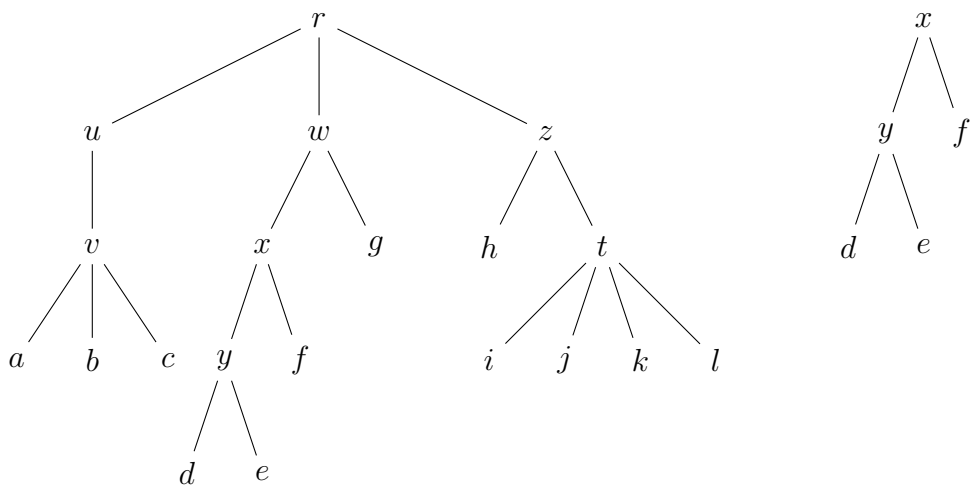


Figura 2.3: Árbol T con raíz r y subárbol T_x.

imposible. Entonces, dado que ambos $mca(y,z)$ y $mca(x,y)$ están en P_y, debe ocurrir que $mca(y,z) <_T mca(x,y)$, pero esto implica que $mca(x,y)$ es un ancestro común de z y de x con $mca(x,y) <_T mca(x,z)$, lo cual también es absurdo. De este modo concluimos la imposibilidad de que $mca(y,z) <_T mca(x,z)$, pero como ya mostramos que $mca(y,z) \leq_T mca(x,z)$, se debe cumplir que $mca(y,z) = mca(x,z)$. ■

Proposición 2.2.4. *Sean $\langle T, r\rangle$ un árbol enraizado y $n \in V_T$. Para cualesquiera $u \in V_T \setminus V_{T_n}$ y $v \in V_{T_n}$, se satisface que $mca(u,v) = mca(u,n)$.*

Demostración. Como tanto $mca(u,v)$ como $mca(u,n)$ están en P_u, se debe tener $mca(u,n) <_T mca(u,v)$ o $mca(u,v) \leq_T mca(u,n)$. No es posible que $mca(u,n) <_T mca(u,v)$, pues, por la Proposición 2.2.3, tendríamos que $n = mca(n,v) = mca(u,v)$, pero esto implicaría que n es ancestro de u y por lo tanto $u \in V_{T_n}$, lo cual es absurdo. Así que, tenemos $mca(u,v) \leq_T mca(u,n)$. Observemos que tampoco puede ocurrir $mca(u,v) <_T mca(u,n)$, ya que de tenerse esto, por la Proposición 2.2.3 de nuevo, tendríamos que $n = mca(v,n) = mca(u,n)$, implicando otra vez que $u \in V_{T_n}$, lo cual sabemos que es falso. Por lo tanto, podemos concluir que $mca(u,v) = mca(u,n)$. ■

2.3. Búsqueda en amplitud

El siguiente algoritmo es una de las versiones en pseudo-código del algoritmo conocido como **BFS** (*Breadth-first search*). Este algoritmo recibe como entrada a

una gráfica conexa con un vértice particular de esa gráfica, posteriormente construye varias funciones, una de ellas es la función de parentesco p, tal función se utiliza para construir un árbol generador de la gráfica. Antes de continuar, necesitamos abordar algunos conceptos de los que hace uso este algoritmo y algoritmos posteriores.

La estructura de datos empleada en BFS se llama **cola**, o queue en inglés. Primero, la manera en la que una cola Q almacena datos es conocida como **FIFO** (*First In First Out*), para explicar lo anterior, supongamos que se almacenaron n datos en Q, digamos $x_1, \ldots, x_n$, y se insertaron en ese orden; o sea, x_1 primero, luego x_2, etcétera, hasta finalizar con x_n. La propiedad FIFO significa que cuando Q retire de su almacenamiento a los datos almacenados, esta lo hará en el mismo orden en el que entraron, es decir, primero sale x_1, luego x_2, etcétera, de ahí el término *First In First Out*. Segundo, como ya mencionamos, las colas cuentan con operaciones o métodos, los cuales describimos aquí utilizando la notación Q para hacer referencia a una cola genérica. La cola cuenta con una operación para determinar si no está vacía, en general la vamos a denotar simplemente como $Q \neq \varnothing$. Para añadir datos a Q tenemos la operación $add(Q, x)$, que agrega a x en el almacenamiento de Q siguiendo la regla FIFO. Se pueden eliminar datos de Q, siguiendo la regla FIFO, con la operación $del(Q)$, así, $del(Q)$ elimina al elemento de Q con más antigüedad. Con la operación $head(Q)$ recuperamos, sin eliminarlo de Q, al elemento de mayor antigüedad que almacena Q. No es difícil ver que, una cola la podemos modelar con una lista doblemente ligada, en donde todas las inserciones se hacen al principio y las eliminaciones se hacen al final. Desde luego, se sigue que todas las operaciones de una cola se hacen en tiempo constante.

Mencionamos en esta parte otra estructura de datos, la **cola de prioridades**. Una cola de prioridades es una estructura de datos Q, la cual es una cola cuyos elementos tienen un orden parcial, digamos $\trianglelefteq$, definido en todo momento. Como en toda cola, se debe seguir el orden FIFO para agregar o quitar elementos, sin embargo, antes de considerar FIFO, se considera la prioridad determinada por $\trianglelefteq$, de allí el nombre. O sea, $head(Q)$ es el elemento más antigüo de Q, pero entre los $\trianglelefteq$-mínimos, mientras que $tail(Q)$ es el elemento más reciente de Q, pero entre los $\trianglelefteq$-máximos. Así, la operación $del(Q)$ elimina a $head(Q)$ de Q, mientras que $add(Q, x)$ debe insertar a Q en el lugar que le corresponde considerando a $\trianglelefteq$, pero también la regla FIFO.

Dentro de un algoritmo, es usual dotar de colores a ciertos objetos para marcarlos. Por ejemplo, se utiliza la instrucción "colorear a x de azul", para indicar que desde ese momento x tiene un color y ese color es azul, esto nos permite hacer posteriormente una verificación en tiempo constante de si tal o cual objeto ya tiene tal o cual color. La acción de colorear objetos toma tiempo constante, pues, de cierto

modo es equivalente a realizar varias asignaciones, ya que podemos simular el color con un atributo booleano que simplemente indique si el objeto en cuestión está ya coloreado o no.

Algoritmo 3: Breadth First Search

Input: Una gráfica conexa G con un vértice distingido r.
Output: Funciones de parentesco p, nivel ℓ y tiempo de exploración t.

```
1  Q ← ∅; i ← 0;
2  i ← i + 1;
3  colorear a r de negro;
4  add(Q, r);
5  t(r) ← i, p(r) ← ∅, ℓ(r) ← 0;
6  while Q ≠ ∅ do
7  |  x ← head(Q);
8  |  if x tiene un vecino y sin colorear then
9  |  |  i ← i + 1;
10 |  |  colorear a y de negro;
11 |  |  add(Q, y);
12 |  |  t(y) ← i, p(y) ← x, ℓ(y) ← ℓ(x) + 1;
13 |  else
14 |  |  /* Como x = head(Q), se elimina a x de Q                */
15 |  |  del(Q);
16 |  end
17 end
18 return (p, ℓ, t)
```

Lo que vamos a querer del Algoritmo 3, y en lo que vamos a basar su correctud, es en que podamos utilizar la función p que devuelve para poder construir un árbol enraizado en r. Dicho árbol va a ser de hecho generador y además, el nivel de cada vértice en el árbol enraizado será justamente la distancia de ese vértice hacia r. Por otro lado, vamos a mostrar que la complejidad temporal del Algoritmo 3 es de $\mathcal{O}(n+m)$, donde n es el orden de la gráfica y m es el tamaño. Comenzaremos con la correctud, pero para esto utilizaremos un resultado previo.

Lema 2.3.1. *Sean G una gráfica conexa, $r \in V_G$ y (p, ℓ, t) las funciones que devuelve la aplicación del Algoritmo 3 a la gráfica G con vértice distinguido r. Se satisface lo siguiente:*

1. *Todo vértice de G es coloreado de negro en algún momento, o sea, todo vértice es explorado.*

2. *Si $u, v \in V_G$ son tales que $t(u) < t(v)$, entonces $\ell(u) \leq \ell(v)$.*

3. *Para cualquier $uv \in E_G$, se satisface que $|\ell(u) - \ell(v)| \leq 1$.*

Demostración.

1. Procedamos por inducción sobre la distancia hacia r, esto basta para cubrir a todos los vértices de G, pues G es conexa. Para el caso base, observemos que r es el único vértice de G con distancia 0 hacia él mismo, y claramente r se colorea de negro al principio del algoritmo. Supongamos válido el enunciado para $k \in \mathbb{N}$ fijo y mostremos su validez para $k+1$. Si $x \in V_G$ es tal que $d(x, r) = k + 1$, tomemos a z como el penúltimo vértice en una rx-trayectoria de longitud $k + 1$. Se debe cumplir que $d(z, r) = k$, de esta manera, utilizando la hipótesis inductiva tenemos la certeza de que eventualmente z se colorea de negro. Por lo cual, z entra en algún punto a Q, y por ende, hay un momento en el que $z = \mathit{head}(Q)$. En tal instante en el que comienza a satisfacerse $z = \mathit{head}(Q)$, por la constitución del Algoritmo 3, se colorean tarde o temprano todos los vecinos aún no coloreados de z, así que se va a colorear eventualmente a x, o bien x ya estaba coloreado en ese instante.

2. Fijemos $k \in \mathbb{N} \setminus \{0\}$ y consideremos el enunciado $\varphi(k)$: Para todo $l \geq k$ y cualesquiera $u, v \in V_G$ tales que $t(u) = k$ y $t(v) = l$, se satisface que $\ell(u) \leq \ell(v)$. Veamos que todos los naturales mayores que 0 cumplen φ usando inducción fuerte sobre k. Si $k = 1$, el único vértice cuya imagen bajo t es 1 resulta ser r y claramente $\ell(r) = 0 \leq i$ para cualquier $i \geq 0$. La hipótesis de inducción es que para $k > 1$ fijo, todos los naturales menores a k y mayores a 0 satisfacen φ. Para continuar con el paso inductivo, supongamos que $u, v \in V_G$ y $l \geq k$ son tales que $t(u) = k$ y $t(v) = l$. Denotemos por x al vértice que satisfacía $x = \mathit{head}(Q)$ cuando u fue insertado en Q, de la misma forma denotemos por z a la cabeza de Q en el momento en que v entró a Q. Como $t(u) < t(v)$, el vértice v todavía no está coloreado cuando u es insertado en Q. Si ocurriera $t(z) < t(x)$, en el momento de agregar a u en Q, o sea mientras $x = \mathit{head}(Q)$, el vértice z ya tendría que haber salido de Q y por ende, el vértice v ya tuvo que haber sido coloreado. Dada la imposibilidad de lo anterior, se sigue que $t(x) \leq t(z)$. Usando la hipótesis inductiva con $t(x) < t(u) = k$, podemos deducir que $\ell(x) \leq \ell(z)$, de donde se sigue $\ell(u) = \ell(x) + 1 \leq \ell(z) + 1 = \ell(v)$.

3. Mostraremos usando inducción fuerte que para cualquier $k > 0$ se satisface lo siguiente: si $u, v \in V_G$ son tales que $uv \in E_G$, $t(u) = k$ y $t(u) < t(v)$, entonces $\ell(v) \leq \ell(u) + 1$. Tenemos para el caso base $t(r) = 1$, pero este caso es sencillo, porque al ser r el primer vértice en entrar a Q, todos los vecinos de r entran a Q cuando $r = head(Q)$. Por lo tanto, si $vr \in E_G$, entonces $\ell(v) = \ell(r) + 1 = 1$. Supongamos el enunciado válido para todos los naturales positivos menores a k, con fijo $k > 1$ fijo. Dados $u, v \in V_G$ tales que $uv \in E_G$, $t(u) = k$ y $t(u) < t(v)$, podemos considerar el vértice z que era cabeza de Q en el momento en el que v fue insertado en Q. Tenemos dos opciones para v cuando u se pone a la cabeza de Q. La primera opción es que v ya esté coloreado, lo que implica que se coloreó cuando z era cabeza de Q, pero esto ya pasó, o lo que es lo mismo, z ya salió de Q. La segunda opción es que v todavía no haya sido coloreado aún y por lo tanto será coloreado mientras u es cabeza de Q, pues v es un vecino de u. Del primer caso se desprende que $t(z) < t(u)$, utilizando que $zv \in E_G$ y la hipótesis inductiva con $t(z)$, concluimos en este caso que $\ell(v) \leq \ell(z) + 1$, pero por la compatibilidad de las funciones t y ℓ respecto al orden, tenemos que $\ell(z) \leq \ell(u)$, o sea que $\ell(v) \leq \ell(u) + 1$. En el segundo caso que es $u = z$, tenemos directamente por la definición de z que $\ell(v) = \ell(z) + 1 = \ell(u) + 1$.

■

Proposición 2.3.2. *Sean G una gráfica conexa, $r \in V_G$ y (p, ℓ, t) la tercia de funciones devueltas por la ejecución del Algoritmo 3 con entrada G y r. El algoritmo BFS es correcto. Es decir, si T es la gráfica inducida por el conjunto de aristas $\{up(u) \colon u \in V_G \setminus \{r\}\}$, resulta que T es un árbol generador de G, donde el nivel de los vértices en el árbol enraízado $\langle T, r\rangle$ coincide con los valores de la función ℓ, que a su vez coincide con la distancia hacia r en G.*

Demostración. Vamos a verificar que cualquier $u \in V_G$ está conectado con r en T, y que además $\ell(u) = d_G(u, r)$, esto usando inducción fuerte sobre los valores de ℓ. El único vértice cuya imagen bajo ℓ es 0, es el vértice r, y claramente r está conectado con r mismo en T, además se tiene $\ell(r) = 0 = d_G(r, r)$. Supongamos que para $k > 0$ fijo se satisface la hipótesis inductiva, esto es, que cualquier $v \in V_G$ tal que $\ell(v) < k$, debe estar conectado con r en T y también debe satisfacer la igualdad $d_G(v, r) = \ell(v)$. Supongamos que $u \in V_G$ es tal que $\ell(u) = k$, consideremos adicionalmente a $z \in V_G$, como el vértice que satisfacía $z = head(Q)$ en el momento en el que u se añadió a Q. Tenemos que $\ell(z) < \ell(z) + 1 = \ell(u)$, así que por hipótesis inductiva, z está conectado con r en T y $d_G(z, r) = \ell(z)$. Como $z = p(u)$, la arista uz se encuentra en T, y por lo tanto u está conectado con r en T. Por otro lado, tenemos que $d_G(u, r) \leq d_G(z, r) + 1 = \ell(z) + 1 = \ell(u)$, esto por la existencia de la arista uz. Ahora, sean

$m = d_G(u, r)$ y $(x_0 = r, \ldots, x_m = u)$ cualquier ru-trayectoria en G de longitud m. Por el Lema 2.3.1, se tiene que $\ell(u) = \ell(u) - \ell(r) = \sum_{i=1}^{m} [\ell(x_i) - \ell(x_{i-1})] \leq \sum_{i=1}^{m} 1 = m$, esto prueba que $m = d_G(u, r) = \ell(u)$.

Una observación algo obvia es que T no puede tener lazos, ni ciclos formados por dos aristas paralelas, esto por la condición del `if`. Para ver que T no tiene ciclos de mayor longitud, busquemos una contradicción, esto suponiendo que existe un ciclo $C = (x_0, \ldots, x_k)$ en T con $k \geq 3$. Sea $j \in \{0, \ldots, k-1\}$ tal que $\ell(x_j) = \text{máx}\{\ell(x_i) \colon 0 \leq i < k\}$. Según el Algoritmo 3, para todo $u \in V_G \setminus \{r\}$, se cumple que $\ell(u) = \ell(p(u)) + 1 > \ell(p(u))$. Como $\ell(x_j)$ es máximo en los valores de ℓ sobre los vértices de C, no es posible que $p(x_{j+1}) = x_j$, pero C es subgráfica de T, así que por definición de T y lo anterior, se sigue que $p(x_j) = x_{j+1}$. En el siguiente argumento, que es casi idéntico al previo, vamos a suponer que u es x_{j-1} si $j > 0$, mientras que u será x_{k-1} si $j = 0$. De nuevo, al tenerse $ux_j \in E_T$, se satisface $p(u) = x_j$, o bien, se cumple $p(x_j) = u$. Sin embargo, ya establecimos que $p(x_j) = x_{j+1}$, con $x_{j+1} \neq u$ porque $k \geq 3$. Por lo que, no puede ocurrir que $p(x_j) = u$, pues, eso implicaría que x_j tiene dos padres, contradiciendo que p es una función. De lo anterior, se sigue que $p(u) = x_j$, o sea que $\ell(x_j) < \ell(x_j) + 1 = \ell(u)$, contradiciendo la cualidad de máximo que tiene $\ell(x_j)$. Por lo tanto, T debe ser acíclica, esto junto al párrafo previo, nos permite concluir que T es un árbol generador de G, además de que la función ℓ coincide con la distancia sobre G hacia r.

Queda por ver que el nivel de los nodos en el árbol enaraizado $\langle T, r\rangle$, coincide con los valores de la función ℓ. Para que no haya confusión, utilizaremos el símbolo ℓ simple para referirnos a la función ℓ, parte de la salida del Algoritmo 3, mientras que ℓ_T hará referencia al nivel de los nodos en el árbol enraizado $\langle T, r\rangle$. Vamos a usar inducción sobre el nivel de los nodos en el árbol. El único vértice cuyo nivel en $\langle T, r\rangle$ es 0, resulta ser r, y desde luego se satisface que $\ell_T(r) = 0 = \ell(v)$. Fijemos $k > 0$, y supongamos el resultado válido para todos los vértices que tengan nivel menor que k en $\langle T, r\rangle$. Si u resulta ser un nodo de $\langle T, r\rangle$ tal que $\ell_T(u) = k$, podemos tomar a v, el segundo nodo en P_u, es decir, el que sigue después de u. Dado que $P_v = vP_ur$, se sigue que $\ell_T(v) = k - 1$. Aplicando la hipótesis inductiva a v, tenemos que $\ell(v) = \ell_T(v)$. Como $uv \in E_T$, se debe satisfacer $p(u) = v$ o $p(v) = u$, aunque $p(v) = u$ no es cierto, ya que $\ell(v) = d_G(v, r) \leq d_G(u, r) = \ell(u)$. Por lo tanto, se satisface $p(u) = v$, de donde se sigue que $\ell_T(u) = \ell_T(v) + 1 = \ell(v) + 1 = \ell(p(u)) + 1 = \ell(u)$. ■

Proposición 2.3.3. *Denotemos por n al orden de una gráfica y por m a su tamaño. Resulta que el Algoritmo 3 es $\mathcal{O}(n + m)$.*

Demostración. Supongamos que la entrada es una gráfica conexa G y un vértice distinguido $r \in V_G$. Desde el comienzo del algoritmo, hasta el momento previo de

comenzar el `while`, el algoritmo solo hace un número constante de operaciones, llamemos c_1 a dicho número. Lo siguiente es una explicación más detallada de cómo podemos verificar la condición del bloque `if-then`. Esta descripción la damos para garantizar que el tiempo de ejecución no excede la cota que queremos; en el Algoritmo 3 no se encuentra especificado esto para no volver el pseudo-código más pesado. Primero notemos que podemos saber cuándo es la primera iteración del `while` donde un vértice x es la cabeza de Q, esto porque tal iteración sucede inmediatamente después de que quitamos a la antigua cabeza de Q. Para verificar la condición del bloque `if-then`, en la primera iteración del `while` donde x es la cabeza de Q, podemos crear una lista con los elementos de $N(x)$ que todavía no están coloreados, simplemente se trata de irlos agregando al principio de la lista conforme recorremos la vecindad de x. Como las inserciones se hacen al principio de la lista, la creación de esta lista nos lleva solo el tiempo en que recorremos $N(x)$, más la verificación de color y las inserciones, así, toma solo $c_2|N(x)|$ operaciones hacer la lista, donde c_2 es una constante. Con esta lista construida, dado que x será la cabeza de Q en todas las iteraciones del `while` de este momento hasta que todos los vecinos de x estén coloreados, para verificar que x todavía tenga vecinos sin colorear, podemos preguntar si la lista es vacía, lo cual lleva tiempo constante. Por otro lado, si la lista no es vacía, para elegir un vecino de x sin colorear, podemos tomar simplemente al primer elemento de lista y quitarlo, esto también toma tiempo constante. Por lo tanto, hacer este proceso adicional, nos permite hacer la verificación del bloque `if-else` en tiempo constante, con el detalle de que realizamos $c_2|N(x)|$ operaciones creando la lista en la ejecución cuando x se convierte en la cabeza de Q.

En el `if-then`, además de verificar la condición, se hacen seis operaciones en caso de entrar al bloque `if`, y se hace solo una al aterrizar en el bloque `else`. Independientemente de si la condición del bloque `if-then` se satisface o no, dentro de dicho bloque se realiza a lo más un número constante de operaciones, número que vamos a llamar c_3. En cada iteración del `while`, además de las operaciones dentro del bloque `if-then`, se hace solo una asignación de tiempo constante, la asignación de x como la cabeza actual de Q. Por lo que, en cada iteración del `while` usamos a lo sumo $c_3 + 1$ unidades de tiempo, más las necesarias para crear la respectiva lista si es una iteración especial, como ya lo habíamos discutido.

Ahora, evidenciemos que el bloque `while` realiza a lo más $2n - 1$ iteraciones. Podemos distinguir dos tipos de iteraciones del `while`, las primeras son, en las que se evalúa a verdadero la condición del `if`, mientras que las segundas son en las que se evalúa a falso, llegando a la claúsula `else`. En cada una de las iteraciones del `while` perteneciente al primer tipo, se colorea un vértice y que es un vecino todavía no coloreado de x, siendo x la cabeza de Q en ese momento. Si en una iteración se

colorea y que es vecino de x, podemos relacionar dicha iteración con la arista yx. Así, podemos homologar cada iteración del primer tipo, con la correspondiente arista $yx = yp(y)$, donde y es el vértice que se colorea en dicha iteración. Esta forma de asociar las iteraciones del primer tipo con las aristas del árbol generador $G[\{yp(y) \colon y \in V_G\}]$, resulta ser una función inyectiva, ya que los vértices de G se colorean exactamente una vez. De lo anterior, tenemos que el número de iteraciones del `while` perteneciente al primer tipo es, a lo más, el número de aristas que tiene el árbol generador que construimos a partir de p, o sea, $n-1$ deacuerdo con el Corolario 2.1.3. Por otro lado, la cantidad de iteraciones del bloque `while` correspondiente al segundo tipo, lo podemos homologar al número de veces que se elimina un vértice de Q, este número está claramente acotado superiormente por n, pues, un vértice no puede entrar más de una vez a Q. Concluimos entonces que, el número de iteraciones que realiza el bloque `while`, es a lo más $(n-1)+n = 2n-1$.

Por la Proposición 1.1.1, tenemos que $\sum_{x\in V_G} |N(x)| \leq \sum_{x\in V_G} d(x) = 2m$. Todo el desarrollo anterior nos permite garantizar que, toda la ejecución del algoritmo toma a lo más $c_1 + (2n-1)(c_3+1) + \sum_{x\in V_G} c_2|N(x)| \leq (c_3+1)2n - (c_3+1) + c_2 2m$ pasos. Como la función $(c_3+1)2n - (c_3+1) + c_2 2m \leq 2(c_3+1+c_2)(n+m)$ es $\mathcal{O}(n+m)$ por la Proposición 1.9.3, podemos concluir que el Algoritmo 3 también es $\mathcal{O}(n+m)$. ■

Capítulo 3

Cográficas

3.1. Familia de cográficas

En esta sección estudiaremos la definición de cográfica y de coárbol, junto con algunas de sus propiedades más importantes. Entremos de lleno con la definición de cográfica.

Definición 3.1.1. Definimos a la familia de cográficas, como la familia mínima por contención que satisface las siguientes tres propiedades:

1. Las gráficas vacías de un solo vértice están en la familia.
2. Si $G_1, \ldots, G_k$ son gráficas en la familia, $k \geq 2$, entonces su unión ajena $\sum_{i=1}^{k} G_i$ pertenece también a la familia.
3. Si $G_1, \ldots, G_k$ son gráficas en la familia, $k \geq 2$, entonces su unión completa $\bigoplus_{i=1}^{k} G_i$ pertenece también a la familia.

Cabe mencionar, que no estamos siendo del todo formales en la definición previa, pues, puede no existir tal familia con la propiedad de ser mínima respecto a la contención, o bien, si existiera podríamos estar considerando clases en lugar de conjuntos, de hecho, la colección de todas las gráficas triviales es una clase. En cualquier caso, lo que pretende la definición al hablar sobre familias de gráficas, es solo considerar sus esquemas, o sea su estructura, homologando gráficas que sean isomorfas. No hay que preocuparnos por esto, ya que, a pesar de no haber especificado lo anterior, se puede solventar fácilmente restringiendo las familias de gráficas, de tal manera que sus vértices y aristas estén delimitados en un conjunto bien específico, por ejemplo en $\mathbb{N}$. Naturalmente, definimos una **cográfica** como una gráfica perteneciente a la

familia de cográficas definida en la Definición 3.1.1. Algo que vale la pena mencionar es que las cográficas son gŕaficas simples, esto se desprende directamente de la definición. En la Figura 3.1, podemos ver el ejemplo de varias cográficas. Ahí, podemos ver claramente que, $\mathscr{C}_1$ es cográfica por ser la unión completa de $\overline{K_3}$ y dos gráficas isomorfas a $K_1 + K_2$, mientras que $\mathscr{C}_2$ es cográfica por ser la unión completa de K_3, K_1 y $\overline{K_3}$.

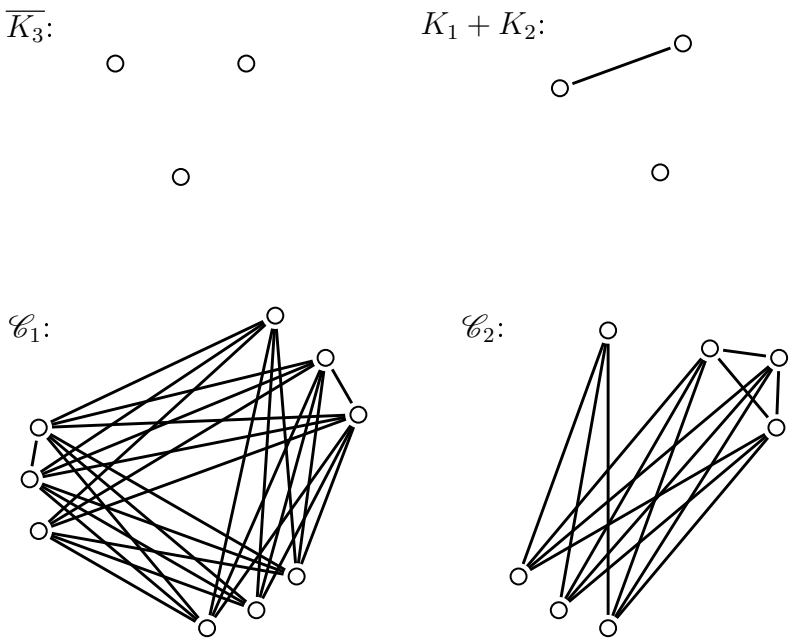


Figura 3.1: Ejemplos de cográficas.

Una característica relevante de las cográficas, es que se puede llevar una especie de registro de las operaciones que se usan para construirlas, para esto existe un árbol donde los nodos representan una unión ajena o una unión completa. En aras de facilitar la definición del coárbol, primero daremos otra definición auxiliar, esta definición no es estándar y solo nos es útil para hacer la redacción de la definición de coárbol más legible. Dado un árbol enraizado no trivial $\langle T, r\rangle$, los árboles $T^1, \ldots, T^k$ denotaran a los subárboles $T_{s_1}, \ldots, T_{s_k}$, donde $s_1, \ldots, s_k$ son los hijos de r, no importa el orden en el que se tomen. Como el árbol no es trivial, sabemos que r tiene al menos un hijo y por lo tanto, sí tiene sentido esta definición.

Definición 3.1.2. El coárbol de una cográfica, es un árbol enraizado, con sus nodos etiquedados y donde todos sus nodos internos tienen más de un hijo. Las hojas están etiquetadas con los vértices de la cográfica, mientras que los nodos internos están

etiquetados con 0 o 1. El etiquetado de los nodos en el coárbol siempre satisface que, los nodos internos, dentro de cualquier rama, alternan sus etiquetas entre 0 y 1. Damos la definición precisa del **coárbol** de una cográfica, utilizando recursión sobre la Definición 3.1.1.

1. Si G es una gráfica vacía y con un solo vértice, su coárbol es el árbol trivial y con su único nodo, o sea la raíz, etiquetado con el único vértice de G.

2. Sea G una cográfica con más de un vértice, que cumple $G = \sum_{i=1}^{k} G_i$, donde $k \geq 2$ y $G_1, \ldots, G_k$ son cográficas con respectivos coárboles $T_1, \ldots, T_k$. Vamos a suponer, sin pérdida de generalidad, que las etiquetas de las hojas en los coárboles $T_1, \ldots, T_k$, no se repiten dos a dos. Para cada $i \in \{1, \ldots, k\}$, definimos la colección F_i como $\{T_i\}$, si la etiqueta de la raíz de T_i es 1 o es un vértice de G_i; por otro lado, definimos a F_i como $\{(T_i)^1, \ldots, (T_i)^{k_i}\}$, si la etiqueta de la raíz de T_i es 0, aquí k_i denota al número de hijos de la raíz de T_i. Un coárbol de G, se define como el árbol $\langle T, r\rangle$, resultante de tomar un nodo r, que no esté presente en ninguno de los árboles $T_1, \ldots, T_k$, y luego crear el árbol T conectando el nodo r con cada raíz de cada árbol en F_i, para todo $0 \leq i \leq k$. Posteriormente se le adjunta a r la etiqueta 0, dejando a los demás nodos internos con las etiquetas que tenían en sus respectivos uniendos.

3. Sea G una cográfica con más de un vértice, que cumple $G = \bigoplus_{i=1}^{k} G_i$, donde $k \geq 2$ y $G_1, \ldots, G_k$ son cográficas con respectivos coárboles $T_1, \ldots, T_k$. Vamos a suponer, sin pérdida de generalidad, que las etiquetas de las hojas en los coárboles $T_1, \ldots, T_k$, no se repiten dos a dos. Para cada $i \in \{1, \ldots, k\}$, definimos la colección F_i como $\{T_i\}$, si la etiqueta de la raíz de T_i es 0 o es un vértice de G_i; por otro lado, definimos a F_i como $\{(T_i)^1, \ldots, (T_i)^{k_i}\}$, si la etiqueta de la raíz de T_i es 1, aquí k_i denota al número de hijos de la raíz de T_i. Un coárbol de G, se define como el árbol $\langle T, r\rangle$, resultante de tomar un nodo r, que no esté presente en ninguno de los árboles $T_1, \ldots, T_k$, y luego crear el árbol T conectando el nodo r con cada raíz de cada árbol en F_i, para todo $0 \leq i \leq k$. Posteriormente se le adjunta a r la etiqueta 1, dejando a los demás nodos internos con las etiquetas que tenían en sus respectivos uniendos.

En la Definición 3.1.2, no se muestra que los coárboles satisfagan todas las condiciones que impusimos en un principio. Esto se soluciona mostrando por inducción sobre la Definición 3.1.1, que en los coárboles el etiquetado satisface lo necesario y que los nodos internos, en caso de tratarse de un coárbol no trivial, tienen más de un hijo. La parte de que los nodos internos tienen al menos dos hijos, se puede ver utilizando hipótesis inductiva en los casos 2-3 y, notando que por la construcción, r

tiene al menos k hijos, con $k \geq 2$ por hipótesis. Ver que el etiquetado de los nodos internos alterna entre 0 y 1 también es fácil, simplemente hay que observar que si la raíz t de un coárbol tiene etiqueta $x \in \{0, 1\}$, entonces sus hijos no van a tener etiqueta x, es decir, los hijos de t van a tener por fuerza la etiqueta $1 - x$, o van a tener la etiqueta de un vértice. Esto muestra que los coárboles satisfacen lo establecido en la Definición 3.1.2. Dado un vértice x de una cográfica, la mayoría de las veces haremos indistintamente el uso del nombre x para denotar al vértice x, o para denotar a la hoja que está etiquetada con x en el coárbol de la cográfica.

Otra propiedad importante de los coárboles, es su unicidad respecto a la cográfica asociada. Esto es, si G es una cográfica, hay un único coárbol de G, salvo isomorfismo. El enunciado anterior está plasmado en el siguiente resultado, que bien lo podríamos desarrollar en unas cuantas líneas aquí mismo. Pero, al ser un resultado destacado, conviene más enmarcarlo aparte.

Teorema 3.1.3. *Sea G una cográfica. Cualesquiera dos coárboles de G son idénticos, incluso en el etiquetado de sus nodos.*

Demostración. Como la definición de cográfica es una definición recursiva, esta demostración la vamos a desarrollar por inducción sobre las Definiciones 3.1.1 y 3.1.2. Si G es una gráfica vacía con un solo vértice, por definición el coárbol es el árbol trivial cuyo nodo está etiquetado con el único vértice de G. Sea $\bigstar$ un operador, ya sea $\sum$ o $\bigoplus$. Supongamos que $G = \bigstar_{i=1}^{i=k} G_i$ con $k \geq 2$ y $G_1, \ldots, G_k$ cográficas. Por definición, un coárbol de G es un árbol obtenido al colgar de un nuevo nodo r a algunos subárboles de coárboles correspondientes a $G_1, \ldots, G_k$. Entonces, un coárbol de G está determinado por los coárboles de $G_1, \ldots, G_k$, pero, la hipótesis inductiva garantiza que $G_1, \ldots, G_k$ tienen un único coárbol. Por lo tanto, el coárbol de G también es único. Solo notemos para terminar que la unicidad del etiquetado también está implicita, pues, el etiquetado es único en los coárboles de $G_1, \ldots, G_k$, y además la etiqueta de r está determinada solo por la identidad de $\bigstar$. ■

El resultado anterior establece que a cualquier cográfica, le corresponde un único árbol enraizado que satisface ciertas propiedades, o sea, su coárbol. Por otro lado, tomemos cualquier árbol enraizado no trivial $\langle T, r \rangle$ de tal modo que todos los nodos internos tengan más de un hijo. Elijamos una etiqueta para r, ya sea 0 o 1, después, etiquetemos a los nodos alternando las etiquetas entre 0 y 1, desde la raíz hasta las hojas. Por último, demos etiquetas distintas dos a dos a cada una de las hojas de T. Si seguimos el proceso anterior, vamos a terminar con el árbol enraizado $\langle T, r \rangle$ con sus nodos etiquetados, y resulta que éste árbol enraizado va a corresponder al coárbol de una cográfica. Dicha cográfica que le corresponde al coárbol $\langle T, r \rangle$ etiquetado, la

podemos construir siguiendo el orden índicado de las operaciones que determinan las etiquetas de los nodos. Comenzamos con las hojas que representan a K_1, luego cada nodo etiquetado con 0 indica la operación $+$, mientras que cada nodo etiquetado con 1 indica la operación $\oplus$. Entonces, existe una relación biunívoca entre las cográficas y cierto tipo de árboles enraizados. En la Figura 3.2 podemos observar los coárboles de las gráficas $\mathscr{C}_1$ y $\mathscr{C}_2$, correspondientes a la Figura 3.1

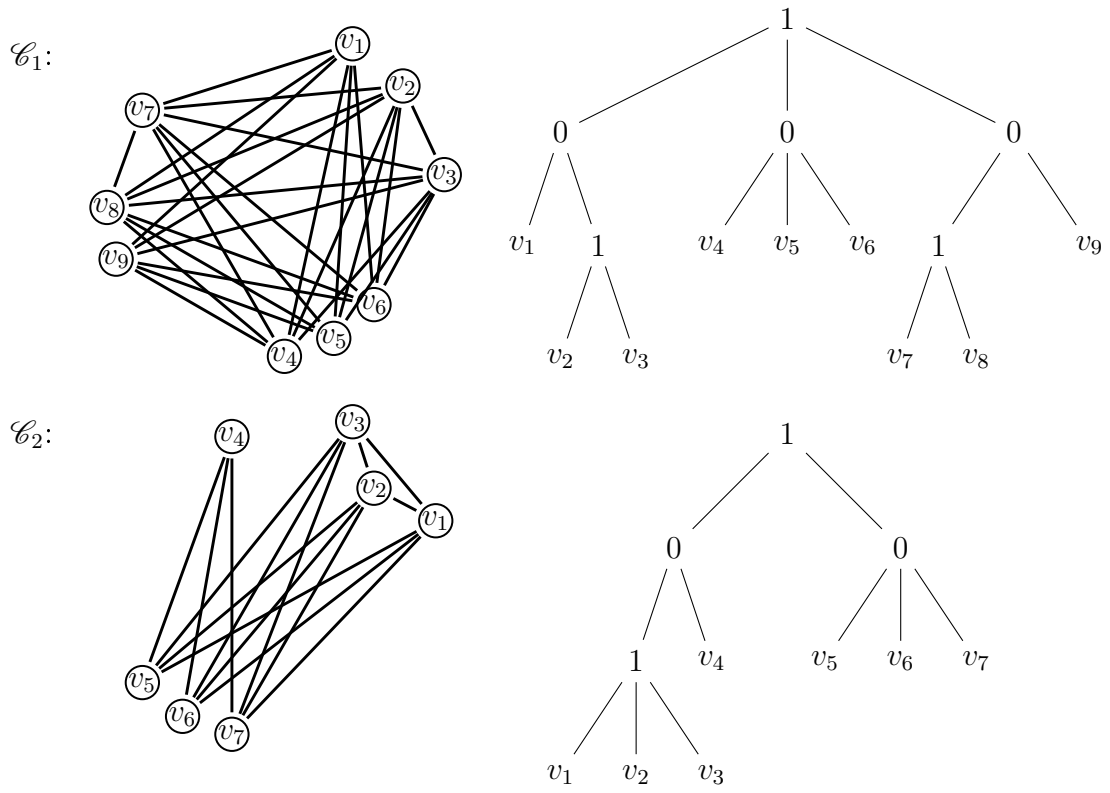


Figura 3.2: Coárboles de cográficas.

Como ya lo habíamos anticipado, cerraremos la sección con algunas propiedades que satisface la familia de cográficas. Para esto, necesitamos la definición de subgráfica correspondiente a un nodo, la cual proporcionamos después del siguiente resultado.

Proposición 3.1.4. *Sean G una cográfica con coárbol $\langle T, r\rangle$ y n_1, n_2 nodos de T tales que n_1 es ancestro de n_2. Si denotamos por H_1 y H_2 a las únicas cográficas salvo isomorfismo que tienen coárboles T_{n_1} y T_{n_2} respectivamente, entonces H_1 es una subgráfica inducida de H_2.*

Demostración. Procedamos por inducción sobre la distancia entre n_1 y n_2 en T. Si la distancia es 0, o sea, si $n_1 = n_2$, entonces $H_1 = H_2$. Si en cambio n_1 es padre de n_2, notemos que H_1 es por definición, la unión ajena o la unión completa de las cográficas correspondientes a los hijos de n_1 en T, entre ellas H_2. Como la unión completa y la unión ajena no agrega aristas entre vértices de un mismo operando, se sigue que H_2 es una subgráfica de H_1 inducida por vértices. Ahora, tomemos $k \in \mathbb{N}$ y supongamos que el enunciado se satisface para nodos cuya distancia entre ellos sea k. Para el paso inductivo, hay que suponer que la distancia entre n_1 y n_2 en T es $k+1$. Como $k+1 > 0$, P_{n_2} no es trivial y tiene al nodo n_1 entre sus vértices. Sea m el nodo que está en P_{n_2} y que es hijo de n_1. Por hipótesis inductiva, se tiene que H_2 es una subgráfica de H inducida por vértices, donde H es la cográfica correspondiente al nodo m. Dado que n_1 es padre de m, también se satisface que H es una subgráfica de H_1 inducida por vértices. Por lo tanto, tenemos que H_2 es una subgráfica de H_1 inducida por vértices. ■

Sean G cualquier cográfica con coárbol $\langle T, r\rangle$ y n cualquier nodo de T. Un corolario de la Proposición 3.1.4 tomando a los nodos r y n es que, la única cográfica salvo isomorfismo que tiene a T_n como coárbol, es una subgráfica de G inducida por vértices, de hecho, inducida por el conjunto de vértices $H(T_n)$. A esta cográfica la llamamos **cográfica correspondiente** al nodo n y la denotamos con G_n, o sea, simplemente como la cográfica del coárbol principal con el nodo como subíndice.

Proposición 3.1.5. *Sean G una cográfica con coárbol $\langle T, r\rangle$ y u, v dos vértices distintos de G. La arista uv está en E_G, si y solamente si, el nodo $mca_T(u, v)$ tiene etiqueta* 1.

Demostración. Definamos a n como el hijo de $mca(u, v)$ en la trayectoria P_u y denotemos por m al hijo de $mca(u, v)$ en la trayectoria P_v. La gráfica $G_{mca(u,v)}$ es la unión ajena o la unión completa de las gráficas correspondientes a los hijos de $mca(u, v)$ en T, entre los operandos de esta unión, ya sea ajena o completa, se encuentran G_n y G_m. En $G_{mca(u,v)}$, hay aristas entre vértices de dos operandos distintos, si y solamente si, la operación es una unión completa. Dado que $u \in V_{G_n} = H(T_n)$ y $v \in V_{G_m} = H(T_m)$, podemos concluir que $uv \in E_G$ si y solo si, el nodo $mca(u, v)$ está etiquetado con 1. ■

Para afianzar el concepto de gráfica correspondiente a un nodo, hacemos la observación de que en una cográfica G y su coárbol $\langle T, r\rangle$, siempre se satisface que G_r es G misma y, para $x \in V_G$, la cográfica G_x es isomorfa a la gráfica trivial. De forma suplementaria damos la Figura 3.3, donde se toma un nodo n de la cográfica $\mathscr{C}$ y se plasma la cográfica correspondiente al nodo n junto con su coárbol.

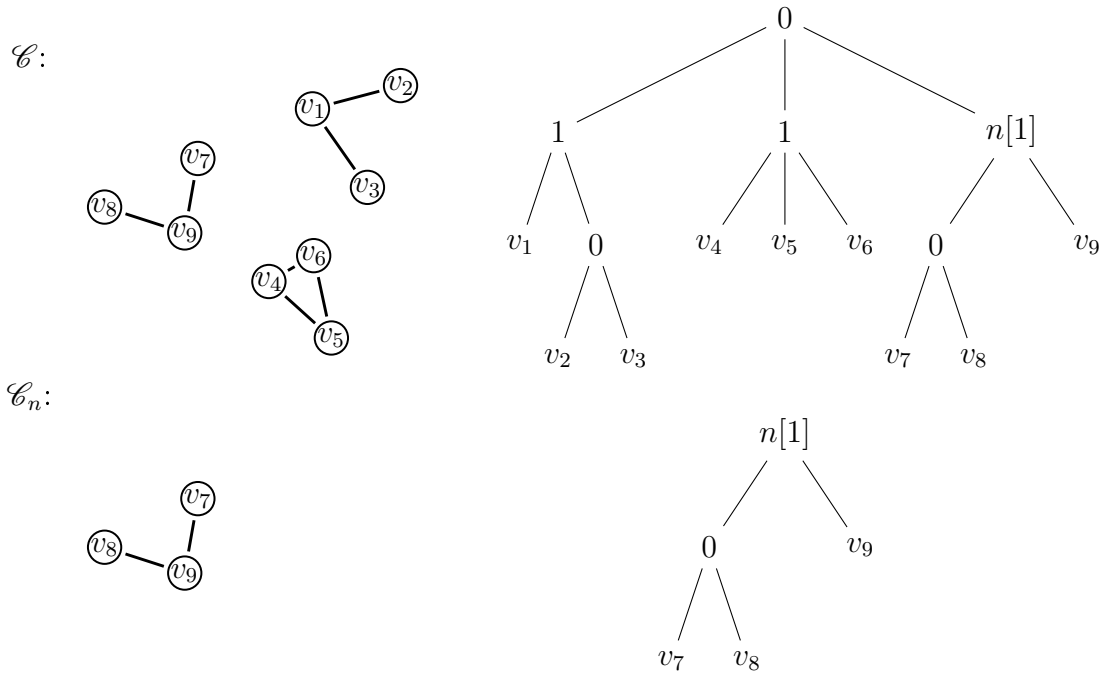


Figura 3.3: Cográfica C con su coárbol y la subgráfica correspondiente al nodo n.

Proposición 3.1.6. *Sea G una cográfica con coárbol $\langle T, r\rangle$. Se satisface lo siguiente:*

1. *La gráfica $\overline{G}$ es también una cográfica y su coárbol es justamente $\langle T, r\rangle$, pero con el etiquetado invertido, o sea, los nodos tendrán etiqueta 0 si en el coárbol de G tenían etiqueta 1 y viceversa.*

2. *Cualquier subgráfica de G inducida por vértices, es también una cográfica.*

Demostración.

1. Primero abordemos el caso en el que T es trivial. Si T es trivial, claramente G es trivial, lo anterior implica que $\overline{G}$ también es una cográfica y $\langle T, r\rangle$ es su coárbol, esto porque $\overline{G}$ también debe ser trivial. Libre del caso anterior, supongamos de ahora en adelante que T no es trivial. Denotemos por $\overline{T}$ al árbol descrito en el enunciado, esto es, al árbol T enraizado en r, pero con las etiquetas de sus nodos internos cambiadas. Sabemos que $\overline{T}$ es el coárbol de una única cográfica salvo isomorfismo, llamemos a dicha cográfica H. Por la Proposición 3.1.5, dadas x y z dos hojas de $\overline{T}$, tenemos que $mca_T(x, z)$ tiene etiqueta 0, si y solo si $xz \in E_G$. Pero, por la definición de $\overline{T}$, se tiene que

$mca_{\overline{T}}(x,z)$ tiene etiqueta 1, si y solo si, $mca_T(x,z)$ tiene etiqueta 0. Definamos $f\colon V_H \to V_G$, como la función biyectiva dada por $f(x) = z$, solo cuando x y z corresponden a la misma hoja de T, o lo que es lo mismo, de $\overline{T}$. Con las observaciones anteriores tendremos para $x, y \in V_H$ que $xy \in E_H$ si y solo si $f(x)f(y) \notin E_G$, es decir, f es un isomorfismo entre H y $\overline{G}$.

2. Vamos a usar inducción fuerte sobre el número de vértices que inducen a la subgráfica. Si tenemos un unitario $\{x\} \subseteq V_G$, es obvio que la subgráfica inducida, o sea la trivial, es una cográfica. Sea $X \subseteq V_G$ arbitrario y con más de un elemento, también elijamos $x \in X$. Como $r \in V_{P_x}$ y $X \subseteq H(T_r) = V_G$, podemos considerar a n como el $\leq_T$-mínimo nodo en P_x que satisface $X \subseteq H(T_n)$. Demos una enumeración $n_1, \ldots, n_k$ de los hijos de n en T, pero únicamente de los hijos m que satisfacen $H(T_m) \cap X \neq \varnothing$. Por la propiedad mínima de n, se debe cumplir $k \geq 2$. Para cada $i \in \{1, \ldots, k\}$, le daremos el nombre de X_i al conjunto $H(T_{n_i}) \cap X$. Ahora, por hipótesis inductiva, tenemos que para cada $i \in \{1, \ldots, k\}$, la subgráfica inducida $G[X_i]$ es una cográfica. Gracias a la Proposición 3.1.5, es fácil observar que dados $u \in X_i$ y $v \in X_j$, con $i \neq j$, se cumple $uv \in E_G$, si y solo si $mca_T(u,v)$ tiene etiqueta 1. Lo anterior implica que la subgráfica inducida $G[X] = G[X \cap H(T_n)]$ es la unión ajena o la unión completa, dependiendo la etiqueta de n, de las cográficas $G[X_1], \ldots, G[X_k]$. Por lo tanto, también $G[X]$ es una cográfica.

■

3.2. Módulos

Los módulos en las gráficas son conjuntos que rescatan la noción de "comportarse de la misma forma" afuera del conjunto mismo. El concepto de módulo está ampliamente estudiado y tiene varias aplicaciones en la teoría de gráficas. Sin embargo, nos conformaremos en esta sección con mencionar las propiedades principales, básicamente lo necesario para profundizar en nuestro algoritmo de reconocimiento.

Definición 3.2.1. Sea G una gráfica. Decimos que un subconjunto no vacío M de V_G es un **módulo** en G, si y solamente si, para cualesquiera $x, y \in M$ se satisface $N_G(x) \setminus M = N_G(y) \setminus M$. Adicionalmente, decimos que M es un **módulo fuerte** en G, si además de ser módulo, para cualquier módulo C en G que satisfaga $M \cap C \neq \varnothing$, también se cumple que $M \subseteq C$ o $C \subseteq M$.

Para ejemplificar la definición de módulo en una gráfica G, notemos directamente de la definición que V_G siempre es un módulo, también observemos que el conjunto

$\{x, y\}$, compuesto por dos vértices distintos, es un módulo solo cuando $N(x)\setminus\{x, y\} = N(y) \setminus \{x, y\}$. Si la condición anterior se satisface, es decir, si dos vértices distintos $x, y \in V_G$ cumplen $N(x) \setminus \{x, y\} = N(y) \setminus \{x, y\}$, decimos que x y y son vértices **gemelos** en G. Esa condición se reduce simplemente a $N(x) \setminus \{y\} = N(y) \setminus \{x\}$ cuando G no tiene lazos.

Resulta que en las cográficas, los módulos se pueden caracterizar de una manera no tan compleja, esto es el contenido del par de proposiciones que presentamos en esta sección.

Proposición 3.2.2. *Sea G una cográfica con coárbol $\langle T, r\rangle$. Un conjunto de vértices M es un módulo en G, si y solamente si, existe un conjunto no vacío de nodos hermanos en T, digamos U, tal que $M = \bigcup_{u\in U} H(T_u)$.*

Demostración. Si G es trivial, T es trivial también. En este caso, el si y solo si es inmediato, pues el único módulo en G sería V_G. Entonces, supongamos que G no es trivial y consecuentemente, T tampoco es trivial. Para demostrar la suficiencia del primer enunciado, elijamos $x \in M$. Como $r \in V_{P_x}$ y $M \subseteq H(T_r) = V_G$, podemos tomar a un nodo n como el $\leq_T$-mínimo entre los nodos de P_x, con la propiedad de cumplir $M \subseteq H(T_n)$. Demos una enumeración $n_1, \ldots, n_k$ de los hijos de n en T, pero únicamente de los hijos m que satisfacen $H(T_m) \cap M \neq \varnothing$. Observemos que la condición de mínimo que tiene n, implica $k \geq 2$. Por la definición de n y sus hijos seleccionados, es claro que $M \subseteq \bigcup_{i=1}^{k} H(T_{n_i})$, así que, solo hace falta ver la otra contención. La contención anterior nos permite tomar un índice $j \in \{1, \ldots, k\}$ tal que $x \in H(T_{n_j})$, es fácil observar que ese índice es único, ya que de haber más de uno se violaría la propiedad de trayectorias únicas en un árbol. Sea $i \in \{1, \ldots, k\}$ arbitrario y elijamos $y \in H(T_{n_i})$ también arbitrario.

Tomemos cualquier índice $l \in \{1, \ldots, k\}$ distinto de i. Ahora, elijamos cualquier $u \in H(T_{n_i})\cap M$ y cualquier $v \in H(T_{n_l})\cap M$. Si $u = y$, entonces ya estaría demostrado lo que queríamos, así que enfoquémonos en el caso $u \neq y$. Como $u, v \in M$ y M es un módulo, se tiene que $N(u)\setminus M = N(v)\setminus M$, en otras palabras $(N(u)\setminus N(v))\cup(N(v)\setminus N(u)) \subseteq M$. Por lo que, en virtud de la Proposición 3.1.5, se sigue que, el conjunto de vértices cuyos máximos comunes ancestros, con u por un lado, y con v por el otro, tengan etiqueta distinta, debe estar contenido en M. Ahora, como u y y son hojas distintas de T, ambas en el conjunto $H(T_{n_i})$, no es posible que n_i sea una hoja, además, n_i tiene etiqueta diferente a n porque son padre e hijo. Nótese que por la Proposición 2.2.3, se cumple que $mca(u, v) = mca(y, v) = n$. Si $mca(y, u) = n_i$, por las deducciones anteriores, se satisface que $y \in M$, mientras que si $mca(y, u) \neq n_i$, se cumple por fuerza que $mca(y, u) <_T n_i$. Sabemos que de los hijos de n_i, solo hay uno que está en la trayectoria de $mca(y, u)$ a n_i en T, entonces, elijamos m

cualquier hijo de n_i que sea distinto al que está en la trayectoria de $mca(y,u)$ a n_i en T. Con esta construcción, tenemos que cualquier hoja $h \in H(T_m)$, va a satisfacer que $mca(h,u) = n_i$, mientras que por la Proposición 2.2.4 se tiene $mca(h,v) = n$. Entonces, ocurre que $h \in M$, pero notemos que, de nuevo por la Proposición 2.2.4, se tiene $mca(h,y) = n_i$. Concluimos de lo anterior, por tenerse $v, h \in M$, que y también debe ser un elemento de M.

Ahora, para la suficiencia del segundo enunciado, supongamos que U es cualquier conjunto no vacío de nodos hermanos en T. Hay que probar que $\bigcup_{u\in U} H(T_u)$ es un módulo en G. Sean $x, y \in \bigcup_{u\in U} H(T_u)$ distintos, y tomemos $v, w \in U$ tales que $x \in H(T_v)$ y $y \in H(T_w)$. Como U es un conjunto de hermanos, hay un nodo n que es padre de cada nodo en U. Supongamos que z es cualquier vértice que no está en el conjunto $\bigcup_{u\in U} H(T_u)$. Si n no es ancestro de z, por la Proposición 2.2.3, se satisface $mca(z,x) = mca(z,n) = mca(z,y)$, así que se cumple: $z \in N(x)$ si y solo si $z \in N(y)$. Por otro lado, si n sí es un ancestro de z, podemos definir m como el hijo de n que está en la trayectoria de z a n en T. Por la elección de z ocurre que $m \notin U$, así que, usando la Proposición 2.2.3 de nuevo, deducimos que $mca(z,x) = mca(m,v) = n$ y $mca(z,y) = mca(m,w) = n$. Por lo tanto, $z \in N(x)$ si y solo si $z \in N(y)$. Al ser x y y arbitrarios, concluimos que $\bigcup_{u\in U} H(T_u)$ es un módulo en G. ■

Proposición 3.2.3. *Sea G una cográfica con coárbol $\langle T, r\rangle$. Un conjunto de vértices M es un módulo fuerte en G, si y solamente si, existe un nodo n en T, tal que $M = H(T_n)$.*

Demostración. Para la suficiencia del primer enunciado, supongamos que M es un módulo fuerte en G. Por la Proposición 3.2.2, sabemos que existe un conjunto no vacío de nodos hermanos en T, digamos U, que satisface $M = \bigcup_{u\in U} H(T_u)$. Sea n el padre de todos los nodos en U y elijamos cualquier $w \in U$. Si U tiene a todos los hijos de n, lo que queríamos demostrar está consumado, porque $\bigcup_{u\in U} H(T_u) = H(T_n)$. Entonces, vamos a suponer de ahora en adelante que U no tiene a todos los hijos de n. Consideremos a m como cualquier hijo de n que no pertenezca a U y definamos $C = \{m, w\}$. Podemos usar otra vez la Proposición 3.2.2, para garantizar que el conjunto $N = \bigcup_{u\in C} H(T_u) = H(T_w) \cup H(T_m)$ es un módulo en G. Como $w \in U$, tenemos un módulo N tal que $N \cap M \neq \varnothing$. Al ser M un módulo fuerte, se debe tener $M \subseteq N$ o $N \subseteq M$. Como $m \notin U$, los nodos en $H(T_m)$ no pueden tener como ancestro a un nodo en U, así que es imposible la contención $N \subseteq M$. Por lo tanto $M \subseteq N$, de donde se sigue que $U \subseteq \{w, m\}$. Sin embargo, sabemos que m no pertenece a U, o sea que $M = H(T_w)$.

Sigamos con la suficiencia del segundo enunciado, entonces, sea n cualquier nodo en T. Sabemos por la Proposición 3.2.2, que $\bigcup_{u\in\{n\}} H(T_u) = H(T_n)$ es un módulo

en G. Para ver que M es fuerte, tomemos cualquier otro módulo en G que intersecte a M, digamos N. Otra aplicación de la Proposición 3.2.2, nos permite garantizar la existencia de un conjunto no vacío U, conformado por nodos hermanos en T, de tal suerte que $N = \bigcup_{u \in U} H(T_u)$. Si algún miembro de U es ancestro de n, claramente $M \subseteq N$, así que supongamos que ningún elemento de U es ancestro de n. Seleccionemos cualquier $x \in M \cap N$, esa elección implica que para algún $w \in U$ se satisface $x \in H(T_w)$, también que $x \in H(T_n)$ desde luego. Así que, tanto n como w están en P_x, por lo que uno debe ser ancestro del otro. Recordando que ningún elemento de U es ancestro de n, deducimos que n es un ancestro propio de w. Si definimos a p como el padre de todos los nodos en U, en particular p es padre de w. Se cumple entonces que $p \leq_T n$, o sea que todos los nodos en U son descendientes de n. Con lo anterior concluimos directamente que $N \subseteq M$, y por lo tanto que M es fuerte. ■

3.3. Otras definiciones y atributos de gráficas

Esta sección la utilizaremos para desarrollar todos los conceptos necesarios para nuestro teorema de caracterización. También la vamos a aprovechar para explorar los conceptos definidos desde otra perspectiva, una perspectiva más intuitiva que pretende vislumbrar como se conectan los conceptos entre sí, aparte de su nexo formal mostrado en su equivalencia lógica.

Sea G una gráfica. Decimos que $I \subseteq V_G$ es un **conjunto independiente** de G, si y solo si, es no vacío y no hay aristas entre vértices de I. Además, cuando G es simple, tenemos el concepto de clan. Si $C \subseteq V_G$ es no vacío y tiene todas las posibles aristas entre vértices de C, es decir $G[C]$ es completa, decimos que C es un **clan** de G. Adicionalmente, si I es un conjunto independiente de G con k vértices, decimos que I es un k-conjunto independiente. Análogamente, cuando C es un clan de G con k vértices, decimos que C es un k-clan. Un **conjunto independiente máximo por contención** de G, es un conjunto independiente $I \subseteq V_G$, de tal forma que para cualquier otro conjunto independiente J de G que contenga a I, se satisface por fuerza que $I = J$. Del mismo modo, un **clan máximo por contención** de G, es un clan $C \subseteq V_G$, de tal forma que para cualquier otro clan K de G que contenga a C, se satisface por fuerza que $C = K$. Decimos que la gráfica simple G satisface la **propiedad CK**, cuando cualquier conjunto independiente máximo por contención I y cualquier clan máximo por contención C, se intersectan en exactamente un vértice, esto es $|I \cap C| = 1$.

Primero hagamos una observación inmediata. En cualquier gráfica simple, si un conjunto independiente se intersecta con un clan, esta intersección consiste en a lo más un vértice. La validez de la afirmación anterior surge al notar que, por definición,

dos vértices del clan son adyacentes, así que no pueden estar ambos a la vez en el conjunto independiente. Lo anterior indica que la propiedad CK, es equivalente a pedir que cualquier conjunto independiente máximo por contención intersecte a cualquier clan máximo por contención. Hay gráficas donde no se satisface la propiedad CK, un ejemplo es cualquier trayectoria P_n con $n \geq 4$, en particular P_4. Reflexionando un poco acerca de la propiedad CK y su conexión con las cográficas, podemos intuir el porqué las cográficas deben satisfacer la propiedad CK.

Gracias a la definición recursiva de la familia de cográficas, sabemos que si G es una cográfica no trivial, entonces G es la unión, ajena o completa, de dos o más cográficas. La clave es que la propiedad CK se va transmitiendo desde los uniendos hacia la gráfica G. Si G es una unión ajena, dados un clan máximo por contención y un conjunto independiente máximo por contención, ambos en G, el clan debe ser un clan máximo por contención en uno de los uniendos; pero, el conjunto independiente debe tener vértices en dicho uniendo, de hecho, los vértices de ese uniendo que están en el conjunto independiente, conforma un conjunto independiente máximo por contención en el uniendo. Si G es una unión completa, cambiando los papeles del clan y el conjunto independiente, funciona de manera análoga el argumento anterior. Por lo tanto, el hecho de que los uniendos satisfagan la propiedad CK, en virtud de un argumento inductivo sobre la familia de cográficas, implica que G también la va a cumplir. La suficiencia de la propiedad CK para que una gráfica sea una cográfica, es un tanto más compleja, así que la dejaremos para el teorema de equivalencia.

Decimos que una gráfica simple G es una **gráfica Dacey**, solo cuando se satisface lo siguiente: dados $u, v \in V_G$ distintos y un clan máximo por contención C en G, la contención $C \subseteq N(u) \cup N(v)$ implica que u y v son adyacentes. Cuando toda subgráfica inducida por vértices de G es una gráfica Dacey, decimos que G es una gráfica hereditariamente Dacey, o **HD-gráfica** para acortar. Resulta que ser una HD-gráfica, es una condición tanto suficiente como necesaria para ser cográfica. Igual que en el caso de la propiedad CK, dejaremos para el teorema de caracterización la exploración de la suficiencia. Por otro lado, para la necesidad del enunciado, tenemos lo siguiente.

Supongamos que tenemos una cográfica G con coárbol T, un clan máximo por contención C en G, y $u, v \in V_G$ distintos tales que $C \subseteq N(u) \cup N(v)$. Si u fuera adyacente a todos los vértices en $C \setminus \{u\}$, tendríamos que $u \in C$ y por lo tanto $u \in N(v)$. Algo análogo ocurre cuando v es adyacente a todos los vértices en C. El caso que falta por considerar, es que para algunos $x \in C \setminus \{u\}$ y $z \in C \setminus \{v\}$, se cumpla $xu, zv \notin E_G$, o lo que es lo mismo en virtud de la Proposición 3.1.5, que $mca(x, u)$ y $mca(z, v)$ tengan etiqueta 0. Como $C \subseteq N(u) \cup N(v)$, se tiene forzosamente $xv, zu \in E_G$, o sea que $mca(x, v)$ y $mca(z, u)$ tienen etiqueta 1. Uti-

lizando la contrapositiva de la Proposición 2.2.3 y que $mca(x,z)$ tiene etiqueta 1, podemos inferir que $mca(x,z) \leq_T mca(x,v)$ y $mca(x,z) \leq_T mca(z,u)$. Usando de la misma forma la Proposición 2.2.3, podemos deducir que $mca(x,v) \leq_T mca(u,v)$ y $mca(z,u) \leq_T mca(u,v)$. De las cuatro desigualdades anteriores, podemos afirmar que $mca(x,v) = mca(z,u)$, de donde se sigue que $mca(u,v) \leq_T mca(x,v)$. Teniendo las dos desigualdades necesarias, concluimos que $mca(u,v) = mca(v,x)$. Justamente lo anterior significa que u y v son adyacentes; es importante destacar que los argumentos anteriores vienen dados por la estructura del coárbol.

Recordemos rápidamente que en un conjunto parcialmente ordenado $\langle X, \leq \rangle$, tenemos para un elemento $x \in X$ la siguiente notación: $(x, \rightarrow) = \{y \in X : x < y\}$ y $(\leftarrow, x) = \{y \in X : y < x\}$. También utilizamos $[x, \rightarrow)$ y $(\leftarrow, x]$ cuando consideramos el orden reflexivo inducido por $<$, o sea $\leq$, resultando así en los mismos conjuntos, pero con x incluido. Decimos que un conjunto parcialmente ordenado $(X, \leq)$ es un **multiárbol**, solo cuando se satisfaga que para cualesquiera $w, x \in X$, se tiene $w \leq x$, o bien, se cumple para todo $y \in [w, \rightarrow) \setminus [x, \rightarrow)$ y para todo $z \in [w, \rightarrow) \cap [x, \rightarrow)$ que $y \leq z$. Si tenemos un conjunto parcialmente ordenado $\langle X, \leq \rangle$, podemos definir la gráfica simple que tiene conjunto de vértices X y conjunto de aristas $\{\{x, y\} \in [X]^2 : x < y \text{ o } y < x\}$, la gráfica anterior se llama **gráfica de comparabilidad** del conjunto parcialmente ordenado $(X, \leq)$ y la vamos a denotar como $C_{(X,\leq)}$. En la Figura 3.4, podemos ver la representación esquemática de la propiedad que se dio antes y que caracteriza a los multiárboles, solo el caso cuando $w \not\leq x$, ahí las flechas indican la relación de orden. Por otro lado, en la Figura 3.5 podemos observar un ejemplo de gráfica de comparabilidad; se representa un conjunto parcialmente ordenado y se plasma su gráfica de comparabilidad correspondiente.

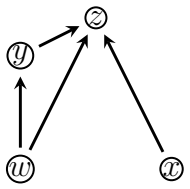


Figura 3.4: Propiedad de un multiárbol.

Cuando en la notación anterior hablemos de un solo orden, solo pondremos el conjunto como subíndice. Como explicar la relación de los conceptos anteriores con el de cográfica no es tan simple, optaremos por relacionar el concepto de multiárbol con el de no tener subgráficas inducidas que sean isomorfas a P_4. Si tenemos un orden parcial $(X, \leq)$ y su gráfica de comparabilidad C, la definición un tanto críptica de multiárbol implicaría que, de ser $(X, \leq)$ un multiárbol, la gráfica C no puede

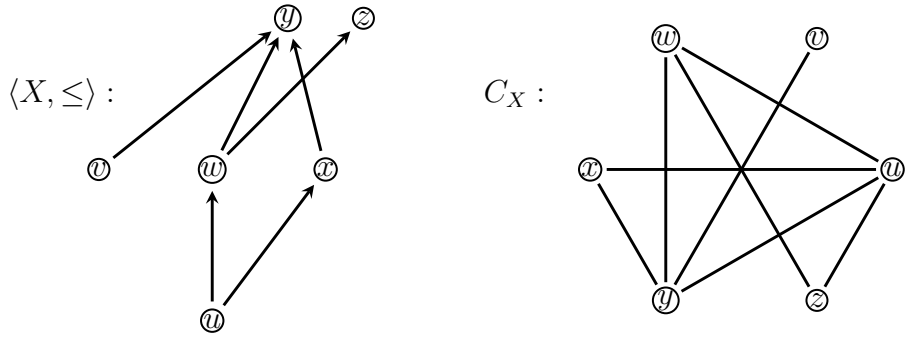


Figura 3.5: Gráfica de comparabilidad de $\langle X, \leq \rangle$.

tener subgráficas inducidas que sean isomorfas a P_4. Lo anterior se vuelve claro haciendo un análisis exhaustivo de todas las posibilidades. Si tenemos una subgráfica $H = C[\{w, x, y, z\}]$ que sea isomorfa a la trayectoria (w, x, y, z), hay tres posibilidades para x y z. Las dos triviales son que $x < z$ o que $z < x$, decimos triviales porque de ocurrir estas, tendríamos que $xz \in E_C$ y por lo tanto H no es inducida. El último caso es que x y z no son comparables. En este caso se debe satisfacer que y es menor a ambos x y z, o que y sea mayor a ambos x y z. El caso en el que y es mayor a x y a z ya va tomando la forma de nuestra hipótesis, pues, esto significa que $y \in (x, \rightarrow) \cap (z, \rightarrow)$. Ahora, por la existencia de la arista wx, se debe tener $x \leq w$ o $w < x$, pero $x \leq w$ es lo mismo que $w \in (x, \rightarrow)$. Si $wz \in E_C$, se sigue que H no es inducida, en cambio, si $wz \notin E_C$, eso querría decir que $w \in (x, \rightarrow) \setminus (z, \rightarrow)$. O sea que, por la definición de multiárbol, se tiene $w \leq y$ implicando $wy \in E_C$, es decir, H no es inducida. Claro que faltó considerar varios casos, pero lo dado es suficiente para crear una noción intuitiva.

Sea $k \in \mathbb{N}$ arbitrario. Una **k-coloración** de una gráfica sin lazos G, es una función $c\colon V_G \to \{1, \ldots, k\}$. La idea que subyace en esta definición es que estamos asignando, para cada vértice de G, un color de entre los k disponibles. Si la coloración c satisface además que, ningún par de vértices adyacentes en G tienen la misma imagen bajo c, diremos que c es una **coloración propia** de G. Hacemos aquí la observación de que al hablar de coloraciones de ahora en adelante, suponemos que las gráficas no tienen lazos. Alternativamente, pudimos haber definido una coloración de la gráfica G, como una familia ordenada $C = \{C_1, \ldots, C_k\}$ de subconjuntos de V_G, de tal forma que $C \setminus \{\varnothing\}$ sea una partición de V_G. Es clara la equivalencia entre ambas definiciones, pues, dada la partición $\{C_1, \ldots, C_k\}$ podemos definir para $x \in V_G$ a $c(x)$ como el índice de la parte a la que pertenece x, por otro lado, si c es una coloración, basta definir la partición $\{c^{-1}[1], \ldots, c^{-1}[k]\} \setminus \{\varnothing\}$. Además, si consideramos una coloración

como la familia de subconjuntos de V_G dada por $C = \{C_1, \ldots, C_k\}$, podemos también representar el concepto de coloración propia exigiendo que los elementos no vacíos de C sean conjuntos independientes de G. El decantarnos por una definición es arbitrario en realidad, así que usaremos cualquiera de estas definiciones para una coloración indistintamente. Dada una gráfica simple G, definimos el **número de clan** de G como el natural positivo

$$\omega(G) = \text{máx}\{|C| \colon C \text{ es un clan en } G\}.$$

Es obvio que a una gráfica G se le puede dar una $|V_G|$-coloración propia, solo hay que darle a cada vértice de G un color distinto. Por otro lado, es fácil ver que las únicas gráficas que cuentan con una 1-coloración son las gráficas vacías, o sea sin aristas. Resulta natural preguntarnos por el mínimo número de colores necesarios para dotar a G de una coloración propia con este número de colores. A ese número mínimo de colores lo llamamos **número cromático** de G, y lo denotamos por $\chi(G)$. Es decir,

$$\chi(G) = \text{mín}\{k \in \mathbb{N} \colon G \text{ tiene una } k\text{-coloración propia}\}.$$

Sea $c \colon V_G \to \{1, \ldots, k\}$ una coloración propia de G suprayectiva, o sea que usa todos los colores, tal que para cualesquiera $i, j \in \{1, \ldots, k\}$ que cumplan $i < j$, y para cada $x \in c^{-1}[j]$ se satisface que $N(x) \cap c^{-1}[i] \neq \varnothing$; en estas circunstancias, diremos que c es una **coloración Grundy** de G. El ser una coloración Grundy lo que garantiza es ser, esencialmente, una coloración glotona, es decir, una coloración que se puede construir vértice por vértice asignando el mínimo de los colores que no esté ocupado por un vecino. En relación a las coloraciones Grundy tenemos el siguiente resultado.

Lema 3.3.1. *Dada una gráfica arbitraria G, existe una $\chi(G)$-coloración para G que es Grundy.*

Demostración. Sea g una $\chi(G)$-coloración propia para G. Consideremos una enumeración $\{v_1, \ldots, v_n\}$ de V_G con las siguientes propiedades: para todo i en el conjunto $\{1, \ldots, \chi(G)\}$, los vértices en $g^{-1}[i]$ aparecen juntos en la enumeración, y dados $i, j \in \{1, \ldots, \chi(G)\}$ distintos, los vértices en $g^{-1}[i]$ aparecen antes que los vértices en $g^{-1}[j]$, si y solo si $i < j$. Definamos la función $c \colon V_G \to \mathbb{N}^+$ recursivamente como sigue: establecemos $c(v_1) = 1$, y para $k > 1$ definimos $c(v_k)$ como el mínimo natural que no es imagen de ninguno de los vértices en $N(v_k)$, claro está, de entre $v_1, \ldots, v_{k-1}$. Usando inducción fuerte, vamos a probar que para todo $k \in \{1, \ldots, \chi(G)\}$, se satisface que $\text{máx}\{c(x) \colon x \in g^{-1}[k]\} \leq k$, una vez probado lo anterior, tendremos que los valores de c están acotados por g. A su vez, dentro de esta prueba inductiva, podríamos usar inducción sobre la enumeración dentro de los conjuntos $g^{-1}[k]$, sin

embargo, vamos a optar por evitar esta argumentación tediosa y remitirnos a observar que $g^{-1}[k]$ siempre es un conjunto independiente. Para $k = 1$, se sigue por definición que c toma solo el valor 1 en todos los elementos de $g^{-1}[1]$, esto por la independencia de este último conjunto. Supongamos para k fijo, mayor que 1 y a lo más $\chi(G)$, que todos los números dentro del rango, que son menores a k, cumplen lo requerido. Para cualquier $x \in g^{-1}[k]$, suponiendo que $x = v_l$ y que el valor de c en $v_1, \ldots, v_{l-1}$ ya está definido, tendremos que x solo puede ser adyacente a vértices que no estén en $g^{-1}[k]$, esto por la independencia de $g^{-1}[k]$. De esta forma, si $y = v_m$ es un vecino de x con $m < l$, se sigue que $y \in g^{-1}[i]$ para algún $i < k$, o sea que $c(y) \leq i < k$. De lo anterior se deduce que $c(x) \leq k$, así que, al ser x arbitrario, podemos concluir que $\max\{c(x) \colon x \in g^{-1}[k]\} \leq k$. Como ya habíamos anticipado, la inducción anterior nos habilita la propiedad de que c está acotada por g, o sea que $c \colon V_G \to \{1, \ldots, t\}$ es una coloración de G con $t \leq \chi(G)$. Es fácil ver que c es una coloración propia, pues de no serlo significa que para algunos $i, j \in \{1, \ldots, n\}$, con $i < j$, se tiene $v_i v_j \in E_G$, pero $c(v_i) = c(v_j)$, lo anterior sería una contradicción a la definición de c en v_j. Por lo que, al ser c una t-coloración propia de G con $t \leq \chi(G)$, se sigue que $t = \chi(G)$ y que c es suprayectiva. Solo queda por ver que c es Grundy, pero esto es inmediato de la definición de c, pues dado cualquier $x \in V_G$, para que $c(x)$ tenga el valor que tiene, se debe cumplir que para cualquier $1 \leq i < c(x)$, hay un vértice y con $yx \in V_G$ y $c(y) = i$. ■

Dado que toda gráfica G tiene una $\chi(G)$-coloración Grundy, no tiene sentido, como en el caso de las coloraciones propias, preguntarse cuál es el mínimo número de colores con los que se obtiene una coloración Grundy, pues este número será siempre $\chi(G)$. Sin embargo, al ser las coloraciones Grundy suprayectivas, sabemos que toda coloración Grundy de G debe usar a lo más $|V_G|$ colores. Así pues, definimos el **número Grundy** de la gráfica G como

$$\gamma(G) = \max\{k \in \mathbb{N} \colon G \text{ tiene una } k\text{-coloración Grundy}\}.$$

En una coloración propia, los clanes no pueden tener dos vértices de un mismo color, así que tenemos la desigualdad $\omega(G) \leq \chi(G)$. Como toda coloración Grundy es en particular una coloración propia, tenemos también la desigualdad $\chi(G) \leq \gamma(G)$.

Proposición 3.3.2. *Sean G una gráfica y k un número tal que $\chi(G) \leq k \leq \gamma(G)$. La gráfica G tiene una k-coloración Grundy.*

Demostración. Procedamos por inducción fuerte sobre el orden. Si una gráfica tiene solo un vértice, es obvio que el número Grundy y el cromático son ambos 1, así que el resultado es inmediato. Supongamos válido el resultado para todos los

naturales positivos menores a $n > 1$, donde n es arbitrario y fijo. Consideremos una gráfica G de orden n y recordemos que k es un número cualquiera que satisface $\chi(G) \leq k \leq \gamma(G)$. Primero, tomemos $c\colon V_G \to \{1, \ldots, \gamma(G)\}$ una coloración Grundy. Ahora, definamos para cada $j \in \{1, \ldots, \gamma(G)+1\}$, el número s_j como el mínimo número de colores que puede utilizar una coloración de G, pero que extienda a c en el conjunto de vértices $\bigcup_{i<j} c^{-1}[i]$, o sea que coincida con c en dicho conjunto. Se desprende de la definición que, s_1 es $\chi(G)$ y $s_{\gamma(G)+1}$ es $\gamma(G)$. También, para todo $j \in \{1, \ldots, \gamma(G)+1\}$, definamos G_j como la subgráfica $G[\bigcup_{i\geq j} c^{-1}[i]]$. Analicemos algunas propiedades de lo que acabamos de definir, para eso, consideremos $j \in \{1, \ldots, \gamma(G)+1\}$. Podemos dar una coloración de G que coincida con c en el conjunto $\bigcup_{i<j} c^{-1}[i]$, simplemente coloreando los vértices restantes, o sea los vértices de G_j, utilizando colores distintos a los primeros $j-1$ utilizados; lo anterior muestra que $s_j \leq j - 1 + \chi(G_j)$. Sin embargo, notemos que cualquier coloración que extienda a c en el conjunto $\bigcup_{i<j} c^{-1}[i]$, debe utilizar en los vértices de G_j por fuerza, colores distintos a los que utilizó en los vértices de $\bigcup_{i<j} c^{-1}[i]$, esto porque al ser c una coloración Grundy, cualquier vértice de G_j es adyacente a un vértice en $c^{-1}[i]$ solo con satisfacer $1 \leq i < j$. Podemos concluir, de hecho, que $s_j = j - 1 + \chi(G_j)$. Si $j \in \{1, \ldots, \gamma(G)\}$, tenemos que G_{j+1} es subgráfica de G_j, así que $\chi(G_{j+1}) \leq \chi(G_j)$, por otro lado, $\chi(G_j) - 1 \leq \chi(G_{j+1})$ porque $c^{-1}[j]$ es un conjunto independiente. Otra observación importante, es que dado lo anterior, tenemos para $j \in \{1, \ldots, \gamma(G)\}$ que $s_j = j - 1 + \chi(G_j) \leq j + \chi(G_{j+1}) = s_{j+1} \leq j + \chi(G_j) = s_j + 1$.

Al ser c una coloración Grundy, c usa a todos los colores, o sea que para todo $1 < j \leq \gamma(G)+1$, la gráfica G_j tiene menos vértices que G. Si $k = \gamma(G)$, la coloración c es una k-coloración. Supongamos que $k < \gamma(G)$ y tomemos a m como el mínimo de los números en $\{1, \ldots, \gamma(G)+1\}$ que satisfacen $s_m \geq k$, tal m existe porque $s_{\gamma(G)+1} = \gamma(G) > k$. Además, m no coincide con $\gamma(G)+1$ porque $k \leq \gamma(G) - 1 = s_{\gamma(G)+1} - 1 \leq s_{\gamma(G)}$. Si $m = 1$, es claro que $s_m = s_1 = \chi(G) = k$. De tenerse $m > 1$, como $s_m - 1 \leq s_{m-1}$, no es posible que $s_m > k$ por la propiedad de mínimo que tiene m. Lo que queremos mostrar con lo anterior, es que siempre se cumple $s_m = k$. Si $k = \chi(G)$, podemos usar el Lema 3.3.1 para exhibir una k-coloración Grundy de G. Por otro lado, si $\chi(G) < k$, se sigue que $m > 1$, entonces, podemos aplicar la hipótesis inductiva para garantizar que la gráfica G_m tiene una $\chi(G_m)$-coloración Grundy, digamos g. Denotemos por f a la función c restringida a $\bigcup_{i<m} c^{-1}[i]$. Tenemos que la función $g + (m-1)$ tiene imagen $\{m, \ldots, \chi(G_m) + (m-1)\}$, y resulta que la función $c' = f \cup (g + (m-1))$ es una s_m-coloración propia de G, porque $(m-1) + \chi(G_m) = s_m$. Además, del hecho de que c y g sean Grundy, se sigue que c' también es Grundy. Por lo tanto, acabamos de definir c' una k-coloración Grundy para G, pues recordemos que $s_m = k$. ■

Si bien, las demostraciones anteriores tienen su extensión y complejidad, son enunciados que vale la pena explorar, ya que nos permite familiarizarnos con la noción de coloración Grundy en contraste a las coloraciones propias no Grundy.

Una relación que tienen las cográficas con las coloraciones, es que en una gráfica G, la igualdad $\chi(G) = \gamma(G)$ resulta ser una condición necesaria y suficiente para que G sea una cográfica. La igualdad anterior en las cográficas es una propiedad que viene transmitida desde las cográficas más pequeñas que conforman a una cográfica dada, de manera similar a lo que ocurre con la propiedad CK. Para adentrarnos un poco en lo anterior, mencionemos algunas cosas que, aunque no vamos a demostrar, son bastante simples. Si $G = \sum_{i=1}^{k} G_i$, entonces $\chi(G) = \text{máx}\{\chi(G_i)\colon 1 \leq i \leq k\}$, pero, si $G = \bigoplus_{i=1}^{k} G_i$, se tiene $\chi(G) = \sum_{i=1}^{k} \chi(G_i)$. Ahora, si $G = \sum_{i=1}^{k} G_i$, tomemos c cualquier coloración Grundy de G, al restringir c a G_i para $1 \leq i \leq k$, obtenemos una coloración propia de G_i, además, esta coloración debe ser Grundy también porque c lo es y los vértices de G_i no pueden ser adyacentes a vértices fuera de G_i. Esto implica que la coloración c restringida a G_i utiliza a lo más, los primeros $\gamma(G_i)$ colores. Por lo tanto, la coloración c de usar a lo más, $\text{máx}\{\gamma(G_i)\colon 1 \leq i \leq k\}$ colores. De lo anterior, podemos ver como la igualdad $\chi(G_i) = \gamma(G_i)$ para todo i, va a implicar que $\chi(G) \leq \gamma(G) \leq \chi(G)$. En caso de que $G = \bigoplus_{i=1}^{k} G_i$, tomemos c cualquier coloración Grundy de G y observemos que c no puede usar el mismo color en vértices de G_i y en vértices de G_j, si $i \neq j$. Así que, podemos modificar el codominio, manteniendo el orden, de la restricción de c a G_i para obtener una coloración propia suprayectiva, además, esa coloración va a ser Grundy porque, como ya se mencionó, los colores de G_i solo se utilizan en G_i. De lo anterior tenemos que c usa a lo más $\sum_{i=1}^{k} \gamma(G_i)$ colores. De nuevo, podemos ver como la igualdad $\chi(G_i) = \gamma(G_i)$ para todo i, va a implicar que $\chi(G) \leq \gamma(G) \leq \chi(G)$. Dada una gráfica G, decimos que G es **$\omega\gamma$-perfecta**, solo cuando toda subgráfica inducida H de G satisface $\omega(H) = \gamma(H)$. De modo similar, diremos que G es **$\chi\gamma$-perfecta** , si toda subgráfica inducida H de G satisface $\chi(H) = \gamma(H)$.

Vamos con el último concepto que presentaremos, este es el de clique-width. Una gráfica etiquetada es una pareja $\langle G, t\rangle$, donde G es una gráfica simple y t es una función de V_G en un conjunto finito de etiquetas S, este conjunto de etiquetas lo vamos a considerar siempre como un subconjunto de $\mathbb{N}^+$. La definición anterior es muy simple, pero no es la única que podemos dar, de hecho, la que daremos a continuación será la que utilizaremos con más frecuencia. Podemos definir una **gráfica etiquetada** como una $(l+1)$-tupla $\langle G, V_1, \ldots, V_l\rangle$, donde G es una gráfica simple y $V_1, \ldots, V_l$ satisfacen que $\{V_1, \ldots, V_l\} \setminus \{\varnothing\}$ es una partición de V_G, en esta definición, se pretende representar los vértices etiquetados con cierta etiqueta s_i como los vértices que estan en el conjunto V_i. En general, el conjunto de etiquetas es un segmento

inicial de $\mathbb{N}^+$, de hecho, cuando $p \in \mathbb{N}^+$, nos referimos con el nombre de **p-gráfica** a las gráficas etiquetadas, que tienen etiquetas en el conjunto $\{1, \ldots, p\}$. Notemos que, si una gráfica no es una gráfica etiquetada, siempre la podemos considerar como una, pues, la podemos ver como una gráfica etiquetada donde todos los vértices tienen la misma etiqueta, es decir, una 1-gráfica.

Sean G y H dos gráficas etiquetadas. No escribimos sus etiquetados porque queremos hacer notar lo siguiente. Si G tiene etiquetas en un conjunto S_1 y H tiene etiquetas en un conjunto S_2, siempre podemos considerar a ambas gráficas con etiquetas en el conjunto $S_1 \cup S_2$. Más aún, si p es cualquier natural tal que $p \geq |S_1 \cup S_2|$, siempre podemos suponer, sin pérdida de generalidad, que G y H son gráficas con etiquetas en el conjunto $\{1, \ldots, p\}$, o sea p-gráficas. Visto lo anterior, podemos escribir las gráficas etiquetadas como $\mathcal{G} = \langle G, V_1, \ldots, V_p \rangle$ y $\mathcal{H} = \langle H, U_1, \ldots, U_p \rangle$. Vamos a extender la unión ajena para que sea una operación que tome dos gráficas etiquetadas y devuelva otra gráfica etiquetada, para distinguirla de la operación sobre gráficas sin etiquetar, utilizaremos el símbolo $\dot{\cup}$. Supondremos, sin pérdida de generalidad, que los conjuntos de vértices de G y H son ajenos, ya que siempre podemos considerar una gráfica H' isomorfa a H que satisfaga $V_G \cap V_{H'} = \varnothing$. Definimos la unión ajena de $\mathcal{G}$ con $\mathcal{H}$ como la gráfica etiquetada

$$\mathcal{G} \dot{\cup} \mathcal{H} = \langle G + H, V_1 \cup U_1, \ldots, V_p \cup U_p \rangle.$$

También vamos a definir dos operaciones unarias que tomen una gráfica etiquetada y devuelvan otra gráfica etiquetada, para esto tomaremos a $\mathcal{G}$ definida como antes. Dadas i, j etiquetas distintas, definimos $\eta_{i \to j}(\mathcal{G})$ como la gráfica etiquetada $\langle G', V_1, \ldots, V_p \rangle$, donde

$$G' \text{ es la gráfica que satisface } E_{G'} = E_G \cup \{uv \colon u \in V_i, v \in V_j\}.$$

Esto es, la gráfica $\eta_{i \to j}(\mathcal{G})$ es la gráfica obtenida de añadir en G las aristas entre vértices con etiqueta i y etiqueta j, dejando las etiquetas como están. Dadas i, j etiquetas distintas, definimos $\rho_{i \to j}(\mathcal{G})$ como la gráfica etiquetada $\langle G, V'_1, \ldots, V'_p \rangle$, donde

$$V'_t = V_t \text{ para toda etiqueta } t \notin \{i, j\};\ V'_i = \varnothing \text{ y } V'_j = V_j \cup V_i.$$

Esto es, la gráfica $\rho_{i \to j}(\mathcal{G})$ es la gráfica G, pero cambiando las etiquetas de los vértices marcados con i para que tengan etiqueta j. También, si i es cualquier etiqueta y v cualquier objeto, definimos la gráfica etiquetada $v(i)$ como la gráfica trivial, cuyo único vértice es v, que tiene su único vértice etiquetado con i. O sea que, si $i \leq p$, podemos decir que $v(i)$ es una p-gráfica.

Con las operaciones anteriores, podemos definir sin meternos en el término formal, la siguiente gramática que describe expresiones.

1. Para cualquier v y cualquier i, $v(i)$ es una expresión.

2. Para cualquier expresión φ y cualesquiera i, j etiquetas distintas, $\eta_{i\to j}(\varphi)$ es una expresión.

3. Para cualquier expresión φ y cualesquiera i, j etiquetas distintas, $\rho_{i\to j}(\varphi)$ es una expresión.

4. Para cualesquiera φ_1, φ_2 expresiones y cualesquiera i, j etiquetas distintas, $(\varphi_1 \dot{\cup} \varphi_2)$ es una expresión.

Como podemos constatar, de acuerdo con el conjunto de reglas anteriores, tenemos que

$$\eta_{2\to 3}(\rho_{2\to 3}((\rho_{1\to 2}(u(1))\dot{\cup}\eta_{2\to 1}((v(1)\dot{\cup}w(2)))))\dot{\cup}x(2))$$

forma una expresión que, según la semántica que dimos a las operaciones definidas, representa, o define, a la gráfica etiquetada de la Figura 3.6.

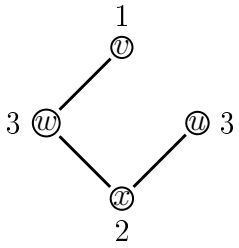


Figura 3.6: Diagrama de la gráfica etiquetada que está definida por la expresión: $\eta_{2\to 3}(\rho_{2\to 3}((\rho_{1\to 2}(u(1))\dot{\cup}\eta_{2\to 1}((v(1)\dot{\cup}w(2)))))\dot{\cup}x(2))$.

La expresión anterior solo utiliza, por así decirlo, ciertas etiquetas, a saber 1,2 y 3. Esto tiene un nombre que definiremos a continuación, para esto fijemos $k \in \mathbb{N}^+$ arbitrario. Decimos que una expresión φ, generada por la gramática anterior, es una **k-expresión**, si todas las etiquetas presentes en φ están dentro del conjunto $\{1, \ldots, k\}$. Con "las etiquetas presentes en φ", nos referimos, desde luego, a las etiquetas utilizadas en subexpresiones del tipo $v(i)$, $\eta_{i\to j}(t)$ y $\rho_{i\to j}(t)$. De la definición anterior se desprende que, si $\mathcal{G}$ es una gráfica etiquetada que utiliza la etiqueta p en alguno de sus vértices y φ es una k-expresión que define a $\mathcal{G}$, entonces $p \leq k$. También podemos notar que, si existe una p-expresión que define a la gráfica etiquetada $\mathcal{G}$, entonces claramente $\mathcal{G}$ es una p-gráfica. A la clase de p-gráficas, para alguna p, que pueden ser definidas por una k-expresión la denotamos por $\mathcal{C}(k)$.

Proposición 3.3.3. *Sea G una gráfica de orden n. Existe una n-gráfica cuya primera componente es G y dicha n-gráfica está en la clase $\mathcal{C}(n)$. Además, tal n-gráfica es de la forma $\mathcal{G} = \langle G, V_1, \ldots, V_n\rangle$ de tal modo que para cada $1 \leq i \leq n$, se cumple $|V_i| = 1$.*

Demostración. Procedamos por inducción sobre n. Si $n = 1$, G solo tiene un vértice, digamos v, entonces $v(1)$ es una 1-expresión que define a una gráfica como la que queremos. Supongamos el enunciado válido para $n \in \mathbb{N}^+$, además, supongamos que G tiene orden $n+1$. Tomemos $v \in V_G$ y apliquemos la hipótesis inductiva para obtener una n-gráfica $\mathcal{H} = \langle G - v, V_1, \ldots, V_n\rangle$ tal que $|V_i| = 1$ para cada $1 \leq i \leq n$, y que además $\mathcal{H}$ esté en $\mathcal{C}(n)$. Consideremos también ψ una n-expresión que defina a la gráfica $\mathcal{H}$. Tomemos el conjunto de etiquetas $S = \{i \in \{1, \ldots, n\} \colon V_i \cap N(v) \neq \varnothing\}$ y demos una enumeración cualquiera $\{s_1, \ldots, s_l\}$ de S. Claramente se satisface que $N(v) = \bigcup_{i \in S} V_i$, por lo que, si definimos la expresión φ como $(v(n+1)\dot{\cup}\psi)$, tendremos que $\eta_{(n+1)\to s_1}(\ldots \eta_{(n+1)\to s_l}(\varphi))$ es una $(n+1)$-expresión que define a la $(n+1)$-gráfica $\langle G, V_1, \ldots, V_n, \{v\}\rangle$. ■

La Proposición 3.3.3 nos dice que, para cualquier gráfica simple G de orden n, se puede encontrar una n-gráfica de la forma $\mathcal{G} = \langle G, V_1, \ldots, V_n\rangle$, donde, para todo $i \in \{1, \ldots, n\}$ se satisface $|V_i| = 1$. Por lo tanto, si tenemos cualquier gráfica etiquetada $\mathcal{H}$ de orden n, de tal forma que su primera componente (su gráfica simple asociada) sea G, podemos obtener una expresión que la genere, simplemente cambiando el etiquetado en la n-expresión descrita en la Proposición 3.3.3 que genera a $\mathcal{G}$, o sea, aplicando los operadores $\rho_{i,j}$ de manera adecuada. Ahora, si k es la máxima etiqueta presente en $\mathcal{H}$, la expresión que nos da como resultado $\mathcal{H}$ podemos tomarla con toda certeza como una m-expresión, donde $m = \text{máx}\{n, k\}$. Todo lo anterior justifica el siguiente corolario.

Corolario 3.3.4. *Si $\mathcal{G}$ es una p-gráfica de orden n, entonces $\mathcal{G}$ pertenece a la clase $\mathcal{C}(\text{máx}\{n, p\})$.*

Tomemos G cualquier gráfica simple y definamos $\mathcal{E}(G)$ como el conjunto de las gráficas etiquetadas que tienen como primera componente a G, esto es, las que son de la forma $\langle G, V_1, \ldots, V_p\rangle$ para algún $p \in \mathbb{N}^+$. Lo que nos asegura la Proposición 3.3.3 es que siempre existen $\mathcal{G}$ en $\mathcal{E}(G)$ y $k \in \mathbb{N}^+$ tales que $\mathcal{G}$ está en la clase $\mathcal{C}(k)$. Así que, podemos definir el clique-width de la gráfica G como el mínimo entre los naturales que satisfagan esto. Es decir, podemos definir el **clique-width** de G como el número

$$cwd(G) = \text{mín}\{k \in \mathbb{N}^+ \colon \text{existe } \mathcal{G} \in \mathcal{E}(G), \mathcal{G} \text{ está en } \mathcal{C}(k)\}.$$

Uno de los aspectos más estudiados del clique-width es, dado $k \in \mathbb{N}^+$ caracterizar a la clase de gráficas que satisfacen $cwd \leq k$. Por ejemplo, tenemos que la clase $\{G\colon cwd(G) = 1\}$ es la clase de gráficas vacías. Vamos a argumentar esto rápidamente haciendo ciertas observaciones clave. Si G es una gráfica vacía con $V_G = \{v_1, \ldots, v_n\}$, claramente la 1-expresión $(\cdots(v_1(1)\dot{\cup}v_2(1))\cdots\dot{\cup}v_n(1))$ define a una gráfica en $\mathcal{E}(G)$ si $n > 1$, mientras que la 1-expresión $v_1(1)$ define a una gráfica en $\mathcal{E}(G)$ si $n = 1$. Por otro lado, sean G cualquier gráfica simple con $cwd(G) = 1$, $\mathcal{G} \in \mathcal{E}(G)$ que esté en $\mathcal{C}(1)$ y φ cualquier 1-expresión que defina a $\mathcal{G}$. No es posible que φ tenga subexpresiones del tipo $\eta_{i\to j}$ para $i \neq j$, pues no hay presencia de dos etiquetas distintas en φ. Por lo que, de acuerdo con la construcción de $\mathcal{G}$ a partir de φ, la gráfica que funge como primera componente de $\mathcal{G}$, o sea la gráfica G, no puede tener aristas. Así queda caracterizada la clase de gráficas con clique-width 1, o lo que es lo mismo, con clique-width a lo más 1, como las gráficas vacías. Resulta que la clase de gráficas con clique-width a lo más 2 es justamente la clase de cográficas, este hecho forma parte de nuestro teorema de caracterización.

3.4. Caracterización de cográficas

Con todos los conceptos de la sección anterior a la mano, vamos a demostrar un teorema que contiene varias caracterizaciones de la familia de cográficas.

Teorema 3.4.1. *Sea G una gráfica simple. Los siguientes enunciados son equivalentes:*

1. *G es una cográfica.*
2. *Toda subgráfica inducida no trivial de G tiene, por lo menos, un par de gemelos.*
3. *Toda subgráfica inducida de G tiene la propiedad CK.*
4. *G no tiene subgráficas inducidas que sean isomorfas a P_4.*
5. *Todas las subgráficas inducidas, conexas y no triviales de G tienen complemento inconexo.*
6. *G es una HD-gráfica.*
7. *Toda subgráfica inducida y conexa de G tiene diámetro menor que 3.*
8. *G es la gráfica de comparabilidad de algún multiárbol.*

9. *G es una gráfica $\omega\gamma$-perfecta.*

10. *G es una gráfica $\chi\gamma$-perfecta.*

11. *G tiene clique-width a lo más 2.*

Demostración.

(1)⇒(2): Sean G una cográfica y $H \subseteq G$ inducida, no trivial. Sabemos que H también es una cográfica, así que podemos considerar a su coárbol T. Elijamos dos hojas de T que tengan un padre común, digamos $x, y \in V_H$ con padre p. Consideremos cualquier $z \in V_H \setminus \{x, y\}$ y planteemos dos casos. Si p es padre de z, entonces $mca(z, x) = p = mca(z, y)$. Por otro lado, si $z \notin H(T_p)$, por la Proposición 2.2.4 tenemos que $mca(z, x) = mca(z, p) = mca(z, y)$. En cualquiera de los dos casos $mca(z, x)$ y $mca(z, y)$ tienen la misma etiqueta, de acuerdo la Proposición 3.1.5 eso implica que $z \in N_H(x)$ si y solo $z \in N_H(y)$. Por lo tanto x y y son gemelos en H.

(2)⇒(3): Sea G una gráfica tal que toda subgráfica inducida tiene al menos un par de gemelos y sea H una subgráfica inducida de G. Procedamos por inducción sobre el orden de H, digamos m. Para $m \in \{1, 2\}$ es inmediato ya que H es completa o $\overline{H}$ es completa. Ahora, supongamos el enunciado válido para $m \geq 2$.

Supongamos que H tiene orden $m + 1$ y consideremos un clan máximo por contención $C \subseteq V_H$, y un conjunto independiente máximo por contención $I \subseteq V_H$. Sean por hipótesis x y y un par de gemelos en H. Si $x \notin C \cup I$, entonces, C e I son un clan máximo por contención y un conjunto independiente máximo por contención en $H - x$ respectivamente. Así que, al tener $H - x$ orden m se sigue que $C \cap I \neq \varnothing$. Un argumento similar es aplicable en caso de que $y \notin C \cup I$. Por lo tanto, el caso que queda por ver es en el que $x, y \in C \cup I$. Vamos a ver que es imposible que $|C \cap \{x, y\}| = 1$, para esto, supongamos por un momento que sí ocurre, eso implicaría que $|I \cap \{x, y\}| = 1$. Supongamos sin pérdida de generalidad que $x \in C$ y $y \in I$. Si x y y son adyacentes, debe existir $z \in C \setminus \{x\}$ no adyacente a y, porque si $C \subseteq N_H(y)$, C no sería un clan máximo por contención, entonces por ser x y y gemelos, z tampoco es adyacente a x, lo cual es absurdo ya que $x, z \in C$. Por otro lado, si x y y no son adyacentes, debe existir $z \in I$ adyacente a x, porque si $N_H(x) \subseteq V_H \setminus I$, se cumple que I no sería un conjunto independiente máximo por contención, entonces por ser x y y gemelos, z también es adyacente a y, lo cual es absurdo ya que $y, z \in I$. Con lo anterior, podemos concluir que es imposible tener $|C \cap \{x, y\}| = 1$, o sea que, $x, y \in C \setminus I$ o bien $x, y \in I \setminus C$. Si $x, y \in C$, se tiene que $C \setminus \{x\}$ es un clan máximo por contención en $H - x$, esto porque si existiera $z \in V_{H-x} \setminus C$ tal que $C \setminus \{x\} \subseteq N_{H-x}(z)$, entonces como x y y son gemelos y $yz \in E_{H-x}$, se sigue que x y z

serían adyacentes en H, de lo que deducimos que $C \cup \{z\}$ es una subgráfica completa inducida de H, pero esto contradice que C es un clan máximo por contención en H. Por lo tanto, $C \setminus \{x\}$ es un clan máximo por contención en $H - x$ y es inmediato que I es un conjunto independiente máximo por contención en $H - x$, lo que implica que por inducción $(C \setminus \{x\}) \cap I \neq \varnothing$. Un argumento análogo muestra que, si $x, y \in I$, entonces C e $I \setminus \{x\}$ son un clan máximo por contención y un conjunto independiente máximo por contención respectivamente en $H - x$, de donde se deduce de nuevo por inducción que $C \cap (I \setminus \{x\}) \neq \varnothing$.

(3)⇒(4): Procedamos por contrapuesta y supongamos que G es una gráfica que contiene a P_4 como subgráfica inducida, digamos que los vértices u_0, u_1, u_2 y u_3 inducen la trayectoria (u_0, u_1, u_2, u_3) en G. Por lo tanto, en esta subgráfica inducida de G, el clan máximo por contención $\{u_2, u_3\}$ y el conjunto independiente máximo por contención $\{u_1, u_4\}$ no se intersectan, es decir, $G[\{u_0, u_1, u_2, u_3\}]$ no tiene la propiedad CK.

(4)⇒(5): Sea G una gráfica que no contiene a P_4 inducida y sea H una subgráfica inducida, no trivial y conexa de G. La prueba será por inducción sobre el orden de H, digamos m. Para $m \in \{2, 3\}$, es claro que $\overline{H}$ es inconexa. Ahora, supongamos el enunciado cierto para $m \geq 3$ y supongamos también que H tiene orden $m + 1$.

Como H es conexa, por la Proposición 2.1.4, H tiene un árbol generador. Así, podemos considerar $u \in V_H$ tal que $H - u$ sigue siendo conexa y podemos tomar también un vecino de u en H, digamos $v \in N_H(u)$. Por hipótesis inductiva, $\overline{H - u} = \overline{H} - u$ es inconexa, así que tomemos sus componentes conexas $H_1, \ldots, H_k$. Supondremos que $\overline{H}$ es conexa para alcanzar una contradicción. Para cada $i \in \{1, \ldots, k\}$, debe existir $w_i \in V_{H_i}$ de tal modo que $w_i u \in E_{\overline{H}}$. Supongamos, sin pérdida de generalidad, que el índice j satisface que $v \in V_{H_j}$. Como H_j es conexa, debe existir una vw_j-trayectoria $(v = x_1, \ldots, x_l = w_j)$, ahora, como $vu \in E_H$ y $w_j u \notin E_H$, se sigue la existencia de un índice $t \in \{1, \ldots, l - 1\}$ tal que $x_t u \in E_H$ y $x_{t+1} u \notin E_H$. Resumiendo todo lo anterior, obtenemos que si i es cualquier índice distinto a j, se cumple que $ux_t, x_t w_i, w_i x_{t+1} \in E_H$ y $ux_{t+1}, uw_i, x_t x_{t+1} \notin E_H$, esto significa que $H[\{u, x_t, x_{t+1}, w_i\}]$ es isomorfa a P_4, lo cual resulta absurdo. Por lo tanto, $\overline{H}$ debe ser inconexa.

(5)⇒(6): Sea G una gráfica tal que, todas sus subgráficas inducidas, conexas y no triviales tienen complemento inconexo. Consideremos H cualquier subgráfica inducida de G. Para verificar que H es Dacey, supongamos que $u, v \in V_H$ son distintos y que C es un clan máximo por contención en H tal que $C \subseteq N_H(u) \cup N_H(v)$. Si $u \in C$, entonces $u \in N_H(v)$ porque $u \notin N_H(u)$. Análogamente, si $v \in C$, se cumple $v \in N_H(u)$; en cualquiera de los casos anteriores obtenemos que $uv \in E_H$, entonces,

supongamos que $u, v \notin C$. Como $\{u\} \cup (N_H(u) \cap C)$ es un clan en H y C es un clan máximo por contención, es imposible que $C \subseteq N_H(u)$, del mismo modo $C \nsubseteq N_H(v)$. Por lo que, existen $x \in C \setminus N_H(u)$ y $y \in C \setminus N_H(v)$, esto implica que $x \in N_H(v)$ y $y \in N_H(u)$. Notemos que $H[u, v, x, y]$ es conexa porque $xv, xy, yu \in E_H$, así que $\overline{H[u, v, x, y]} = \overline{H}[u, v, x, y]$ es inconexa, pero el único caso en el que esto ocurre es en el que $uv \notin E_{\overline{H}}$, pues $xu, yv \in E_{\overline{H}[u,v,x,y]}$. Por lo tanto $uv \in E(H)$.

(6)⇒(7): Procedamos por contrapuesta y supongamos que G es una gráfica que contiene una subgráfica inducida y conexa de diámetro al menos 3. Así, es inmediato que existe una trayectoria (u, v, w, x) en G de modo que $H = G[\{u, v, w, x\}]$ es isomorfa a P_4. Notemos que $\{v, w\}$ es un clan máximo por contención en H claramente contenido en $N_H(u) \cup N_H(x)$, sin embargo, $ux \notin E_H$. Por lo tanto G no es una HD-gáfica.

(7)⇒(8): Antes de todo, verifiquemos que si $\langle X, \leq \rangle$ es un conjunto parcialmente ordenado tal que su gráfica de comparabilidad C_X satisface que, todas sus subgráficas inducidas y conexas tienen diámetro menor que 3, entonces $\langle X, \leq \rangle$ es necesariamente un multiárbol. Sean $w, x \in X$ arbitrarios, tales que $(w, x) \notin \leq$. Supongamos también que $y, z \in X$ satisfacen que $y \in [w, \to) \setminus [x, \to)$ y $z \in [w, \to) \cap [x, \to)$. Tenemos que $(x, w), (y, x), (z, y) \notin \leq$, porque de estar alguno de estos pares en la relación, podríamos utilizar la transitividad para derivar una contradicción. De todo lo observado, podemos garantizar que $x \neq w$, $x \neq y$, $x \neq z$, $z \neq w$ y $z \neq y$. De esto se desprenden dos casos, el primero es que $w = y$, de donde se sigue que $(y, z) \in \leq$. El otro caso, que veremos hasta el final del argumento, es que $w \neq y$. Este caso implica que todos los elementos en $\{w, x, y, z\}$ son distintos. Como C_X es la gráfica de comparabilidad de $\langle X, \leq \rangle$, deducimos que $xz, zw, wy \in E_{C_X}$ y $xw, xy \notin E_{C_X}$. Por lo que $C_X[\{x, z, w, y\}]$ es una gráfica conexa, y por lo tanto tiene diámetro menor que 3, pero $xw, xy \notin E_{C_X}$, así que para tener $d_{C_X}(x, y) < 3$, necesariamente se debe cumplir $yz \in E_{C_X}$. Sin embargo, $(z, y) \notin \leq$, o sea que debe ocurrir $(y, z) \in \leq$. Al ser w y x arbitrarios, concluimos que $\langle X, \leq \rangle$ es un multiárbol.

Sea G una gráfica cuyas subgráficas inducidas y conexas tienen diámetro menor que 3. Podemos suponer sin pérdida de generalidad que G es conexa, esto porque de no serlo, podemos aplicar el caso conexo a cada una de sus componentes conexas G_i para deducir que debe de ser la gráfica de comparabilidad de algún multiárbol $\langle X_i, \leq_i \rangle$. Luego, fácilmente podemos demostrar, proceso que omitiremos por su inmediatez, que el conjunto parcialmente ordenado $\langle \bigcup_i X_i, \bigcup_i \leq_i \rangle$ es un multiárbol que tiene gráfica de comparabilidad G. Procedamos por inducción fuerte sobre el orden de G, digamos n. Si $n = 1$, entonces G es la gráfica de comparabilidad del multiárbol trivial. Ahora, supongamos que $n > 1$ es fijo y que para todo $k < n$ se satisface el

enunciado.

Fijemos $u \in V_G$ y definamos el conjunto $A = \{v \in V_G \colon d(v,u) = 2\}$. Por hipótesis, $V_G = \{u\} \cup N(u) \cup A$. En caso de que $A \neq \varnothing$, consideremos $A_1, \ldots, A_k$ las componentes conexas de $G[A]$ y usemos la hipótesis inductiva para hallar órdenes parciales $\trianglelefteq_1, \ldots, \trianglelefteq_k$ de tal forma que $\langle V_{A_1}, \trianglelefteq_1\rangle, \ldots, \langle V_{A_k}, \trianglelefteq_k\rangle$ son multiárboles con gráficas de comparabilidad $A_1, \ldots, A_k$ respectivamente. Si $A \neq \varnothing$, podemos definir $\trianglelefteq_A = \bigcup_{i=1}^{k} \trianglelefteq_i$ y se satisface que $\langle V_A, \trianglelefteq_A\rangle$ es un conjunto parcialmente ordenado con gráfica de comparabilidad $G[A]$.

Definamos la relación sobre $N(u)$, que claramente es de equivalencia, dada por

$$\sim = \{(x,y) \colon N(x) \cap A = N(y) \cap A\}.$$

Para cada clase de equivalencia $C \subseteq N(u)$, por hipótesis inductiva, podemos hallar un orden parcial sobre $G[C]$, digamos $\trianglelefteq_C$, de tal modo que $\langle G[C], \trianglelefteq_C\rangle$ es un multiárbol con gráfica de comparabilidad $G[C]$. Definimos la relación sobre $N(u)$ dada por

$$\trianglelefteq_{N(u)} = \Big(\bigcup_{C \in (N(u)/\sim)} \trianglelefteq_C\Big) \cup \{(x,y) \colon x,y \in N(u), x \nsim y, xy \in E_G, N(y) \cap A \subseteq N(x) \cap A\}.$$

Nuestro objetivo es demostrar que $\trianglelefteq_{N(u)}$ es un orden parcial sobre $N(u)$. Por un lado, tenemos que $\trianglelefteq_{N(u)}$ es reflexiva porque si $x \in N(u)$, entonces $(x,x) \in \trianglelefteq_{[x]_\sim} \subseteq \trianglelefteq_{N(u)}$. También es antisimétrica porque si $x, y \in N(u)$ son tales que $(x,y), (y,x) \in \trianglelefteq_{N(u)}$, hay dos opciones por la definición de $\trianglelefteq_{N(u)}$. La primera es que $x \sim y$, si esto ocurre, se deduce $x = y$ porque $(x,y), (y,x) \in \trianglelefteq_{[x]_\sim}$, y $\trianglelefteq_{[x]_\sim}$ es un orden parcial. La segunda opción, que no puede ocurrir por cierto, es que $x \nsim y$. De ocurrir lo anterior, tenemos que $N(x) \cap A \subseteq N(y) \cap A$ y $N(y) \cap A \subseteq N(x) \cap A$, o sea $N(x) \cap A = N(y) \cap A$, lo cual implica que $x \sim y$. Por lo tanto $\trianglelefteq_{N(u)}$ es antisimétrica.

Solo falta ver que $\trianglelefteq_{N(u)}$ es transitiva, para esto tomemos $x, y, z \in N(u)$ de tal modo que $(x,y), (y,z) \in \trianglelefteq_{N(u)}$. Observemos que la definición de la relación, se deduce que $N(y) \cap A \subseteq N(x) \cap A$ y $N(z) \cap A \subseteq N(y) \cap A$, por lo tanto $N(z) \cap A \subseteq N(x) \cap A$. Tenemos dos casos, el primero es que $N(z) \cap A = N(x) \cap A$, si esto fuera verdad se tiene que $N(x) \cap A = N(y) \cap A = N(z) \cap A$ y por ende $(x,y), (y,z) \in \trianglelefteq_{[x]_\sim}$, de esta manera, al ser $\trianglelefteq_{[x]_\sim}$ un orden parcial ocurriría que $(x,z) \in \trianglelefteq_{[x]_\sim} \subseteq \trianglelefteq_{N(u)}$. El otro caso es que $N(z) \cap A \subsetneq N(x) \cap A$. Tomemos $w \in (N(x) \setminus N(z)) \cap A$. Por hipótesis el diámetro de $G[\{u,x,z,w\}]$ es menor que 3 por ser conexa, sin embargo, $zw \notin E_G$ y $u \in N(z) \setminus N(w)$, por lo tanto $x \in N(z) \cap N(w)$. De esto concluimos que $(x,z) \in \trianglelefteq_{N(u)}$ porque $z, x \in N(u)$ satisfacen $zx \in E_G, x \nsim z$ y $N(z) \cap A \subseteq N(x) \cap A$. Por lo tanto $\trianglelefteq_{N(u)}$ es un orden parcial para $N(u)$.

Demostremos ahora que $G[N(u)]$ es la gráfica de comparabilidad de $\langle N(u), \trianglelefteq_{N(u)}\rangle$. Sean $x, y \in N(u)$ vértices adyacentes. Puede ocurrir que $x \sim y$, en cuyo caso

$(x,y) \in \trianglelefteq_{[x]_\sim}$ o $(y,x) \in \trianglelefteq_{[x]_\sim}$ porque $G[[x]_\sim]$ es la gráfica de comparabilidad del multiárbol $\langle [x]_\sim, \trianglelefteq_{[x]_\sim} \rangle$. Ahora abordemos el caso en el que $x \nsim y$. Observemos que debe cumplirse alguna de las contenciones $N(x) \cap A \subseteq N(y) \cap A$ o $N(y) \cap A \subseteq N(x) \cap A$, para ver que esto es cierto, supongamos que no se satisface ninguna de las contenciones y derivemos una contradicción. Como no se cumple ninguna de las contenciones anteriores, deben existir $w \in (N(x) \setminus N(y)) \cap A$ y $z \in (N(y) \setminus N(x)) \cap A$. Dado que la gráfica $G[\{w,x,y,z\}]$ es conexa, debe tener diámetro menor que 3, pero, como $x \in N(w) \setminus N(z)$ y $y \in N(z) \setminus N(w)$, debe ocurrir $wz \in E_G$. Sin embargo, con lo anterior, la gráfica $G[\{z,w,x,u\}]$ es conexa y por ende tiene diámetro menor que 3, pero $zu \notin E_G, w \in N(z) \setminus N(u)$ y $x \in N(u) \setminus N(z)$, lo cual es absurdo. Por lo tanto, debe ser cierto que $N(x) \cap A \subseteq N(y) \cap A$ o que $N(y) \cap A \subseteq N(x) \cap A$. De lo anterior, deducimos que $(x,y) \in \trianglelefteq_{N(u)}$ o $(y,x) \in \trianglelefteq_{N(u)}$. Por otro lado, supongamos ahora que $(x,y) \in \trianglelefteq_{N(u)}$, con $x \neq y$. Tenemos los mismos dos casos, comencemos suponiendo que $x \sim y$, o sea que $(x,y) \in \trianglelefteq_{[x]_\sim}$; en este caso podemos afirmar que $xy \in E_G$ porque $G[[x]_\sim]$ es la gráfica de comparabilidad de $\langle [x]_\sim, \trianglelefteq_{[x]_\sim} \rangle$. El otro caso es que $x \nsim y$, en este caso, por la definición de $\trianglelefteq_{N(u)}$ se debe satisfacer que $xy \in E_G$. Por lo tanto $G[N(u)]$ es la gráfica de comparabilidad de $\langle N(u), \trianglelefteq_{N(u)} \rangle$.

Ahora, definamos la relación

$$\trianglelefteq = Id_{V_G} \cup \trianglelefteq_{N(u)} \cup \trianglelefteq_A \cup \{(x,u) : x \in N(u)\} \cup \{(x,y) : x \in N(u), y \in A, xy \in E_G\}.$$

Probemos que es un orden parcial para G. Es inmediato que $\trianglelefteq$ es reflexiva por que contiene a Id_{V_G}. Para probar la antisimetría, tomemos $x,y \in V_G$ tales que $(x,y),(y,x) \in \trianglelefteq$. Si $x = u$, por la definición de $\trianglelefteq$ tendríamos que $u = x = y$. Si $x \in A$, por la definición de $\trianglelefteq$ junto con el hecho de que $(x,y) \in \trianglelefteq$, se sigue que $y \in A$. Tenemos que $x = y$, porque $(x,y),(y,x) \in \trianglelefteq_A$, y esta última relación es un orden parcial. Por último, si $x \in N(u)$, se sigue que $y \in N(u)$ por la definición de $\trianglelefteq$ y el hecho de que $(y,x) \in \trianglelefteq$; se sigue que $x = y$, porque $(x,y),(y,x) \in \trianglelefteq_{N(u)}$ y esta última relación es un orden parcial.

Terminemos por probar la transitividad de $\trianglelefteq$, para esto consideremos $x,y,z \in V_G$ tales que $(x,y),(y,z) \in \trianglelefteq$. Los casos inmediatos son que $x = u$ o que $x \in A$, pues en el primero $x = y = z = u$ y en el segundo $x,y,z \in A$, esto gracias a la definición de $\trianglelefteq$. Abordemos el último caso, que es $x \in N(u)$. Una vez en este supuesto podemos identificar otro caso trivial, este es cuando $y = u$ ya que eso implica que $z = u$ y por ende $(x,z) \in \trianglelefteq$. Exploremos que ocurre cuando $y \in A$; podemos notar que se deduce $z \in A$, más aún, se sigue que $(y,z) \in \trianglelefteq_A$. Al ser $G[A]$ la gráfica de comparabilidad de $\langle A, \trianglelefteq_A \rangle$, tenemos dos opciones, la primera es que $y = z$ lo cual implica $(x,z) \in \trianglelefteq$, y la segunda es que $y \neq z$ lo cual deriva que $yz \in E_G$. Además, de la definición de $\trianglelefteq$ junto con el hecho de que $(x,y) \in \trianglelefteq$, podemos grarantizar

que $xy \in E_G$. Con lo anterior, podemos ver que $G[\{u,x,y,z\}]$ es conexa y por lo tanto debe tener diámetro menor que 3. Sin embargo, $uz \notin E_G$ y $y \in N(z) \setminus N(u)$, así que $x \in N(u) \cap N(z)$, en particular $xz \in E_G$; resulta que por la definición de $\trianglelefteq$, se cumple $(x,z) \in \trianglelefteq$. Por último, supongamos que $y \in N(u)$, notemos que esta suposición implica $(x,y) \in \trianglelefteq_{N(u)}$. Podemos desembarazarnos del caso inmediato, esto es, cuando $z \in N(u)$, pues de ocurrir esto tendriamos $(x,y),(y,z) \in \trianglelefteq_{N(u)}$ con $\trianglelefteq_{N(u)}$ un orden parcial. Otro caso sería que $z = u$, esto implica que $(x,z) \in \trianglelefteq$ porque $x \in N(u)$. En el caso final tenemos que $z \in A$, pero recordemos de la definición de $\trianglelefteq_{N(u)}$ que, $(x,y) \in \trianglelefteq_{N(u)}$ implica $N(y) \cap A \subseteq N(x) \cap A$. Como $yz \in E_G$. esto porque $(y,z) \in \trianglelefteq$ con $y \in N(u)$ y $z \in A$, se sigue que $xz \in E_G$. Así, recurriendo de nuevo a la definición de $\trianglelefteq$, obtenemos que $(x,z) \in \trianglelefteq$. Por lo tanto, $\trianglelefteq$ es un orden parcial para V_G.

Para ver que la gráfica G es en efecto la gráfica de comparabilidad de $\langle V_G, \trianglelefteq\rangle$, tomemos $x,y \in V_G$ adyacentes. Si $x,y \in N(u)$ o bien $x,y \in A$, se sigue que $(x,y) \in \trianglelefteq$ o que $(y,x) \in \trianglelefteq$, porque $G[N(u)]$ es la gráfica de comparabilidad de $\langle N(u), \trianglelefteq_{N(u)}\rangle$ con $\trianglelefteq_{N(u)} \subseteq \trianglelefteq$ y, $G[A]$ es la gráfica de comparabilidad de $\langle A, \trianglelefteq_A\rangle$ con $\trianglelefteq_A \subseteq \trianglelefteq$. Si $u \in \{x,y\}$, podemos suponer sin pérdida de generalidad que $y = u$. Como $xy \in E_G$, se cumple que $x \in N(u)$ y por lo tanto $(x,y) \in \trianglelefteq$. Solo falta discutir el caso donde $|\{x,y\} \cap N(u)| = 1 = |\{x,y\} \cap A|$, en este caso supongamos sin pérdida de generalidad que $y \in A$ y $x \in N(u)$. Tenemos directamente por definición que $(x,y) \in \trianglelefteq$, pues $xy \in E_G$. Reciprocamente, si $(x,y) \in \trianglelefteq$ con $x \neq y$, tenemos por definición de $\trianglelefteq$ los siguientes cuatro casos: tanto x como y están en $N(u)$ y $(x,y) \in \trianglelefteq_{N(u)}$; ambos vértices están en A y $(x,y) \in \trianglelefteq_A$; $y = u$ y $x \in N(u)$; $x \in N(u)$, $y \in A$ y $xy \in E_G$. En los primeros dos casos, se sigue que $xy \in E_G$ porque $G[N(u)]$ y $G[A]$ son las gráficas de comparabilidad de $\langle N(u), \trianglelefteq_{N(u)}\rangle$ y $\langle A, \trianglelefteq_A\rangle$ respectivamente, mientras que en los dos últimos casos es obvio que $xy \in E_G$. Ya podemos concluir que G es la gráfica de comparabilidad de $\langle V_G, \trianglelefteq\rangle$, por otro lado, el primer párrafo de la demostración de esta implicación, garantiza que $\langle V_G, \trianglelefteq\rangle$ es un multiárbol.

(8)⇒(9): Primero consideremos a cualquier gráfica que sea la gráfica de comparabilidad de un multiárbol arbitrario, digamos la gráfica H tal que es gráfica de comparabilidad de $\langle V_H, \trianglelefteq\rangle$. Observemos que H no puede tener una gráfica inducida isomorfa a P_4. Para ver esto supongamos que sí, es decir, supongamos que tiene a la trayectoria (w,x,y,z) de tal modo que $H[\{w,x,y,z\}]$ es isomorfa a P_4. Consideremos $X = \{(a,b) \colon a,b \in \{w,x,y,z\}\} = \{w,x,y,z\}^2$. Es fácil ver que los únicos casos posibles son $X \cap \trianglelefteq = \{(w,x),(y,x),(y,z)\}$ o $X \cap \trianglelefteq = \{(x,w),(x,y),(z,y)\}$, esto porque $wx, xy, yz \in E_H$, pero $wy, xz, wz \notin E_H$. En el primer caso, por la hipótesis de multiárbol y debido a que $(y,w) \notin \trianglelefteq$, $z \in [y,\rightarrow) \setminus [w,\rightarrow)$, pero $x \in [y,\rightarrow) \cap [w,\rightarrow)$, se debe tener $(z,x) \in \trianglelefteq$, lo cuál es absurdo. En el segundo caso, tenemos que $(x,z) \notin \trianglelefteq$,

$w \in [x, \to) \setminus [z, \to)$ y $y \in [x, \to) \cap [z, \to)$, así que por la hipótesis de multiárbol, se sigue que $(w, y) \in \trianglelefteq$, algo absurdo. Por lo tanto, podemos concluir que H no contiene subgráficas inducidas que sean isomorfas a P_4.

Sean G la gráfica de comparabilidad de un multiárbol cualquiera y H una subgráfica inducida de G. Gracias al primer párrafo de la prueba, podemos garantizar que G no tiene subgráficas inducidas que sean isomorfas a P_4. Consideremos también $C_1, \ldots, C_{\gamma(H)}$ una coloración Grundy para H con $\gamma(H)$ colores, desde luego, pensando la coloración como una partición de V_H. Supongamos que $\omega(H) < \gamma(H)$ para llegar a una contradicción. Definamos φ como la afirmación sobre los números en $\{1, \ldots, \gamma(H)\}$ dada por $\varphi(t)$: existe un $(\gamma(H) + 1 - t)$-clan en H, digamos $K \subseteq V_H$, tal que para toda $i \in \{1, \ldots, \gamma(H)\}$ se cumple que, $K \cap C_i \neq \varnothing$ si y solo si, $i \geq t$.

Observemos que $\varphi(\gamma(H))$ se satisface, esto porque cualquier subconjunto unitario de $C_{\gamma(H)}$ es un 1-clan, o sea un $(\gamma(H)+1-\gamma(H))$-clan, que satisface lo requerido. Así, podemos definir $m = \text{mín}\{t \in \{1, \ldots, \gamma(H)\} \colon \varphi(t) \text{ se cumple}\}$. Notemos que $m > 1$, porque no existen $(\gamma(H)+1-1)$-clanes en H. Elijamos $K \subseteq V_H$ un $(\gamma(H)+1-m)$-clan, de tal modo que para cualquier $i \in \{1, \ldots, \gamma(H)\}$, $K \cap C_i \neq \varnothing$ si y solo si, $i \geq m$. Como $m > m - 1 \geq 1$, podemos tomar $w \in C_{m-1}$ de tal forma que, entre todos los vértices de C_{m-1}, w maximiza la cardinalidad de $N(w) \cap K$. Notemos que $K \cup \{w\}$ no puede ser un clan porque, dada la propiedad de mínimo que tiene m, $\varphi(m-1)$ no se puede cumplir. Por lo que existen $j \geq m$, y $y \in C_j \cap K$, tales que $wy \notin E_H$. Por ser la coloración Grundy, podemos hallar $z \in C_{m-1}$ de tal forma que $yz \in E_H$. Por la elección de w en C_{m-1}, no es posible que $N(w) \cap K \subsetneq N(z) \cap K$, esto junto con el hecho de que $y \in (N(z) \cap K) \setminus (N(w) \cap K)$, implica la existencia de $x \in K$ que satisface $x \in N(w) \setminus N(z)$. Por lo tanto $wx, xy, yz \in E_H$ y $wy, xz, wz \notin E_H$, sin embargo, esto significa que $H[\{w, y, z, x\}]$ es una subgráfica inducida de H, y por ende subgráfica inducida de G, que es isomorfa a P_4, una contradicción. La contradicción anterior deviene por la suposición de que $\omega(H) < \gamma(H)$, de donde tenemos $\omega(H) = \gamma(H)$. Como H era arbitraria, podemos concluir que G es una gráfica $\omega\gamma$-perfecta.

(9)⇒(10): Es inmediato por las desigualdades $\omega(G) \leq \chi(G) \leq \gamma(G)$ que se satisfacen para cualquier gráfica simple G.

(10)⇒(1): Primero que nada, demostremos que si H es una gráfica isomorfa a P_4, entonces $\chi(H) = 2 < 3 = \gamma(H)$. Supongamos que la gráfica H tiene vértices x_1, x_2, x_3, x_4 y la trayectoria inducida (x_1, x_2, x_3, x_4). Por un lado, $\gamma(H) \leq 3$ porque H no tiene vértices de grado 3, por otro lado, se cumple $\gamma(H) \geq 3$ porque la 3-coloración dada por $C_1 = \{x_1, x_4\}, C_2 = \{x_2\}, C_4 = \{x_3\}$ es Grundy. El número cromático de H es 2 debido a que $C_1 = \{x_1, x_3\}, C_2 = \{x_2, x_4\}$ es una coloración propia de H, mientras que claramente $\chi(H) \geq 2$. De lo anterior, tenemos para cualquier

gráfica H donde se cumpla $\chi(H) = \gamma(H)$ la imposibilidad de que H sea isomorfa a P_4. El corolario que vamos a utilizar de lo anterior, es que las gráficas $\chi\gamma$-perfectas no tienen subgráficas inducidas que sean isomorfas a P_4.

Sea G una gráfica arbitraria $\chi\gamma$-perfecta. La prueba será por inducción fuerte sobre el orden de G. Si G tiene orden 1 es inmediato, pues las gráficas vacías con un solo vértice son cográficas. Tomemos $n > 1$ fijo y supongamos que para cualquier $1 \leq l < n$ se satisface el enunciado. Supongamos también que G tiene orden n. Observemos que si G es inconexa y $G_1, \ldots, G_k$ son sus componentes conexas, para cada $i \in \{1, \ldots, k\}$, la gráfica G_i también es $\chi\gamma$-perfecta, así que por hipótesis inductiva es una cográfica. Como G es la unión ajena de $G_1, \ldots, G_k$, se sigue que G es también cográfica. De aquí en adelante veremos el caso conexo, o sea, supondremos que G es conexa.

Tomemos $C_1, \ldots, C_{\gamma(G)}$ una coloración Grundy de G con $\gamma(G)$ colores, y sea $u \in C_{\gamma(G)}$ arbitrario. Como toda subgráfica inducida de $G-u$ es una subgráfica inducida de G, la gráfica $G-u$ también es $\chi\gamma$-perfecta. Apliquemos la hipótesis inductiva para hallar el coárbol T de la cográfica $G-u$. El primer caso a considerar es que la raíz de T tenga etiqueta 0, esto implica que la gráfica $G-u$ es la unión ajena de algunas cográficas, digamos de $G_1, \ldots, G_k$ con $k \geq 2$. Como G es conexa, para todo $i \in \{1, \ldots, k\}$ existe $v_i \in V_{G_i}$ de tal modo que $uv_i \in E_G$. Probaremos que $N(u) = V_G \setminus \{u\}$, para esto, supondremos que existe $x \in V_G \setminus \{u\}$ no adyacente a u, así vamos a poder inferir una contradicción. Sea $j \in \{1, \ldots, k\}$ tal que $x \in V_{G_j}$. Como G_j es conexa, existe $(x = w_0, \ldots, v_j = w_l)$ una xv_j-trayectoria en G_j, pero ya que $xu \notin E_G$ y $v_j u \in E_G$, podemos tomar $t = \text{mín}\{i \in \{0, \ldots, l-1\} \colon w_i u \notin E_G \text{ y } w_{i+1} u \in E_G\}$. Por lo tanto, para cualquier $i \in \{1, \ldots, k\}$ distinto de j tenemos que $w_t v_i, w_{t+1} v_i, w_t u \notin E_G$ y $w_t w_{t+1}, w_{t+1} u, uv_i \in E_G$, pero esto implica que $G[w_t, w_{t+1}, u, v_i]$ es isomorfa a P_4, lo cual es absurdo. Por lo tanto $N(u) = V_{G-u}$. Así pues, G resulta ser una cogŕafica por ser la unión completa de $G - u$ con u.

El otro caso es que la raíz de T tenga etiqueta 1, esto quiere decir que $G - u$ es la unión completa de algunas cográficas, digamos de $G_1, \ldots, G_k$ con $k \geq 2$. Si $N(u) = V_G \setminus \{u\}$, entonces G es la unión completa de $G_1, \ldots, G_k, G[\{u\}]$. Ahora, supongamos que existe $x \in V_G \setminus \{u\}$ tal que $xu \notin E_G$ y sea $j \in \{1, \ldots, k\}$ tal que $x \in V_{G_j}$. Adicionalmente consideremos $i \in \{1, \ldots, k\}$ distinto a j. Probaremos que u es adyacente a todos los vértices de G_i, para esto, supongamos que $y \in V_{G_i}$. Primero veamos el caso en el que $x \notin C_{\gamma(H)}$. Por ser la coloración Grundy, podemos hallar $z \in V_G$ del mismo color que x y adyacente a u. Se debe tener que $z \in V_{G_j}$ porque z es del mismo color que x y $G - u = \bigoplus_{s=1}^{k} G_s$. Si se tuviera $uy \notin E_G$, se tendría que $uz, zy, yx \in E_G$ y $ux, uy, xz \notin E_G$, de donde se sigue que $G[\{u, x, y, z\}]$ es isomorfa a P_4, una contradicción. Por lo tanto, garantizamos que $uy \in E_G$. El

otro caso es que $x \in C_{\gamma(H)}$, en este caso $y \notin C_{\gamma(H)}$ porque $xy \in E_G$. Si suponemos que $uy \notin E_G$, por ser la coloración Grundy, podemos encontrar $z \in V_G$ del mismo color que y de tal manera que $uz \in E_G$. Resulta que $z \in V_{G_i}$ por ser z del mismo color de y. De este modo, tendríamos que $uz, xy, xz \in E_G$ y $ux, uy, yz \notin E_G$, esto es absurdo por implicar que $G[u, x, y, z]$ es isomorfa a P_4. Por lo tanto, $uy \in E_G$. Como ya cubrimos todos los casos y el vértice y fue arbitrario, concluimos que u es adyacente a todos los vértices de V_{G_i}. A su vez, i era arbitrario, por lo que podemos concluir que $\bigcup_{i \neq j} V_{G_i} \subseteq N(u)$. Con todo lo anterior, podemos utilizar la hipótesis inductiva para garantizar que $H = G[V_{G_j} \cup \{u\}]$ es una cográfica. Como G es la unión completa de las cográficas $G_1, \ldots, G_{j-1}, H, G_{j+1}, \ldots, G_k$, se deduce que G es una cográfica también.

(1)⇒(11): Primero, vamos a probar que si G y H son gráficas con clique-width a lo más 2, entonces, el clique-width de las gráficas $G + H$ y $G \oplus H$ es a lo más 2. Sean $\mathcal{G} = \langle G, V_1, V_2 \rangle$ y $\mathcal{H} = \langle H, W_1, W_2 \rangle$ gráficas etiquetadas y ajenas por vértices tales que, las 2-expresiones φ y ψ corresponden a las gráficas $\mathcal{G}$ y $\mathcal{H}$ respectivamente. Como la 2-expresión $(\varphi \dot{\cup} \psi)$ corresponde a $\langle G + H, V_1 \cup W_1, V_2 \cup W_2 \rangle$ y la 2-expresión $\eta_{1\to 2}((\rho_{2\to 1}(\varphi) \dot{\cup} \rho_{1\to 2}(\psi)))$ corresponde a $\langle G \oplus H, V_1 \cup V_2, W_1 \cup W_2 \rangle$, podemos concluir que $cwd(G + H) \leq 2$ y $cwd(G \oplus H) \leq 2$. Con un argumento inductivo, lo anterior se generaliza fácilmente a uniones ajenas, o uniones completas, de más de dos gráficas.

Procedamos por inducción fuerte sobre el orden. Si G es una cográfica de orden 1, entonces G es una gráfica vacía de un solo vértice, digamos con vértice x; resulta que la 1-expresión $x(1)$ atestigua que $cwd(G) = 1 < 2$. Supongamos que $n > 1$ es fijo y que el enunciado se satisface para culaquier $l < n$. Consideremos una cográfica G de orden n, como el orden de G es mayor a 1, la cográfica G es la unión ajena de otras cográficas, o bien, es la unión completa de otras cográficas. Utilizando la hipótesis inductiva y el primer párrafo de la demostración, se sigue que G tiene clique-width a lo más 2.

(11)⇒(4): Primero vamos a probar que, si φ es una 2-expresión que genera a la gráfica etiquetada $\mathcal{G} = \langle G, V_1, V_2 \rangle$, entonces G no contiene subgráficas inducidas e isomorfas a P_4. La demostración va a ser por inducción fuerte sobre la longitud de la expresión, digamos ℓ.

Es claro por la definición de las expresiones que el caso base es para $\ell = 4$, caso que corresponde a una expresión del tipo $x(i)$, con i una etiqueta. Una expresión del tipo mencionado está asociada a una gráfica etiquetada trivial, así que se cumple el enunciado. Ahora, tomemos $\ell > 4$ y supongamos válido el enunciado para toda $m < \ell$. Si φ es una 2-expresión de longitud $\ell > 4$, hay tres casos, el primero es que φ sea de la forma $(\psi \dot{\cup} \phi)$ para ψ y ϕ expresiones, en este caso tanto ψ como ϕ son 2-

expresiones por ser subexpresiones de la 2-expresión φ. Sean $\mathcal{G}_\psi = \langle G_\psi, V_1, V_2\rangle$ y $\mathcal{G}_\phi = \langle G_\phi, W_1, W_2\rangle$ las gráficas correspondientes a las expresiones ψ y ϕ respectivamente. Como ψ y ϕ tiene longitud menor que ℓ, podemos aplicar la hipótesis inductiva para garantizar que G_ψ y G_ϕ no tienen subgráficas inducidas isomorfas a P_4. Tenemos que la gráfica correspondiente a $\varphi = (\psi \dot\cup \phi)$ es la gráfica etiquetada $\langle G_\psi + G_\phi, V_1 \cup W_1, V_2 \cup W_2\rangle$, y resulta que $G_{\psi\dot\cup\phi}$ no tiene subgráficas inducidas e isomorfas a P_4 porque cualquier subgráfica inducida de $G_{\psi\dot\cup\phi}$ es, o bien una subgrafica inducida de G_ψ o de G_ϕ, o bien es una subgráfica inducida con vértices de G_ψ y G_ϕ en cuyo caso es inconexa. Los otros dos casos son que φ sea de la forma $\mathcal{E}(\psi)$ con ψ una 2-expresión y $\mathcal{E}$ uno de los símbolos $\rho_{i\to j}$ o $\eta_{i\to j}$ para $i \neq j$. Sea $\mathcal{G} = \langle G, V_1, V_2\rangle$ la gráfica correspondiente a ψ. Si $\mathcal{E}$ es $\rho_{i\to j}$, la gráfica etiquetada de φ es $\langle G, \varnothing, V_1 \cup V_2\rangle$ cuando $i = 1$, y es $\langle G, V_1 \cup V_2, \varnothing\rangle$ cuando $i = 2$, así que φ satisface lo que queremos. Si $\mathcal{E}$ es $\eta_{i\to j}$, entonces la gráfica etiquetada asociada a φ es $\langle H, V_1, V_2\rangle$, donde $V_H = V_G$ y $E_H = E_G \cup \{uv \colon u \in V_1, v \in V_2\}$. Consideremos una trayectoria (a, b, c, d) en H. Si suponemos que $H[\{a, b, c, d\}]$ es isomorfa a P_4, se debe tener que $ad, ac, bd \notin E_H$. Denotando por t a la etiqueta de a, lo que nos dice la ausencia de las aristas ad, ac, bd en H es que las etiquetas de b, c y d son también t, o sea que (a, b, c, d) ya era una trayectoria en G y por ende G tiene una subgráfica inducida isomorfa a P_4, a saber $G[\{a, b, c, d\}]$, pero esto contradice que G no tiene subgráficas inducidas e isomorfas a P_4, hecho que sabemos por la hipótesis inductiva aplicada a ψ.

Por lo tanto, si G es una gráfica con clique-width a lo más 2, existe una 2-expresión φ y una gráfica etiquetada $\langle G, V_1, V_2\rangle$ asociada a φ. Por todo lo anterior, G no tiene subgráficas inducidas e isomorfas a P_4. ■

Capítulo 4

BFS-Lexicográfico y algoritmo de reconocimiento

4.1. BFS-Lexicográfico

En esta sección estudiaremos el algoritmo de BFS-lexicográfico, para ello vamos a repasar algunos conceptos. Como ya vimos en la Sección 1.9, dado X cualquier conjunto finito, un ordenamiento de X es cualquier función biyectiva $\tau\colon X \to |X|$, donde $|X|$ es el conjunto cardinal de X, o sea $|X| = \{0, \ldots, |X|-1\}$. Teniendo nuestro conjunto finito X y τ un ordenamiento de X, podemos definir el orden total para X inducido por τ, esto es la relación $\leq_\tau = \{(x, y)\colon \tau(x) \leq \tau(y)\}$. En la Sección 1.9 introdujimos las estructuras de datos y algunas instancias de estas, en particular discutimos las listas doblemente ligadas. Recordando, si L es una lista doblemente ligada y c es una celda de L, $valor_L(c)$ o simplemente $valor(c)$ devuelve el valor almacenado en la celda c; cuando hay celdas en L depués de c podemos obtener la inmediata siguiente con $siguiente_L(c)$; cuando hay celdas en L antes de c podemos obtener la inmediata anterior con $anterior_L(c)$. Si L es vacía solo podemos agregar elementos, así que en este caso tenemos las operaciones $addL(L, x)$ y $addR(L, x)$ que hacen ecxatamente lo mismo, que es guardar a x en una celda y añadir a L dicha celda. Si L no es vacía tenemos las operaciones $addL(L, c, x)$ y $addR(L, c, x)$, la primera guarda a x en una celda y agrega a L dicha celda exactamente a la izquierda de c, la segunda operación guarda a x en una celda y añade a L dicha celda pero justo a la derecha de c. Además también tenemos la operación $del(L, c)$ cuando L no es vacía, esta operación elimina a la celda c de L, asegurando la integridad de la lista, o sea, conectando las celdas siguiente y anterior para "rellenar" el espacio generado. Cuando L no es vacía podemos designar dos operaciones más, estas son $primero(L)$

y $ultimo(L)$ que devuelven el primer y el último elemento de L respectivamente. También es menester tener en cuenta que, por su definición, todas las operaciones anteriores se llevan a cabo en tiempo constante.

Si L es una lista, podemos definir un orden sobre las celdas de la lista utilizando simplemente la noción natural de su sucesión, es decir, dadas dos celdas de L, digamos c_1 y c_2, decimos que c_1 está antes de c_2 en L, escrito como $c_1 \leq_L c_2$, cuando podamos llegar de la celda c_1 a la celda c_2 aplicando sucesivamente, cero o más veces, la operación $siguiente_L$ a c_1. Otro comentario pertinente que debemos hacer es que muchas veces vamos a omitir la notación sobre las acciones en una lista, es decir, podemos escribir, por poner un ejemplo, la frase "cambiamos de lugar en L a c con su celda anterior, y agregamos al elemento x exactamente entre ellas" en lugar de escribir el engorroso bloque de instrucciones

1 $addR(L, c, valor_L(anterior_L(c)))$;

2 $del(L, anterior_L(c))$;

3 $addR(L, c, x)$;

También abusaremos mucho, cuando el contexto disperse la ambigüedad, de homologar las celdas de una lista con sus respectivos valores, esto con el fin de matener liviana la redacción. Es decir, si sabemos, por dar un ejemplo, que todos los valores de celdas en la lista son números, escribiremos para dos celdas c_1 y c_2 de L la expresión $c_1 + c_2$ para referirnos al número $valor_L(c_1) + valor_L(c_2)$. Como una última cosideración, concensuemos en que en este capítulo, como en la mayor parte de este trabajo, nos referiremos con gráfica a una gráfica simple, a menos que se especifique lo contrario. Sin más dilación, demos nuestra primera versión del algoritmo BFS-lexicográfico, abreviado simplemente como **LexBFS**, la cual se presenta en el Algoritmo 4.

En la Figura 4.1 podemos ver el estado de la lista y la construcción de σ en una ejecución del algoritmo LexBFS, utilizando como entrada a la gráfica H y al ordenamiento $\tau = (v_3, v_2, v_4, v_5, v_1)$.

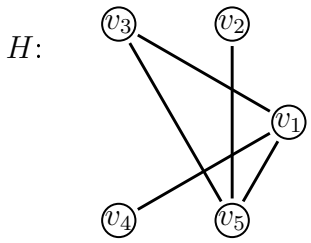
Podemos observar en el Algoritmo 4 que en cada iteración del `while`, las celdas de L cuyo valor es no vacío, forman una partición del conjunto de vértices que no tienen asignado un valor para σ aún. Lo que hace, a grandes rasgos, el Algoritmo 4 es refinar en cada iteración del `while` a cada uno de los conjuntos que forman la partición dada por L, esta refinación se hace respecto a la vecindad del vértice en turno. O sea que, dado $0 < k < |V_G|$, el algoritmo toma dos consideraciones para decidir a cual de los vértices restantes asignar k como su imagen bajo σ, estos factores

Algoritmo 4: LexBFS(G,τ).

Input: Una gráfica G y un ordenamiento τ de V_G
Output: Un ordenamiento σ de V_G

```
1  list L;
2  integer i ← 0;
3  addR(L, V_G);
4  while Exista un elemento no vacío en L do
5  |  Tomar P, el primer elemento no vacío de L;
6  |  Tomar x, el elemento τ-mínimo de P;
7  |  Sustituir a P por P \ {x} en L;
8  |  σ(x) ← i;
9  |  i ← i + 1;
10 |  for Q en L, tal que P ≤_L Q do
11 |  |  Q' ← N(x) ∩ Q;
12 |  |  if Q' no es vacío y no es Q then
13 |  |  |  Añadir a Q' justo a la izquierda de Q;
14 |  |  |  Sustituir a Q por Q \ Q' en L;
15 |  |  end
16 |  end
17 end
18 return σ
```

son, en orden de prioridad: los vecinos en el conjunto $\{\sigma^{-1}(0), \ldots, \sigma^{-1}(k-1)\}$, o sea los vértices ya explorados, y además ser el τ-mínimo de entre todos los candidatos. Es decir, si $\sigma(x) = k$, entonces para cualquier $y \neq x$ no explorado hasta el punto donde se asigna $\sigma(x) = k$, se debe cumplir que el primer vértice en $\{\sigma^{-1}(0), \ldots, \sigma^{-1}(k-1)\}$ que distingue a x y y, debe ser vecino de x; o en caso de que no haya vértices en $\{\sigma^{-1}(0), \ldots, \sigma^{-1}(k-1)\}$ que distingan a x y y, se sigue $\tau(x) < \tau(y)$. De lo anterior se puede ver la relación que tiene este algoritmo con BFS, esta relación es que en el momento en el que se explora un vértice x, tenemos la certeza de que se van a explorar los vértices en $N(x) \cap R$ antes que los vértices en $R \setminus N(x)$, donde R es el conjunto de los vértices no explorados hasta ese momento. Respecto al comportamiento del ordenamiento σ que devuelve el algoritmo tenemos varios resultados. Como vamos a utilizar mucho el término, vamos a definir explícitamente a qué nos referimos con refinar, aunque es bastante intuitivo. Dados S un conjunto y L una lista doblemente ligada cuyas celdas almacenan conjuntos, el **refinamiento** de L respecto a S es la



L:	$\{v_1, v_2, v_3, v_4, v_5\}$			$\sigma = ()$
L:	$\{v_1, v_5\}$	$\{v_2, v_4\}$		$\sigma = (v_3)$
L:	$\{v_1\}$	$\{v_2\}$	$\{v_4\}$	$\sigma = (v_3, v_5)$
L:	$\varnothing$	$\{v_2\}$	$\{v_4\}$	$\sigma = (v_3, v_5, v_1)$
L:	$\varnothing$	$\varnothing$	$\{v_4\}$	$\sigma = (v_3, v_5, v_1, v_2)$
L:	$\varnothing$	$\varnothing$	$\varnothing$	$\sigma = (v_3, v_5, v_1, v_2, v_4)$

Figura 4.1: Ejecución LexBFS$(H, (v_3, v_2, v_4, v_5, v_1))$.

lista obtenida de L al reemplazar cada celda c de L, que satisfaga $\varnothing \subsetneq c \cap S \subsetneq S$, por las celdas consecutivas $c \cap S$ y $c \setminus S$, justo en ese orden. O sea que en el Algoritmo 4, en cada iteración del `while`, se refina a L respecto $N(x)$.

Para dar un par de definiciones, tomemos una gráfica G. Dado un ordenamiento σ de V_G, diremos que σ es un **ordenamiento LexBFS** de V_G, solo cuando exista un ordenamiento de V_G (posiblemente σ mismo) tal que σ es el ordenamiento devuelto por LexBFS(G,τ). Decimos que un ordenamiento σ de V_G satisface la **condición de los cuatro puntos** , abreviada como **FPC**, si y solo si, para cualesquiera $x, y, z \in V_G$ tales que $x <_\sigma y <_\sigma z$, $xy \notin E_G$ y $xz \in E_G$, se tiene la existencia de $v \in V_G$ que satisface $v <_\sigma x$, $vy \in E_G$ y $vz \notin E_G$. Cuando hagamos referencia al momento de exploración de un vértice x en la ejecución del Algoritmo 4, estaremos haciendo referencia al instante en el que inicia la iteración del `while` en donde se asigna $\sigma(x)$. En relación a lo anterior, decimos que x es el **pivote** de una iteración del `while`, cuando x sea el vértice al cual se asigna su valor de σ en dicha iteración, y en consecuencia el vértice con la vecindad respecto a la cual se refinan las celdas de L.

Teorema 4.1.1. *Para cualquier gráfica G y cualquier ordenamiento σ de V_G se cumple que, σ es un ordenamiento LexBFS si y solo si σ satisface FPC.*

Demostración. Utilizaremos la notación L_w, donde L es la variable de lista que se utiliza en el Algoritmo 4 y w es cualquier vértice que esté en algún elemento de L. Lo que significa L_w es justamente el elemento de L en donde está w. El considerar estos

conjuntos podría acarrear ciertos problemas, pues L y las celdas de L van cambiando a lo largo de la ejecución, sin embargo, utilizaremos el contexto para solventar esta dificultad.

Comencemos suponiendo que σ es un ordenamiento LexBFS, o sea que existe un ordenamiento τ de V_G tal que σ es devuelta por LexBFS(G, τ). Sean $x, y, z \in V_G$ tales que $x <_\sigma y <_\sigma z$, $xy \notin E_G$ y $xz \in E_G$. Cuando se asigna $\sigma(x)$, los vértices y y z deben estar en elementos distintos de L, pues de lo contrario, al refinar las celdas de L con $N(x)$, se reemplazaría L_y por $L_y \setminus N(x)$ y se insertaría a $N(x) \cap L_y$ justo a la izquierda, pero esto implicaría que al finalizar la ejecución del algoritmo se debería tener $\sigma(z) < \sigma(y)$, lo cual es falso. Por lo tanto, en dicho momento en el que se explora a x, se debe tener $L_y \neq L_z$. Lo anterior implica que en algún momento previo a la exploración de x, los vértices y y z dejaron de estar en la misma parte; supongamos que tal momento fue en la exploración del vértice v. Al comienzo de la iteración del `while` donde se explora a v, se tenía $L_y = L_z$, pero al refinarse esa parte se separaron y y z. No pudo haber ocurrido que $y \in L_y \setminus N(v)$ y $z \in L_y \cap N(v) = L_z \cap N(x)$ porque esto implicaría de nuevo que $\sigma(z) < \sigma(y)$. Así, debió ocurrir que $z \in L_y \setminus N(v)$ y $y \in L_y \cap N(v)$, y por ende $vy \in E_G$ y $vz \notin E_G$. Por lo tanto, v satisface lo que queremos, puesto que al explorarse v antes que x, se sigue que $v <_\sigma x$.

Ahora supongamos que σ satisface FPC. Denotemos por n al orden de G y por τ al ordenamiento que devuelve la ejecución de LexBFS(G, σ). Cuando hablemos de la ejecución del algoritmo en la demostración, nos referiremos obviamente a la ejecución LexBFS(G, σ). Vamos a demostrar usando inducción sobre $\{0, \ldots, n-1\}$ que, para todo $k \in \{0, \ldots, n-1\}$, $\sigma^{-1}(k) = \tau^{-1}(k)$. Para $k = 0$ es obvio, porque el vértice, al que el algoritmo asigna el valor 0 como imagen bajo τ, es justamente el σ-mínimo de V_G, o sea $\tau^{-1}(0)$. Tomemos $k \in \{1, \ldots, n-1\}$ y supongamos que para todo $i < k$, $\sigma^{-1}(i) = \tau^{-1}(i)$. Sea $x = \tau^{-1}(k)$; por hipótesis inductiva se debe satisfacer $\sigma(x) \geq k$, pues para $i < k$ se cumple $\sigma^{-1}(i) = \tau^{-1}(i) \neq \tau^{-1}(k) = x$; para derivar una contradicción supondremos que $\sigma(x) > k$. Definamos $y = \sigma^{-1}(k)$ que es un vértice distinto de x bajo nuestro supuesto. En la iteración del `while` en donde asignamos el valor para $\tau(x)$, o sea en la exploración de x, se tomó como pivote a x justo porque estaba en la primera celda no vacía de L y además era el σ-mínimo de los vértices en esta celda. O sea que y no puede estar en la primera celda no vacía de L, es decir L_x, porque se tiene $\sigma(y) = k < \sigma(x)$. Como x y y ya no están en la misma celda en la iteración donde x es pivote, o sea la iteración $(\sigma(x)+1)$-ésima del `while`, tuvo que haber una iteración anterior del `while`, digamos la iteración l-ésima, donde por primera vez x y y quedaron en partes distintas después del refinamiento. Supongamos que v era el pivote en esta iteración l-ésima; al inicio de esta iteración se tenia $L_x = L_y$, pero $|N(v) \cap \{x, y\}| = 1$. Observemos que al

ser v el primer vértice, respecto a τ, en donde x y y dejaron de estar en la misma celda de L, ningún vértice antes de v distingue a x y y, escrito lo anterior de forma más técnica sería que, para todo $0 \leq i < \tau(v)$ se cumple $x \in N(\tau^{-1}(i))$ si y solo si $y \in N(\tau^{-1}(i))$. Recordando nuestra hipótesis inductiva podemos escribir lo anterior como, para todo $0 \leq i < \tau(v)$ se cumple $x \in N(\sigma^{-1}(i))$ si y solo si $y \in N(\sigma^{-1}(i))$. Por lo anterior, y dado que $v <_\sigma y <_\sigma x$, no es posible que $vx \in E_G$ y $vy \notin E_G$ porque σ satisface FPC. Por lo tanto se debe cumplir $vx \notin E_G$ y $vy \in E_G$, haciendo que para la posteridad las celdas de L donde está y queden a la izquierda de las celdas donde está x, o sea implicando que $\tau(y) < \tau(x)$. Esto es una contradicción que viene de suponer que $\sigma(x) > k$, de donde se deduce que $\sigma(x) = k$ y por lo tanto $\sigma^{-1}(k) = \tau^{-1}(k)$. De la implicación anterior podemos deducir un poco más de lo que queríamos, además de concluir que σ es un ordenamiento LexBFS, podemos deducir que cualquier ordenamiento LexBFS pasado como argumento en una ejecución del Algoritmo 4 resulta en el mismo ordenamiento. ■

El resultado anterior nos da una caracterización para un ordenamiento LexBFS que no depende de su construcción mediante el Algoritmo 4, esto nos será útil algunas veces. Teniendo una gráfica G y un ordenamiento σ de V_G, diremos que una tripleta $(x, y, z) \in V_G \times V_G \times V_G$ es un **paraguas** del ordenamiento σ, solo cuando se satisfaga $x <_\sigma y <_\sigma z$, $xy \notin E_G$, $xz \in E_G$ y $yz \notin E_G$.

Teorema 4.1.2. *Sean G una cográfica y σ un ordenamiento LexBFS de V_G, el ordenamiento σ no puede tener paraguas.*

Demostración. Probemos una contrapositiva para el enunciado. Vamos a suponer que σ sí tiene paraguas, además de que σ es un ordenamiento LexBFS de V_G para llegar a que G no puede ser una cográfica. Sea $x \in V_G$ el vértice σ-mínimo con la propiedad de que para algunos $u, w \in V_G$, (x, u, w) es un paraguas de σ. Consideremos además $y, z \in V_G$ tales que (x, y, z) es un paraguas de σ. De la definición de paraguas tenemos que $x <_\sigma y <_\sigma z$, las aristas xy y yz no están en G, pero la arista xz sí está en G. Por satisfacer FPC el ordenamiento σ, garantizamos la existencia de $v \in V_G$ tal que $v <_\sigma x$, $vy \in E_G$ y $vz \notin E_G$. No puede ser que $vx \notin E_G$, pues de ocurrir esto la tripleta (v, x, y) sería un paraguas con $v <_\sigma x$, lo que contradice la elección de x. Por lo tanto, $yv, vx, xz \in E_G$ y $xy, yz, vz \notin E_G$, o sea que $G[\{y, v, x, z\}]$ es isomorfa a P_4 y por lo tanto G no es una cográfica. ■

Sean G una gráfica de orden n y σ un ordenamiento de V_G. Para cualesquiera $i \in \{0, \ldots, n\}$ y $x \in V_G$, definimos el conjunto $N_i^\sigma(x)$ como

$$N_i^\sigma(x) = \{v \in N(x) \colon \sigma(v) < i\}.$$

La mayoría de las veces, omitiremos el superíndice σ en la definición anterior cuando sepamos de antemano a que ordenamiento hacemos referencia. Claramente, de la definición se sigue que $N_0(x) = \varnothing$ y $N_n(x) = N(x)$. Decimos que un subconjunto de vértices S es una **rebanada** de σ, si y solo si, S es un conjunto no vacío de vértices consecutivos respecto a σ, digamos $S = \{v \in V_G \colon i \leq \sigma(v) \leq j\}$ para algunos $0 \leq i \leq j \leq n-1$, de tal forma que también satisface las siguientes condiciones: Para todo $x \in S$ se cumple $N_i(x) = N_i(\sigma^{-1}(i))$ y para todo $y \in \sigma^{-1}[\{j+1, \ldots, n-1\}]$ se tiene $N_i(y) \neq N_i(\sigma^{-1}(i))$. Resulta que todo ordenamiento LexBFS siempre tiene rebanadas, más aún, sabemos exactamente cuales son.

Teorema 4.1.3. *Sean G una gráfica y σ un ordenamiento LexBFS de G. Las rebanadas de σ son justamente las primeras celdas no vacías de L respecto a la ejecución del Algoritmo 4, una por cada iteración del* `while`*. Es decir, las rebanadas son $L_{\sigma^{-1}(i)}$, con $0 \leq i \leq n-1$, donde la celda $L_{\sigma^{-1}(i)}$ se toma al inicio de la iteración $(i+1)$-ésima del* `while`*. Todas las referencias dentro de lo que ocurre en LexBFS serán tomadas sobre cualquier ejecución LexBFS(G, τ) que dé como resultado el ordenamiento σ.*

Demostración. Primero veamos que los conjuntos que proponemos como rebanadas, sí son rebanadas. Fijemos $i \in \{0, \ldots, n-1\}$ y llamemos S a la primera celda no vacía de L al comienzo de la $(i+1)$-ésima iteración del `while`, o sea, a $L_{\sigma^{-1}(i)}$. Primero observemos que por la definición del Algoritmo 4, S es en efecto un conjunto de vértices consecutivos respecto a σ, pues el algoritmo siempre explora primero a todos los vértices de la primera celda no vacía antes de explorar vértices de otras celdas, de hecho tenemos que $S = \{v \in V_G \colon i \leq \sigma(v) \leq j\}$ con $j = \max \sigma[S]$. Tomemos cualquier $x \in S$. Si $x = \sigma^{-1}(i)$, evidentemente se satisface lo que queremos. Si $x \neq \sigma^{-1}(i)$, no puede existir $u <_\sigma \sigma^{-1}(i)$ tal que $|N(u) \cap \{\sigma^{-1}(i), x\}| = 1$, pues, esto implicaría que en la iteración $(\sigma(u)+1)$-ésima del `while`, después de refinar las celdas de L con $N(u)$, los vértices x y $\sigma^{-1}(i)$ quedaron en celdas distintas de L. Lo anterior es absurdo, porque al tenerse $\sigma(u) + 1 < i + 1$, se debería tener, al inicio de la iteración $(i+1)$-ésima, a x y $\sigma^{-1}(i)$ en celdas diferentes de L. Por lo tanto, para todo $u <_\sigma \sigma^{-1}(i)$ se cumple $x \in N(u)$ si y solo si $\sigma^{-1}(i) \in N(u)$, es decir, $N_i(\sigma^{-1}(i)) = N_i(x)$. Ahora, si z es un vértice σ-mayor al σ-máximo de S, se sigue que z no está en S. Tuvo que haber una iteración anterior del `while`, digamos la iteración l-ésima con $l < i + 1$, donde z y $\sigma^{-1}(i)$ dejaron de estar en la misma celda de L. O sea que, si u era el pivote de esta l-ésima iteración, se cumple que $|N(u) \cap \{z, \sigma^{-1}(i)\}| = 1$ y por lo tanto $N_i(z) \neq N_i(\sigma^{-1}(i))$. Con esto concluimos que los conjuntos que propusimos sí son rebandas de σ, solo queda por ver que son las únicas.

Sea S una rebanada arbitraria de σ, digamos $S = \{v \in V_G \colon i \leq \sigma(v) \leq j\}$ para algunos $0 \leq i \leq j \leq n-1$. Sea $x = \sigma^{-1}(i)$ y consideremos la celda L_x al inicio de la iteración $(i+1)$-ésima del `while`, o sea la iteración en donde x es tomado como pivote. Consideremos cualquier $z \in L_x$. Por el desarrollo del párrafo anterior, podemos garantizar que $N_i(z) = N_i(x)$, entonces, por ser S rebanada se cumple que $\sigma(z) \leq j$. Notemos que $\sigma(z) \geq i$, pues de lo contrario z ya hubiera sido explorado y por ende no podría estar en L_x en ese momento. Así, concluimos por un lado que $z \in S$, y al ser z arbitrario, también que $L_x \subseteq S$. Por otro lado, dado $z \in S$, por definición se satisface $N_i(z) = N_i(x)$. Así que z debe estar en L_x, pues de lo contrario existiría $u <_\sigma x$ tal que $|N(u) \cap \{x, z\}| = 1$, o sea $N_i(x) \neq N_i(z)$, lo cual es absurdo. Concluimos entonces que $L_x = L_{\sigma^{-1}(i)} = S$. ■

Teorema 4.1.4. *Sean, G una gráfica, σ un ordenamiento LexBFS de V_G, y $S = \{v \in V_G \colon i \leq \sigma(v) \leq j\}$ una rebanada de σ, con $0 \leq i \leq j \leq n-1$. El ordenamiento $\sigma - i$ es un ordenamiento LexBFS de los vértices de la gráfica $G[S]$.*

Demostración. En virtud el Teorema 4.1.1, basta con probar que $\sigma - i$ satisface FPC. Sean $x, y, z \in V_{G[S]} = S$ tales que $x <_{\sigma - i} y <_{\sigma - i} z$, $xy \notin E_G$ y $xz \in E_G$. Claramente, para vértices en S, el ordenamiento $\sigma - i$ preserva el orden de σ, así que $x <_\sigma y <_\sigma z$. Entonces, podemos usar que σ satisface FPC para garantizar la existencia de $v \in V_G$ tal que $v <_\sigma x$, $vy \in E_G$ y $vz \notin E_G$. Para mostrar que se satisface FPC para y, z y x, solo hace falta mostrar que v está en S, pues v ya satisface lo que queremos. Como $N_i(y) = N_i(\sigma^{-1}(i)) = N_i(z)$, forzosamente ocurre que $\sigma(v) \geq i$, además $\sigma(v) < \sigma(x) \leq j$, de donde concluimos que $v \in S$. ■

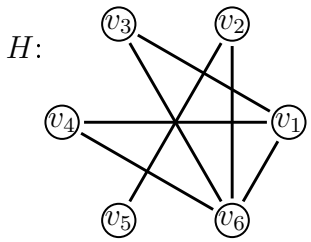
Vamos a introducir más notación, para esto consideraremos G una gráfica, σ un ordenamiento LexBFS de V_G y x un vértice de G. Las rebanadas tomadas a continuación, serán entendidas como rebanadas de σ en cualquier ejecución LexBFS(G, τ) que dé como resultado el ordenamiento σ. Por el Teorema 4.1.3, todas las rebanadas están dadas por las celdas $L_{\sigma^{-1}(i)}$, estas celdas tomadas al inicio de la iteración $(i+1)$-ésima del `while` en el Algoritmo 4. Definimos la **rebanada correspondiente** a x, o simplemente la rebanada de x, como la única rebanada que comienza con x, la cual denotamos por $S_\sigma(x)$. Es decir, la rebanada $S_\sigma(x)$ es simplemente la celda L_x al inicio de la iteración $(\sigma(x)+1)$-ésima del Algoritmo 4. Cuando por el contexto sea claro de cuál ordenamiento hablamos, omitiremos el subíndice σ en la notación $S_\sigma(x)$. Diremos que un conjunto S_1 es una subrebanada de una rebanada S_2, cuando S_1 también sea rebanada y $S_1 \subseteq S_2$. Por ejemplo, si $N(x) \cap S(x)$ no es vacío, entonces $N(x) \cap S(x)$ es una subrebanada de $S(x)$, pues de hecho, siguiendo el flujo del algoritmo podemos ver que $N(x) \cap S(x) = S(\sigma^{-1}(\sigma(x)+1))$. Este conjunto $S(x) \cap N(x)$

tiene importancia, sea vacío o no, así que tiene una notación particular la cual es $S^A_\sigma(x) = S_\sigma(x) \cap N(x)$. Definamos al conjunto $S^N_\sigma(x) = (S_\sigma(x) \setminus N(x)) \setminus \{x\}$. Cuando por el contexto sea claro de cual ordenamiento hablamos, omitiremos el subíndice σ en las notaciones $S^A_\sigma(x)$ y $S^N_\sigma(x)$. De las definiciones tenemos directamente que $S(x) = \{x\} \cup S^A(x) \cup S^N(x)$. Si $S^N(x)$ no es vacío, todos los vértices de $S^A(x)$ (si tiene vértices) son explorados antes de explorar los vértices de $S^N(x)$. Así que cada vez que se realiza la iteración correspondiente a un vértice $z \in S^A(x)$ como pivote, se refinan consecutivamente las celdas, que pueden ser ya refinamientos, de $S^N(x)$. O sea que, al terminar la última iteración del `while` que tiene como pivote un vértice en $S^A(x)$, tenemos para algún k que las primeras k celdas no vacías sucesivas de L, digamos $S_1, \ldots, S_k$, forman una partición para $S^N(x)$. Estas celdas también reciben un nombre y una notación especial, se llaman **x-celdas** y se denota a la i-ésima de estas celdas por $S^\sigma_i(x)$, omitiendo el superíndice cuando no haya duda del ordenamiento. También le damos notación particular al vértice σ-mínimo de $S^\sigma_i(x)$, lo denotamos por x^σ_i, omitiendo el superíndice cuando no haya duda del ordenamiento. De la definición de las x-celdas, se deduce de inmediato que en caso de que $S^N(x) \neq \varnothing$, $S_1(x)$ es una subrebanada de $S(x)$ porque $S_1(x)$ es justamente la primera celda no vacía de L al terminar de explorar a todos los vértices en $S^A(x)$. Los siguientes resultados nos proporcionan características más específicas de las x-celdas.

Para ejemplificar los conceptos anteriores, exploramos en la Figura 4.2 una ejecución LexBFS, utilizando como entrada la gráfica H ahí definida y considerando el ordenamiento $\tau = (v_3, v_5, v_1, v_6, v_2, v_4)$. Como ya observamos, el elemento de la lista que tiene a un vértice x al momento de tomarlo como pivote es la rebanada de x, esta rebanada la delimitamos por $\{\ldots\}$ en el ejemplo, como todos los demás elementos de la lista, pero el vértice x al que corresponde la rebanada estará subrayado. Por otro lado, dentro de la rebanada de un vértice x, delimitamos al conjunto $S^A(x)$ con $(\ldots)$ y al conjunto $S^N(x)$ con $[\ldots]$. Para visualizar el concepto de x-celdas vamos a tener que centrarnos en un solo vértice, que, por ser el único vértice que las tiene, elegimos a v_3, así que las v_3-celdas están delimitadas por $\langle\ldots\rangle$; como se sigue por definición, las v_3-celdas aparecen después de procesar a los vértices en $S^A(v_3)$.

De nuevo vamos a hacer una observación de suma importancia, pero esta será la última vez en la que hacemos hincapié en ella. Todos los conceptos anteriores dependen de una ejecución concreta LexBFS(G, τ) que devuelva al ordenamiento σ. Sin embargo, todos los resultados que enunciaremos son válidos sin importar quién sea el ordenamiento τ, así que omitiremos mencionarlo la mayoría de las veces, exceptuando algunos casos explícitos donde τ es un ordenamiento específico.

Proposición 4.1.5. *Tomemos G una gráfica, x un vértice de G, σ un ordenamiento LexBFS de V_G y $k = \max\{\sigma(v) \colon v \in \{x\} \cup S^A(x)\}$. Para cualesquiera $j > 0$ y*



L:	$\{\underline{v_3}, (v_1, v_6), [v_2, v_4, v_5]\}$				$\sigma = ()$
L:	$\{\underline{v_1}, (v_6)\}$	$\{v_2, v_4, v_5\}$			$\sigma = (v_3)$
L:	$\{\underline{v_6}\}$	$\{v_4\}$	$\{v_2, v_5\}$		$\sigma = (v_3, v_1)$
L:	$\varnothing$	$\langle\{\underline{v_4}\}\rangle$	$\langle\{v_2\}\rangle$	$\langle\{v_5\}\rangle$	$\sigma = (v_3, v_1, v_6)$
L:	$\varnothing$	$\varnothing$	$\{\underline{v_2}\}$	$\{v_5\}$	$\sigma = (v_3, v_1, v_6, v_4)$
L:	$\varnothing$	$\varnothing$	$\varnothing$	$\{\underline{v_5}\}$	$\sigma = (v_3, v_1, v_6, v_4, v_2)$
L:	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\sigma = (v_3, v_1, v_6, v_4, v_2, v_5)$

Figura 4.2: Ejecución LexBFS$(H, (v_3, v_5, v_1, v_6, v_2, v_4))$.

$u, w \in S_j(x)$, se satisface $N_{k+1}(u) = N_{k+1}(w)$. O sea que, las vecindades de u y w coinciden en el conjunto de vértices σ-menores a los vértices de $S^N(x)$.

Demostración. Hay que probar que, para todo $i \in \{0, \ldots, k\}$, se cumple que $\sigma^{-1}(i) \in N(u)$ si y solo si $\sigma^{-1}(i) \in N(w)$, para esto, tomemos $i \leq k$ arbitrario. Como $N_{\sigma(x)}(u) = N_{\sigma(x)}(w)$ por estar u y w en la rebanada $S(x)$, solo hay que considerar el caso en el que $\sigma(x) \leq i \leq k$. Es claro que $S^A(x) = \{v \in V_G \colon \sigma(x) < \sigma(v) \leq k\}$, de esta manera, si definimos $z = \sigma^{-1}(i)$, tendremos que $z \in \{x\} \cup S^A(x)$. Para el caso $z = x$, se tiene que $xu, xw \notin E_G$, pues $u, w \in S^N(x)$. Para el caso $z \in S^A(x)$, tenemos que $zu \in E_G$ si y solo si $zw \in E_G$ porque, de tenerse $|N(z) \cap \{u, w\}| = 1$, se seguiría, por definición de las x-celdas, que u y w no pueden estar ambos en $S_j(x)$. Por lo tanto $N_{k+1}(u) = N_{k+1}(w)$. ■

Plasmamos en el siguiente corolario otra forma equivalente de escribir la Proposición 4.1.5.

Corolario 4.1.6. *Tomemos G una gráfica, x un vértice de G y σ un ordenamiento LexBFS de V_G. Para cualquier índice j tal que $S_j(x)$ exista y cualesquiera $u, w \in S_j(x)$, se satisface $N(u) \cap (\sigma^{-1}[\{0, \ldots, \sigma(x)\}] \cup S^A(x)) = N(w) \cap (\sigma^{-1}[\{0, \ldots, \sigma(x)\}] \cup S^A(x))$, en particular $N(u) \cap S^A(x) = N(w) \cap S^A(x)$.*

Teorema 4.1.7. *Sea G una gráfica con σ un ordenamiento LexBFS de V_G. Para cualquier $x \in V_G$ y cualesquiera índices $j < k$ tales que existan las x-celdas $S_j(x)$ y $S_k(x)$, se satisface lo siguiente. Si hay un vértice $z \in S_k(x) \setminus \{x_k\}$ que tiene un vecino en $S_j(x)$, digamos $u \in S_j(x)$, deben existir $1 \leq i \leq j$ y $w \in S_i(x)$ tales que $wx_k \in E_G$ y $wz \notin E_G$, más aún, $w \leq_\sigma u$.*

Demostración. Si x_k y u son vecinos, el mismo u satisface lo que queremos. Si x_k no es vecino de u, tenemos $u <_\sigma x_k <_\sigma z$, $ux_k \notin E_G$ y $uz \in E_G$, así que por FPC debe existir $w <_\sigma u$ tal que $wx_k \in E_G$ y $wz \notin E_G$. Como, por el Corolario 4.1.6, las vecindades de x_k y z coinciden en vértices del conjunto $\sigma^{-1}[\{0, \ldots, \sigma(x)\}] \cup S^A(x)$, se tiene por fuerza que $w \in S^N(x)$, o sea que para algún índice i, se tiene $w \in S_i(x)$. Para todo índice $l > j$ y para todo $y \in S_l(x)$, se cumple que $u <_\sigma y$, de donde se sigue $i \leq j$. ■

Proposición 4.1.8. *Sean G una gráfica con σ un ordenamiento LexBFS de V_G, y x un vértice de G. Supongamos que j es tal que la x-celda $S_j(x)$ existe y $N(S_j(x)) \cap (V_G \setminus N(x))$ no es vacío. Si z es el vértice σ-mínimo de $V_G \setminus N(x)$ que es adyacente a algún vértice de $S_j(x)$, y además se cumple $z <_\sigma x_j$, entonces $z \in N(x_j)$.*

Demostración. Si z está en el conjunto $\sigma^{-1}[\{0, \ldots, \sigma(x)\}] \cup S^A(x)$, por el Corolario 4.1.6, tenemos que $z \in N(x_j)$. El otro caso es que z esté en $S_k(x)$ para algún $k < j$, en este caso aplicamos el Teorema 4.1.7 para garantizar la existencia de $i \leq k$ y $w \in S_i(x)$ que satisfagan $wx_j \in E_G$ y $w \leq_\sigma z$. Por la propiedad de mínimo que tiene z, se debe tener $z = w$. Por lo tanto $z \in N(x_j)$. ■

Sean G una gráfica de orden n y σ un ordenamiento LexBFS de V_G. Consideremos a S, un subconjunto de vértices consecutivos respecto a σ, o sea que S tiene la forma $\{v \in V_G \colon i \leq \sigma(v) \leq j\}$, para algunos $0 \leq i \leq j \leq n-1$. Definimos la **vecindad izquierda** de S respecto a σ como el conjunto

$$N^\sigma_<(S) = \{v \in \bigcap_{z \in S} N(z) \colon v <_\sigma x\}, \text{ donde } x \text{ es el } \sigma\text{-mínimo de } S.$$

Desde luego que omitiremos tanto el superíndice en la notación, como la referencia al ordenamiento en el nombre, si es que la situación lo evidencia por el contexto. Además de las propiedades que ya presentamos de las x-celdas, podemos garantizar todavía más cuando se trata de ordenamientos LexBFS sobre cográficas. Cuando hablamos de las vecindades izquierdas en un ordenamiento LexBFS$^-$, utilizamos la notación $\overline{N}_<(S)$ para hacer explícito que estamos en el contexto de LexBFS$^-$.

Teorema 4.1.9. *Sean G una cográfica y σ un ordenamiento LexBFS de V_G. Supongamos que para $x \in V_G$ tenemos la existencia de dos x-celdas, digamos $S_i(x)$ y $S_j(x)$, con $i < j$. Resulta que no hay adyacencias entre vértices de $S_i(x)$ y de $S_j(x)$, además se satisface $N_<(S_j(x)) \subsetneq N_<(S_i(x))$.*

Demostración. Para ver que no hay aristas entre vértices de $S_i(x)$ y vértices de $S_j(x)$, vamos a proceder por contrapositiva; supondremos que no es cierto y llegaremos a que G no es una cográfica. Sean $u \in S_i(x)$ y $w \in S_j(x)$ tales que $uw \in E_G$. Por la definición de las x-celdas, debe haber un vértice en $S^A(x)$ que distingue a u y w. Tomemos a z como el vértice σ-mínimo de $S^A(x)$ tal que $|N(z) \cap \{u, w\}| = 1$. Por la propiedad de mínimo que tiene z aunado a que $u <_\sigma w$, se debe cumplir de hecho que $zu \in E_G$ y $zw \notin E_G$. Como $u, w \in S^N(x)$, $z \in S^A(x)$, $zu, uw \in E_G$ y $zw \notin E_G$, se sigue que $G[\{x, z, u, w\}]$ es isomorfa a P_4 y por lo tanto G no es una cográfica.

Veamos ahora la contención de las vecindades izquierdas que queremos, para esto, tomemos $z \in N_<(S_j(x))$ arbitrario. Por lo que acabamos de demostrar en el párrafo anterior, z no puede estar en $S_i(x)$ para alguna $0 \leq i < j$. Sabemos que $z <_\sigma x_j$, lo que implica que $z \in S^A(x)$ o $z \leq_\sigma x$. Si $z \leq_\sigma x$, por ser $S(x)$ rebanada y tenerse $zx_j \in E_G$, se sigue que $S(x) \subseteq N(z)$, en particular $S_i(x) \subseteq N(z)$ y por lo tanto $z \in N_<(S_i(x))$. Abordemos ahora el caso en el que $z \in S^A(x)$. Dado cualquier $u \in S_i(x)$, se tiene $z <_\sigma u <_\sigma x_j$, $zx_j \in E_G$ y $ux_j \notin E_G$, pero como por el Teorema 4.1.2 σ no tiene paraguas, se sigue que $zu \in E_G$. Al ser u arbitrario y tenerse $z \in S^A(x)$, concluimos que $z \in N_<(S_i(x))$. Para ver que la contención es propia, observemos que, tomando $z \in S^A(x)$ tal que $zx_i \in E_G$ y $zx_j \notin E_G$, se deduce por el Corolario 4.1.6 que $S_i(x) \subseteq N(z)$ y $S_j(x) \cap N(z) = \varnothing$, o sea $z \in N_<(S_i(x))$ y $z \notin N_<(S_j(x))$. ■

Del Teorema 4.1.9 podemos extraer el siguiente corolario muy importante.

Corolario 4.1.10. *Sean G una cográfica y σ un ordenamiento LexBFS de V_G. Si hay x-celdas para $x \in V_G$, todas las x-celdas $S_1(x), \ldots, S_k(x)$ son rebandas de σ.*

Dediquemos solo un par de líneas a la validez del Corolario 4.1.10. Supongamos que tenemos las x-celdas $S_1(x), \ldots, S_k(x)$ respecto a un ordenamiento LexBFS σ de una cográfica G. Ya observamos al definir las x-celdas que $S_1(x)$ siempre es una rebanada. Ahora, si $1 < j \leq k$, al refinar utilizando como pivote a los vértices de $S_i(x)$ para $i < j$, nunca refinamos a $S_j(x)$ porque del Teorema 4.1.9 tenemos que no hay aristas entre vértices de $S_i(x)$ y vértices de $S_j(x)$. Al terminar todas las iteraciones donde un elemento de $\bigcup_{0 \leq i < j} S_i(x)$ era el pivote, la primera celda no vacía de L en el Algoritmo 4 será justamente $S_j(x)$.

Algoritmo 5: LexBFS$^-$(G,τ).

Input: Una gráfica G y un ordenamiento τ de V_G
Output: Un ordenamiento σ^- de V_G

```
1  list L;
2  integer i ← 0;
3  addR(L, V_G);
4  while Exista un elemento no vacío en L do
5  |  Tomar P, el primer elemento no vacío de L;
6  |  Tomar x, el elemento τ-mínimo de P;
7  |  Sustituir a P por P \ {x} en L;
8  |  σ⁻(x) ← i;
9  |  i ← i + 1;
10 |  for Q en L, tal que P ≤_L Q do
11 |  |  Q' ← N(x) ∩ Q;
12 |  |  if Q' no es vacío y no es Q then
13 |  |  |  Añadir a Q' justo a la derecha de Q;
14 |  |  |  Sustituir a Q por Q \ Q' en L;
15 |  |  end
16 |  end
17 end
18 return σ⁻
```

Introduzcamos otra versión de LexBFS, a la que llamaremos **LexBFS$^-$**.

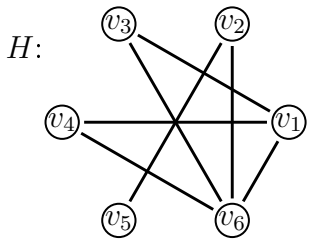
Observemos que LexBFS$^-$, es decir el Algoritmo 5, tiene exactamente la misma estructura que LexBFS, o sea, que el Algoritmo 4; los algoritmos solo tienen dos diferencias. La primera de estas diferencias es que se cambia la variable σ por σ^- para hacer patente que se trata de un ordenamiento devuelto por LexBFS$^-$. La segunda diferencia es la palabra subrayada, en LexBFS se inserta a Q', celda que tiene valor $N(x) \cap Q$, a la izquierda de Q, mientras que en LexBFS$^-$ se inserta Q' a la derecha de Q. El insertar a Q' a la derecha de Q en lugar de a la izquierda de Q puede parecer un cambio trivial, sin embargo, al observar que $N_G(x) \cap A = A \setminus N_{\overline{G}}(x)$ y que $A \setminus N_G(x) = N_{\overline{G}}(x) \cap A$, podemos ver que el cambio es bastante fuerte. Resulta que este cambio nos permite calcular el ordenamiento que devuelve LexBFS$(\overline{G}, \tau)$ simplemente calculando el ordenamiento que devuelve LexBFS$^-(G, \tau)$, esto sin necesidad de tener la información de $\overline{G}$ codificada. Lo anterior es claro de lo ya explicado, quizás solo notando también que, el insertar una celda A exactamente

a la izquierda de otra celda B equivale a insertar la celda B exactamente a la derecha de A.

También tenemos notación de todos los conceptos que definimos para LexBFS, solo que relativos a LexBFS$^-$. Dada una gráfica G y un ordenamiento σ^- de V_G, decimos que σ^- es un ordenamiento LexBFS$^-$ si y solo si, existe τ un ordenamiento de V_G tal que σ^- es el ordenamiento arrojado por la ejecución LexBFS$^-(G,\tau)$. Consideremos una gráfica G, σ^- un ordenamiento LexBFS$^-$ de G y $x \in V_G$. Aunque LexBFS$^-$ es un algoritmo distinto a LexBFS, podemos extender de una manera natural los conceptos de rebanada, de x-celdas, etcétera. En lugar de escribir de nuevo las definiciones y ver, por ejemplo, quienes son todas las rebanadas de un ordenamiento LexBFS$^-$, vamos a utilizar el hecho de que una ejecución LexBFS$^-$ equivale a una LexBFS sobre el complemento de la gráfica. De modo que, si τ es tal que σ^- es el ordenamiento devuelto por LexBFS$^-(G,\tau)$, sabemos que σ^- coincide con el ordenamiento arrojado por el barrido LexBFS$(\overline{G},\tau)$. Denotamos por $\overline{S}(x)$ a la rebanada de σ^- que comienza en x, pero respecto a la ejecución LexBFS$(\overline{G},\tau)$, no a la ejecución LexBFS(G,τ) como sería el caso de la rebanada normal $S(x)$. Es decir, $\overline{S}(x)$ es otra forma de escribir $S_{\sigma^-}(x)$ pero respecto al barrido LexBFS sobre $\overline{G}$. Denotamos por $\overline{S^A}(x)$ al conjunto dado por $\overline{S}(x) \setminus (N_G(x) \cup \{x\})$ y escribimos $\overline{S^N}(x)$ para referirnos al conjunto $\overline{S}(x) \cap N_G(x)$. Como podemos notar fácilmente, $\overline{S^A}(x)$ coincide con $S^A_{\sigma^-}(x)$ pero en la ejecución LexBFS$(\overline{G},\tau)$, mientras que $\overline{S^N}(x)$ coincide con $S^N_{\sigma^-}(x)$ pero respecto a la ejecución LexBFS$(\overline{G},\tau)$. Para i tal que existe la x-celda $S_i^{\sigma^-}(x)$, pero considerada en la ejecución LexBFS$(\overline{G},\tau)$, definimos la i-ésima x-celda respecto a LexBFS$^-$ justamente como $\overline{S}_i(x) = S_i^{\sigma^-}(x)$. O sea que las x-celdas en la ejecución LexBFS$^-(G,\tau)$ también están ordenadas como $[\overline{S}_1(x), \ldots, \overline{S}_k(x)]$ y forman una partición para $\overline{S^N}(x)$, en caso de que este último conjunto no sea vacío. Tenemos resultados análogos a los de LexBFS pero aplicados a los conceptos de LexBFS$^-$, sin embargo, solo los mencionaremos cuando sea necesario. Anticipamos que para desarrollar el algoritmo de reconocimiento, vamos a utilizar a LexBFS y LexBFS$^-$ combinados, pues, calcular un ordenamiento σ que sea LexBFS y luego alimentar a LexBFS$^-$ con σ como ordenamiento base nos proporciona de propiedades interesantes.

Como en la Figura 4.2, vamos a explorar en la Figura 4.3 una ejecución para ejemplificar los conceptos que acabamos de definir, pero para LexBFS$^-$. También vamos a utilizar como entrada la misma gráfica H y para seguir el espíritu de este trabajo, vamos a utilizar el ordenamiento $(v_3, v_1, v_6, v_4, v_2, v_5)$ que obtuvimos en el ejemplo de la Figura 4.2. Como en el otro ejemplo, para los vértices x, las rebanadas $\overline{S}(x)$ están delimitadas por $\{\ldots\}$, además, el vértice x al que corresponde la rebanada estará subrayado. Por otro lado, dentro de la rebanada de un vértice x, delimitamos

al conjunto $\overline{S^A}(x)$ con $(\dots)$ y al conjunto $\overline{S^N}(x)$ con $[\dots]$. También esta vez, vamos a elegir a v_3 para marcar las celdas $\overline{S_j}(v_3)$ con $\langle\dots\rangle$; como se sigue por definición, las v_3-celdas aparecen después de procesar a los vértices en $\overline{S^A}(v_3)$.



L:	$\{\underline{v_3}, (v_2, v_4, v_5), [v_1, v_6]\}$			$\sigma^- = ()$
L:	$\{\underline{v_4}, (v_2, v_5)\}$	$\{v_1, v_6\}$		$\sigma^- = (v_3)$
L:	$\{\underline{v_2}, [v_5]\}$	$\{v_1, v_6\}$		$\sigma^- = (v_3, v_4)$
L:	$\{\underline{v_5}\}$	$\langle\{v_1\}\rangle$	$\langle\{v_6\}\rangle$	$\sigma^- = (v_3, v_4, v_2)$
L:	$\varnothing$	$\{\underline{v_1}\}$	$\{v_6\}$	$\sigma^- = (v_3, v_4, v_2, v_5)$
L:	$\varnothing$	$\varnothing$	$\{\underline{v_6}\}$	$\sigma^- = (v_3, v_4, v_2, v_5, v_1)$
L:	$\varnothing$	$\varnothing$	$\varnothing$	$\sigma^- = (v_3, v_4, v_2, v_5, v_1, v_6)$

Figura 4.3: Ejecución $\text{LexBFS}^-(H, (v_3, v_1, v_6, v_4, v_2, v_5))$.

Teorema 4.1.11. *Sean G una gráfica, M un módulo en G y σ un ordenamiento LexBFS de G. Consideremos σ^- el ordenamiento que se obtiene de la ejecución $\text{LexBFS}^-(G, \sigma)$. Resulta que $x \in V_G$ es el vértice σ-mínimo de M, si y solo si x es el vértice σ^--mínimo de M. Más aún, de satisfacerse lo anterior, $S(x)$ y $\overline{S}(x)$ son las rebanadas mínimas por contención de σ y σ^- respectivamente, con la propiedad de contener a M.*

Demostración. Sean u el vértice σ-mínimo de M y w el vértice σ^--mínimo de M. Como M es un módulo, la vecindad de u y la vecindad de w coinciden fuera de M, en particular, para cualquier $v <_{\sigma^-} w$ se cumple que $v \in N(u)$ si y solo si $v \in N(w)$. Antes de que se tomara a w como pivote en la ejecución de $\text{LexBFS}^-(G, \tau)$, todas las iteraciones del `while` y todos los refinamientos efectuados mantuvieron a u y a w en la misma celda de L. O sea que al momento de tomar a w como pivote en la ejecución $\text{LexBFS}^-(G, \sigma)$, se satisface que $L_w = L_u$ y de hecho $L_w = L_u$ es la primera celda no vacía de L. Pero, w fue elegido como pivote porque w es el σ-mínimo de L_w, de donde se sigue que $w \leq_\sigma u$. Como se tiene $u \leq_\sigma w$ por la elección de u, podemos concluir que $u = w$.

Como ya sabemos que el σ-mínimo de M coincide con el σ^--mínimo de M, llamemos x a dicho mínimo. Veamos que $M \subseteq S(x)$, para esto tomemos $v \in M$ arbitrario. Por ser x el σ-mínimo de M, se tiene $x \leq_\sigma v$. Por otro lado, al ser M módulo y tenerse $x, v \in M$, se cumple $N(x) \setminus M = N(z) \setminus M$, en particular $N_\sigma(x) = N_\sigma(v)$. De lo anterior, se deduce por la definición de rebanada que $v \in S(x)$. La prueba de que $M \subseteq \overline{S}(x)$ es completamente análoga, considerando claro que x es el σ^--mínimo de M y que $\overline{S}(x)$ es la rebanada de x en la ejecución LexBFS$(\overline{G}, \sigma)$.

Ya sabemos que $S(x)$ y $\overline{S}(x)$ son rebanadas de σ y σ^- respectivamente que contienen al módulo M, entonces solo queda por ver que son mínimas por contención con esa propiedad. Supongamos que $S(z)$ es una rebanda de σ que también contiene a M. Sea u cualquier vértice en $S(x)$. Primero, observemos que $x \in M \subseteq S(z)$ implica por un lado que $z \leq_\sigma x$, o sea $z \leq_\sigma u$, por otro lado implica que $N_{\sigma(z)}(z) = N_{\sigma(z)}(x)$ según la definición de rebanada. Por estar u en la rebanada $S(x)$, para cualquier $i \leq \sigma(x)$ se satisface $N_i(x) = N_i(u)$. Entonces, de todo lo anterior deducimos que $N_{\sigma(z)}(z) = N_{\sigma(z)}(x) = N_{\sigma(z)}(u)$, pero lo anterior garantiza justamente la definición para que u pertenezca a la rebanada $S(z)$. Como en el párrafo previo, la prueba de que $\overline{S}(x)$ es mínima por contención con la propiedad de contener a M es análoga. ■

Para finalizar esta sección, vamos a ver cómo se relacionan los barridos LexBFS y LexBFS$^-$ respecto a la presencia de gráficas isomorfas a P_4 en una gráfica.

Teorema 4.1.12. *Sean G una gráfica, σ un ordenamiento LexBFS de G y $S(x)$ la rebanada mínima por contención de σ que contiene los vértices de $P \cong P_4$. Si x no es vértice de ningún P_4 en G, entonces $S^A(x) \cap V_P$ tiene solo a los puntos intermedios de P, o sea, a los vértices que no son extremos de la trayectoria P.*

Demostración. No es posible que $V_P \subseteq S^A(x)$, pues $S^A(x)$ es una subrebanada propia de $S(x)$, así que $S^N(x) \cap V_P \neq \varnothing$. De hecho $|V_P \cap S^N(x)| > 1$, porque, de tenerse $V_P \cap S^N(x) = \{u\}$, podríamos tomar P' el P_3 en $\overline{G}$ que está contenido en $\overline{P}$ y tiene a u como extremo, de esto se sigue la contradicción de que el conjunto $V_{P'} \cup \{x\}$ induce un P_4 en $\overline{G}$ y por ende induce un P_4 en G. De lo anterior deducimos que $|V_P \cap S^N(x)| > 1$. Por otro lado, tampoco es posible que $V_P \subseteq S^N(x)$, para ver que no puede ocurrir esto, supongamos que se cumple la contención anterior y encontremos una contradicción. Dado que hay vértices en $S(x)$ que son adyacentes a ciertos vértices de P pero no a todos, por ejemplo los mismos vértices de P, podemos elegir $v \in S(x)$ el vértice σ-mínimo de $S(x)$ con esta propiedad. Debido a que $V_P \subseteq S^N(x)$, tenemos $x <_\sigma v$. Como v es adyacente a ciertos vértices de P mas no a todos, podemos elegir $y, z \in V_P \subseteq S^N(x)$ tales que $\{v, y, z\}$ inducen un P_3 en G, así, $\{x, v, y, z\}$ induciría un P_4 en G, una contradicción. Por lo tanto, v no

puede estar en $S^A(x)$, o sea que v está en $S^N(x)$. Hagamos una observación; dados $j > i > 0$, $y \in S_i(x)$ y $z \in S_j(x)$, se tiene $yz \notin E_G$. Lo anterior se satisface porque, de tenerse $yz \in E_G$, podríamos tomar $u \in S_A(x)$ tal que $uy \in E_G$ y $uz \notin E_G$, por lo que tendríamos que el conjunto $\{x, u, y, z\}$ induce un P_4 en G lo que deriva otra contradicción. Por lo tanto, no hay aristas entre x-celdas distintas. Para todo $u \in S^A(x)$, se tiene $u <_\sigma v$, por lo que $V_P \subseteq N(u)$ o $V_P \cap N(u) = \varnothing$ por la propiedad de mínimo que tiene v. De lo anterior, se deduce que para algún $j > 0$, se cumple $V_P \subseteq S_j(x)$. Ahora, si $i < j$ y $u \in S_i(x)$, por lo ya observado antes tenemos que $V_P \cap N(u) = \varnothing$. Concluimos así que $v \in S_j(x)$, más aún, concluimos que $S_j(x)$ es la rebanada de x_j, pero esta es una contradicción porque $S_j(x)$ resulta ser una subrebanada propia de $S(x)$ que contiene a V_P. Podemos garantizar ahora que $V_P \nsubseteq S^N(x)$, o sea que $V_P \cap S^A(x) \neq \varnothing$. También se cumple $|V_P \cap S^A(x)| > 1$, porque si ocurriera $V_P \cap S^A(x) = \{u\}$, podríamos tomar P' el P_3 contenido en P que tiene a u como extremo y el conjunto $V_{P'} \cup \{x\}$ induciría un P_4 en G, lo que deriva en una contradicción. Como se satisface $|S^A(x) \cap V_P| = 2 = |S^N(x) \cap V_P|$, solo hay que ver que en $S^A(x) \cap V_P$ no puede haber puntos extremos de P. Demos nombre a P, digamos que $P = (a, b, c, d)$. Comencemos a argumentar a partir de suponer que $a \in S^A(x)$ para llegar a una contradicción. Como $\{x, a, b, c\}$ no puede inducir un P_4, se satisface $b \in S^A(x)$ o $c \in S^A(x)$, sin embargo, $b \in S^A(x)$ implica que $c \in S^A(x)$ o $d \in S^A(x)$ porque $\{x, b, c, d\}$ tampoco induce un P_4, pero esto ya desbordaría la cardinalidad de $S^A \cap V_P$, así que no puede ocurrir. De esta manera, tenemos por fuerza que $a \in S^A(x)$ implica $S^A(x) \cap V_P = \{a, c\}$, de donde se sigue que (a, x, c, d) es un P_4 inducido, una contradicción. La prueba de que d tampoco puede pertenecer a $S^A(x)$ es completamente simétrica, por lo que concluimos que $S^A(x) \cap V_P$ es el conjunto de puntos intermedios de P, o sea $\{b, c\}$. ■

Teorema 4.1.13. *Sean G una gráfica, σ un ordenamiento LexBFS de G y σ^- el ordenamiento devuelto por la ejecución de $LexBFS^-(G, \sigma)$. Consideremos $w \in V_G$ el vértice σ-mínimo que es vértice de algún P_4 inducido en G y P cualquier P_4 inducido en G que tenga a w como vértice. Si w es un extremo de P, entonces, el vértice σ^--mínimo de $\overline{P}$ debe ser un punto intermedio de $\overline{P}$, o sea un extremo de P.*

Demostración. Sea $\overline{S}(x)$ la rebanada mínima por contención que contiene a V_P. Si x no es vértice de ningún P_4 inducido en G, no puede ser tampoco vértice de ningún P_4 inducido en $\overline{G}$. Por el Teorema 4.1.12, $\overline{S^A}(x) \cap V_P$ debe ser el conjunto de puntos medios de $\overline{P}$, o sea que, el σ^--mínimo de V_P, que coincide con el σ^--mínimo de $\overline{S^A}(x) \cap V_P$, debe ser un punto intermedio de $\overline{P}$. Por otro lado, si existe un P_4 inducido que tenga a x como vértice, se debe tener $w \leq_\sigma x$ por la propiedad de

mínimo que tiene w. Como, por definición de la rebanada $\overline{S}(x)$, x es justamente el σ-mínimo de $\overline{S}(x)$, se sigue que $x \leq_\sigma w$ y por lo tanto $x = w$. O sea que, el σ^--mínimo de V_P es de hecho $x = w$, el cual sabemos por hipótesis que es un extremo de P, o sea un punto intermedio de $\overline{P}$. ■

4.2. LexBFS lineal

En la sección anterior estudiamos varias propiedades de los ordenamiento LexBFS, esta clase de ordenamientos serán esenciales para la construcción del algoritmo de reconocimiento. Dado que estos ordenamientos se obtienen con la ejecución del algoritmo LexBFS, no es ninguna sorpresa que vamos a usar el algoritmo LexBFS como subrutina en el algoritmo de reconocimiento. Sin embargo, como el nombre de nuestro trabajo lo índica, el algoritmo que vamos a definir va a ser de tiempo lineal sobre el orden y el tamaño de una gráfica, por lo que, si queremos utilizar a LexBFS como subrutina, debemos garantizar que este último algoritmo tome tiempo lineal. Por desgracia, al menos para el autor de este trabajo, no es evidente que el Algoritmo 4 se ejecute en tiempo lineal, pues en principio debemos acotar el número de veces que se hacen los refinamientos y, más importante aún, debemos garantizar que la toma del vértice σ-mínimo no desborde la complejidad temporal.

Dadas estas circunstancias, antes de continuar con la sección dedicada a la construcción del algoritmo de reconocimiento, vamos a dedicar la presente sección para convencernos de que, en efecto, LexBFS tiene complejidad temporal lineal. Lo anterior lo vamos a llevar a cabo, primero, definiendo objetos especiales para poder obtener los mínimos en tiempo constante, y segundo, construyendo un algoritmo que hace lo mismo que el Algoritmo 4, un poco más complejo, sobre el cual sí exhibiremos directamente la complejidad temporal lineal. A lo que llamamos objetos especiales, será de hecho una estructura de datos, así que se recomienda revisar la parte de estructuras de datos, la cual se encuentra en la la Sección 1.9.

Vamos a llamar lista con intervalos a la estructura que introducimos. La vamos a usar para refinar las celdas de una lista doblemente ligada, respecto a un conjunto S en tiempo $O(|S|)$. Primero hablemos un poco de **apuntadores**. Un apuntador es un objeto p que representa a una referencia, es decir, el objeto p solo almacena un valor que es la referencia del objeto al que apunta, recalquemos que una referencia a un objeto no es lo mismo que el objeto en sí. Por poner un ejemplo, supongamos que p apunta al objeto x, entonces, p no almacena al objeto x, solo almacena una referencia al objeto x. Esto nos da la ventaja de poder acceder y por ende manipular a x, pero sin tener a x arraigado a p. Supongamos que p es un apuntador. Tenemos que p puede ser un apuntador nulo, o sea no apuntar a nada, o bien, ser un apuntador no

nulo, en cuyo caso podemos acceder al objeto al que referencia p con la instrucción ref(p), la cual nos proporciona el objeto al que apunta p en tiempo constante. Para asignar un nuevo valor a p, o sea una nueva referencia a algún objeto x, simplemente usamos la asiganación ref(p) $\leftarrow x$. Decimos que una celda c en una lista doblemente ligada es una **celda final** o simplemente final, si es la última celda de L, o sea, si $siguiente_L(c)$ es un apuntador nulo. De modo similar, la celda c en una lista doblemente ligada es una **celda inicial** o simplemente inicial, si es la primera celda de L, o sea, si $anterior_L(c)$ es un apuntador nulo. Por último, dado cualquier objeto x, utilizaremos la notación x^* para construir un apuntador hacia x, es decir, x^* es un apuntador que referencia al objeto x.

Definición 4.2.1. Un **intervalo** es un objeto $I = \langle v, p\rangle$ que almacena un valor númerico v y un apuntador p, a los cuales les daremos la notación de I_v e I_p respectivamente.

Definición 4.2.2. Una **lista con intervalos** sobre un conjunto X es una estructura de datos $\mathcal{L} = \langle L, C, A, \preceq\rangle$. Aquí tenemos lo siguiente: L es una lista doblemente ligada, sin repeticiones de valores, cuyas celdas tienen valores en X; C es una lista doblemente ligada de intervalos; $\preceq$ es un orden lineal sobre X; A es un arreglo tal que para $v \in X$, tenemos la entrada $A[v]$ en A que almacena tres registros, a los cuales nos referiremos como $A[v](1)$, $A[v](2)$ y $A[v](3)$. El objeto $A[v](1)$ es un apuntador que referencia a una celda de C, también $A[v](2)$ es un apuntador, pero este apunta a una celda de L, por último, $A[v](3)$ es un booleano que nos va a servir para saber si v ya fué explorado en LexBFS o no. Además, $\mathcal{L}$ satisface las siguientes condiciones.

1. Para cualquier celda I de C, I_p apunta a un elemento de L. También se satisface que $(primero(C))_p$ apunta a $primero(L)$.

2. Para cualesquiera dos elementos I y J de C, se cumple que, $I <_C J$ si y solo si ref$(I_p) <_L$ ref(J_p).

3. Para cualquier I con sucesor J en C, las celdas x de L en el segmento ref$(I_p) \leq_L x <_L$ ref(J_p), satisfacen que $A[valor(x)](1)$ apunta a I. También, para F en C final, las celdas x de L en el segmento ref$(I_p) \leq_L x$, satisfacen que $A[valor(x)](1)$ apunta a F.

4. Para cualquier I con sucesor J en C, si x y z son celdas en L que satisfacen ref$(I_p) \leq_L x <_L z <_L$ ref(J_p), entonces $valor(x) \preceq valor(z)$. También, para F en C final, si x y z son celdas en L que satisfacen ref$(I_p) \leq_L x <_L z$, entonces $valor(x) \preceq valor(z)$.

5. Para toda celda x de L, $A[valor(x)](2)$ apunta a x.

Expliquemos un poco la idea detrás de la Definición 4.2.2, para esto, optaremos por explicar las cinco propiedades de la definición, pues con eso bastará para comprender la utilidad de los objetos adoptados. El objetivo es tener en L todos los valores de interés, mientras que los intervalos en C representen bloques $\preceq$-ordenados de celdas contiguas en L, de tal modo que todos los intervalos de C juntos conformen a la totalidad de L.

1. Lo único que garantiza esta condición es que, los mínimos de los elementos de C están representados por celdas de L, y en particular que el mínimo del segmento representado por $primero(C)$, o sea el primer intervalo, está representado por el mínimo de L.

2. Este requisito es para garantizar que los intervalos de C estén ordenados de la misma forma como segmentan a L. Es decir, para garantizar que no se traslapen.

3. Este inciso es para establecer que los elementos de C en efecto son intervalos, en el sentido de que tienen segmentos de L continuos, en otras palabras, están conformados por ciertas celdas de L consecutivas.

4. Esto es solo para asegurar que los segmentos en L representados por los intervalos de C, sí están $\preceq$-ordenados.

5. Con este inciso poseemos una manera de encontrar en tiempo constante, para cualquier valor presente en L, su posicionamiento en L, o sea, la única celda que almacena ese valor.

Una vez definidos los intervalos y las listas con intervalos, damos un algoritmo que será utilizado como subrutina por la versión de LexBFS lineal. Este algoritmo es el núcleo de LexBFS, pues es el encargado de hacer los refinamientos de cada una de las partes. El algoritmo es **Refina**$(\mathcal{L}, S, j)$; toma como entrada una lista con intervalos $\mathcal{L}$ sobre un conjunto X, un subconjunto $S \subseteq X$ y un entero j que satisface ciertas condiciones. La operación consiste en reemplazar cada intervalo I de $\mathcal{L}$, por la pareja de intervalos contiguos $I \cap S$ e $I \setminus S$ justo en ese orden. Un comentario importante para entender mejor el algoritmo es que el entero j sirve para identificar cuales intervalos son nuevos y cuales ya estaban al principio de la ejecución, o sea, para identificar que intervalos no hace falta refinar. Refina$(\mathcal{L}, S, j)$ es el Algoritmo 6.

Algoritmo 6: Refina$(\mathcal{L}, S, j)$.

Input: Lista con intervalos $\mathcal{L} = \langle L, C, A, \preceq \rangle$ sobre un conjunto X; una lista doblemente ligada y sin repeticiones S que tiene valores en X, y además el orden de S es compatible con $\preceq$; un entero j de tal suerte que para cada $I \in C$ se cumple $j > I_v$

Output: No tiene

```
1  /* La exploración del for se lleva a cabo en el orden de S, y
      solo para las celdas con su valor presente en L                */
2  for g celda en S, A[valor(g)](3) = 1 do
3      s ← valor(g);
4      I ← ref(A[s](1));
5      x ← ref(A[s](2));
6      /* Si I tiene otro elemento además de x                       */
7      if ref(I_p) no es final en L ∧ A[valor(siguiente_L(ref(I_p)))](1) apunta a I
        then
8          /* Se crea el intervalo correspondiente a S ∩ I de ser
              necesario                                              */
9          if I es inicial en C ∨ (anterior_C(I))_v < j then
10             Insertar un nuevo intervalo J = ⟨j, x*⟩ a la izquierda de I;
11         end
12         /* Se hace el cambio de intervalo para x                  */
13         if I_p apunta a x then
14             /* Si la celda x era la mínima en el intervalo I      */
15             Apuntar I_p a siguiente_L(x);
16         end
17         Apuntar A[s](1) a anterior_C(I);
18         Cambiar x justo a la izquierda de ref(I_p) en L;
19     else if I no es inicial en C ∧ (anterior_C(I))_v = j then
20         Apuntar A[s](1) a anterior_C(I);
21         del(C, I);
22     end
23 end
```

Proposición 4.2.3. *El Algoritmo 6 es correcto. Es decir, después de la ejecución Refina$(\mathcal{L}, S, j)$, para cada intervalo I en C, visto como el segmento de valores en L que representa, quedan dos intervalos $\preceq$-ordenados $I \cap S$ e $I \setminus S$, esto solo en caso*

de que $\varnothing \subsetneq I \cap S \subsetneq I$.

Demostración. Sea I cualquier intervalo que estaba presente en C antes de iniciar la ejecución. Si I solo contempla una celda de L, no se hace nada pues no se satisfacen ni la cláusula del `if` ni la del `else`. Por este motivo, desde aquí hasta el final de la demostración supongamos que I abarca más de una celda. Si $I \cap S = \varnothing$, tampoco hay cambios en I, así que supongamos también que $I \cap S \neq \varnothing$. Consideremos el primer valor s de S, tal que $A[s](1)$ apunta a I. Lo primero que se hace es insertar un nuevo intervalo, digamos J, a la izquierda de I que representa a $I \cap S$, aquí hay dos escenarios. Si I_p apunta a la celda de s, es decir, si s es el valor mínimo cubierto por I, se corrige al mínimo de I, es decir, se modifica el apuntador I_p para que ahora apunte a la celda siguiente de la correspondiente a s en L. También se designa a J como nuevo intervalo que abarca el valor s, y como la celda de s ya está a la izquierda del nuevo mínimo de I (la siguiente celda) no se mueve a la celda de s de posición en L. Si I_p no apunta a la celda de s, se apunta $A[s](1)$ a J y se cambia a la celda de s para estar exactamente a la izquierda de la celda $\text{ref}(I_p)$. En ninguno de los escenarios se altera el orden de las celdas en I, pues solo desmarcamos a una celda y la quitamos del segmento. Tampoco hay problema con el orden de J, ya que hasta este punto solo tiene la celda correspondiente a s. Posteriormente, cuando exploremos otro elemento t de S tal que $A[t](1)$ apunte a I, volvemos a tener los dos escenarios anteriores, y volvemos a hacer lo mismo. El orden de I no sufre cambio alguno porque, de nuevo, solo estamos desmarcando la celda de t y quitándola del segmento que abarca I. Por su parte, el orden de J se mantiene, porque todos los elementos en J hasta este punto eran valores $\preceq$-menores a t, pues S está $\preceq$-ordenada, así que el insertar la celda de t justo a la izquierda de $\text{ref}(I_p)$, o sea a la derecha de todos estos valores, no corrompe el orden de J. El caso que merece especial mención es en el que se entra al `else`. Este caso se alcanza cuando todos lo elementos que cubría I terminaron por recorrerse al dominio de J, aquí simplemente se termina de trasladar el último elemento que ocupa I para eliminarse el intervalo I, ya vacío, de C. Por el mismo argumento esto no revuelve el orden interno que tiene J. Para terminar, observemos que todos los valores de J son valores que estaban en un principio en I, así que jamás va a ocurrir que en la exploración de un valor s en S se tenga que $A[s](1)$ apuntando a J antes de explorarlo, esto ocurre después. Una parte formal de la demostración es verificar que el estado en el que queda la estructura $\mathcal{L}$ después de la ejecución sigue preservando las condiciones de lista con intervalos. Sin embargo, no dearrollaremos más, pues creemos que los detalles pendientes son, o bastante inmediatos, o se siguen fácilmente de lo anterior. Por lo tanto, una vez que se terminan de explorar con el `for` todos los valores presentes en S, tendremos la certeza de que J tiene los valores

de $I \cap S$ mientras que las celdas que abarca I, al final del algoritmo, son solamente las de $I \setminus S$. ■

Proposición 4.2.4. *La ejecución de Refina$(\mathcal{L}, S, j)$ en el Algoritmo 6 toma tiempo del orden $\mathcal{O}(|S|)$, donde $|S|$ es la longitud de S.*

Demostración. Todas las inserciones y eliminaciones que se realizan en listas toman tiempo constante, además, las otras operaciones son asiganciones y comparaciones, las cuales toman también tiempo constante. Por lo tanto, cada iteración del `for` toma tiempo $\mathcal{O}(1)$. Como hay una iteración del `for` por cada celda de S y la exploración se lleva a cabo en el orden de S, podemos concluir que la ejecución toma tiempo $\mathcal{O}(|S|)$. ■

En el Algoritmo 6, para cualquier I en C, el nuevo intervalo J, que representa a $I \cap S$, siempre se inserta a la izquierda del intervalo I, pues eso es lo que hace justamente LexBFS. Podríamos pensar en cambiar esa línea del algoritmo para que la inserción se realice a la derecha en lugar de la izquierda, justo como lo hace LexBFS$^-$. Parecería que al cambiar la línea que índica el lado de inserción nos resulta en una subrutina que podemos utilizar para LexBFS$^-$, sin embargo hay que ser cuidadosos, pues podrían surgir problemas al perder el orden interno de los intervalos. Así que también debemos modificar la línea en donde se cambia a la celda x que tiene al valor en exploración, ahora la queremos cambiar a la derecha, por que ahí es donde está J. Pero, si queremos preservar el orden de los elementos que ya tiene J, hay que pasar a x hasta el final del intervalo J, puesto que por el orden de S la celda x tiene un valor que es $\preceq$-mayor a todos los que actualmente están en J. Otra alternativa a lo anterior, es ir recorriendo los elementos a la derecha y ponerlos al inicio del intervalo J, pero esto ocasionaría que terminaramos con el orden invertido en J, así que la clave sería considerar a S con el orden invertido también. Otros cambios menores son necesarios en las condiciones de los condicionales y en otras líneas. En el Algoritmo 7 podemos ver la rutina Refina$^-(\mathcal{L}, S, j)$ que hace exactamente lo mismo que el Algoritmo 6, pero haciendo las inserciones a la derecha.

El Algoritmo 7 y el Algoritmo 6 son casi idénticos. Como optamos por el primer enfoque de recorrer las celdas hasta la derecha del nuevo intervalo cuando las exploramos, tuvimos que separar dos casos con el `if-else` de la línea `18`, los otros cambios están subrayados. Como las pruebas son sumamente similares a las de Refina$(\mathcal{L}, S, j)$, simplemente establecemos sin mayor desarrollo que Refine$^-(\mathcal{L}, S, j)$ toma tiempo del orden $\mathcal{O}(|S|)$ y sustituye cada intervalo I de C por el par de intervalos $I \setminus C$, $I \cap C$ en ese orden, cuando $\varnothing \subsetneq I \cap C \subsetneq I$.

Algoritmo 7: Refina$^{-}(\mathcal{L}, S, j)$.

Input: Lista con intervalos $\mathcal{L} = \langle L, C, A, \preceq \rangle$ sobre un conjunto X; una lista doblemente ligada y sin repeticiones S que tiene valores en X, y además el orden de S es compatible con $\preceq$; un entero j de tal suerte que para cada $I \in C$ se cumple $j > I_v$.

Output:

```
1  /* La exploración del for se lleva a cabo en el orden de S, y
      solo para las celdas con su valor presente en L            */
2  for g celda en S, A[valor(g)](3) = 1 do
3      s ← valor(g);
4      I ← ref(A[s](1));
5      x ← ref(A[s](2));
6      /* Si I tiene otro elemento además de x                   */
7      if ref(I_p) no es final en L ∧ A[valor(siguiente_L(ref(I_p)))](1) apunta a I
        then
8          /* Se crea el intervalo correspondiente a S ∩ I de ser
              necesario                                          */
9          if I es final en C ∨ (siguiente_C(I))_v < j then
10             Insertar un nuevo intervalo J = ⟨j, x*⟩ a la derecha de I;
11         end
12         /* Se hace el cambio de intervalo para x              */
13         if I_p apunta a x then
14             /* Si la celda x era la mínima en el intervalo I  */
15             Apuntar I_p a siguiente_L(x);
16         end
17         Apuntar A[s](1) a siguiente_C(I);
18         if siguiente_C(I) es final en C then
19             Cambiar x hasta el final de L;
20         else
21             Cambiar x justo a la izquierda de
                ref((siguiente_C(siguiente_C(I)))_p);
22         end
23     else if I no es final en C ∧ (siguiente_C(I))_v = j then
24         Apuntar A[s](1) a siguiente_C(I);
25         if siguiente_C(I) es final en C then
26             Cambiar x hasta el final de L;
27         else
28             Cambiar x justo a la izquierda de
                ref((siguiente_C(siguiente_C(I)))_p);
29         end
30         del(C, I);
31     end
32 end
```

Es momento de dar la versión explícitamente lineal de LexBFS, la cual llamaremos **LxBFS** para distinguirla de la otra que ya hemos ido explorando. La idea en LxBFS(G, τ) es utilizar listas con intervalos $\mathcal{L} = \langle L, C, A, \preceq \rangle$ y la rutina Refina$(\mathcal{L}, S, j)$. Lo que haremos será almacenar en las celdas de L a los vértices que quedan por explorar, mientras que los intervalos de C serán los conjuntos que se van refinando; como es de sospechar, $\preceq$ va a estar dado por $\leq_\tau$. Vamos a utilizar el Algoritmo 2, así que el lector hará bien en revisarlo de ser necesario. El algoritmo LxBFS lo podemos ver en el Algoritmo 8.

En la Definición 4.2.2 no profundizamos en el orden $\preceq$, en particular, no explicamos cómo se codifica para pasarlo como la componente de la lista con intervalos. Esto lo dejamos al aire adrede para no cargar la definición de lista con intervalos, sin embargo, lo vamos a aclarar aquí, pues de hecho lo utiliza el Algoritmo 8. La forma en la que vamos a códificar los ordenes, y los ordenamientos, será como una lista doblemente ligada. Es decir, el orden lineal $\preceq$ de un conjunto X, lo podemos codificar como una lista doblemente ligada cuyas celdas almacenan a los elementos de X, y naturalmente la primera celda en esa lista tiene al $\preceq$-mínimo, la siguiente celda tiene al elemento que sigue en el orden, etcétera. En particular, el orden $\leq_\tau$ que usamos en el Algoritmo 8, lo podemos considerar como la lista doblemente ligada $[\tau^{-1}(0), \ldots, \tau^{-1}(n-1)]$. Esta representación de los órdenes nos es útil para garantizar la cota de tiempo.

Teorema 4.2.5. *El Algoritmo 8 es correcto, es decir, LxBFS hace lo mismo que LexBFS.*

Demostración. Lo que queremos demostrar es que, en todo momento del algoritmo los subconjuntos de V_G representados por los intervalos de la lista con intervalos $\mathcal{L}$, son justamente los valores no vacíos de las celdas de L en el Algoritmo 4, y desde luego, que están en el mismo orden. Lo anterior basta porque $\sigma(v)$ se define en función a la iteración correspondiente para v del `while`, esto en ambos algoritmos. Vamos a hacer inducción sobre las iteraciones del `while`, donde la propiedad que queremos inducir, y que es también la hipótesis inductiva, es que los intervalos de la lista con intervalos $\mathcal{L}$ satisfacen lo ya mencionado. Lo primero que hacemos en el algoritmo es ordenar las listas de adyacencias de G llamando a la rutina ordena(G, τ), esto lo podemos hacer sin resquemor, pues podemos suponer sin pérdida de generalidad que V_G es justamente el conjunto $\{0, \ldots, n-1\}$. De esta manera, G' es la gráfica G, pero con sus listas de adyacencias ya ordenadas respecto a τ. En las líneas siguientes, hasta antes del `while`, se crea una lista con intervalos, cuyo único intervalo representa a todo V_G, así que el paso base, que es el estado antes de entrar al `while` en ambos algoritmos, queda demostrado. Al comienzo de cada iteración del `while`, se toma

Algoritmo 8: LxBFS(G, τ).

Input: Una gráfica G de orden n, codificada por sus listas de adyacencias, y un ordenamiento τ de G

Output: Un ordenamiento σ de V_G

```
1  /* Se ordenan las listas de adyacencias respecto a ≤_τ          */
2  gráfica G' ← ordena(G, τ);
3  /* Se crea una lista con intervalos que represente la
      configuración inicial de LexBFS, o sea (V_G)                 */
4  intervalo I ← (0, p);
5  lista C ← Una lista doblemente ligada cuya única celda tenga valor I;
6  lista L ← Una lista doblemente ligada vacía;
7  n-arreglo A ← Un arreglo cuyos registros almacenan, cada uno, tres
   apuntadores A[i](1), A[i](2) y A[i](3);
8  entero i ← 0;
9  /* Se prepara la lista L con los vértices, y se registra el
      intervalo y celda actual de cada vértice                     */
10 while i < n do
11 |  Insertar una celda con valor τ^{-1}(i) al final de L;
12 |  Apuntar A[τ^{-1}(i)](1) hacia primero(C) y apuntar A[τ^{-1}(i)](2) hacia
   |   ultimo(L);
13 |  A[τ^{-1}(i)](3) ← 1;
14 |  i ← i + 1;
15 end
16 Apuntar (primero(C))_p hacia primero(L);
17 lista con intervalos L ← ⟨L, C, A, ≤_τ⟩;
18 /* Se toma consecutivamente al primer vértice de L y se usa su
      vecindad para refinar todos los intervalos                   */
19 entero j ← 1;
20 i ← 0;
21 while i < n do
22 |  u ← primero(L);
23 |  Jp ← A[valor(u)](1);
24 |  /* Se quita a u de la lista con intervalos                   */
25 |  if u es final en L ∨ A[valor(siguiente_L(u))](1) no apunta a ref(Jp) then
26 |  |  Quitar a ref(Jp) de C;
27 |  else
28 |  |  Apuntar (ref(Jp))_p hacia siguiente_L(u);
29 |  end
30 |  Quitar a u de L y hacer A[valor(u)](3) ← 0;
31 |  σ(valor(u)) ← i;
32 |  j ← j + 1, i ← i + 1;
33 |  Refina(L, G'[valor(u)], j);
34 end
35 return σ;
```

a u como la primera celda de L, que por hipótesis inductiva y por la definición de lista con intervalos, debe coincidir con la celda que tiene como valor al vértice $\leq_\tau$-mínimo del primer intervalo. Acto seguido, considerando a I como el intervalo que contempla a u, al quitar a u de la lista con intervalos, sustituimos al intervalo I por $I \setminus \{valor(u)\}$ en caso de que I no abarque únicamente a u, si no, simplemente quitamos al intervalo I. Gracias a la Proposición 4.2.3, podemos garantizar que después de efectuar la subrutina Refina$(\mathcal{L}, G'[valor(u)], j)$ los intervalos I de $\mathcal{L}$ que satisfacen $\varnothing \subsetneq I \cap G'[valor(u)] \subsetneq I$, son reemplazados por los intervalos consecutivos $I \cap N(valor(u))$, $I \setminus N(valor(u))$, pues $G'[valor(u)]$ no es otra cosa que el conjunto $N(valor(u))$ en forma de lista. Por lo tanto, en el `while` hacemos lo mismo que en LexBFS, con lo que queda completa la inducción. ■

Teorema 4.2.6. *La ejecución LxBFS(G, τ) del Algoritmo 8, se ejecuta en tiempo $\mathcal{O}(n+m)$, donde n es el orden de G y m es el tamaño de G.*

Demostración. Primero ejecutamos ordena(G, τ) para obtener G', y por la Proposición 1.9.6, sabemos que nos toma tiempo $\mathcal{O}$ de $n + \sum_{i=0}^{n-1} |G[i]| = n + \sum_{i=0}^{n-1} d(i) = n + 2m$, pero esta última función es claramente $\mathcal{O}(n+m)$. Hasta antes del primer `while`, se realizan únicamente operaciones de tiempo constante. Entrando en el primer `while`, el considerar a τ como una lista doblemente ligada, nos permite recorrerla en orden, avanzando sobre ella en cada iteración de dicho `while`, o sea que obtener $\tau^{-1}(i)$ en la iteración del primer `while` toma tiempo constante. Así, en el primer `while`, también se hacen solo operaciones de tiempo constante, así que el tiempo en el que se termina de ejecutar el primer `while` resulta ser $\mathcal{O}(n)$. Después tenemos el incremento a i, operación que toma una unidad de tiempo. Finalmente tenemos el segundo `while`, aquí también tenemos en su mayoría operaciones de tiempo constante, solo hay que desarrollar un poco acerca de las instrucciones Refina$(\mathcal{L}, G'[valor(u)], j)$ y $\sigma(valor(u)) \leftarrow i$. En la primera instrucción, sabemos por la Proposición 4.2.4, que tomamos tiempo $\mathcal{O}(|G'[valor(u)]|)$, o lo que es lo mismo, tiempo $\mathcal{O}(d(valor(u)))$, mientras que el hacer $\sigma(valor(u)) \leftarrow i$ toma tiempo constante, pues al considerar a σ como una lista, el efectuar esa línea equivale a insertar una celda al final de la lista σ. Dado que en cada iteración del segundo `while` se explora una celda por cada vértice, podemos concluir que el segundo `while` toma tiempo $\mathcal{O}$ de $\sum_{i=0}^{n-1} d(i) = 2m$, o sea, toma tiempo $\mathcal{O}(m)$. Por lo tanto, todo el algoritmo se ejecuta en $\mathcal{O}(n+m)$ pasos. ■

Hagamos la observación respectiva para $LexBFS^-$. Ya no escribiremos el algoritmo que, siguiendo nuestro patrón, se debería llamar **LxBFS$^-$** . Sin embargo, es muy fácil notar que a partir del Algoritmo 8, simplemente utilizando Refina$^-$ en lugar de

Refina, podemos obtener la versión del Algoritmo 8 pero para LexBFS$^-$. O sea que a partir de este momento, asumiremos sin ningún pesar, que la versión LexBFS$^-(G, \tau)$ del Algoritmo 5, también toma tiempo $\mathcal{O}(n + m)$.

4.3. Algoritmo de reconocimiento para cográficas

En esta sección vamos a presentar la mayoría de preliminares necesarios para el algoritmo de reconocimiento en tiempo lineal que viene dado en [2]. Como es de esperar, echaremos mano a todas las herramientas que desarrollamos antes, desde los árboles enraízados y cográficas hasta LexBFS y su relación con LexBFS$^-$. Claro que todavía necesitamos dar más conceptos y más resultados, toda esta teoría que nos queda por ver nos dará la clave de como caracterizar a las cográficas en términos de ordenamientos LexBFS y LexBFS$^-$.

Sean G una gráfica y σ un ordenamiento LexBFS de V_G. Definimos la **vecindad local** de una x-celda en σ, digamos $S_i(x)$, como el conjunto

$$N^l_\sigma(S_i(x)) = \{v \in S(x) \colon v <_\sigma x_i, N(v) \cap S_i(x) \neq \varnothing\}.$$

Como de costumbre, casi nunca escribiremos el subíndice σ, a menos que necesitemos hacer explícito el ordenamiento, o sea que solo escribiremos $N^l(S_i(x))$. Veamos un resultado respecto a la definición anterior que es bastante inmediato.

Proposición 4.3.1. *Sean G una cográfica, σ un ordenamiento LexBFS de V_G y $S_i(x)$ cualquier x-celda en σ. La vecindad local $N^l(S_i(x))$ coincide con $N_<(S_i(x)) \cap S^A(x)$.*

Demostración. Siguiendo las definiciones nada más, podemos ver que $N_<(S_i(x)) \cap S^A(x) \subseteq N^l(S_i(x))$. Ahora, si $z \in N^l(S_i(x))$, se tiene $z \in S^N(x) \cup S^A(x) \cup \{x\}$. Pero $z \neq x$ porque $S_i(x) \subseteq S^N(x)$, tampoco se puede tener $z \in S^N(x)$ por el Teorema 4.1.9 y tomando en cuenta que $z <_\sigma x_i$. Con lo anterior deducimos que $z \in S^A(x)$. Por otro lado, z no puede ser a la vez adyacente a algún vértice de $S_i(x)$, y no ser adyacente a otro vértice de $S_i(x)$, pues eso implicaría por la definición de x-celda que tales vértices deberían pertenecer a x-celdas distintas. Así, z es adyacente a todos los vértices en $S_i(x)$, o bien no es adyacente a ninguno. Sin embargo, el tener que $z \in N^l(S_i(x))$ implica que z tiene un vecino en $S_i(x)$, por lo que $S_i(x) \subseteq N(z)$. Por lo tanto, $z \in N_<(S_i(x)) \cap S^A(x)$. ■

Definición 4.3.2. Sean G una gráfica y σ un ordenamiento LexBFS de V_G devuelto por la ejecución LexBFS(G, τ). Decimos que σ satisface la **propiedad del subconjunto vecindad** respecto a τ, o para abreviar, satisface **NSP** respecto a τ, si y solo

si, considerando las x-celdas en la ejecución LexBFS(G, τ), se cumple lo siguiente: Para todo $x \in V_G$ y cualesquiera $i < j$ tales que existe la x-celda $S_j(x)$, se satisface $N^l(S_j(x)) \subsetneq N^l(S_i(x))$.

La definición anterior tiene la incomodidad de depender del ordenamiento τ utilizado para calcular σ, así que diremos que un ordenamiento σ satisface NSP a secas, cuando exista un ordenamiento τ respecto al cual σ satisface NSP. A lo largo de las pruebas en esta sección, todas las rebanadas y celdas son consideradas en una ejecución LexBFS arbitraria que culmine en el ordenamiento σ, pues es irrelevante, casi siempre, el ordenamiento τ que se utiliza para calcular σ en el barrido LexBFS.

Lema 4.3.3. *Sean G una gráfica y σ un ordenamiento LexBFS de V_G. Supongamos que P es una subgráfica inducida de G que es isomorfa a P_4 y consideremos $S(x)$ una rebanada de σ que contiene a los vértices de P. Si $S^A(x) \cap V_P$ es el conjunto de puntos intermedios de P o x es un extremo de P, entonces σ no satisface NSP.*

Demostración. Primero supongamos que x es un extremo de P, donde P es la trayectoria (x, w, y, z). Así, $w \in S^A(x)$ y $y, z \in S^N(x)$. Como $wy \in E_G$ y $wz \notin E_G$, se tiene para algunos $i \neq j$ que $y \in S_i(x)$ y $z \in S_j(x)$. Si $i < j$, entonces $y \in N^l(S_j(x)) \setminus N^l(S_i(x))$. Si $j < i$, por el Corolario 4.1.6, para cualquier $v \in S_j(x)$ se tiene $wv \notin E_G$ porque $wz \notin E_G$, entonces $w \in N^l(S_i(x)) \setminus N^l(S_j(x))$. Por lo tanto σ no satisface NSP.

Supongamos ahora que x no es extremo de P y $S^A(x) \cap V_P$ es el conjunto de puntos intermedios de V_P. Escribamos a P como la trayectoria (v, w, y, z). Ocurre que $w, y \in S^A(x)$ y $v, z \in S^N(x)$. Dado que $wv \in E_G$ y $wz \notin E_G$ con $w \in S^A(x)$, deben haber $i \neq j$ tales que $v \in S_i(x)$ y $z \in S_j(x)$. Como $w \in S^A(x)$, $wv \in E_G$ y $wz \notin E_G$, se satisface por el Corolario 4.1.6 que $S_i(x) \subseteq N(w)$ y $S_j(x) \cap N(w) = \varnothing$. Un argumento exactamente simétrico al anterior nos permite concluir que $S_j(x) \subseteq N(y)$ y $S_i(x) \cap N(y) = \varnothing$. Así que $w \in N^l(S_i(x)) \setminus N^l(S_j(x))$ y $y \in N^l(S_j(x)) \setminus N^l(S_i(x))$. Por lo tanto, en cualquiera de los casos $i < j$ o $j < i$, el ordenamiento σ no cumple NSP. ■

Hagamos algunos comentarios acerca de NSP y los ordenamientos LexBFS$^-$. En un principio, la definición de la propiedad NSP la dimos solo para ordenamientos LexBFS. Si tenemos una gráfica G y un ordenamiento LexBFS$^-$ de V_G, digamos σ^-, hay un ordenamiento τ de V_G de tal suerte que σ^- es el ordenamiento que arroja la ejecución LexBFS$^-(G, \tau)$. Sin embargo, recordemos que lo anterior es exactamente lo mismo que σ^- sea el ordenamiento devuelto por la llamada LexBFS$(\overline{G}, \tau)$. Por lo que diremos que σ^- satisface NSP, si y solo si, vista como un ordenamiento LexBFS de la gráfica $\overline{G}$, satisface NSP.

Teorema 4.3.4. *Sean G una gráfica, σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento LexBFS$^-$ que devuelve la ejecución LexBFS$^-(G,\sigma)$. Ambos ordenamientos σ y σ^- satisfacen NSP, si y solamente si, G es una cográfica.*

Demostración. Vamos a utilizar el Teorema 3.4.1, en particular la equivalencia entre ser cográfica y no tener subgráficas inducidas que sean isomorfas a P_4. Para ver que ser cográfica es una condición necesaria en el enunciado del teorema, vamos a llevar la prueba por contrapositiva, supondremos que G tiene alguna subgráfica inducida e isomorfa a P_4. Sean x el vértice σ-mínimo con la propiedad de estar en algún P_4 inducido de G, y tomemos P cualquier P_4 inducido de G que tenga a x como vértice. Consideremos también a $S(v)$ como la rebanada mínima por contención de σ que contiene a los vértices de P. Por la elección de la rebanada, debemos tener $v \leq_\sigma x$, así que vamos a contemplar dos casos. Si $v <_\sigma x$, por la propiedad de mínimo que tiene x, el vértice v no puede estar en ningún P_4 inducido de G, por lo que el Teorema 4.1.12 implica que $S^A(v) \cap V_P$ es el conjunto de puntos intermedios de P. De lo anterior ya se deduce que σ no cumple NSP por el Lema 4.3.3. El segundo caso es que $x = v$, en este caso consideramos otros dos subcasos. Si x es extremo de P, como $V_P \subseteq S(x)$, podemos usar de nuevo el Lema 4.3.3 para concluir que σ no satisface NSP. Ahora, si x es un punto intermedio de P, tomemos $\overline{S}(u)$ la rebanada de σ^- mínima por contención con la propiedad de contener a V_P. Como $x \in \overline{S}(u)$, por la definición de rebanada se debe cumplir $u \leq_\sigma x$, pues σ es el ordenamiento con el que se calcula σ^-. Si $u = x$, entonces x es un extremo del P_4 inducido en $\overline{G}$ dado por $\overline{P}$, así que el Lema 4.3.3 garantiza el incumplimiento de NSP por parte de σ^-. Si tuvieramos $u <_\sigma x$, por la propiedad de mínimo que posee x podemos asegurar que no hay ningún P_4 inducido en G que tenga a u como vértice, por ende no hay ningún P_4 inducido en $\overline{G}$ que tenga a u como vértice. El Teorema 4.1.12 nos dice que $\overline{S^A}(u) \cap V_{\overline{P}}$ es el conjunto de puntos intermedios de $\overline{P}$, pero esto junto el Lema 4.3.3 implica que σ^- no satisface NSP.

La otra parte es la suficiencia de que G sea cográfica. Lo que vamos a demostrar es que σ satisface NSP por el simple hecho de ser el ordenamiento devuelto por LexBFS(G,τ), para algún τ, con G una cográfica. Sean $x \in V_G$ e $i \neq j$ arbitrarios tales que la x-celda $S_j(x)$ existe. Por la Proposición 4.3.1 y el Teorema 4.1.9, tenemos que $N^l(S_j(x)) = N_<(S_j(x)) \cap S^A(x) \subseteq N_<(S_i(x)) \cap S^A(x) = N^l(S_i(x))$. Además, por el Teorema 4.1.9 otra vez y por estar contenidos $S_i(x)$ y $S_j(x)$ en $S^N(x)$, la contención anterior es propia. Por lo tanto σ satisface NSP. Ahora, observemos que σ^- es el ordenamiento devuelto por LexBFS$(\overline{G},\sigma)$ con $\overline{G}$ cográfica, así que, usando un argumento análogo concluimos que σ^- también cumple NSP. ■

Como podemos intuir del resultado anterior, para que nuestro algoritmo de reco-

nocimiento decida si una gráfica es o no una cográfica, en su ejecución va a tener que verificar que un ordenamiento LexBFS de la gráfica satisfaga NSP y luego verificar que un ordenamiento LexBFS$^-$ satisfaga NSP. Así que nuestro objetivo por ahora es mostrar que la verificación de la propiedad NSP puede ser realizada en tiempo lineal, para este fin tenemos el siguiente resultado.

Proposición 4.3.5. *Sean G una gráfica y σ un ordenamiento LexBFS de G que no satisface NSP. Se cumple lo siguiente:*

1. *Sea $x \in V_G$ de tal forma que hay dos x-celdas de σ las cuales no satisfacen la contención de sus vecindades locales que indica NSP. Consideremos $i > 0$ el mínimo índice con la propiedad de que, existe $j > i$ tal que $N^l(S_j(x)) = N^l(S_i(x))$ o $N^l(S_j(x)) \setminus N^l(S_i(x)) \neq \varnothing$. Se cumple que $N_<(\{x_j\}) \cap S(x) = N_<(\{x_i\}) \cap S(x)$ o $(N_<(\{x_j\}) \setminus N_<(\{x_i\})) \cap S(x) \neq \varnothing$.*

2. *Sea $x \in V_G$ y supongamos que $0 < i < j$ satisfacen que, $N_<(\{x_i\}) \cap S(x) = N_<(\{x_j\}) \cap S(x)$ o $(N_<(\{x_j\}) \setminus N_<(\{x_i\})) \cap S(x) \neq \varnothing$, supongamos además que i es mínimo con la propiedad de que existe tal j mayor a i cumpliendo lo anterior. Entonces, se satisface que $N^l(S_j(x)) = N^l(S_i(x))$ o $N^l(S_j(x)) \setminus N^l(S_i(x)) \neq \varnothing$.*

Demostración.

1. Primero supongamos que se satisface $N^l(S_j(x)) \setminus N^l(S_i(x)) \neq \varnothing$ y tomemos u en $N^l(S_j(x)) \setminus N^l(S_i(x))$. Si $ux_j \in E_G$, entonces $u \in N_<(\{x_j\})$. Para ver que $u \notin N_<(\{x_i\})$ consideremos dos casos, el primero es que $x_i \leq_\sigma u$, en este caso, por definición de vecindad izquierda, claramente $u \notin N_<(\{x_i\})$. Por otro lado, si $u <_\sigma x_i$, el hecho de que $u \notin N^l(S_i(x))$ significa que u no es adyacente a ningún vértice de $S_i(x)$, en particular no es adyacente a x_i, así que $u \notin N_<(\{x_i\})$. El segundo caso es que $ux_j \notin E_G$. Como u está en $S^N(x)$, por ser vecino de algún vértice en $S_j(x)$ pero no de todos, podemos usar el Teorema 4.1.7. Entonces, si $k < j$ es tal que $u \in S_k(x)$, deben existir $h \leq k$ y $v \in S_h(x)$ de tal modo que $v \leq_\sigma u$ y $vx_j \in E_G$, de hecho $v <_\sigma u$ porque $ux_j \notin E_G$. Si $x_i \leq_\sigma v$, entonces claramente $v \in N_<(\{x_j\}) \setminus N_<(\{x_i\})$. Si $v <_\sigma x_i$, se sigue que $h < i$ y como $x_h \leq_\sigma v$, se tiene $v \notin N^l(S_h(x))$. Por tanto, no es posible que $v \in N^l(S_i(x))$, pues esto derivaría en la contradicción $N^l(S_i(x)) \setminus N^l(S_h(x)) \neq \varnothing$ con $h < i$. Por lo tanto $v \in N_<(\{x_j\}) \setminus N_<(\{x_i\})$, pues el tener $v \notin N^l(S_i(x))$ implica que $v \notin N_<(\{x_i\})$.

   Ahora supongamos que $N^l(S_i(x)) = N^l(S_j(x))$. Tomemos $u \in (N_<(\{x_j\}) \cap S(x)) \subseteq N^l(S_j(x))$. Por la contención anterior $u \in N^l(S_i(x))$, o sea que $u <_\sigma x_i$. No es posible que $u \in S^N(x)$, pues de ocurrir esto último, tendríamos que

para algún $k < i$, $u \in S_k(x)$ y por lo tanto $u \in N^l(S_i(x)) \setminus N^l(S_k(x)) \neq \varnothing$ contradiciendo la propiedad de mínimo que posee i. De lo anterior se sigue que $u \in S^A(x)$ y por ende u es adyacente a todos los vértices de $S_i(x)$, o sea que $u \in (N_<(\{x_i\}) \cap S(x))$. Es el turno de tomar $u \in (N_<(\{x_i\}) \cap S(x)) \subseteq N^l(S_i(x))$ arbitrario. Por hipótesis $u \in N^l(S_j(x))$. Por el mismo argumento utilizado en la otra contención, se debe tener $u \in S^A(x)$, pues $u <_\sigma x_i$. Se sigue que u debe ser adyacente no solo a un vértice en $S_j(x)$, sino a todos. Por lo tanto $u \in N_<(\{x_j\}) \cap S(x)$.

2. Primero supongamos que se satisface $(N_<(\{x_j\}) \setminus N_<(\{x_i\})) \cap S(x) \neq \varnothing$ y tomemos u en el conjunto anterior. Por definición, tenemos que $u \in N^l(S_j(x))$, si u no pertenece a $N^l(S_i(x))$, ya podemos afirmar que $N^l(S_j(x)) \setminus N^l(S_i(x)) \neq \varnothing$. Veamos que no puede ocurrir $u \in N^l(S_i(x))$. Si $u \in N^l(S_i(x))$, como u es adyacente a vértices de $S_i(x)$, mas no es adyacente a todos porque $ux_i \notin E_G$, tenemos que $u \in S^N(x)$. Como $u <_\sigma x_i$, se sigue la existencia de $k < i$ tal que $u \in S_k(x)$, o sea que $u \in (N_<(\{x_j\}) \setminus N_<(\{x_k\})) \cap S(x)$ con $k < i$, una contradicción. Por lo tanto $u \in N^l(S_j(x)) \setminus N^l(S_i(x))$.

   Ahora supongamos que $N_<(\{x_j\}) \cap S(x) = N_<(\{x_i\}) \cap S(x)$. Consideremos $u \in N^l(S_j(x))$, tenemos dos casos, el primero es que $ux_j \in E_G$, de donde se sigue que $u \in (N_<(\{x_j\}) \cap S(x)) = (N_<(\{x_i\}) \cap S(x)) \subseteq N^l(S_i(x))$. El segundo caso es que $ux_j \notin E_G$, este caso es imposible, pero supondremos que se da para alcanzar una contradicción. Tenemos que $u \in S^N(x)$, porque u es vecino de vértices en $S_j(x)$ pero no es vecino de x_j. Tomemos $k < j$ de tal modo que $u \in S_k(x)$. Por el Teorema 4.1.7, deben existir $h \leq k$ y $v \in S_h(x)$ de tal modo que $v \leq_\sigma u$ y $vx_j \in E_G$, de hecho $v <_\sigma u$ porque $ux_j \notin E_G$. O sea que $v \in N_<(\{x_j\}) \cap S(x) = N_<(\{x_i\}) \cap S(x)$, es decir, $v \in (N_<(\{x_i\}) \setminus N_<(\{x_h\})) \cap S(x)$. La pertenencia anterior implica que $v <_\sigma x_i$, por lo tanto $h < i$, una contradicción a la propiedad mínima que tiene i. De la contradicción anterior podemos afirmar que $ux_j \in E_G$, así que por lo ya argumentado antes tenemos $u \in N^l(S_i(x))$. Solo falta ver la otra contención, así que vamos a suponer que $u \in N^l(S_i(x))$. De nuevo, si $ux_i \in E_G$, se cumple $u \in N_<(\{x_i\}) \cap S(x) = N_<(\{x_j\}) \cap S(x) \subseteq N^l(S_j(x))$. Por otro lado, es imposible que $ux_i \notin E_G$, pues con argumentos análogos a los de la otra contención, podemos garantizar la existencia de $k < i$ y de $v \in S_k(x)$ de tal modo que $vx_i \in E_G$, o sea que $v \in (N_<(\{x_i\}) \setminus N_<(\{x_h\})) \cap S(x)$, pero esto es absurdo porque $h < i$. Por lo tanto, siempre se tiene $ux_i \in E_G$, implicando que $u \in N^l(S_j(x))$ por el caso anterior.

■

Utilizando la Proposición 4.3.5 podemos deducir el siguiente corolario.

Corolario 4.3.6. *Sean G cualquier gráfica y σ un ordenamiento LexBFS de G. El ordenamiento σ satisface NSP, si y solamente si, para todo $x \in V_G$ y para cualesquiera $i < j$ tales que las x-celdas $S_i(x)$ y $S_j(x)$ existen, se satisface la contención propia $N_<(\{x_j\}) \cap S(x) \subsetneq N_<(\{x_i\}) \cap S(x)$.*

El Corolario 4.3.6, junto con algunas observaciones que haremos a continuación, nos brindan la clave para convencernos de que el tiempo en la verificación de NSP es lineal. Primero, veamos como calcular las vecindades izquierdas de los σ-mínimos en cada x-celda. Como parte de este cálculo lo planeamos hacer dentro de la misma ejecución LexBFS en donde se calcula el ordenamiento LexBFS, no vamos a poner el pseudocódigo, pues tendríamos que transcribir otra vez la amplia versión del Algoritmo 8 junto con algunas líneas más. Por lo tanto, vamos a optar simplemente por explicar que cambios son necesarios, esto no resulta ser muy problemático, ya que los cambios necesarios no son tan complejos. Antes de comenzar, recordemos que las listas con intervalos nos permiten obtener los mínimos de los intervalos en tiempo constante.

Un paréntesis para introducir la EDD pila es más que pertinente. Una **pila** Q es una colección de datos que satisface la condición LIFO (Last In First Out) y que cuenta con únicamente cuatro operaciones, a saber $empty(Q)$, $push(Q, x)$, $pop(Q)$, $peak(Q)$. Por razones prácticas, evitaremos definir explícitamente la condición LIFO, y daremos la definición de pila como una clase de listas particulares. Pensaremos una pila Q simplemente como una lista doblemente ligada que tiene sus operaciones habituales "desactivadas", y en cambio, cuenta únicamente con las operaciones $empty(Q)$, $push(Q, x)$, $pop(Q)$ y $peak(Q)$. La operación $empty(Q)$ solo evalua si la lista Q es vacía. Al realizar la instrucción $push(Q, x)$ se almacena al valor x en una celda que es añadida al final de Q, por su parte $pop(Q)$ elimina de Q a su última celda (si $Q = \varnothing$ no hace nada), y por último, $peak(Q)$ nos devuelve una referencia a la última celda de Q, la celda que está en esta posición se llama tope de la pila. De ahora en adelante, cuando insertemos un elemento x a una pila Q, estaremos pensando que lo hacemos con la operación $push(Q, x)$. De manera similar, si decimos que quitamos un elemento de una pila Q, estaremos pensando que lo hacemos con la operación $pop(Q)$. Hacemos la observación de que todas las operaciones sobre una pila se ejecutan en tiempo constante, pues son operaciones sobre listas.

Situémonos en una ejecución de LexBFS de la versión del Algoritmo 8. Vamos a necesitar, para cada $x \in V_G$, una lista U_x que almacenará las cardinalidades de las x-celdas. Consideremos a cualquier vértice x. Cuando tomamos a x como pivote, tenemos que el primer intervalo, o sea $primero(C)$ es justamente la rebanada $S(x)$.

Haciendo uso de las herramientas que tenemos en la ejecución, mientras refinamos los intervalos de C con $N(x)$, podemos darnos cuenta cuando encontramos al primer vértice u en la lista $N(x)$ que está en el intervalo $primero(C)$, tal u resulta ser el σ-mínimo de $S^A(x)$. Lo que nos interesa en particular, es que podemos "prender" una bandera en esa iteración para indicar que $S^A(x)$ no es vacío, si nunca prendemos esa bandera mientras refinamos significa que $S^A(x)$ es vacío, en otras palabras, significa que $S(x) \setminus \{x\} = S^N(x)$. De manera semejante, podemos activar una bandera para indicar si $S^N(x)$ es vacío o no, esta bandera debe apagarse cuando se elimine de C al intervalo $siguiente_C(primero(C))$ por quedar vacío, pues si este intervalo, o sea el segundo de C, queda vacío, significa que $S(x) \setminus \{x\} = S^A(x)$. Así, con las banderas anteriores podemos saber en tiempo constante si $S^A(x)$ es vacío o no, y también si $S^N(x)$ es vacío o no.

Aquí viene uno de los cambios más relevantes, es relevante porque implica la modificación de la estructura misma de lista con intervalos. Recordemos que un intervalo I es de la forma $\langle v, p\rangle$, donde v es el valor númerico que nos ayuda a saber si un intervalo es nuevo o viejo en el Algoritmo 8, mientras que p es un apuntador a la celda de la lista que funge como mínimo de I. Podemos añadir a I un atributo más para poder saber el número de celdas que está contemplando I, este atributo de I lo denotamos por I_k, en consonancia con las notaciones I_v e I_p. Una vez que equipamos a los intervalos con este contador de cardinalidad, podemos también agregar a la definición de lista con intervalos unas claúsulas que garanticen que el contador en efecto tiene el significado que queremos dar. Lo anterior desemboca en cambios sobre el Algoritmo 6 para que incremente la cardinalidad de los intervalos cuando se añadan elementos, o la decremente cuando se quiten elementos. De este modo, en la ejecución LexBFS ya podemos contar con que, dado un intervalo I, podemos obtener su cardinalidad, es decir, el número de celdas en L que está contemplando, lo anterior en tiempo constante.

Volvamos con la ejecución LexBFS. Observemos que, después de la iteración donde x fue pivote ocurre lo siguiente: si $S^A(x) \neq \varnothing$, $primero(C)$ es $S^A(x)$; si $S^A(x) = \varnothing$ pero $S^N(x) \neq \varnothing$, $primero(C)$ es $S^N(x) = S_1(x)$; desde luego, si $S^A(x)$ y $S^N(x)$ son no vacíos ambos, entonces $primero(C)$ es $S^A(x)$ y $siguiente_C(primero(C))$ es $S^N(x)$. Lo que haremos, justo depués de explorar a x, será almacenar en dos registros asociados a x, $R_x(1)$ y $R_x(2)$ las cardinalidades respectivas (posiblemente nulas) de $S^A(x)$ y $S^N(x)$.

En la ejecución de LexBFS, podemos tener una lista doblemente ligada que actúe como una pila, cuya finalidad sea almacenar ciertos contadores $count_x$ para $x \in V_G$. Estos contadores $count_x$ tienen como valor numérico la cardinalidad de $S^A(x)$, o sea lo almacenado en $R_x(1)$. Expliquemos que pretendemos con estos contadores y como

los vamos a manejar. Situémonos después de utilizar al vértice x como pivote, y después de llenar los registros $R_x(1)$ y $R_x(2)$. Tenemos dos casos, el primero de ellos es que la pila sea vacía, aquí simplemente vamos a meter a $count_x$ en la pila, es decir, vamos a hacer un push con $count_x$. El otro subcaso es que la pila no sea vacía, o sea que tenga tope, digamos que tiene a $count_v$ como tope, más adelante veremos que necesariamente $count_v$ debe ser mayor que 0, en ésta situación, antes de meter a $count_x$ como nuevo tope en la pila, vamos a restar $count_x + 1$ de $count_v$, es decir, realizaremos la asignación

$$count_v \leftarrow count_v - count_x - 1.$$

El insertar a $count_x$ en la pila significa que estamos explorando a los vértices en $S^A(x)$ y que, de hecho, los próximos $count_x$ vértices que vamos a explorar, son los vértices que restan en $S^A(x)$.

Lo anterior describe la actividad del algoritmo justo después de explorar a x, inmediatamente después de esto, vamos a realizar lo descrito a continuación. Primero observemos que la pila no es vacía, puesto que $count_x$ es el tope en ese momento. Volvemos a bifurcarnos en casos, el primero es que el tope de la pila, o sea $count_x$, sea mayor que 0, en este caso proseguimos con la ejecución sin hacer nada más. Por otro lado, si $count_x = 0$, entramos en la fase para "recolectar" las cardinalidades de las x-celdas y también de otras v-celdas (ya veremos cuáles). Antes de explicar como hacemos esta recolección, demos un poco de semántica a las instrucciones anteriores.

Interpretemos lo que estamos haciendo, esto para que sirva como prueba de correctud. Los contadores que almacena la pila, son signo de los vértices cuyos conjuntos S^A están actualmente en exploración. Supongamos que en cierto momento, $count_u > 0$ es el tope de la pila justo antes de comenzar a refinar utilizando al vértice v como pivote, no necesariamente siendo $count_u$ la cantidad $|S^A(u)|$, es decir, en un momento en el que posiblemente ya se ha decrementado varias veces el contador de u. Ahora supongamos que ya se terminó de refinar usando a v como pivote. Como $count_u$ era mayor que 0, no habíamos terminado de explorar a los vértices en $S^A(u)$, de hecho, $v \in S^A(u)$ y por ende, gracias a la naturaleza de LexBFS, se tiene que $S^A(v)$ es subconjunto de los vértices en $S^A(u)$ que faltan por explorar en ese momento (después de explorar a v). Lo anterior garantiza que $count_v < count_u$, así que $count_u - count_v - 1$ siempre es al menos 0, es decir, siempre tiene sentido cambiar el valor de $count_u$ como lo indicamos en un párrafo anterior. Lo que significa el cambiar $count_u$ por el valor $count_u - count_v - 1$ es que, una vez que terminamos de explorar tanto a v como a los vértices en $S^A(v)$, nos quedan $count_u - count_v - 1$ vértices en $S^A(u)$ que hacen falta de explorar. De hecho, al terminar de explorar el vértice σ-máximo de $S^A(v)$, o bien, después de explorar a v si $S^A(v) = \varnothing$, $count_u$ va a quedar de nuevo como tope de la

pila, y esta vez con un valor $count_u - count_v - 1$ como esperaríamos. Lo anterior fue suponiendo que $count_u > 0$, sin embargo, al describir la fase de recolección, vamos a ver que no puede ocurrir que $count_u$ sea 0 antes de explorar v, pues en la fase de recolección siempre vamos a quitar contadores de la pila hasta encontrar uno que no sea nulo, o bien, hasta agotar la pila. Todo lo anterior debería bastar para convencernos de que, si $count_u$ es tope de la pila despúes de una ronda de refinamiento, después de aplicar las acciones descritas en los párrafos anteriores, y después de efectuar la fase de recolección, entonces, los siguientes $count_u$ vértices que se van a explorar son los vértices restantes en $S^A(u)$. En particular, la observación que nos interesa y que se deriva de la anterior, es que, en el momento en el que el tope de la pila $count_u$ queda reducido a 0, ya hemos acabado de explorar los vértices en $S^A(u)$. Justamente lo anterior, es lo que nos ayuda a calcular las cardinalidades de las v-celdas al entrar en la fase de recolección, la cual vamos a explicar a continuación.

Dejamos hasta el final la llamada fase de recolección porque es la que involucra más instrucciones. Primero resumamos lo que llevamos hasta ahora. De nuevo, supongamos que el algoritmo acaba de utilizar a $x \in V_G$ como pivote. También supongamos que, para cualquier $v \in V_G$ tal que $count_v$ ya está o estuvo en la pila, el algoritmo ya tiene grabados los dos registros de R_v (incluidos los de R_x). Acto seguido, se resta $count_x + 1$ al contador que es tope de la pila, antes de insetar a $count_x$ claro está. Por lo que nuestro tope nuevo ya es $count_x$, y dijimos que, si $count_x > 0$, simplemente seguimos con la ejecución. La fase de recolección llega cuando $count_x = 0$. El hecho de que $count_x$ sea 0, dado que x es el vértice que acabamos de pivotar, significa que $S^A(x) = 0$, es decir, $S(x) \setminus \{x\} = S^N(x)$; esto no nos interesa tanto, lo que nos interesa es que ya terminamos de explorar a todos los vértices en $S^A(x)$, aunque en este caso no haya habido ninguno. Debido a esto, hacemos lo siguiente. Si $R_x(2) = |S^N(x)| = 0$, significa que x no tiene x-celdas, por lo que, al dejar la lista U_x vacía, ya tendríamos por vacuidad a la lista de las cardinalidades de las x-celdas de forma ordenada. Si $R_x(2) > 0$ comenzamos a recolectar; primero hacemos $I \leftarrow primero(C)$ y establecemos un contador κ en 0, notemos que el intervalo I representa a $S_1(x)$, así que guardamos en U_x a la cardinalidad de $I = S_1(x)$, o sea a I_k y actualizamos a κ con el valor I_k. Acto seguido, evaluamos si $\kappa < R_x(2)$, si se da esta desigualdad, hacemos $I \leftarrow siguiente_C(I)$, es decir, ahora I es el intervalo siguiente del que teníamos a la mano, el intervalo I es entonces $S_2(x)$; ahora guardamos al final de U_x a la cardinalidad de $I = S_2(x)$. Por la igualdad $R_x(2) = |S^N(x)| = \sum_{i>0} |S_i(x)|$, si no se da $\kappa < S_x(2)$, debe ocurrir entonces que $\kappa = R_x(2)$, por lo que ya contemplamos a todas las x-celdas, así que dejamos de recolectar para x. En caso de $\kappa < R_x(2)$, recolectamos al final de U_x a $|S_3(x)|$ y actualizamos κ con el valor $\kappa + |S_3(x)|$, y así seguimos sucesivamente hasta obtener

$\kappa = R_x(2)$. Observemos que esta recolección de las cardinalidades de las x-celdas, solo para las x-celdas, toma tiempo $O(Z_x)$, donde Z_x es el número de x-celdas que hay.

Una vez que terminamos de recolectar las cardinalidades de las x-celdas en U_x, podemos sacar a $count_x$ de la pila. Si la pila queda vacía, termina la fase de recolección y proseguimos con la ejecución del algoritmo. Si la pila no se vacía y queda con un tope, digamos $count_v$, evaluamos si $count_v$ es 0 o no. En caso de que $count_v$ sea mayor que 0, solo salimos de la recolección y continuamos con la ejecución, pero, si $count_v = 0$, seguimos en la fase de recolección. Seguimos con la fase de recolección, pero esta vez para v, es decir, aplicaremos exactamente lo mismo que en el párrafo anterior, solo que en lugar de hacerlo con x lo hacemos con v, todo de la misma forma, desde luego utilizando ahora $R_v(2)$ para actualizar el contador κ y almacenado las cardinalidades de las v-celdas en U_v, una vez que terminemos hay que retirar a $count_v$ de la pila. Algo interesante de notar es que, como modificamos a $count_v$ por el valor $count_v - count_x - 1$ antes de guardar a $count_x$ en la pila, y gracias a la desigualdad $count_v \geq count_x + 1$ que explicamos antes, tenemos que: $count_v = 0$ implica que $S^N(x) = \varnothing$. Una vez que $count_v$ salió de la pila, continuamos recolectando las cardinalidades de las u-celdas mientras que, dentro de la fase de recolección, nos encontremos con que el tope de la pila $count_u$ es 0, así hasta encontrar la pila vacía o un tope de la pila que no sea un contador nulo para poder salir de la fase de recolección. Por la observación que hicimos recién, en cuanto nos encontremos con un tope $count_u = 0$ tal que $R_u(2) = |S^N(u)| > 0$, saldremos por fuerza de la fase de recolección después de preparar U_u.

Con la amplia explicación de las últimas páginas, podemos considerar una rutina **LexBFS-NSP**(G, τ) que, paralelamente construye el ordenamiento LexBFS respecto a τ, o sea el ordenamiento σ, y además un arreglo U, donde cada registro $U[x]$ almacena a la lista U_x teniendo las cardinalidades de las x-celdas $S_1(x), \ldots$ (o siendo vacía cuando $S^N(x) = \varnothing$). Algo que también necesitaremos para todo $x \in V_G$, es el vértice σ-mínimo del conjunto $S^N(x)$ y la cardinalidad de $S^N(x)$, en la descripción del nuevo algoritmo lo omitimos para no saturar demasiado, sin embargo, es algo necesario para calcular las vecindades izquierdas que necesitamos. No obstante, esto es muy sencillo de hacer, para poder obtener el σ-mínimo de $S^N(x)$, en caso de que $S^N(x) \neq \varnothing$, podemos guardar el σ-mínimo de $S_1(x)$ cuando guardamos la cardinalidad de $S_1(x)$ en U_x. Por otro lado, para tener la cardinalidad de $S^N(x)$, simplemente guardamos en un arreglo el registro $R_x(2)$. Debido a lo anterior, podemos suponer que la ejecución LexBFS-NSP(G, τ) devuelve $\langle \sigma, U, R, M \rangle$, donde, σ es el ordenamiento LexBFS respecto a τ, U es el arreglo que tiene a las listas U_x de cardinalidades de las x-celdas, R es el arreglo que tiene las cardinalidades $|S^N(x)|$, y M es un arreglo

tal que, para cada $x \in V_G$, $M[x]$ es nulo si $S^N(x) = \varnothing$, o bien, $M[x]$ es el σ-mínimo de $S^N(x)$ en otro caso. Además, como anticipamos al explicar la fase de recolección, la rutina LexBFS-NSP(G, τ) toma el tiempo que utliza el Algoritmo 8, más cierto número constante c de instrucciones por cada iteración del `while` en el que se refina, más el tiempo que tomamos al entrar en cada fase de recolección. Dado que, para $x \in V_G$, el recolectar las cardinalidades de las x-celdas nos toma tiempo $\mathcal{O}(Z_x)$, donde Z_x es el número de x-celdas que hay, podemos concluir que LexBFS-NSP es $\mathcal{O}((n+m) + cn + \sum_{x \in V_G} Z_x)$. Para acotar esta función de tiempo, debemos acotar $\sum_{x \in V_G} Z_x$, lo cual haremos a continuación.

Lema 4.3.7. *Sea G una gráfica de orden n y τ un ordenamiento de V_G. En la ejecución de LexBFS tomando como entrada a G y τ, se generan a lo más n celdas, contando sobre todos los vértices. Es decir, $|\{S_j(x) \colon x \in V_G, j > 0\}| \leq n$.*

Demostración. Vamos a suponer que el ordenamiento LexBFS devuelto por la ejecución es σ. Utilizaremos inducción sobre n para demostrarlo, así, observemos que claramente se satisface para $n \in \{1, 2\}$. Ahora, tomemos $n > 2$ fijo y supongamos válido el enunciado para todo $k < n$. Definamos $u = \sigma^{-1}(n-1)$ y, para cada $x \in V_G$, consideremos a $S^*(x)$ y a $S_j^*(x)$ como la rebanada de x y la j-ésima x-celda respectivamente, pero, generadas en la ejecución LexBFS tomando como entrada a $G - u$ y τ. Por hipótesis inductiva, se cumple que $|\{\{S_j^*(x) \colon x \in V_G \setminus \{u\}, j > 0\}\}| \leq n - 1$. Fijemos $x \in V_G \setminus \{u\}$ y observemos que, la rebanada $S^*(x)$ es $S(x)$, o bien, la rebanada $S^*(x)$ coincide con $S(x) \setminus \{u\}$, esto porque u es el σ-máximo de V_G. Además, si $u \notin S^N(x)$ y $S^N(x) \neq \varnothing$, podemos deducir que $u \notin S(x)$, así pues, en este caso, para todo $j > 0$, la celda $S_j^*(x)$ coincide con $S_j(x)$. Ahora, si $u \in S^N(x)$, por fuerza se debe cumplir que $u \in S_l(x)$, donde $S_l(x)$ es la última x-celda, de aquí tenemos dos casos. Si $\{u\} \subsetneq S_l(x)$, entonces para todo $0 < j < l$, $S_j^*(x) = S_j(x)$ y $S_l^*(x) = S_l(x) \setminus \{u\}$. El otro caso es que $\{u\} = S_l(x)$, en esta situación, para todo $0 < j < l$ se cumple $S_j^*(x) = S_j(x)$, además, no existe la rebanada $S_l^*(x)$. Se satisface que, cuando $S^N(x) \neq \varnothing$, pasa que $|\{S_j(x) \colon j > 0\}| \leq |\{S_j^*(x) \colon j > 0\}| + 1$. Lo anterior se cumple para $x \neq u$ arbitrario, pero podemos decir más aún. Si $x \in V_G$ es tal que, la última de sus celdas, digamos $S_l(x)$ coincide con $\{u\}$, entonces u ya no puede ser parte de una rebanada $S^*(v)$ para $v \in V_G \setminus \{u\}$ con $x <_\sigma v$. De lo anterior se sigue que, existe a lo más un vértice $x \in V_G \setminus \{u\}$ que satisface $|\{S_j(x) \colon j > 0\}| = |\{S_j^*(x) \colon j > 0\}| + 1$. Concluimos entonces que

$$|\{S_j(x) \colon x \in V_G, j > 0\}| \leq |\{S_j^*(x) \colon x \in V_G \setminus \{u\}, j > 0\}| + 1 \leq n.$$

■

Lo que observamos en el párrafo justo antes del Lema 4.3.7, nos permite establecer el tiempo de LexBFS-NSP en el siguiente corolario.

Corolario 4.3.8. *Sean G una gráfica de orden n y tamaño m, además de τ un ordenamiento de V_G. La ejecución LexBFS-NSP(G, τ) toma tiempo $\mathcal{O}(n+m)$.*

El siguiente paso es dar un algoritmo que utilice a LexBFS-NSP para calcular las vecindades izquierdas que necesitamos, luego, dar otro algoritmo para verificar la contención de esas vecindades izquierdas, o sea, para verificar NSP. El construir las vecindades y verificar NSP lo podríamos hacer en la misma rutina. Sin embargo, como nuestra prioridad es la transparencia, haremos esas dos cosas en etapas, o sea en rutinas, distintas.

Lema 4.3.9. *Suponiendo que estamos en la ejecución de LexBFS para una gráfica G con ordenamiento τ, se satisface lo siguiente. Para cualesquiera $x, y \in V_G$ distintos y cualesquiera $i, j > 0$ tales que $S_j(x)$ y $S_i(y)$ existen, se cumple $x_j \neq y_i$. Además, para cualesquiera $a, b > 0$ tales que $S_a(x)$ y $S_b(x)$ existen, se cumple que $x_a \neq x_b$.*

Demostración. Supongamos sin pérdida de generalidad que $x <_\sigma y$. Si $y \notin S(x)$, se tiene $S(y) \cap S(x) = \varnothing$, y si $y \in S^A(x)$, se sigue que $S(y) \subseteq S^A(x)$, en ambos casos se cumple claramente $x_j \neq y_i$. El caso que nos queda por ver es cuando $y \in S^N(x)$. En dicho caso existe $t > 0$ tal que $y \in S_t(x)$, así que $S(y) \subseteq S_t(x)$. Si la x-celda S_{t+1} existe, se cumple $x_t \leq_\sigma y <_\sigma y_i <_\sigma x_{t+1}$, de donde se sigue que $x_j \neq y_i$. Por otro lado, si $S_t(x)$ es la última x-celda, entonces $x_j <_\sigma y_i$. Ahora, el que $x_a \neq x_b$ es obvio de que las x-celdas son ajenas. ■

Corolario 4.3.10. *Suponiendo que estamos en la ejecución de LexBFS para una gráfica G con ordenamiento τ, se satisface que, cualesquiera dos vértices de la colección $\{x_j \colon x \in V_G, j > 0\}$ son distintos.*

A continuación presentamos el algoritmo **calculaLvec** para calcular las vecindades izquierdas. Vamos a describir aquí la entrada y la salida que tiene el Algoritmo 9, ésto para hacer eficiente el uso de espacio dentro del algoritmo. Recordemos también que, al representar a las gráficas como arreglos de listas, los vértices de las gráficas están representados por números, números que corresponden a los registros de los arreglos.

Entrada: Una gráfica G, de orden n y tamaño m, codificada por sus listas de adyacencias; un ordenamiento LexBFS σ de V_G codificado como un arreglo cuyo i-ésimo registro tiene a $\sigma^{-1}(i)$; tres arreglos U, R y M, tales que para todo $x \in V_G$,

$R[x] = |S^N(x)|$; si $S^N(x) = \varnothing$, la entrada $M[x]$ es nula y la lista $U[x]$ es vacía, en cambio para $S^N(x) \neq \varnothing$, $M[x]$ tiene al σ-mínimo de $S^N(x)$ y $U[x]$ tiene la lista en orden con las cardinalidades de las x-celdas.

Salida: Un arreglo N de tal forma que, para todo $x \in V_G$, $N[x]$ guarda una lista que a su vez almacena los conjuntos $N_<(\{x_j\}) \cap S(x)$ de forma σ-ordenada, para $j > 0$, lo anterior si $S^N(x)$ no es vacío, en otro caso $N[x]$ es vacía. Es decir, $N[x]$ es una lista de listas, de las listas $N_<(\{x_j\}) \cap S(x)$.

Proposición 4.3.11. *El Algoritmo 9 es correcto. Es decir, siendo N el arreglo que devuelve la ejecución $calculaLvec(G, \sigma, U, M)$, para todo $x \in V_G$, la lista $N[x]$ almacena en orden a las listas $N_<(\{x_j\}) \cap S(x)$ para $j > 0$. Al decir que las almacena en orden, nos referimos a que si $i < j$, entonces la lista $N_<(\{x_i\}) \cap S(x)$ aparece primero que la lista $N_<(\{x_j\}) \cap S(x)$ en $N[x]$. Además, para todo $j > 0$, la lista $N_<(\{x_j\}) \cap S(x)$ aparece σ-ordenada en $N[x]$.*

Demostración. En el primer `while`, construimos el arreglo s que satisface lo siguiente: para todo $i \in \{0, \ldots, n-1\}$, $s[i]$ almacena el valor $\sigma(i)$. Para convencernos de que este `while` hace de verdad esto, simplemente notemos que, estando en la iteración i-ésima del `while`, $\sigma[k]$ almacena justamente a $\sigma^{-1}(k)$, por lo que σ calculada sobre el valor de $\sigma[k]$ coincide con k, y en efecto, k es el valor que guardamos en $s[\sigma[k]]$.

En el primer `for` se realiza el cálculo de las vecindades izquierdas. Para seguir con la prueba, fijemos a $x \in V_G$ y supongamos que estamos dentro de la iteración del `for` que corresponde a x. Dentro del `for`, lo primero que hacemos es decidir si dejamos el registro $N[x]$ nulo, en caso de que $S^N(x) = \varnothing$, o bien, preparamos una lista vacía en $N[x]$ cuando $S^N(x) \neq \varnothing$, en éste último caso también apuntamos a p hacia $primero(U[x])$. Recordemos que, cuando $S^N(x) \neq \varnothing$, $primero(U[x])$ tiene a la cardinalidad de $S_1(x)$. Al `while` siguiente, entramos solo cuando $R[x] > 0$, es decir, solo cuando $S^N(x) \neq \varnothing$, en otro caso solo pasamos a la siguiente iteración del `for`. Ahora supongamos que para x sí se satisface $R[x] > 0$, y por lo tanto sí entramos al `while`. Es muy importante notar que entramos en el `while` con $k = 0$. En la iteración k-ésima del `while` calculamos $N_<(\{x_k\}) \cap S(x)$, para ver que todo funciona bien, podemos usar inducción sobre un invariante de ciclo en el `while`. El invariante de ciclo que vamos a usar es que, en la iteración k-ésima del `while`, k tiene el valor $\sum_{i<k} |S_i(x)|$, y además p apunta a la k-esima celda de $U[x]$. Este invariante se satisface para la primera iteración porque, como ya lo mencionamos antes, entramos al `while` con $k = 0$ que coincide con la suma vacía $\sum_{i<1} |S_i(x)|$, además, antes de

Algoritmo 9: calculaLvec(G, σ, U, R, M).

```
1  arreglo[n] N;
2  /* Se ordenan las listas de adyacencias respecto a ≤_σ          */
3  gráfica G' ← ordena(G, σ);
4  /* Se construye un arreglo para que, dado x, podamos calcular
       σ(x) con el registro s[x] en tiempo constante                */
5  entero k ← 0;
6  arreglo[n] s;
7  while k < n do
8  |   s[σ[k]] ← k;
9  |   k ← k + 1;
10 end
11 apuntador p;
12 entero v;
13 /* Este for explora los vértices de V_G en el orden dictado por
       G como arreglo                                               */
14 for x ∈ V_G do
15 |   if R[x] > 0 then
16 |   |   Guardar en N[x] una lista doblemente ligada y vacía;
17 |   |   Apuntar a p hacia primero(U[x]);
18 |   else
19 |   |   Establecer como nulo el registro N[x];
20 |   end
21 |   k ← 0;
22 |   while k < R[x] do
23 |   |   /* Si estamos en la l-ésima iteración de éste while, v es
       |   |      x_l                                                */
24 |   |   v ← σ[s[M[x]] + k];
25 |   |   Guardar al final de N[x] una celda con una lista vacía;
26 |   |   /* También la lista G'[v] se recorre en orden           */
27 |   |   for u ∈ G'[v], s[u] < s[v] do
28 |   |   |   if s[x] < s[u] < s[v] then
29 |   |   |   |   Agregar a u hasta el final de la lista almacenada en
   |   |   |   |    ultimo(N[x]);
30 |   |   |   end
31 |   |   end
32 |   |   /* Se suma a k la cardinalidad |S_l(x)|                */
33 |   |   k ← k + valor(ref(p));
34 |   |   Apuntar a p hacia siguiente_{U[x]}(ref(p));
35 |   end
36 end
37 return N;
```

entrar al `while` apuntamos p hacia $primero(U[x])$. Por otro lado, las instrucciones $k \leftarrow k + \text{valor}(\text{ref}(p))$ y $p \leftarrow siguiente_{U[x]}(\text{ref}(p))$ al final del bloque garantizan lo deseado para la iteración siguiente.

Fijemos también una iteración para el `while`, supongamos que estamos en la iteración l-ésima. Al principio del `while` asignamos a v el valor $\sigma[s[M[x]] + k]$, sin embargo, este último valor coincide con x_l. Para ver lo anterior, notemos que todas las x-celdas son conjuntos de vértices consecutivos respecto a σ, por lo cual, dado que $s[M[x]]$ es la imagen de x_1 bajo σ, $s[M[x]] + |S_1(x)|$ es la imagen del primer vértice fuera de $S_1(x)$; más aún, si $S_2(x)$ existe, $s[M[x]] + |S_1(x)|$ es la imagen de x_2 bajo σ y por lo tanto, $s[M[x]] + |S_1(x)| + |S_2(x)|$ es la imágen del primer vértice fuera de $S_2(x)$. Para no extender la prueba innecesariamente, omitiremos la formalización del argumento inductivo anterior, solo nos quedaremos con la conclusión que nos brinda, esta conclusión es: como $k = \sum_{i<l} |S_i(x)| < R[x]$, $S_l(x)$ existe, y por ende, $s[M[x]] + k$ es la imagen de x_l bajo σ, o sea que $\sigma[s[M[x]] + k]$ es justamente x_l. Después de definir a v, ponemos una lista vacía al final de $N[x]$ dispuesta para almacenar $N_<(\{x_l\}) \cap S(x)$. He aquí la evidencia de que $N[x]$ guarda las vecindades izquierdas en orden, esto porque al poner a $N_<(\{x_l\}) \cap S(x)$ justo al final de la lista $N[x]$, estamos haciendo que las listas $N_<(\{x_i\}) \cap S(x)$, para $i < l$, aparezcan primero en $N[x]$. Finalicemos analizando el `for` siguiente. No es difícil ver que, en este ciclo recorremos a $G'[v]$ en orden, o sea, recorremos a $N(x_l)$ en el orden de σ, y agregamos a la lista correspondiente a $N_<(\{x_l\}) \cap S(x)$ solo a los vértices u que cumplen $s[x] < s[u] < s[v]$, o lo que es lo mismo $x <_\sigma u <_\sigma x_l$. Observando que $\{u \in V_G \colon x <_\sigma u <_\sigma x_l\} \cap N(x_l)$ coincide con $N_<(\{x_l\}) \cap S(x)$, podemos concluir que, en efecto, este `for` construye, ordenada respecto a σ, a la vecindad izquierda de x_l intersectada con $S(x)$. ■

Proposición 4.3.12. *La ejecución $calculaLvec(G, \sigma, U, M)$ en el Algoritmo 9 toma tiempo $\mathcal{O}(n + m)$, donde n y m son el orden y el tamaño de G respectivamente.*

Demostración. Por la Proposición 1.9.6, la llamada $\text{ordena}(G, \sigma)$ toma tiempo $\mathcal{O}(n + m)$. En adelante, solo hay que abordar el tiempo que toman las estructuras de control, puesto que las operaciones efectuadas fuera de éstas son instrucciones de tiempo constante. En cada iteración del primer `while`, el que construye s, se hacen dos operaciones nada más, por lo que la ejecución de todo el `while` toma únicamente $2n$ unidades de tiempo. Como es usual, comenzaremos desde los ciclos más profundos en el anidamiento, o sea que vamos a comenzar por el segundo `for`. Como ya se observó en la prueba de correctud, este `for` recorre la lista correspondiente a $N(v)$; todo lo que ocurre dentro del `for` es de tiempo constante, así que la ejecución completa del `for` toma tiempo $\mathcal{O}(|N(v)|)$, o sea $\mathcal{O}(d(v))$. Las operaciones dentro del segundo

`while` que están después del `for` son dos asignaciones, en las que gastamos tiempo constante. También, antes del `for`, tenemos la asignación de v y la creación de la lista vacía al final de $N[x]$, lo que conlleva más tiempo constante. Como notamos en la prueba de correctud, en la iteración l-ésima del `while`, el valor de v es x_l, o sea que el tiempo que toma la iteración l-ésima del `while` es $\mathcal{O}(d(x_l))$. Por lo tanto, todo el `while` se ejecuta en tiempo $\mathcal{O}(\sum_{j>0} d(x_j))$. Si $x \in V_G$ es tal que $S^N(x) \neq \varnothing$, o sea, si en la iteración correspondiente a x del primer `for` sí accesamos al `while`, entonces, el tiempo que toma dicha iteración del `for` es $\mathcal{O}(\sum_{j>0} d(x_j))$. En cambio, si $S^N(x) = \varnothing$, solo usamos el tiempo en verificar el `if` y en anular el registro $N[x]$, por lo que, en este caso la iteración toma tiempo constante. De lo anterior, deducimos que la ejecución de todo el primer `for` toma tiempo $\mathcal{O}(n+\sum_{x\in V_G, R[x]>0}(\sum_{j>0} d(x_j)))$. Aquí utilizamos el Corolario 4.3.10 para afirmar que $\sum_{x\in V_G, R[x]>0}(\sum_{j>0} d(x_j)) \leq \sum_{x\in V_G} d(x) = 2m$, y así deducir que todo el primer `for` toma tiempo $\mathcal{O}(n+m)$. ■

Con lo anterior, para concretar la verificación de NSP, solo nos hace falta verificar las contenciones $N_<(\{x_j\}) \subsetneq N_<(\{x_i\})$ para cualquier $x \in V_G$ y cualesquiera $0 < i < j$. Para comprobar estas contenciones, vamos a usar el algoritmo **contencion**, este algoritmo es una rutina estándar que toma poco tiempo, cuando tenemos la suerte de que las listas que se pasan como entrada están ordenadas. El Algoritmo 10 es al que nos referimos.

Proposición 4.3.13. *El Algoritmo 10 es correcto. Es decir,* contencion(L_1, L_2) *en efecto devuelve* 0 *si todos los valores de* L_1 *están presentes en* L_2*, y de lo contrario, devuelve el mínimo índice* b *tal que el valor de la* b*-ésima celda en* L_1 *no está presente en* L_2*.*

Demostración. Si entramos en el primer `if`, ambas listas son la misma. Si entramos en el segundo `if`, los valores de la lista L_1 están en L_2 por vacuidad, y L_2 tiene valores que no están en L_1. Si en cambio entramos al tercer `if`, eso significa que L_1 no es vacía pero L_2 sí lo es, así que la primera celda de L_1 cuyo valor no está en L_2 es la primera. Ahora, si saltamos los tres bloques `if` del principio, podemos estar seguros de que ni L_1 ni L_2 son vacías, así que podemos apuntar a x hacia la primera celda de L_1, y apuntar a y hacia la primera celda de L_2. En el primer `while`, podemos establecer el siguiente invariante de ciclo: al iniciar una iteración, x está señalando la b-ésima celda de L_1. Claramente lo anterior se satisface al entrar en la primera iteración, ahora, justo antes de salir del `while`, se reasigna x a la celda siguiente y se incrementa b en 1, por lo que se satisface nuestro invariante al principio de la siguiente iteración. Como, la condición del `while` es que la celda ref(x) no sea final en L_1, además, x apunta a la primera celda de L_1 en la primera iteración del `while` y, antes de salir del `while`

Algoritmo 10: contencion(L_1, L_2).

Input: Dos listas L_1 y L_2 sin repeticiones, ambas con sus valores, si los tienen, ordenados de manera creciente respecto a algún orden arbitrario

Output: Un número b, el cual vale 0 si $L_1 \subsetneq L_2$; vale i cuando $L_1 \not\subseteq L_1$, siendo i el mínimo índice tal que el valor de la celda i-ésima de L_1 no está presente en la lista L_2; y vale -1 si L_1 y L_2 tienen los mismos valores

```
1  apuntador x, y;
2  entero b;
3  if L_1 = L_2 = ∅ then
4  |  return −1;
5  end
6  if L_1 = ∅ ∧ L_2 ≠ ∅ then
7  |  return 0;
8  end
9  if L_1 ≠ ∅ ∧ L_2 = ∅ then
10 |  return 1;
11 end
12 Apuntar x hacia primero(L_1) y apuntar a y hacia primero(L_2);
13 b ← 1;
14 while ref(x) no sea final en L_1 do
15 |  while valor(ref(x)) ≠ valor(ref(y)) do
16 |  |  if ref(y) es final en L_2 then
17 |  |  |  return b;
18 |  |  end
19 |  |  Apuntar a y hacia siguiente_{L_2}(ref(y));
20 |  end
21 |  b ← b + 1;
22 |  Apuntar a x hacia a siguiente_{L_1}(ref(x));
23 end
24 while valor(ref(x)) ≠ valor(ref(y)) do
25 |  if ref(y) es final en L_2 then
26 |  |  return b;
27 |  end
28 |  Apuntar a y hacia siguiente_{L_2}(ref(y));
29 end
30 if |L_1| < |L_2| then
31 |  return 0;
32 else
33 |  return −1;
34 end
```

redireccionamos a x hacia la celda siguiente, podemos deducir que este `while` realiza una iteración por cada celda en L_1 exceptuando la última. Supongamos que estamos en una iteración cualquiera. Lo primero que ocurre es que entramos en el segundo `while`, aquí hacemos iteraciones hasta que se satisfaga $valor(\text{ref}(x)) = valor(\text{ref}(y))$. Lo que hacemos dentro de este segundo `while`, es comparar los valores a los que apuntan x y y, en caso de que no coincidan simplemente avanzamos a y hacia la siguiente celda de L_2 hasta que lleguemos a la última, pero en caso de que sí sean los mismos, salimos del `while`. Lo anterior significa que el segundo `while` está dedicado a buscar el valor de ref(x) en la lista L_2, pero solo en los valores que siguen a partir de la celda ref(y). Sin embargo, basta con buscar al valor de ref(x) en ese segmento, justifiquemos esto último. Si x apunta a la primera celda de L_1, entonces, entrando al segundo `while` tenemos que y apunta hacia la primera celda de L_2, por lo que en el segundo `while` se busca al valor de ref(x) en toda la lista L_2. Por otro lado, si x ya no apunta a la primer celda de L_1, es porque el valor de $anterior_{L_1}(\text{ref}(x))$ ya se encontró en la lista L_2, de hecho y apunta en este preciso momento hacia la celda en L_2 que tiene dicho valor. Por lo que, no es necesario buscar al valor de ref(x) antes de ref(y), ya que todas las celdas anteriores a ref(y) van a tener un valor menor a $valor(\text{ref}(y)) < valor(\text{ref}(x))$, y por lo tanto distinto al de ref(x), pues recordemos que las listas están ordenadas. De lo anterior, podemos concluir que el segundo `while` garantiza que, devolvemos b si no encontramos en L_2 el valor de ref(x), y simplemente salimos del `while` en caso de haberlo encontrado. El último `while` hace exactamente lo mismo que el segundo, pero esta vez, buscando al valor de $ultimo(L_1)$ en la lista L_2. Por lo tanto, si llegamos al último bloque `if-else`, significa que $L_1 \subseteq L_2$, por lo que basta verificar si $|L_1| < |L_2|$ para saber si $L_1 \subsetneq L_2$. ■

Proposición 4.3.14. *La ejecución* contencion(L_1, L_2) *en el Algoritmo 10 toma tiempo* $\mathcal{O}(|L_1| + |L_2|)$*, donde* $|L_1|$ *y* $|L_2|$ *denotan a las longitudes respectivas de* L_1 *y* L_2*.*

Demostración. Ignoraremos todas las acciones realizadas fuera del primer y último `while`, incluyendo a los tres bloques `if` del principio, pues es evidente que consumen solo tiempo constante. Primero hay que desembarazarnos del bloque `if-else` que está al final, esto es fácil, pues, para verificar $|L_1| < |L_2|$ basta con recorrer ambas listas una vez, lo que toma tiempo $\mathcal{O}(|L_1| + |L_2|)$. Observemos que, dentro de cada iteración del segundo `while`, el que está anidado dentro del primero, se avanza el apuntador y a la siguiente celda de L_2, además, todo lo que se hace dentro de este `while`, el segundo, toma tiempo constante. Por otro lado, en las iteraciones del útimo `while`, también se avanza el apuntador y a la celda siguiente y, también son operaciones constantes las que se hacen dentro del último `while`. Lo anterior, aunado a que nunca se apunta a y hacia una celda anterior, nos permite establecer que, el tiempo

de ejecución que transcurre dentro del último `while`, o dentro del segundo `while` (el anidado), es $\mathcal{O}(|L_2|)$. Como ya mencionamos en la Proposición 4.3.13, el primer `while` se encarga de revisar que el valor de la celda ref(x), para todas las celdas de L_1 exceptuando la última, esté presente en la lista L_2, pero solo buscándolo de la celda ref(y) en adelante. O sea que el primer `while` hace $|L_1| - 1$ iteraciones, las cuales, omitiendo el `while` que tiene anidado, solo emplean operaciones de tiempo constante. Es decir, el tiempo en el que se ejecuta el primer `while`, sin contar el tiempo del `while` que tiene anidado, es $\mathcal{O}(|L_1|)$. Podemos contar el número total de operaciones que hacemos en la ejecución, separando el tiempo que transcurre dentro del segundo y el último `while`, del tiempo que transcurre fuera de estos, este último monto de tiempo solo es el que se tiene dentro del primer `while` pero fuera del segundo a la vez. Por lo tanto, debido a todo el análisis anterior, podemos afirmar que todo el algoritmo se ejecuta en tiempo $\mathcal{O}(|L_1| + |L_2|)$. ■

Ya estamos listos para proporcionar el algoritmo definitivo que se encarga de verificar NSP, a este algoritmo lo vamos a nombrar **NSP**, justo como la condición NSP, estamos hablando del Algoritmo 11.

Proposición 4.3.15. *Cualquier ejecución NSP(G, τ) del Algoritmo 11 toma tiempo $\mathcal{O}(n+m)$, donde n y m son el orden y el tamaño respectivos de la gráfica que pasamos como entrada.*

Demostración. El Corolario 4.3.8 y la Proposición 4.3.12, nos garantizan que las dos primeras líneas, o sea las llamadas a las subrutinas, se ejecutan en tiempo $\mathcal{O}(n + m)$. Con un vistazo al algoritmo, podemos darnos cuenta de que en todas las líneas del algoritmo que quedan por analizar, exceptuando la que verifica que ref(p) esté contenida propiamente en $siguiente_{N[x]}$(ref(p)), ejecutan una operaciíon de tiempo constante, así que solo hay que analizar el tiempo de los ciclos. Antes de poner nuestra atención en los ciclos, observemos que cuando se valida la condición del `if-else`, o sea, cuando se verifica que ref(p) esté contenida propiamente en $siguiente_{N[x]}$(ref(p)), se utiliza tiempo $\mathcal{O}(|\text{ref}(p)|+|siguiente_{N[x]}(\text{ref}(p))|)$ de acuerdo con la Proposición 4.3.14. Podemos establecer como invariante del `while`, el hecho de que al inicio de la i-ésima iteración, el apuntador p señala a la i-ésima celda de $N[x]$, mientras que t tiene el valor i. Este invariante se satisface claramente para la primera iteración, y en iteraciones posteriores se cumple porque, en la iteración previa, por continuar en la ejecución del algoritmo, debimos haber avanzado el apuntador p hacia la siguiente celda e incrementado el contador t en una unidad. Por lo tanto, el `while` hace una iteración que toma tiempo $\mathcal{O}(|\text{ref}(p)| + |siguiente_{N[x]}(\text{ref}(p))|)$, por cada celda ref($p$) de $N[x]$, exceptuando a la última. Así que, el `while` se ejecuta

Algoritmo 11: NSP(G, τ).

Input: Una gráfica G representada por sus listas de adyacencia y un ordenamiento τ de V_G

Output: El ordenamiento σ devuelto por LexBFS(G, τ). Si σ no satisface NSP, también se devuelve una pareja (x, j) tal que, x es un vértice de G cuyas celdas no cumplen NSP y $j > 0$ es el mínimo índice tal que no se cumple $N_<(\{x_{j+1}\}) \cap S(x) \subsetneq N_<(\{x_j\}) \cap S(x)$; en caso de que σ sí satisfaga NSP, se devuelve la pareja $(-1, -1)$

```
1  Ejecutar LexBFS-NSP(G, τ) y guardar la salida (σ, U, R, M);
2  Ejecutar calculaLvec(G, σ, U, R, M) y guardar la salida N;
3  apuntador p;
4  entero t;
5  for x ∈ V_G do
6      /* Revisamos la conteción propia consecutiva de las
         vecindades izquierdas de x                          */
7      if N[x] no es un registro nulo then
8          /* Según calculaLvec, N[x] no es nulo, si y solo si, N[x]
             guarda la lista no vacía de vecindades izquierdas para
             los x_j                                         */
9          Apuntar a p hacia primero(N[x]);
10         t ← 1;
11         while ref(p) no sea final en N[x] do
12             /* Esta contencion se verifica con el Algoritmo 10   */
13             if siguiente_{N[x]}(ref(p)) está contenida propiamente en ref(p) then
14                 Apuntar p a siguiente_{N[x]}(ref(p));
15                 t ← t + 1;
16             else
17                 return (σ, (x, t))
18             end
19         end
20     end
21 end
22 return (−1, −1);
```

en su totalidad en tiempo $\mathcal{O}(2\sum_{L \text{ celda en } N[x]} |L|)$, o lo que es lo mismo, en tiempo $\mathcal{O}(\sum_{j>0} |N_<(\{x_j\})|)$. Por otro lado, para cada $x \in V_G$, se realiza una iteración del

`for`. En la iteración correspondientre a x del `for`, se hace una única operación cuando $N[x]$ es nulo, mientras que cuando $N[x]$ no es nulo, se ejecutan algunas operaciones constantes junto con el `while`. Entonces, notando que $\sum_{x\in V_G}(1+\sum_{j>0}|N_<(\{x_j\})|) \leq n+\sum_{x\in V_G}\sum_{j>0} d(x_j)$, podemos usar el Corolario 4.3.10 para establecer la desigualdad $n+\sum_{x\in V_G}\sum_{j>0} d(x_j) \leq n+\sum_{x\in V_G} d(x) = n+2m$. Por lo tanto, podemos afirmar que el tiempo de ejecución ocupado por el `for` es $\mathcal{O}(n+m)$, lo que nos permite concluir que todo el algoritmo corre en tiempo $\mathcal{O}(n+m)$. ■

Proposición 4.3.16. *El Algoritmo 11 es correcto. Es decir, si NSP(G,τ) devuelve $(-1,-1)$, entonces el ordenamiento devuelto por LexBFS(G,τ) satisface NSP, mientras que NSP(G,τ) devuelve (x,j) en otro caso, donde $x\in V_G$ y j es el mínimo entero tal que no se cumple $N_<(\{x_{j+1}\})\cap S(x) \subsetneq N_<(\{x_j\})\cap S(x)$.*

Demostración. Por la explicación que dimos de como modificar LxBFS y por la Proposición 4.3.11, podemos tener la certeza de que σ es el ordenamiento devuelto por LexBFS(G,τ) y de que N almacena las vecindades izquierdas. Más específico, N es un arreglo tal que para cualquier $x\in V_G$, $N[x]$ es nulo cuando $S^N(x)=\varnothing$, y en caso de que $S^N(x)\neq\varnothing$, $N[x]$ es una lista de tal modo que la j-ésima celda de $N[x]$ es otra lista que corresponde al conjunto $N_<(\{x_j\})\cap S(x)$ ordenado respecto a σ. Para cada $x\in V_G$, el algoritmo verifica, cuando $N[x]$ no es nulo, que cualquier celda c no final de $N[x]$, cumpla la contención $\mathit{siguiente}_{N[x]}(c)\subsetneq c$, esto es equivalente a verificar que: para todo $j>0$ tal que $S_j(x)$ y $S_{j+1}(x)$ existen, $N_<(\{x_{j+1}\})\cap S(x)\subsetneq N_<(\{x_j\})\cap S(x)$. Utilizando el Corolario 4.3.6, tenemos que σ satisface NSP si y solo si, para todo $x\in V_G$ y cualesquiera $0<i<j$ tales que las x-celdas $S_i(x)$ y $S_j(x)$ existen, se cumple $N_<(\{x_j\})\cap S(x)\subsetneq N_<(\{x_i\})\cap S(x)$. Ahora, es claro entonces que σ satisface NSP si y solo si, para todo $x\in V_G$ y cualquier $j>0$ tales que las x-celdas $S_j(x)$ y $S_{j+1}(x)$ existen, se cumple $N_<(\{x_{j+1}\})\cap S(x)\subsetneq N_<(\{x_j\})\cap S(x)$, es decir, basta con fijarnos en la contención de las vecindades izquierdas de índices consecutivos. Si σ no satisface NSP, deben existir $x\in V_G$ y $j>0$ tales que no se cumple $N_<(\{x_{j+1}\})\cap S(x)\subsetneq N_<(\{x_j\})\cap S(x)$, podemos suponer sin pérdida de generalidad que x es el mínimo vértice de estos respecto al orden que tienen los vértices en la representación en G, también podemos suponer sin pérdida de generalidad que j es el mínimo índice que no satisface la contención propia. Por lo que, el algoritmo devuelve en este caso al par (x,j) por recorrer a los vértices en orden y tener la lista de las vecindades izquierdas ordenadas respecto a σ. Por otro lado, si σ cumple NSP, se verifica que, para cualesquiera $x\in V_G$ y $j>0$ tales que $S_j(x)$ y $S_{j+1}(x)$ existen, $N_<(\{x_{j+1}\})\cap S(x)\subsetneq N_<(\{x_j\})\cap S(x)$, pero, como ya observamos antes, verificar estas contenciones equivale a validar que todas las contenciones que nos permiten entrar al `if`, en cualquier iteración del `for`, se satisfacen. O sea que, la

ejecución sale del `for` incólume, y por lo tanto, terminamos regresando a la pareja $(-1,-1)$. ■

El Algoritmo 11 revisa si el ordenamiento obtenido por LexBFS(G,τ) satisface NSP, así que algunos cambios son necesarios para comprobar que σ^- satisfaga NSP, cambios que comentaremos a continuación. Para empezar, necesitamos una suerte de algoritmo **LexBFS$^-$-NSP** para que, dada G y el ordenamiento LexBFS σ que obtuvimos con LexBFS-NSP, podamos calcular: el ordenamiento LexBFS$^-$; el arreglo U tal que para todo $x \in V_G$, $U[x]$ es una lista con las cardinalidades de las celdas $\overline{S}_j(x)$; el arreglo R tal que para cada $x \in V_G$, $R[x]$ almacena el número $|\overline{S^N}(x)|$; y el arreglo M tal que para todo $x \in V_G$, $M[x]$ almacena a $\overline{x}_1 = \min \overline{S^N}(x)$, si éste mínimo existe. Construir tal algoritmo no es difícil, simplemente hay que acoplar el algoritmo LxBFS$^-$, en lugar de LxBFS, con los cambios necesarios que ya explicamos antes. Con acoplar nos referimos a que, por ejemplo, los contadores $count_x$ que se almacenan en la pila ahora tienen el valor de $|\overline{S^A}(x)| = |\overline{S}(x) \setminus (N_G(x) \cup \{x\})|$, etcétera. Ahora, no es necesario hacer un calculaLvec$^-$, de hecho, el calcular las vecindades izquierdas para σ^- podría disparar nuestra complejidad, porque eso involucraría tener que calcular los complementos de las vecindades. Para lo siguiente, vamos a denotar las vecindades izquierdas en G por $N_<$ como hemos estado trabajando, mientras que las vecindades izquierdas en $\overline{G}$ las denotamos por $\overline{N}_<$. Recordemos también que $(\leftarrow, x)_<$ denota al segmento inicial de x, es decir, a todos los elementos que son $<$-menores que x. Lo que vamos a hacer tiene base en las siguientes observaciones:

1. $\overline{N}^{\sigma^-}_<(\{\overline{x}_{j+1}\}) \cap \overline{S}(x) \subsetneq \overline{N}^{\sigma^-}_<(\{\overline{x}_j\}) \cap \overline{S}(x)$ si y solo si
   $((\leftarrow, \overline{x}_{j+1})_{<_{\sigma^-}} \setminus \overline{N}^{\sigma^-}_<(\{\overline{x}_j\})) \cap \overline{S}(x) \subsetneq ((\leftarrow, \overline{x}_{j+1})_{<_{\sigma^-}} \setminus \overline{N}^{\sigma^-}_<(\{\overline{x}_{j+1}\})) \cap \overline{S}(x)$.

2. $((\leftarrow, \overline{x}_{j+1})_{<_{\sigma^-}} \setminus \overline{N}^{\sigma^-}_<(\{\overline{x}_j\})) \cap \overline{S}(x) = (N^{\sigma^-}_<(\{\overline{x}_j\}) \cap \overline{S}(x)) \cup [\overline{x}_j, \overline{x}_{j+1})_{<_{\sigma^-}}$ y
   $((\leftarrow, \overline{x}_{j+1})_{<_{\sigma^-}} \setminus \overline{N}^{\sigma^-}_<(\{\overline{x}_{j+1}\})) \cap \overline{S}(x) = N^{\sigma^-}_<(\{\overline{x}_{j+1}\}) \cap \overline{S}(x)$.

Por un lado, el algoritmo **calculaLvec** ya nos sirve para calcular las vecindades izquierdas $N^{\sigma^-}_<$; por otro lado, el construir la lista $(N^{\sigma^-}_<(\{\overline{x}_j\}) \cap \overline{S}(x)) \cup [\overline{x}_j, \overline{x}_{j+1})_{<_{\sigma^-}}$ solo nos toma una pasada más a la lista σ^-. Después, podemos verificar la contención contraria de estas, es decir, verificar de forma similar a la del Algoritmo 11 que, para todo $x \in V_G$ y para todo $j > 0$, se cumpla $N^{\sigma^-}_<(\{\overline{x}_j\}) \cap \overline{S}(x) \subsetneq N^{\sigma^-}_<(\{\overline{x}_{j+1}\}) \cap \overline{S}(x)$. Así que, un algoritmo **NSP$^-$**(G,τ) que nos permita decidir en tiempo $\mathcal{O}(n+m)$ si un ordenamiento LexBFS$^-$ satisface NSP, es virtualmente una copia del Algoritmo 11, solo que hay que hacer la primera llamada a la subrutina LexBFS$^-$-NSP en lugar de hacerla a LexBFS-NSP, y luego verificar NSP revisando que se den las contenciones

al revés. De hecho, solo cambia una línea respecto a NSP, la línea donde se verifica la condición del bloque `if-else`. Podemos sintetizar el trabajo de páginas previas en el siguiente corolario.

Corolario 4.3.17. *Sean G una gráfica de orden n y tamaño m, y τ un ordenamiento de V_G. Podemos obtener el ordenamiento σ que resulta de LexBFS(G,τ), y además calcular una pareja (k,l) que es $(-1,-1)$ si σ satisface NSP, o bien, es tal que $k \in V_G$ y l es el mínimo entero que no cumple $N_<(\{k_{l+1}\}) \cap S(k) \subsetneq N_<(\{k_l\}) \cap S(k)$, todo esto en tiempo $\mathcal{O}(n+m)$. Mas aún, podemos hacer lo propio para el ordenamiento devuelto por LexBFS$^-(G,\sigma)$ y también en tiempo $\mathcal{O}(n+m)$.*

4.4. Cálculo de una 4-trayectoria inducida

Ya estamos en la recta final de nuestro algoritmo de reconocimiento. De hecho, utilizando el Algoritmo 11, ya podríamos dar por terminado el trabajo, pues, nos permite armar una rutina que decida en tiempo $\mathcal{O}(n+m)$ si un ordenamiento LexBFS satisface NSP y, justo después, decidir si el ordenamiento LexBFS$^-$ que se calcula usando el ordenamiento que arrojó LexBFS también satisface NSP. En virtud del Teorema 4.3.4, lo que hace la rutina descrita es decidir si estamos pasando una cográfica como entrada o no. Sin embargo, vamos por más en este trabajo, nos disponemos no solo decidir si es una cográfica o no, sino que queremos dar un certificado en la ejecución. Como es de esperar, el certificado en caso de ser cográfica es su coárbol, mientras que el certificado en caso de no ser cográfica es un P_4 inducido de la gráfica, el cual debe existir por el Teorema 3.4.1. Primero, veamos como regresar el P_4 inducido en caso de que la gráfica que se proporciona como entrada no sea una cográfica.

Lema 4.4.1. *Sean G una gráfica y τ un ordenamiento cualquiera de V_G, además, consideremos a σ el ordenamiento arrojado por LexBFS(G,τ). Supongamos que para $x \in V_G$, el índice $j > 0$ es tal que las x-celdas $(j+1)$-ésima y j-ésima existen, y no cumplen la contención propia $N_<(\{x_{j+1}\}) \cap S(x) \subsetneq N_<(\{x_j\}) \cap S(x)$. Sea $w \in S^A(x)$ tal que $wx_j \in E_G$ y $wx_{j+1} \notin E_G$, este vértice existe porque x_{j+1} y x_j están en x-celdas distintas. La existencia de w implica que $N_<(\{x_j\}) \cap S(x) \neq N_<(\{x_{j+1}\}) \cap S(x)$ y nos habilita de considerar a z como el vértice σ-máximo que está en el conjunto $(N_<(\{x_{j+1}\}) \setminus N_<(\{x_j\})) \cap S(x)$. Se satisface lo siguiente:*

1. *Si $xz \notin E_G$, el conjunto $\{x,w,z,x_{j+1}\}$ induce un P_4 en G.*
2. *Si $xz \in E_G$, pero $zw \notin E_G$, el conjunto $\{z,x,w,x_j\}$ induce un P_4 en G.*

3. *Si* $xz, zw \in E_G$, *el conjunto* $\{x_j, w, z, x_{j+1}\}$ *induce un* P_4 *en* G.

Demostración.

1. El hecho de que $w \in S^A(x)$ y $x_j \in S^N(x)$, aunado con la hipótesis de que $xz \notin E_G$, nos permite afirmar que $xz, xx_j, wx_{j+1} \notin E_G$. Ahora, es claro que $xw, zx_{j+1} \in E_G$ por las hipótesis, así que solo hay que justificar que w y z son adyacentes. Tenemos $xz \notin E_G$, $z \in S(x)$ y $z <_\sigma x_{j+1}$, por lo que debe existir $k < j+1$ tal que $z \in S_k(x)$. Veamos primero el caso $k = j$. Por la definición de una x-celda, se cumple que $S_j(x) \subseteq N(w)$ o $S_j(x) \cap N(w) = \varnothing$, sin embargo $x_j \in N(w)$, por lo que $z \in N(w)$. Ahora exploremos el caso $k < j$. Por la propiedad de mínimo que tiene j, se cumple que $N_<(\{x_j\}) \cap S(x) \subsetneq N_<(\{x_{j-1}\}) \cap S(x)$; un argumento inductivo encadenando sucesivas contenciones, el cual omitimos por su inmediatez, muestra que $N_<(\{x_j\}) \cap S(x) \subsetneq N_<(\{x_k\}) \cap S(x)$. Se sigue que $x_k \in N(w)$, así que por la definición de x-celda se sigue que $S_k(x) \subseteq N(w)$, en particular $z \in N(w)$.

2. Las hipótesis derivan de inmediato que $zx, xw, wx_j \in E_G$ y que $zw, xx_j \notin E_G$, solo hay que justificar que z y x_j no son adyacentes. Como x y $z \in S(x)$ son adyacentes, se sigue que $z \in S^A(x)$. Por lo tanto $zx_j \notin E_G$, pues $z <_\sigma x_j$ y $z \in (N_<(\{x_{j+1}\}) \setminus N_<(\{x_j\})) \cap S(x)$.

3. De las hipótesis se sigue de inmediato que $x_jw, wz, zx_{j+1} \in E_G$ y $wx_{j+1} \notin E_G$, así que debemos justificar dos cosas, a saber que x_j y x_{j+1} no son adyacentes, ni tampoco lo son x_j y z. Lo segundo se sigue enseguida de que $zx \in E_G$, pues eso implica $z \in S^A(x)$, y además tenemos que $z \in (N_<(\{x_{j+1}\}) \setminus N_<(\{x_j\})) \cap S(x)$. Por otro lado, x_j y x_{j+1} no son adyacentes porque, de serlo, ocurriría que x_j pertenece al conjunto $(N_<(\{x_{j+1}\}) \setminus N_<(\{x_j\})) \cap S(x)$, pero esto es imposible por la propiedad de máximo que posee z.

■

El resultado anterior nos permite construir la rutina **reporta-P_4** que construye un P_4, en caso de que el ordenamiento LexBFS no satisfaga NSP.

Antes de abordar la correctud y el análisis temporal del Algoritmo 12, hay que explicar un par de abusos que cometimos, cabe aclarar que en aras de la simpleza. El primero es que, la entrada del algoritmo contempla al ordenamiento σ y al arreglo N con las vecindades izquierdas. Rigurosamente hablando, esto es un problema, pues esos atributos los calculamos en la ejecución NSP y no los devolvemos como salida (output) en dicho algoritmo. Sin embargo, esto se soluciona fácil, simplemente

Algoritmo 12: reporta-$P_4(G, \sigma, x, j, x_j, x_{j+1}, Q, R)$.

Input: Una gráfica G representada por sus listas de adyacencias; un ordenamiento LexBFS σ de V_G; un vértice x de G; un índice $j > 0$ mínimo tal que, la contención $N_<(\{x_{j+1}\}) \cap S(x) \subsetneq N_<(\{x_j\}) \cap S(x)$ no se satisface; los vértices x_j y x_{j+1}; R la vecindad izquierda $N_<(\{x_{j+1}\}) \cap S(x)$; Q la vecindad izquierda $N_<(\{x_j\}) \cap S(x)$

Output: Una 4-tupla de vértices (x_1, x_2, x_3, x_4), de tal forma que $\{u_1, u_2, u_3, u_4\}$ induce un P_4 en G

```
1  /* Se ordenan las listas de adyacencias respecto a ≤_σ                    */
2  gráfica G' ← ordena(G, σ);
3  A ← N_<({x_{j+1}}) ∩ S(x), B ← N_<({x_j}) ∩ S(x);
4  /* Encontramos el σ-máximo de (N_<({x_{j+1}}) \ N_<({x_j})) ∩ S(x)       */
5  Guardar en L_1 la reversa de A;
6  Guardar en L_2 la reversa de B;
7  Guardar en un entero l la salida de contencion(L_1, L_2);
8  Guardar en z el valor de la l-ésima celda de L_1;
9  /* Tomamos un vértice en S^A(x) adyacente a x_j, pero no a x_{j+1}       */
10 Guardar en un entero k la salida de contencion(B, A);
11 Guardar en w el valor de la k-ésima celda de B;
12 if x no está en G[z] then
13 |  return (x, w, z, x_{j+1});
14 end
15 if z no está en G[w] then
16 |  return (z, x, w, x_j);
17 else
18 |  return (x_j, w, z, x_{j+1});
19 end
```

modificando el Algoritmo 11 para que también los regrese y podamos seguir trabajando con ellos después de la llamada a **NSP**. El segundo abuso está más o menos relacionado con el primero, o mejor dicho, relacionado también con la entrada que recibe el algoritmo. Nuestra rutina utiliza a los vértices x_j y x_{j+1} como entrada, pero no devolvemos, ni siquiera calculamos, estos vértices en **NSP**. Como calcularlos desde cero implica trabajo en balde que se repite, solo vamos a explicar como lo podemos hacer desde **NSP**. No es nada complicado, lo podemos hacer en la rutina c**alculaLvec**. Al estar ejecutando **calculaLvec**, siempre tenemos acceso a v_l cuando

calculamos la vecindad izquierda $N_<(\{v_l\}) \cap S(v)$. Así, podemos almacenar todos los vértices v_l, variando v sobre todos los vértices y l sobre todas los índices tales que la v-celda $S_l(v)$ existe. Esto no afecta nuestra complejidad temporal en demasía porque, de acuerdo con el Lema 4.3.7, tenemos que $|\{v_l \colon v \in V_G, l > 0\}| \leq |V_G|$. Por lo tanto, podemos pasar todo el registro de vértices v_l a lo que resta de la ejecución de **NSP**, y, en caso de que hallemos x y $j > 0$ tales que rompen la condición NSP, entonces también pasamos en la salida los vértices x_j y x_{j+1}, además de x y j claro está.

Proposición 4.4.2. *El Algoritmo 12 es correcto, es decir, siempre devuelve un cuarteto de vértices que inducen un P_4 en la gráfica de entrada. Además, su ejecución toma tiempo $\mathcal{O}(n+m)$.*

Demostración. Primero observemos que z en efecto es el σ-máximo del conjunto $(N_<(\{x_{j+1}\}) \setminus N_<(\{x_j\})) \cap S(x)$. Al principio del algoritmo establecemos $A = N_<(\{x_{j+1}\}) \cap S(x)$ y $B = N_<(\{x_j\}) \cap S(x)$. Al tomar L_1 y L_2 como las reversas respectivas de A y de B, estamos reordenando las vecindades izquierdas con el orden inverso de $\leq_\sigma$. La llamada a contencion(L_1, L_2) nos arroja el índice l correspondiente a la celda en L_1 que tiene el menor valor y que no está presente en L_2, o sea que, todas las celdas en L_1 anteriores a la de dicho índice sí tienen su valor presente en B. Como L_1 es A con el orden invertido y B tiene los mismos valores que L_2, podemos afirmar que en A, todas las celdas posteriores a la celda en cuestión, sí tienen su valor presente en L_2, de donde se sigue que z es el vértice σ-máximo en $(N_<(\{x_{j+1}\}) \setminus N_<(\{x_j\})) \cap S(x)$. Algo similar pero más sencillo ocurre con w. Al ser w el valor de la celda arrojada por la llamada contencion(B, A), se deduce que w es el vértice σ-mínimo en $B \setminus A$, o sea en $S(x) \cap (N_<(\{x_j\}) \setminus N_<(\{x_{j+1}\}))$. Una vez que garantizamos la identidad de z y w, la correctud del algoritmo se sigue directamente del Lema 4.4.1.

Abordemos ahora la complejidad temporal del algoritmo. La instrucción de ejecutar ordena(G, σ), sabemos que toma tiempo $\mathcal{O}(n+m)$ por la Proposición 1.9.6. Sabemos también, por la Proposición 4.3.14, que la llamada a contencion(L_1, L_2) y la llamada a contencion(B, A), toman ambas tiempo $\mathcal{O}(|A| + |B|)$, o sea tiempo $\mathcal{O}(d(x_j) + d(x_{j+1}))$. Cuando definimos A y B, basta con referenciar A y B a las listas que queremos, porque en relidad nunca las modificamos, si se desea, podemos hacer una copia de las listas que representan a las vecindades izquierdas; en cualquier caso, realizar estas acciones toma tiempo $\mathcal{O}(d(x_j) + d(x_{j+1}))$. Construir las reversas de A y B lo podemos hacer recorriendo las listas al revés, así que también gastamos tiempo $\mathcal{O}(d(x_j) + d(x_{j+1}))$. El buscar la l-ésima celda de L_1 para asignar valor a z lo hacemos en $l \leq |A| \leq d(x_{j+1})$ pasos, del mismo modo, para darle valor a w

tomamos $k \leq |B| \leq d(x_j)$ unidades de tiempo. Para verificar la condición del bloque `if`, hay que buscar un vértice en $N(z)$, mientras que para revisar la condición del bloque `if-else`, necesitamos buscar un vértice en $N(w)$, todo esto toma tiempo $\mathcal{O}(d(z) + d(w))$. Todas las otras instrucciones que no mencionamos nos quitan una cantidad de tiempo constante. Por lo tanto, recordando la identidad $m = \sum_{x \in V_G} d(x)$, concluimos que el algoritmo corre en tiempo $\mathcal{O}(n + m)$. ■

Vamos con nuestro comentario usual respecto al algoritmo anterior y el ordenamiento LexBFS$^-$. No podemos utilizar el Algoritmo 12 para el ordenamiento LexBFS$^-$ como lo usamos para el ordenamiento LexBFS, pues lo que tenemos a la mano son las vecindades izquierdas $N_<$ en G, pero necesitamos las vecindades izquierdas $\overline{N}_<$ en $\overline{G}$. Sin embargo, solo necesitamos los complementos de las vecindades de cuatro vértices, así que las podemos calcular sin incrementar la complejidad temporal. Para calcular los complementos de las vecindades usamos el Algoritmo 13.

Proposición 4.4.3. *La ejecución complemento(L_1, L_2) en el Algoritmo 13 devuelve la lista L, la cual tiene exactamente los valores de L_2 que no están presentes en L_1.*

Demostración. Algo que es inmediato del algoritmo, es que durante toda la ejecución, el apuntador p referencia una celda en L_1, mientras que el apuntador q referencia una celda en L_2. Establezcamos un invariante de ciclo para el primer `while` afirmando que, al entrar en dicho ciclo, el valor que guarda la celda referenciada por p, es el de la celda que apunta q, o bien, es el valor de una celda de L_2 que está después de la celda apuntada por q. Esto se satisface al entrar en la primera iteración porque $L_1 \subseteq L_2$ y p apunta a la primera celda de L_1. Supongamos que se da una iteración posterior. En esa iteración posterior, la condición para salir del `while` que está dentro del que estamos trabajando, fue que los valores de ref(p) y ref(q) coincidieran, además, una vez que salimos del `while` interno avanzamos los apuntadores p y q. Dado que las listas están ordenadas crecientemente, digamos con el orden $<$, al entrar en la nueva iteración del `while`, vamos a tener que $valor(anterior_{L_1}(\text{ref}(q))) = valor(anterior_{L_1}(\text{ref}(p))) < valor(\text{ref}(p))$. Lo anterior implica que, como el valor de ref(p) debe estar presente en L_2 forzosamente, pero no puede estar en las celdas anteriores a la que apunta q por el orden, la celda de L_2 que tiene el mismo valor que ref(p) debe ser ref(q) o debe estar después de ref(p). Con lo anterior, podemos ver que lo que realizan los dos primero `while` en conjunto es lo siguiente. Por cada valor v que este en una celda x no final en L_1, se recorre L_2 desde la celda que tiene al valor v hasta la celda que tiene el valor de $siguiente_{L_1}(x)$, insertando al final de L todos los valores que encuentre en medio. Claramente todos estos valores deben estar en $L_2 \setminus L_1$, por el orden de L_1. El penúltimo `while` hace

Algoritmo 13: complemento(L_1, L_2).

Input: Dos listas L_1 y L_2 sin repeticiones que tienen sus valores ordenados de manera creciente respecto a un orden arbitrario, el mismo en ambas listas; además se satisface que todos los valores de L_1 están presentes en L_2

Output: Una lista L, posiblemente vacía, que tiene exactamente los valores de L_2 que no están presentes en L_1; además, todos los valores en L están ordenados respecto al mismo orden que tienen L_1 y L_2

```
1  apuntador p, q;
2  Apuntar p hacia primero(L1);
3  Apuntar q hacia primero(L2);
4  lista L → ∅;
5  while ref(p) no es final en L1 do
6  |  /* Se introducen a L los valores de L2 hasta llegar al valor
   |     que apunta p en L1                                      */
7  |  while valor(ref(p)) ≠ valor(ref(q)) do
8  |  |  Insertar al final de L, una celda con el valor que tiene ref(q);
9  |  |  Apuntar a q hacia siguiente_{L2}(ref(q));
10 |  end
11 |  Apuntar a p hacia siguiente_{L1}(ref(p));
12 |  Apuntar a q hacia siguiente_{L2}(ref(q));
13 end
14 while valor(ref(p)) ≠ valor(ref(q)) do
15 |  Insertar al final de L, una celda con el valor que tiene ref(q);
16 |  Apuntar a q hacia siguiente_{L2}(ref(q));
17 end
18 while ref(q) no es final en L2 do
19 |  Apuntar a q hacia siguiente_{L2}(ref(q));
20 |  Insertar al final de L, una celda con el valor que tiene ref(q);
21 end
22 return L;
```

exactamente lo mismo, pero en el caso de la última celda de L_1. La labor del último `while` es terminar de introducir en L todos los valores que quedan en L_2 una vez que ya dejamos todos los valores presentes en L_1 atrás; esto es necesario porque, al ser valores mayores al máximo de los valores en L_1, desde luego están en $L_2 \setminus L_1$. Para

finalizar, observemos que L queda ordenada, esto ocurre porque las inserciones las hacemos al final de L y la lista L_2 la recorremos en orden. ■

Proposición 4.4.4. *El tiempo que toma la ejeccución complemento*(L_1, L_2) *en el Algoritmo 13, es* $\mathcal{O}(|L_1| + |L_2|)$.

Demostración. Apoyandonos fuertemenete en el análisis de la Proposición 4.4.3, vamos a hacer un argumento distinto para no meternos con los ciclos por separado. Como dijimos en la prueba de correctud, el algoritmo va introducinedo en L los valores que se encuentran en L_2, y además satisfacen con encontrarse entre dos valores que son consecutivos en L_1. O sea que, por cada celda de L_2 hacemos un número de operaciones constante, las cuales son básicamente: avanzar el apuntador q, añadir un valor al final de L y, verificar la condición del `while` donde nos encontremos. Por otro lado, por cada celda de L_1 también hacemos una cantidad constante de instrucciones, que son: avanzar el apuntador p y verificar la condición de algún `while`. ■

Ahora ya podemos dar la versión **reporta-**P_4^- del algoritmo para reportar los vértices de un P_4 inducido en el caso de LexBFS$^-$. La correctud del Algoritmo 14 se sigue del Lema 4.4.1 y la Proposición 4.4.2, mientras que la complejidad temporal es por las Proposiciones 4.4.2 y 4.4.4, pues el flujo en el algoritmo es casi idéntico al del Algoritmo 12, por lo demás, los únicos cambios los subrayamos.

4.5. Construcción del coárbol

Ahora vamos a explorar el caso en el que la gráfica de entrada sí es una cográfica. Como ya habíamos anticipado, lo que haremos será construir su coárbol y regresarlo como certificado. Para eso necesitamos introducir subárboles especiales del coárbol de una cográfica, asunto que trataremos a continuación. Sean G una cográfica con coárbol T y x un vértice de G. Para cada $i > 0$ tal que hay al menos i nodos con etiqueta 0 en la rama P_x, definimos el **i-ésimo nodo** 0 de la rama P_x como, justamente el i-ésimo nodo con etiqueta 0 que aparace en la xr-trayectoria P_x en T, donde r es el nodo raíz de T. De manera análoga definimos el **i-ésimo nodo** 1 de la rama P_x, cuando hay por lo menos i nodos con etiqueta 1 en P_x, desde luego. Al i-ésimo nodo 0 lo denotamos por 0_i^x, mientras que la notación que damos al i-ésimo nodo 1 es 1_i^x. Para cualquier nodo 0_i^x, definimos el subárbol T_{0i}^x de T como el árbol obtenido al quitar de $T_{0_i^x}$ todos los nodos que tengan un ancestro en la rama P_x distinto a 0_i^x, es decir, si tomamos al subárbol de T que está enraizado en 0_i^x y le quitamos todos los nodos que tienen que subir a un nodo en la rama P_x para poder

Algoritmo 14: reporta-$P_4^-(G, \sigma^-, x, j, \overline{x}_j, \overline{x}_{j+1}, Q, R)$.

Input: Una gráfica G representada por sus listas de adyacencias; un ordenamiento LexBFS$^-$ σ^- de V_G; un vértice x de G; un índice $j > 0$ mínimo tal que, la contención $\overline{N}_<(\{\overline{x}_{j+1}\}) \cap \overline{S}(x) \subsetneq \overline{N}_<(\{\overline{x}_j\}) \cap \overline{S}(x)$ no se satisface; los vértices $\overline{x}_j$ y $\overline{x}_{j+1}$; Q la vecindad izquierda $N_<(\{\overline{x}_j\}) \cap \overline{S}(x)$; R la vecindad izquierda $N_<(\{\overline{x}_{j+1}\}) \cap \overline{S}(x)$

Output: Una 4-tupla de vértices (x_1, x_2, x_3, x_4), de tal forma que $\{u_1, u_2, u_3, u_4\}$ induce un P_4 en $\overline{G}$

1 /* Se ordenan las listas de adyacencias respecto a $\leq_{\sigma^-}$ */
2 gráfica $G' \leftarrow$ ordena(G, σ^-);
3 $A \leftarrow \underline{\text{complemento}}(N_<(\{\overline{x}_{j+1}\}) \cap \overline{S}(x), \sigma^-) \cap [x, \overline{x}_{j+1})_{<_{\sigma^-}}$;
4 $B \leftarrow \underline{\text{complemento}}(N_<(\{\overline{x}_j\}) \cap \overline{S}(x), \sigma^-) \cap [x, \overline{x}_{j+1})_{<_{\sigma^-}}$;
5 /* Encontramos el σ^--máximo de $(\overline{N}_<(\{\overline{x}_{j+1}\}) \setminus \overline{N}_<(\{\overline{x}_j\})) \cap \overline{S}(x)$ */
6 Guardar en L_1 la reversa de A;
7 Guardar en L_2 la reversa de B;
8 Guardar en un entero l la salida de contencion(L_1, L_2);
9 Guardar en z el valor de la l-ésima celda de L_1;
10 /* Tomamos un vértice en $\overline{S^A}(x)$ adyacente en $\overline{G}$ a $\overline{x}_j$, pero no a $\overline{x}_{j+1}$ */
11 Guardar en un entero k la salida de contencion(B, A);
12 Guardar en w el valor de la k-ésima celda de B;
13 **if** x no está en $\underline{\text{complemento}}(G[z], \sigma^-)$ **then**
14 | **return** $(x, w, z, \overline{x}_{j+1})$;
15 **end**
16 **if** z no está en $\underline{\text{complemento}}(G[w], \sigma^-)$ **then**
17 | **return** $(z, x, w, \overline{x}_j)$;
18 **else**
19 | **return** $(\overline{x}_j, w, z, \overline{x}_{j+1})$;
20 **end**

llegar a 0_i^x, lo que obtenemos es justamente el árbol T_{0i}^x. Estos subárboles, variando el índice i, se llaman **árboles** 0 **enraizados en la rama** P_x . De manera similar definimos los **árboles** 1 **enraizados en la rama** P_x y les damos la notación T_{1i}^x cuando estamos considerando al nodo 1_i^x. En la Figura 4.4, tenemos el coárbol de alguna cográfica donde el nodo raíz tiene etiqueta 1. Además, en esa misma figura podemos ver los nodos del subárbol T_{01}^x con letras mayúsculas, mientras que los nodos del subárbol T_{11}^y están etiquetados con letras griegas.

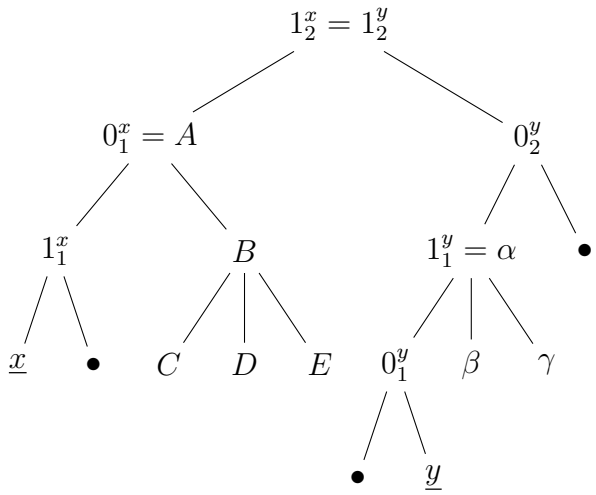


Figura 4.4: Subárboles T_{01}^x y T_{11}^y.

No necesitamos toda la potencia del siguiente resultado, sin embargo plantea algo muy interesante y no tan intuitivo.

Proposición 4.5.1. *Sean G una gráfica, M un módulo de G y τ un ordenamiento de V_G. Consideremos a σ el ordenamiento devuelto por la ejecución LexBFS(G, τ) y a η el ordenamiento devuelto por la ejecución LexBFS$(G[M], \tau|M)$. Para todo $x \in M$, los conjuntos $S(x)$, $S^A(x)$ y $S^N(x)$, todos tomados respecto a la ejecución LexBFS$(G[M], \tau|_M)$, coinciden con sus homólogos respecto a la ejecución LexBFS(G, τ) pero intersectados con M. Además, los órdenes $\leq_{\sigma|M}$ y $\leq_\eta$ son el mismo.*

Demostración. Para no generar confusión, la notación $S(v)$, $S^A(v)$ y $S^N(v)$ será respecto a la ejecución LexBFS(G, τ), mientras que la notación $\mathcal{S}(v)$, $\mathcal{S}^A(v)$ y $\mathcal{S}^N(v)$ la manejaremos para la ejecución LexBFS$(G[M], \tau|M)$. Sea n la cardinalidad de M. Demostramos por inducción sobre $k \in \{0, \ldots, n-1\}$ que el vértice $\eta^{-1}(k)$ satisface

lo que postulamos. Denotemos por x al σ-mínimo de M. Observemos que para todo $y <_\sigma x$, se tiene $M \subseteq N(y)$ o $M \subseteq (V_G \setminus N(y))$ por la definición de módulo. Por lo tanto $M \subseteq S(x)$, de donde se sigue que $x = \text{mín}_\tau M$ y por ende x también es el η-mínimo de M, o sea que $x = \eta^{-1}(0)$. También, de $M \subseteq S(x)$ obtenemos directamente las siguientes igualdades: $\mathcal{S}(x) = M = S(x) \cap M$, $\mathcal{S}^A(x) = S^A(x) \cap M$ y $\mathcal{S}^N(x) = S^N(x) \cap M$.

Nuestra hipótesis inductiva para $k \in \{1, \ldots, n-1\}$ es que los primeros k elementos del orden $\leq_{\sigma|_M}$ coinciden con los primeros k elementos del orden $\leq_\eta$ y están ordenados de la misma forma, además de que para cualquier $i < k$, se satisface $\mathcal{S}(\eta^{-1}(i)) = S(\eta^{-1}(i)) \cap M$, $\mathcal{S}^A(\eta^{-1}(i)) = S^A(\eta^{-1}(i)) \cap M$ y $\mathcal{S}^N(\eta^{-1}(i)) = S^N(\eta^{-1}(i)) \cap M$. Denotemos por x a $\eta^{-1}(k-1)$ y por z al vértice sucesor de x en M según el orden $\sigma|M$, podemos tomar dicho z porque de acuerdo con nuestra hipótesis inductiva solo hay $k < n$ vértices menores o iguales a x con el orden $\sigma|M$, que de hecho son los k vértices menores o iguales a x con el orden η. Mostremos que z es el sucesor de x en M respecto al orden $\leq_\eta$, es decir, que $z = \eta^{-1}(k)$. Para demostrar lo anterior, vamos a proceder por contradicción suponiendo que $u = \eta^{-1}(k)$ y que $x <_\eta u <_\eta z$. Se cumple que $u <_\tau z$ si $z \in \mathcal{S}(u)$, o bien, si $z \notin \mathcal{S}(u)$ debe existir $w <_\eta u$ tal que $wu \in E_{G[M]}$ y $uz \notin E_{G[M]}$. No es posible que $z \notin \mathcal{S}(u)$, porque para cualquier $w <_\eta u$, la hipótesis inductiva aplica para w, pues $w \leq_\sigma x$, así que no es posible que $wu \in E_G$ y $wz \notin E_G$ estando u en M, pues esto implicaría $x <_\sigma u <_\sigma z$, pero z es sucesor de x en M respecto a σ. Ahora, tampoco es posible que $z \in \mathcal{S}(u)$, o sea que $u <_\tau z$, ya que eso significaría que para todo $w \in M$ satisfaciendo $w <_\eta u$, se cumple $u, z \in N(w)$ o $u, z \notin N(w)$, o sea que w no distingue a u y z. Pero, por hipótesis inductiva y utilizando la definición de módulo, tendríamos también que ningún $w <_\sigma z$ distingue a u y z en G, por lo que $u \in S(z)$, de donde tenemos el absurdo de que $z <_\tau u$ por la elección de pivote en $S(z)$. Encontrando una contradicción en cada vía, concluimos que $z = \eta^{-1}(k)$.

De nuevo, utilizamos el mismo argumento que en el caso base, o sea, que al ser M un módulo, ningún vértice y, con $x <_\sigma y <_\sigma z$, puede distinguir a vértices en M. Si $y >_\eta z$ y $y \notin \mathcal{S}(z)$, significa que algún $u \in M$ con $u <_\eta z$, satisface $uz \in E_{G[M]}$ pero $uy \notin E_{G[M]}$, por hipótesis inductiva $u <_\sigma z$ por lo que $y \notin S(z) \cap M$. Ahora, si $y >_\sigma z$ y $y \notin S(z)$, significia que algún $u \in V_G$ con $u <_\sigma z$, cumple $uz \in E_G$ pero $uy \notin E_G$, pero los únicos vértices que distinguen elementos de M son vértices en M, por lo que $u \in M$. Usando la hipótesis inductiva podemos deducir que $u <_\eta z$ y por lo tanto $y \notin \mathcal{S}(z)$. De lo anterior ya podemos concluir que $\mathcal{S}(z) = S(z) \cap M$, de donde se siguen también las igualdades $\mathcal{S}^A(z) = S^A(z) \cap M$ y $\mathcal{S}^N(z) = S^N(z) \cap M$. ■

Los nodos y árboles especiales que definimos anteriormente son muy útiles, pues

tienen una estrecha relación con las celdas de los barridos LexBFS y LexBFS$^-$. A este respecto tenemos los siguientes resultados.

Proposición 4.5.2. *Sean G una cográfica con coárbol T y σ un ordenamiento LexBFS de V_G. Consideremos $x \in V_G$ tal que $\sigma(x) = 0$, es decir, al primer vértice de σ. Para cualquier $i > 0$, la x-celda $S_i(x)$ existe, si y solo si el nodo 0_i^x existe. Además, de cumplirse cualquiera de las condiciones anteriores, los conjuntos $S_i(x)$ y $H(T_{0i}^x)$ coinciden.*

Demostración. Vamos a utilizar mucho la Proposición 3.1.5 en la demostración, por lo que es recomendable revisarla. Sea cualquier $i > 0$ tal que la x-celda $S_i(x)$ existe y tomemos $u, v \in S_i(x)$. Como $u, v \in S^N(x)$, las etiquetas de los nodos $mca_T(x, u)$ y $mca_T(x, v)$ son ambas 0, así que para algunos $k, l > 0$ se cumple que $mca_T(x, u) = 0_k^x$ y $mca_T(x, v) = 0_l^x$. Nótese que $u \in H(T_{0k}^x)$ y $v \in H(T_{0l}^x)$, vamos a mostrar que $k = l$ procediendo por contradicción. Si ocurriera $k < l$, debería haber un nodo η en la rama P_x que tiene etiqueta 1 y también cumple $0_k^x <_T \eta <_T 0_l^x$. Tomando cualquier $z \in V_G$ tal que $mca_T(x, z) = \eta$, vamos a tener por la Proposición 2.2.4, que $mca(u, z) = \eta$ y $mca(v, z) = 0_l^x$. De lo anterior ya obtenemos la contradicción de que $z \in S^A(x)$, $zu \in E_G$, pero $zv \notin E_G$, esto es una contradicción porque u y v están en la misma x-celda. Podemos formular un argumento análogo para ver la imposibilidad de que $l > k$, de donde concluimos que $k = l$. Lo que hemos mostrado es que, para algún $k > 0$ se satisface $S_i(x) \subseteq H(T_{0k}^x)$. Ahora tomemos cualquier $j > 0$ tal que el nodo 0_j^x existe y tomemos también $u, v \in H(T_{0j}^x)$ arbitrarios. Consideremos $z \in S^A(x)$, en caso de que $S^A(x) \neq \varnothing$. La etiqueta del nodo $mca_T(x, z)$ es 1, por lo que $z \notin H(T_{0j}^x)$, de donde se sigue que $mca(u, z) \geq_T 0_j^x$ y $mca(v, z) \geq_T 0_j^x$. Se sigue que, al ser 0_j^x un ancestro común de u y de v, se cumple que $mca(u, z) = mca(v, z)$; es decir, z no distingue a u y v. Como z fue arbitrario, podemos establecer la existencia de $k > 0$ tal que $u, v \in S_k(x)$, más aún, dicho k satisface $H(T_{0j}^x) \subseteq S_k(x)$ porque u y v también fueron arbitrarios. De todo lo anterior, tenemos que toda la x-celda $S_i(x)$ está contenida en el conjunto de hojas de algún árbol T_{0l}^x, sin tener necesariamente $i = l$, y que las hojas de todo árbol T_{0j}^x están contenidas en alguna celda $S_k(x)$, sin tener tampoco $j = k$; lo que falta por hacer es verificar que esos índices en efecto coinciden. El hecho de que para $i \neq j$, los conjuntos $H(T_{0i}^x)$ y $H(T_{0j}^x)$ son ajenos por definición, nos permite hacer la siguiente observación: si $k > 0$ es tal que $H(T_{0i}^x) \subseteq S_k(x)$, podemos tomar $j > 0$ tal que $S_k(x) \subseteq H(T_{0j}^x)$, así que $H(T_{0i}^x) \subseteq H(T_{0j}^x)$, de donde se sigue que $i = j$ y por lo tanto $H(T_{0i}^x) = S_k(x)$. Otra afirmación análoga la podemos hacer para las x-celdas basandonos en que también son ajenas cuando los índices son distintos. Es decir, tenemos que para $i > 0$, si $k > 0$ es tal que $S_i(x) \subseteq H(T_{0k}^x)$, entonces $S_i(x) = H(T_{0k}^x)$.

Supongamos que $0 < k < l$ son tales que los nodos 0_k^x y 0_l^x existen, y consideremos además $u \in H(T_{0k}^x)$ y $v \in H(T_{0l}^x)$. Dado cualquier $z \in S^A(x)$, sin importar que se cumpla $mca(z,x) \leq_T 0_k^x$ o $mca(z,x) >_T 0_k^x$, vamos a tener $mca(z,u) = mca(z,0_k^x)$; de hecho, $mca(z,0_k^x) = 0_k^x = mca(z,u)$ cuando $mca(z,x) \leq_T 0_k^x$, y $mca(z,0_k^x) = mca(x,z) = mca(z,u)$ en otro caso. Lo mismo ocurre para v, es decir, se satisface que $mca(z,v) = mca(z,0_l^x) = 0_l^x$ cuando $mca(z,x) \leq_T 0_l^x$, y $mca(z,v) = mca(z,0_l^x) = mca(z,x)$ en el otro caso. Lo anterior significa que $mca(z,u)$ tiene etiqueta 0, si y solo si $mca(z,x) \leq_T 0_k^x$, simétricamente, $mca(z,v)$ tiene etiqueta 0, si y solo si $mca(z,x) \leq_T 0_l^x$. Así que, si $|N(z) \cap \{u,v\}| = 1$, se debe tener $0_k^x <_T mca(z,x) <_T 0_l^x$, implicando $zu \in E_G$ y $zv \notin E_G$. Esto prueba que, si los índices $i > 0$ y $j > 0$ son tales que $u \in S_i(x)$ y $v \in S_j(x)$, entonces $i < j$.

Concretemos la demostración principal con ayuda de todo lo anterior. Para cada $i > 0$ tal que la x-celda $S_i(x)$ exista, podemos definir $k_i > 0$ como el único índice tal que $S_i(x) = H(T_{0k_i}^x)$. No puede haber un índice $l > 0$ fuera del conjunto $\{k_i : i > 0, S_i(x) \text{ existe}\}$ porque de haberlo, podemos tomar $j > 0$ tal que $H(T_{0l}^x) = S_j(x) = H(T_{0k_j}^x)$ contradiciendo que los conjuntos $H(T_{0l}^x)$ y $H(T_{0k_j}^x)$ son ajenos. Tampoco puede haber más x-celdas que árboles 0 enraizados en la rama P_x, porque existirían dos índices $i \neq j$ tales que $k_i = k_j$, contradiciendo que los conjuntos $S_i(x)$ y $S_j(x)$ son ajenos. Esto basta para mostrar que la i-ésima x-celda existe, si y solamente si, el árbol T_{0i}^x existe. Ahora, si existiera $i > 0$ tal que $i \neq k_i$, podemos suponer sin pérdida de generalidad que i es mínimo con esa propiedad. Así pues, i es menor que k_i, por lo que existe $j > i$ de tal modo que $k_j = i < k_i$, todo esto concretamente significa que $i < j$, $S_i(x) = H(T_{0k_i}^x)$ y $S_j(x) = H(T_{0k_j}^x)$ con $k_j < k_i$, contradiciendo lo demostrado en el párrafo anterior. Por lo tanto, para todo $i > 0$ se cumple $i = k_i$, es decir, $S_i(x) = H(T_{0i}^x)$. ■

Proposición 4.5.3. *Sean G una cográfica con coárbol T y σ^- un ordenamiento LexBFS$^-$ de V_G. Consideremos $x \in V_G$ tal que $\sigma^-(x) = 0$, es decir, al primer vértice de σ^-. Para cualquier $i > 0$, la x-celda $\overline{S}_i(x)$ existe, si y solo si el nodo 1_i^x existe. Además, de cumplirse cualquiera de las condiciones anteriores, los conjuntos $\overline{S}_i(x)$ y $H(T_{1i}^x)$ coinciden.*

Demostración. Sabemos por la Proposición 3.1.6 que $\overline{G}$ también es una cográfica, y su coárbol F es el árbol T con las etiquetas intercambiadas. Como σ^- es un barrido LexBFS de la gráfica $\overline{G}$, podemos usar la Proposición 4.5.2 para garantizar que, para todo $i > 0$ la x-celda $\overline{S}_i(x)$ existe, si y solo si, el nodo 0_i^x existe en F, incluso que los conjuntos $\overline{S}_i(x)$ y $H(F_{0i}^x)$ coinciden en caso de existir. Pero el conjunto $H(F_{0i}^x)$ es justamente $H(T_{1i}^x)$. ■

Consideremos una cográfica G con coárbol T. Si $t \in \{0,1\}$ e $i > 0$ es tal que el nodo t_i^x existe, el conjunto de hojas del árbol T_{ti}^x lo podemos ver como $\bigcup_{n\in A} H(T_n)$, donde A es el conjunto de hijos de t_i^x en T que no están en la rama P_x. De modo que, las Proposiciones 3.2.2, 4.5.2 y 4.5.3 nos permiten establecer el siguiente corolario.

Corolario 4.5.4. *Sea G una cográfica, y sean σ, σ^- ordenamientos LexBFS y LexBFS$^-$ de V_G. Supongamos que x es el primer vértice de los ordenamientos σ y σ^-, es decir, que $\sigma(x) = 0 = \sigma^-(x)$. Para todo $i > 0$, si existen las x-celdas $S_i^\sigma(x)$ y $\overline{S}_i^{\sigma^-}(x)$, deben ser módulos de G.*

El coárbol de una cográfica se puede construir utilizando los subárboles enraizados en la rama que correspone al primer vértice de un ordenamiento LexBFS o LexBFS$^-$.Supongamos que σ es un ordenamiento LexBFS y que $x = \sigma^{-1}(0)$. Entonces, según los resultados anteriores, podemos construir el coárbol utilizando las x-celdas respecto a σ y σ^-. Sin embargo, algo que no podemos saber de los resultados anteriores, es la etiqueta del primer nodo interno que aparece en la rama P_x, o sea la etiqueta del padre de x. Vamos a ver en lo subsecuente ciertos enunciados, uno de los cuales trata con el problema anterior. En particular, el resultado que está a continuación, se sigue inmediatamente de la Proposición 4.5.1, pero creemos que es más instructivo demostrarlo de la forma en la que lo hacemos a continuación.

Proposición 4.5.5. *Sean G una gráfica y M un módulo de G. Consideremos σ y σ^- ordenamientos LexBFS y LexBFS$^-$ de V_G respectivamente. El ordenamiento dado por $\leq_{\sigma|_M}$ es un ordenamiento LexBFS de $G[M]$ y el ordenamiento dado por $\leq_{\sigma^-|_M}$ es un ordenamiento LexBFS$^-$ de $G[M]$.*

Demostración. Denotemos a $\sigma|_M$ simplemente como γ. Por el Teorema 4.1.1 basta con demostrar que γ satisface la condición de los cuatro puntos en $G[M]$. Sean $x, y, z \in M$ tales que $x <_\gamma y <_\gamma z$, y satisfacen además $xy \notin E_G$ y $xz \in E_G$. Sabemos que se verifica la condición de los cuatro puntos para σ, por lo que, existe $w \in V_G$ tal que $w <_\sigma x$, $wy \in E_G$, pero $wz \notin E_G$. Por ser M módulo, se cumple $N(y)\backslash M = N(z)\backslash M$, así que $w \in M$. Por lo tanto $w <_\gamma x$, $wy \in E_{G[M]}$ y $wz \notin E_{G[M]}$, que es lo que queríamos mostrar.

Denotemos a $\sigma^-|_M$ como α. Recordemos que σ^- es un ordenamiento LexBFS$^-$ de $V_{\overline{G}}$, así que aplicando lo anterior y notando que M también es un módulo de $\overline{G}$, se sigue que α es un ordenamiento LexBFS de $\overline{G}[M]$. Ahora, también es fácil convencernos de que, si obtenemos a α a partir de una ejecución LexBFS$(\overline{G}[M], \tau)$, entonces, por la definición de LexBFS$^-$, con la ejecución LexBFS$^-(G[M], \tau)$ obtenemos también el barrido α, pues $G[M] = \overline{\overline{G}[M]}$. ■

Proposición 4.5.6. *Sean G una cográfica, σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento devuelto por LexBFS$^-(G,\sigma)$. Consideremos cualquier módulo en G con su σ-mínimo, digamos M y x respectivamente. Si $a \in S_1(x)$ y $b \in \overline{S}_1(x)$, con las x-celdas tomadas respecto a $\sigma|_M$ y $\sigma^-|_M$, se satisface lo siguiente: El nodo padre de x en el coárbol de $G[M]$ tiene etiqueta 0, si y solo si $ab \in E_G$.*

Demostración. Sea T el coárbol de G y, en virtud de la Proposición 3.1.6, sea $T[M]$ el coárbol de $G[M]$. Recordemos que en la demostración, $S_1(x)$ y $\overline{S}_1(x)$ hacen referencia a las x-celdas de los ordenamientos $\sigma|_M$ y $\sigma^-|_M$, los cuales son ordenamientos LexBFS y LexBFS$^-$ respectivamente, esto según la Proposición 4.5.5. Primero observemos que $S_1(x) \neq \varnothing$, si y solo si, hay nodos con etiqueta 0 sobre la rama P_x en $T[M]$, esto se sigue inmediatamente de la Proposición 4.5.2. Análogamente, podemos establecer que $\overline{S}_1(x) \neq \varnothing$, si y solo si, la rama P_x en $T[M]$ tiene nodos con etiqueta 1, esto por la Proposición 4.5.3. Por la Proposición 3.2.2, sabemos que existe un conjunto no vacío de nodos hermanos en T, digamos U, tal que $M = \bigcup_{u\in U} H(T_u)$. Como $S_1(x)$ y $\overline{S}_1(x)$ no son vacíos, concluimos que $|M| \geq 3$; o sea que, podemos deshacernos del caso en el que U solo tiene un nodo hoja, es decir, en el que $U = \{x\}$ con x una hoja. De este modo, podemos suponer que $|U| > 1$, ya que si U fuera unitario, como no es el unitario de un hoja, podemos considerar al conjunto de hijos del único nodo que pertenece a U, en lugar de considerar a U mismo. Una vez establecido que $|U| > 1$, podemos denotar por p al padre de los nodos en U. Como $M = \bigcup_{u\in U} H(T_u)$, el coárbol $T[M]$ no es otra cosa que los árboles del conjunto $\{T_u \colon u \in U\}$ colgados, o sea enraizados, bajo el nodo p, es decir, $T[M]$ es el subárbol de T_p que obtenemos al quitar todos los nodos que no tienen como ancestro a algún nodo de U. Observemos que, si $a \in S_1(x) = H(T^x_{01}[M])$ y $b \in \overline{S}_1(x) = H(T^x_{11}[M])$, se tiene que $mca_{T[M]}(a,b) = mca_T(a,b) = \max_T\{0^x_1, 1^x_1\}$. Dado que el padre de x es el nodo $\min_T\{0^x_1, 1^x_1\}$, vamos a tener que el padre de x tiene etiqueta 0, si y solo si, el padre de x es el nodo 0^x_1, pero esto ocurre solo cuando $0^x_1 = \min_T\{0^x_1, 1^x_1\}$, equivalentemente, cuando $1^x_1 = \max_T\{0^x_1, 1^x_1\} = mca_T(a,b)$. Utilizando la Proposición 3.1.5, establecemos lo deseado, esto es, que el padre de x tiene etiqueta 0, si y solamente si $ab \in E_G$. ■

Recalquemos una observación que ya está presente en la Proposición 4.5.6 de forma implícita. Dada una cográfica G con coárbol T y cualquier módulo M de G, la Proposición 3.2.2 nos dice que existe un conjunto de nodos hermanos en T, digamos U, tal que $M = \bigcup_{u\in U} H(T_u)$. Entonces, para cualquier $z \in M$, existe $u \in U$ tal que la zu-trayectoria en T también es la zu-trayectoria en el coárbol de $G[M]$. Lo anterior nos permite deducir que el padre de z en el coárbol de $G[M]$ coincide con el padre de z en T, en caso de que T no sea trivial, claro está. Esto junto con las

Proposiciones 4.5.1 y 4.5.6, nos permite formular el siguiente corolario. Este corolario nos deja saber la etiqueta del padre de x en el coárbol de cualquier módulo con solo verificar la adyacencia entre un vértice en la celda $S_1(x)$ y otro en la celda $\overline{S}_1(x)$. Lo importamte aquí, es que las celdas son respecto a los ordenamiento globales σ y σ^-.

Corolario 4.5.7. *Sean G una cográfica con coárbol T, σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento devuelto por LexBFS$^-(G,\sigma)$. Consideremos cualquier módulo en G con su σ-mínimo, digamos M y x respectivamente. Si $a \in S_1(x)$ y $b \in \overline{S}_1(x)$, con las x-celdas tomadas respecto a las ejecuciones σ y σ^-, se satisface lo siguiente: El nodo padre de x en el coárbol de $G[M]$, que es el padre de x en T, tiene etiqueta 0 si y solo si $ab \in E_G$.*

Ya sabemos que $H(T^x_{0i}) = S_i(x)$ y que $H(T^x_{1i}) = \overline{S}_i(x)$, pero solo cuando x es el primer vértice en el ordenamiento. Parte de los resultados anteriores nos explican como se comportan los ordenamientos LexBFS y LexBFS$^-$ cuando los restringimos a un módulo. Resulta que las las v-celdas, para cualquier $v \in V_G$, son un tipo de módulo muy especial, por lo que bajo ciertas condiciones podemos preservar las igualdades anteriores.

Las definiciones de los subárboles enraizados en una rama, es decir, las definiciones de T^v_{0i} y T^v_{1j}, las dimos en general. Sin embargo, vamos a establecer una convención de gran importancia para todos los resultados que restan en este trabajo. Sean G una cográfica, σ un ordenamiento LexBFS de V_G, y σ^- el ordenamiento devuelto por la ejecución LexBFS$^-(G,\sigma)$. Para todo $v \in V_G$ que no sea el primer vértice, o sea, que satisfaga $\sigma(v) > 0$, vamos a considerar a T^v_{0i} y T^v_{1j} como subárboles de $S(v)$ y de $\overline{S}(v)$ respectivamente. Puede que no se comprenda la causa de esto aún, pero es importante hacerlo notar de una vez, pues algunas veces los árboles T^v_{0i} y T^v_{1j} no los vamos a considerar completos; es decir, no los vamos a considerar con la definición del coárbol global. Por ejemplo, supongamos que tenemos el árbol T^v_{0i} utilizando la definición normal, y resulta que 0^v_i es ancestro de la hoja u, o sea que $0^v_i = 0^u_j$ para algún $j > 0$. Casi siempre en lo que resta del trabajo, al considerar T^u_{0j} bajo este contexto, vamos a pensarlo como subárbol de T^v_{0i}, pues, si consideramos a T^u_{0j} con la definición normal, estaríamos contemplando algunos nodos en la rama P_v, y justamente queremos evitar estos nodos. Sobre todo, esta observación hay que tenerla en cuenta en los Lemas 4.5.8, 4.5.9, 4.5.11 y 4.5.12, el Teorema 4.5.13 y la Proposición 4.5.15.

Lema 4.5.8. *Sean G una cográfica con coárbol T. Tomemos σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento devuelto por la ejecución LexBFS$^-(G,\sigma)$. Consideremos $v \in V_G$ e $i > 0$ tales que $S_i(v) = H(T^v_{0i})$. Se satisface para cualquier*

$j > 0$ tal que la j-ésima v_i-celda existe, que $S_j(v_i) = H(T_{0j}^{v_i})$. De modo similar, si $\overline{S}_i(v) = H(T_{1i}^{v})$, entonces, para todo $j > 0$, si la celda $\overline{S}_j(\overline{v_i})$ existe, ésta coincide con $H(T_{1j}^{\overline{v_i}})$. Todas las celdas son consideradas respecto a σ y σ^-.

Demostración. Digamos que la ejecución LexBFS(G, τ) nos devuelve el ordenamiento σ, donde τ es algún ordenamiento de V_G. Suponiendo que $S_i(v) = H(T_{0i}^{x})$, podemos usar la Proposición 3.2.2 para afirmar que $S_i(v)$ es un módulo de G. Sea η el ordenamiento de $S_i(v)$ que induce el orden dado por $\sigma|_{S_i(v)}$, es decir, η es simplemente $\sigma|_{S_i(v)}$ pero reacomodando las imagenes para que estén en $\{0, \ldots, |S_i(v)| - 1\}$. Por la Proposición 4.5.5, sabemos que η es el ordenamiento que devuelve la ejecución LexBFS$(G[S_i(v)], \tau)$. Además, por la Proposición 4.5.5 también, dado que para todo $y \in S_i(v)$ se cumple $S(y) \subseteq S_i(v)$, los conjuntos $S(y)$, $S^A(y)$ y $S^N(y)$ en la ejecución que usa a G como entrada, son los mismos que en la ejecución que usa a $G[S_i(v)]$ como entrada. De hecho, es muy fácil ver que, por el funcionamiento de LexBFS, la ejecución LexBFS(G, τ) hace exactamente lo mismo con la celda $S_i(v)$, a partir de que v_i es tomado como pivote, que la ejecución LesxBFS$(G[S_i(v)], \tau)$. Por lo tanto, para el módulo $S_i(v)$ en particular, sí se satisface que para cualquier $y \in S_i(v)$ y para cualquier $j > 0$, la y-celda $S_j(y)$ es la misma en la ejecución LexBFS(G, τ) y en la ejecución LexBFS$(G[S_i(v)], \tau)$. Ahora, denotemos por F al coárbol de $G[S_i(v)]$ y veamos quién es F. Si podemos asegurar que todos los nodos en T_{0i}^{v} tienen al menos dos hijos, entonces este árbol es un coárbol, y de la igualdad $H(T_{0i}^{v}) = S_i(v)$ deducimos que es justamente el coárbol de $S_i(v)$, o sea F. Por otro lado, si no todos los nodos de T_{0i}^{v} tienen al menos dos hijos, debe ser porque su nodo raíz, o sea 0_i^v, tiene exactamente un hijo en T_{0i}^{v}; en este caso, el coárbol F es el árbol T_g, donde g es el hijo de 0_i^v que está en la rama P_{v_i}. Notemos que en este último caso, 0_i^v solo tiene dos hijos, uno está en la rama P_v y el otro no, este último es al que definimos como g. En cualquiera de los casos anteriores, para cualquier $j > 0$ tal que el nodo $0_j^{v_i}$ existe en F, el árbol $F_{0j}^{v_i}$ coincide con el árbol $T_{0j}^{v_i}$. Por lo tanto, en virtud de la Proposición 4.5.2 y de que v_i es η-mínimo en $S_i(v)$, podemos concluir directamente que la j-ésima v_i-celda en la ejecución LexBFS$(G[S_i(v)], \tau)$, que coincide con $S_j(v_i)$ por lo argumentado antes, es exactamente el conjunto $H(F_{0j}^{v_i}) = H(T_{0j}^{v_i})$. La demostración para ver que $\overline{S}_j(\overline{v_i}) = H(T_{1j}^{\overline{v_i}})$ es análoga a lo que ya desarrollamos, pero utilizando los resultados correspondientes para σ^- y los nodos con etiqueta 1. ■

Las Proposiciones 4.5.2 y 4.5.3 junto con el Lema 4.5.8, nos habilitan el siguiente razonamiento. Para construir el coárbol de una cográfica G dados los ordenamientos σ y σ^-, como ya habíamos observado, podemos tomar el primer vértice de los ordenamientos, digamos x, y construir los subárboles T_{0i}^{x} y T_{1j}^{x}, que junto con la rama P_x resultan en el coárbol de G. Ahora, para construir los subárboles que necesitamos

podemos usar un proceso recursivo con los vértices $x_1, \ldots, x_k$ y $\overline{x}_1, \ldots, \overline{x}_l$, pues sabemos que $H(T^x_{0i}) = S_i(x)$ y $H(T^x_{1j}) = \overline{S}_j(x)$; además de que ya mostramos que para cualesquiera $r, t > 0$ se cumple $H(T^{x_i}_{0r}) = S_r(x_i)$ y $H(T^{\overline{x_j}}_{1t}) = \overline{S}_t(\overline{x_j})$. Sin embargo, también necesitamos asegurar que podemos calcular los árboles $T^{\overline{x_j}}_{0t}$ y $T^{x_i}_{1r}$, de los cuales podemos sospechar, tienen conjuntos de hojas $S_t(\overline{x_j})$ y $\overline{S}_r(x_i)$ respectivamente, pero todavía no hemos demostrado eso.

De una aplicación directa de la Proposición 3.2.2 y el Teorema 4.1.11 obtenemos el siguiente lema.

Lema 4.5.9. *Sean G una cográfica con coárbol T. Tomemos σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento devuelto por la ejecución LexBFS$^-(G, \sigma)$. Consideremos $v \in V_G$ y $j, k > 0$ tales que los nodos 0^v_j y 1^v_k existen. Si u es el σ-mínimo de $H(T^v_{0j})$ y z es el σ-mínimo de $H(T^v_{1k})$, se satisfacen las contenciones $H(T^v_{0j}) \subseteq S(u)$, $H(T^v_{0j}) \subseteq \overline{S}(u)$, $H(T^v_{1k}) \subseteq S(z)$ y $H(T^v_{1k}) \subseteq \overline{S}(z)$. Además, estas rebanadas son las más pequeñas con la propiedad de cumplir dichas contenciones.*

Lema 4.5.10. *Sean G una cográfica con coárbol T. Tomemos σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento devuelto por la ejecución LexBFS$^-(G, \sigma)$. Consideremos $v \in V_G$ e $i > 0$ tales que $S(v)$ es un módulo de G, y además $S_i(v) = H(T^v_{0i})$. Si $\overline{S}(u)$ es la rebanada mínima por contención de σ^- que contiene a $S_i(v)$, entonces $\overline{S}(u) \setminus S_i(v) \subseteq \overline{S^A}(u)$, y para cualquier $z \in \overline{S}(u) \setminus S_i(v)$, se cumple que $N(z) \cap S_i(v) = \varnothing$.*

Demostración. Como $S(v)$ es un módulo, utilizando el Teorema 4.1.11 deducimos que $S(v) \subseteq \overline{S}(v)$. Como los vértices en $S_i(v)$ no son adyacentes a v, se debe tener $S_i(v) \subseteq \overline{S^A}(v)$, y por lo tanto $u \neq v$, pues la rebanada $\overline{S^A}(v) \setminus \{v\}$ contiene a $S_i(v)$. Por la definición de rebanada, se debe cumplir alguna de las siguientes situaciones: $\overline{S}(u) \cap \overline{S}(v) = \varnothing$, $\overline{S}(u) \subseteq \overline{S}(v)$ o $\overline{S}(v) \subseteq \overline{S}(u)$. Dado que $S_i(v) \subseteq \overline{S}(u)$ y $S_i(v) \subseteq S(v) \subseteq \overline{S}(v)$, por la propiedad mínima que tiene $\overline{S}(u)$, lo que debe ocurrir es que $\overline{S}(u) \subseteq \overline{S}(v)$. Por la misma naturaleza de LexBFS$^-$, se debe satisfacer $\overline{S}(u) \subseteq \overline{S^A}(v)$ o $\overline{S}(u) \subseteq \overline{S^N}(v)$. Sin embargo, $S_i(v) \subseteq \overline{S}(u)$ y los vértices en $S_i(v)$ no son adyacentes a v, o sea, están en $\overline{S^A}(v)$; por lo tanto $\overline{S}(u) \subseteq \overline{S^A}(v)$. Deducimos que, para todo $z \in \overline{S}(u)$, el nodo $mca(z, v)$ tiene etiqueta 0 en T. Para demostrar lo que afirma el lema, fijemos cualquier $z \in \overline{S}(u) \setminus S_i(v)$. Por hipótesis, se tiene que $S_i(v) = H(T^v_{0i})$, como $z \notin H(T^v_{0i})$ se sigue que $mca(z, v)$ no puede ser el nodo 0^v_i. Se sigue que, para cualquier $w \in H(T^v_{0i})$, por la Proposición 2.2.3, si $mca(z, v) >_T 0^v_i$, entonces $mca(z, w) = mca(z, v)$, pero si $mca(z, v) <_T 0^v_i$, entonces $mca(z, w) = 0^v_i$. En cualquier caso $mca(z, w)$ es un nodo con etiqueta 0, lo que implica que $zw \notin E_G$ por la Proposición 3.1.5. Como $\overline{S}(u) \setminus \{u\}$ es una rebanada de σ^- más pequeña que

$\overline{S}(u)$, no puede contener a $S_i(v)$. Por lo tanto $u \in S_i(v) = H(T^v_{0i})$. Utilizando lo anterior, dado que $u \in H(T^v_{0i})$, podemos garantizar que $zu \notin E_G$ y por lo tanto $z \in \overline{S^A}(u)$. Esto basta para la contención $\overline{S}(u) \setminus S_i(v) \subseteq \overline{S^A}(u)$, pues z es arbitrario. Ahora, $N(z) \cap S_i(v) = \varnothing$ se satisface porque ya vimos que para cualquier $w \in H(T^v_{0i}) = S_i(v)$, se cumple $zw \notin E_G$. ■

Tenemos también la versión simétrica del lema anterior respecto a σ^- y σ. A pesar de que el siguiente lema también es importante, no vamos a desarrollar su demostración, solo lo vamos a enunciar. Esta decisión la tomamos porque la prueba es la misma que dimos en el Lema 4.5.10, solo hay que intercambiar los conceptos de LexBFS y LexBFS$^-$ según lo necesite el argumento.

Lema 4.5.11. *Sean G una cográfica con coárbol T. Tomemos σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento devuelto por la ejecución LexBFS$^-(G, \sigma)$. Consideremos $v \in V_G$ y $j > 0$ tales que $\overline{S}(v)$ es un módulo de G, y además $\overline{S}_j(v) = H(T^v_{1j})$. Si $S(u)$ es la rebanada mínima por contención de σ que contiene a $\overline{S}_j(v)$, entonces $S(u) \setminus \overline{S}_j(v) \subseteq S^A(u)$, y para cualquier $z \in S(u) \setminus \overline{S}_j(v)$ se cumple que $\overline{S}_j(v) \subseteq N(z)$.*

El siguiente lema complementa lo que establece el Lema 4.5.8, utilizando una hipótesis extra, la cual no conlleva ningún problema para nuestro algoritmo.

Lema 4.5.12. *Sean G una cográfica con coárbol T. Tomemos σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento devuelto por la ejecución LexBFS$^-(G, \sigma)$. Consideremos $v \in V_G$ e $i > 0$ tales que $S(v)$ es un módulo y $S_i(v) = H(T^v_{0i})$. Se satisface para cualquier $j > 0$ tal que la j-ésima v_i-celda $\overline{S}_j(v_i)$ exista, que $\overline{S}_j(v_i) = H(T^{v_i}_{1j})$. Por otro lado, si $\overline{S}(v)$ es un módulo y $\overline{S}_i(v) = H(T^v_{1i})$, entonces, para todo $j > 0$, si la celda $S_j(\overline{v}_i)$ existe, ésta coincide con $H(T^{\overline{v}_i}_{0j})$. Todas las celdas son consideradas respecto a σ y σ^-.*

Demostración. Fijemos $j > 0$; supondremos según el caso que estemos viendo, que la v_i-celda $S_j(\overline{v}_i)$ existe, o bien que la celda $\overline{S}_j(v_i)$ existe. Comencemos probando que $\overline{S}_j(v_i) = H(T^{v_i}_{1j})$ dado que $S_i(v) = H(T^v_{0i})$ y que $S(v)$ es un módulo. Por el Lema 4.5.9, se cumple que $\overline{S}(v_i)$ es la rebanada de σ^- mínima por contención, con la propiedad de contener al módulo $S_i(v) = H(T^v_{0i})$, pues v_i es el mínimo de este módulo (respecto a σ y σ^-). Por lo anterior, junto con los hechos de que $S(v)$ es un módulo y $S_i(v) = H(T^v_{0i})$, ya tenemos todas las hipótesis para utilizar el Lema 4.5.10, garantizando que $\overline{S}(v_i) \setminus S_i(v) \subseteq \overline{S^A}(v_i)$ y, que para cualquier $z \in \overline{S}(v_i) \setminus S_i(v)$ se cumple que $N(z) \cap S_i(v) = \varnothing$. Veamos qué nos dice lo anterior en relación a las v_i celdas de un ordenamiento LexBFS$^-$ de $G[S_i(v)]$. Para esto, vamos a denotar por $\overline{\mathcal{S}^A}(v_i)$ y

por $\overline{\mathcal{S}^N}(v_i)$ a la vecindad y no vecindad dentro de $S_i(v)$ respectivamente, pero en la ejecución LexBFS$^-(G[S_i(v)], \sigma|_{S_i(v)})$, donde hacemos el abuso de considerar a $\sigma|_{S_i(v)}$ como un ordenamiento de los vértices de $G[S_i(v)]$. De igual manera, vamos a denotar por $\overline{\mathcal{S}}_k(v_i)$ a las v_i-celdas en este barrido LexBFS$^-$. Recordemos que lo que define a la celda $\overline{S}_j(v_i)$, tomada respecto a σ^-, son los vértices en $\overline{S^A}(v_i)$ que van separando a los vértices en $\overline{S^N}(v_i)$, hasta que se terminan de explorar los vértices en $\overline{S^A}(v_i)$ y nos quedamos con las v_i-celdas constituidas, en particular con $\overline{S}_j(v_i)$. Tomando los complementos respecto a $\overline{S}(v_i)$ en la contención $\overline{S}(v_i)\setminus S_i(v) \subseteq \overline{S^A}(v_i)$, obtenemos que $\overline{S^N}(v_i) \cup \{v_i\} \subseteq S_i(v)$, garantizando que todos los vértices en $\overline{S^N}(v_i)$ y en particular los de $\overline{S}_j(v_i)$, quedan dentro de $S_i(v) = \overline{\mathcal{S}}(v_i)$, o sea que $\overline{S^N}(v_i) \subseteq \overline{\mathcal{S}^N}(v_i)$. También es menester observar aquí que $\overline{\mathcal{S}^N}(v_i) \subseteq \overline{\mathcal{S}}(v_i) = S_i(v)$, por lo que $\overline{\mathcal{S}^N}(v_i) \subseteq \overline{S^N}(v_i)$, pues $S_i(v) \subseteq \overline{S}(v_i)$. De lo anterior concluimos que $\overline{S^N}(v_i) = \overline{\mathcal{S}^N}(v_i)$. Ahora, para cualquier $z \in \overline{S^A}(v_i)$ hay dos posibilidades, la primera es que $z \in S_i(v) = \overline{\mathcal{S}}(v_i)$, en este caso $z \in \overline{\mathcal{S}^A}(v_i)$ porque $zv_i \notin E_G$. La segunda opción es que $z \notin S_i(v)$, o sea que $z \in \overline{S}(v_i) \setminus S_i(v)$, de donde se sigue que $N(z) \cap S_i(v) = \varnothing$. Lo que nos dice lo anterior, es que los vértices en $\overline{S^A}(v_i)$ que en verdad separan en celdas distintas a los vértices de $\overline{S^N}(v_i) = \overline{\mathcal{S}^N}(v_i) \subseteq S_i(v)$, son los vértices en $\overline{S^A}(v_i) \cap S_i(v)$. Por otro lado, tenemos que $\overline{\mathcal{S}^A}(v_i) \subseteq S_i(v) \cap (V_G \setminus N(v_i)) \subseteq \overline{S^A}(v_i)$, la última contención es porque $S_i(v) \subseteq \overline{S}(v_i)$. Reformulando la afirmación anterior, podemos decir que, los vértices en $\overline{S^A}(v_i)$ que en verdad separan en celdas distintas a los vértices de $\overline{S^N}(v_i) = \overline{\mathcal{S}^N}(v_i)$, son los vértices en $\overline{S^A}(v_i) \cap S_i(v)$, o sea los vértices de $\overline{\mathcal{S}^A}(v_i)$. Toda la argumentación anterior es para mostrar que los barridos LexBFS$^-(G, \sigma)$ y LexBFS$^-(G[S_i(v)], \sigma|_{S_i(v)})$ nos van a dejar con exactamente las mismas v_i-celdas. Esto es, para cada $k > 0$, $\overline{S}_k(v_i)$ existe, si y solo si $\overline{\mathcal{S}}_k(v_i)$ existe, y además coinciden en caso de existir.

Ahora, el coárbol de $G[S_i(v)] = G[H(T^v_{0i})]$, digamos F, es T^v_{0i} o bien T^v_{0i} quitando la raíz 0^v_i en caso de que tenga un solo hijo. Como consecuencia, podemos ver fácilmente que $F^{v_i}_{1k}$ coincide con $T^{v_i}_{1k}$ cuando el nodo $1^{v_i}_k$ existe en F. De acuerdo con la Proposición 4.5.3, tenemos por todo lo desarrollado hasta ahora, que $T^{v_i}_{1j} = F^{v_i}_{1j}$ coincide con $\overline{\mathcal{S}}_j(v_i) = \overline{S}_j(v_i)$. Para demostrar que $S_j(\overline{v}_i)$ es $H(T^{\overline{v}_i}_{0j})$ dado que $S(v)$ es un módulo y $\overline{S}_i(v) = H(T^v_{1i})$, podemos utilizar una argumentación casi idéntica, utilizando los Lemas 4.5.9 y 4.5.11 y la Proposición 4.5.2 para este caso. Por lo tanto, terminamos aquí la demostración. ■

Teorema 4.5.13. *Sean G una cográfica con coárbol T, σ un ordenamiento LexBFS arbitrario de V_G y σ^- el ordenamiento que devuelve LexBFS$^-(G, \sigma)$. Para cualquier vértice $x \in V_G$ tal que $\sigma(x) > 0$, es decir, tal que no sea el primer vértice en σ, se satisface lo siguiente. Existen $v \in V_G$ y $k > 0$ de tal forma que $x = v_k$ o $x = \overline{v}_k$; en*

el primer caso se satisface que $S(x) = S_k(v) = H(T^v_{0k})$*, mientras que en el segundo* $\overline{S}(x) = \overline{S}_k(v) = H(T^v_{1k})$*. Además, para cualesquiera* $i, j > 0$*,* $S_i(x) = H(T^x_{0i})$ *y* $\overline{S}_j(x) = H(T^x_{1j})$ *en el caso de existir dichas* x*-celdas.*

Demostración. Procedamos por inducción fuerte sobre la posición de x en el ordenamiento σ. El caso base es que x sea el segundo vértice del ordenamiento σ. Sea v el primer vértice de σ. Si $x \in S^N(v)$, debe existir $k > 0$ tal que $x \in S_k(v)$, pero x es σ-mínimo en $S(v) \setminus \{v\}$, en particular x es σ-mínimo en $S_k(v)$, por lo que $x = v_k$. En este caso, por el Corolario 4.1.10 y la Proposición 4.5.2, podemos afirmar que $S(x) = S_k(v) = H(T^v_{0k})$. Dado que $S(v) = V_G$ es un módulo, podemos utilizar los Lemas 4.5.8 y 4.5.12 para asegurar que $S_i(x) = H(T^x_{0i})$ y $\overline{S}_j(x) = H(T^x_{1j})$. Algo similar ocurre si $x \in S^A(v) = \overline{S^N}(v)$; tomando $k > 0$ tal que $x \in \overline{S}_k(v)$, se sigue que $x = \overline{v}_k$, pues x es el σ-mínimo de $\overline{S}_k(v)$. Por el Corolario 4.1.10 y la Proposición 4.5.3 tenemos que $\overline{S}(x) = \overline{S}_k(v) = H(T^v_{1k})$. Dado que $\overline{S}(v) = V_G$ es un módulo, podemos utilizar los Lemas 4.5.8 y 4.5.12 para asegurar que $S_i(x) = H(T^x_{0i})$ y $\overline{S}_j(x) = H(T^x_{1j})$. Esto concluye el caso base para nuestra inducción.

Supongamos ahora que para cualquier $y \in V_G$ tal que $0 < \sigma(y) < \sigma(x)$ se satisface el enunciado. Tenemos que x debe estár en $S^N(\sigma^{-1}(0))$ o en $S^A(\sigma^{-1}(0)) = \overline{S^N}(\sigma^{-1}(0))$. Tomemos a v, el vértice σ-máximo tal que $v <_\sigma x$, y además cumple $x \in S^N(v)$ o $x \in \overline{S^N}(v)$. Comencemos viendo el caso en el que $x \in S^N(v)$, este caso nos va a llevar tiempo, así que tengamos en mente que es uno de dos casos posibles. En este caso debe existir $k > 0$ tal que $x \in S_k(v)$. Vamos a suponer que $v_k < x$ para derivar una contradicción. Primero observemos que de ocurrir $\sigma(v) > 0$, el vértice v satisface la hipótesis inductiva, la cual implica, entre otras cosas, que $S(v)$ es un módulo, o bien $\overline{S}(v)$ es un módulo; si v es el primer vértice de σ, claramente $S(v)$ y $\overline{S}(v)$ son ambos módulos. Si $\sigma(v) > 0$, para cualquier $l > 0$ se satisface por la hipótesis inductiva que, $S_l(v) = H(T^v_{0l})$ cuando la v-celda $S_l(v)$ existe, y que $\overline{S}_l(v) = H(T^v_{1l})$ cuando la celda $\overline{S}_l(v)$ existe. Lo anterior también se cumple cuando $\sigma(v) = 0$, esto por las Proposiciones 4.5.2 y 4.5.3. Por la propiedad de máximo que tiene v, no puede ocurrir $x \in S^N(v_k)$, así que $x \in S^A(v_k)$, o sea que $xv_k \in E_G$. Sin embargo, el Teorema 4.1.11 nos dice que $\overline{S}(v_k)$ es la rebanada mínima por contención que contiene al módulo $H(T^v_{0k}) = S_k(v) = S(v_k)$, de donde se sigue que $x \in \overline{S}(v_k)$. Pero que $x \in \overline{S}(v_k)$, implica que $x \in \overline{S^N}(v_k)$, contradiciendo la propiedad de máximo que tiene v. Por lo tanto $x = v_k$, de donde deducimos que $S(x) = S(v_k) = S_k(v) = H(T^v_{0k})$ es un módulo.

Si $S(v)$ es un módulo, entonces por los Lemas 4.5.8 y 4.5.12 se tiene que $S_i(x) = H(T^x_{0i})$ y $\overline{S}_j(x) = H(T^x_{1j})$. Por otro lado, si la rebanada que es módulo resulta ser $\overline{S}(v)$, el Lema 4.5.8 y la hipótesis inductiva nos dicen que, como $S_k(v) = S(x) = H(T^v_{0k})$,

se debe cumplir para todo $i > 0$ tal que la x-celda exista que, $S_i(x) = H(T^x_{0i})$. Solo falta ver que para todo $j > 0$ tal que $\overline{S}_j(x)$ existe, se cumple $\overline{S}_j(x) = H(T^x_{1j})$. Sin embargo, como no contamos con que $S(v)$ sea un módulo necesariamente, no podemos aplicar un argumento directo; así que, vamos a tener que desarrollar desde el principio un argumento casi idéntico al que usamos en el Lema 4.5.10. Si $\sigma(v) = 0$, es decir, si v es el primer vértice de σ, podemos usar lo que explicamos al inicio del párrafo, porque $S(\sigma^{-1}(0))$ y $\overline{S}(\sigma^{-1}(0))$ son ambos módulos al coincidir con V_G. Ahora supongamos que $\sigma(v) > 0$, y apliquemos la hipótesis inductiva a v. Además, supongamos que $S(v)$ no es un módulo, pues este caso ya quedó cubierto. Notemos que la hipótesis inductiva, no solo nos dice que $\overline{S}(v)$ sea un módulo, sino que, como $S(v)$ no es un módulo, nos asegura que $v = \overline{y}_l$ y que $\overline{S}(v) = \overline{S}_l(y) = H(T^y_{1l})$ para algunos $y \in V_G$ y $l > 0$. Utilicemos un argumento por contradicción para ver que $S(x) \subseteq H(T^y_{1l})$, pero antes, nótese que para cualquier $w \in S(x) = S_k(v) = H(T^v_{0k})$, se cumple $y <_\sigma v <_\sigma w$ y $mca(w, v) = 0^v_k$. Si suponemos que $w \in S(x)$ es tal que $w \notin H(T^y_{1l})$, se debe tener $mca(w, y) >_T 1^y_l$ o $mca(w, y) <_T 1^y_l$. Si $mca(w, y) >_T 1^y_l$, utilizando la Proposición 2.2.3 y el hecho de que $mca(v, y) = 1^y_l$, deducimos que $mca(w, y) = mca(w, v) = 0^v_k$. O sea que $yw \notin E_G$ y $yv \in E_G$ con $y <_\sigma v$, implicando así la contradicción de que $w \notin S(v)$. La contradicción que se deriva de suponer $mca(w, y) < 1^y_l = mca(v, y)$ es más directa, pues, esta desigualdad implica por la Proposición 2.2.3 que $0^v_k = mca(w, v) = mca(v, y) = 1^y_l$, algo absurdo claramente. Por lo tanto, se satisface que $w \in H(T^y_{1l}) = \overline{S}(v)$, y por ser w arbitrario, también que $S(x) \subseteq H(T^y_{1l})$. Al ser $S(x) = H(T^v_{0k})$ un módulo, se sigue que $\overline{S}(x)$ es la rebanada mínima por contención con la propiedad de contener a $S(x)$. Por lo tanto, tenemos que $S(x) \subseteq \overline{S}(x) \cap \overline{S}(v)$. Las rebanadas $\overline{S}(v)$ y $\overline{S}(x)$ deben cumplir una, y solo una de las siguientes condiciones: $\overline{S}(v) \cap \overline{S}(x) = \varnothing$, $\overline{S}(v) \subseteq \overline{S}(x)$, o $\overline{S}(x) \subseteq \overline{S}(v)$. Ya vimos que $\overline{S}(x) \cap \overline{S}(v) \neq \varnothing$, así que al ser $\overline{S}(x)$ nínima por contención con la propiedad de contener a $S(x)$, se sigue que $\overline{S}(x) \subseteq \overline{S}(v)$. A partir de aquí, ya podemos replicar el desarrollo que hicimos en el Lema 4.5.10 para concluir que $\overline{S}(x) \setminus S(x) \subseteq \overline{S^A}(x)$; y que para cualquier $u \in \overline{S}(x) \setminus S(x)$, se cumple que $N(u) \cap S(x) = \varnothing$. Utilizando un argumento que ya desarrollamos en el Lema 4.5.12, lo anterior implica que la ejecución $\text{LexBFS}^-(G, \sigma)$ a partir de que x es tomado como pivote, genera las mismas x-celdas que la ejecución $\text{LexBFS}^-(G[S(x)], \sigma|_{S(x)})$, porque $\overline{S^N}(x) \subseteq S(x)$ y los vértices en $\overline{S}(x) \setminus S(x)$ son inocuos para con los vértices de $S(x)$. O sea que para $j > 0$ tal que $\overline{S}_j(x)$ existe, la celda $\overline{S}_j(x)$ coincide con la x-celda respecto al ordenamiento devuelto por la ejecución $\text{LexBFS}^-(G[S(x)], \sigma|_{S(x)})$, pero esta última celda es $H(F^x_{1j})$ por la Proposición 4.5.3, donde F es el coárbol de la cográfica $G[S(x)]$. De nuevo, un argumento ya trillado para nosotros, nos dice que F^x_{1j} es justamente T^x_{1j}, pues $S(x) = T^v_{0k}$.

En caso de que tengamos $x \in \overline{S^N}(v)$, podemos demostrar análogamente que $x = \overline{v}_k$ para algún $k > 0$. También la prueba de este caso es análoga a la del caso anterior, claro, utilizando los lemas y los resultados correspondientes. ■

Del Teorema 4.5.13 podemos extraer el siguiente corolario.

Corolario 4.5.14. *Sean G una cográfica con coárbol T, σ un ordenamiento LexBFS arbitrario de V_G y σ^- el ordenamiento que devuelve LexBFS$^-(G, \sigma)$. Para cualquier vértice $x \in V_G$, $S_i(x) = H(T^x_{0i})$ si $S_i(x)$ existe; también $\overline{S}_j(x) = H(T^x_{1j})$ si $\overline{S}_j(x)$ existe.*

También, podemos aprovechar lo reciente del Teorema 4.5.13 y su demostración que acabamos de desarrollar, para ver de una vez la siguiente observación plasmada en una proposición.

Proposición 4.5.15. *Sean G una cográfica con coárbol T. Consideremos σ un ordenamiento LexBFS arbitrario de V_G y σ^- el ordenamiento de V_G que devuelve LexBFS$^-(G, \sigma)$. Consideremos cualquier vértice $v \in V_G$, y denotemos por $\kappa(v)$ al número de v-celdas respecto a σ; también, denotemos por $\eta(v)$ al número de v-celdas respecto a σ^-. Se satisface una de las siguientes desigualdades: $\kappa(v) - 1 \leq \eta(v) \leq \kappa(v)$, o bien, $\eta(v) - 1 \leq \kappa(v) \leq \eta(v)$.*

Demostración. A lo largo de la prueba vamos a utilizar el Teorema 4.5.13 varias veces. Si v es el primer vértice de σ, por las Proposiciones 4.5.2 y 4.5.3 se satisface lo que queremos. Por otro lado, si $\sigma(v) > 0$, para algún $y \in V_G$ y algún $k > 0$, se cumple que $v = y_k$ o $v = \overline{y}_k$. Consideremos el primer caso. En este caso, tenemos que $S(v) = S_k(y) = H(T^y_{0k})$. Para cualquier $i > 0$ tal que $S_i(v)$ existe, se cumple claramente que $S_i(v) = S(v_i) \subseteq S(v) = H(T^y_{0k})$. Lo anterior implica que, como $S_i(v) = H(T^v_{0i})$, se tiene $0^v_i \leq_T 0^y_k$. Ahora, si $j > 0$ es tal que $\overline{S}_j(v) = \overline{S}(\overline{v}_j)$ existe, podemos argumentar de forma análoga a la demostración del Teorema 4.5.13, para deducir que $\overline{S}_j(v) = \overline{S}(\overline{v}_j) \subseteq S(v) = H(T^y_{0k})$. Podemos concluir, dado que $\overline{S}_j(v) = H(T^v_{1j})$, que $1^v_j \leq_T 0^y_k$. Lo anterior es suficiente para mostrar que $\kappa(v)$ es, a lo más, el número de nodos con etiqueta 0 que aparecen en el segmento de v hasta 0^y_k sobre la rama P_v. Del mismo modo, deducimos que $\eta(v)$ es, a lo más, el número de nodos con etiqueta 1 que aparecen en el segmento de v hasta 0^y_k sobre la rama P_v. Solo resta ver que a todo nodo 0^v_i en la rama P_v, que esté sobre el segmento de v a 0^y_k y tal que $H(T^v_{0i})$ no sea vacío, le corresponde una v-celda. Sea $i > 0$ tal que $0^v_i \leq_T 0^y_k$. Por un lado, $H(T^v_{0i}) \subseteq H(T^y_{0k}) = S(v)$, así que, al ser $H(T^v_{0i})$ un módulo, se cumple $H(T^v_{0i}) \subseteq S^N(v)$, y por ende, debe existir $t > 0$ tal que $H(T^v_{0i}) \subseteq S_t(v)$. Como las v-celdas son ajenas dos a dos, se sigue que t debe ser el mismo i. Podemos hacer el

mismo razonamiento para $j > 0$ tal que $1_j^v \leq_T 0_k^y$ y $H(T_{1j}^v)$ no sea vacío. Solo hay que observar que $H(T_{1j}^v) \subseteq H(T_{0k}^y) = S(v) \subseteq \overline{S}(v)$ por ser $S(v)$ un módulo.

Lo que podemos concluir con lo anterior, es que $\kappa(v)$ es exactamente el número de nodos en P_v, que están en el segmento de v a 0_k^y, que son de la forma 0_i^v y tales que $H(T_{0i}^v)$ no es vacío. Es decir, es el número de nodos con etiqueta 0 que hay en la rama P_v pero considerada en el coárbol de $S(v)$. De igual forma, podemos asegurar que $\eta(v)$ es el número de nodos con etiqueta 1 presentes en el coárbol de $G[S(v)]$. Por lo tanto, se satisface alguno de los pares de desigualdades en cuestión.

El caso en el que $v = \overline{y}_k$ es análogo. Vamos a llegar a que $\kappa(v)$ y $\eta(v)$ son, el número de nodos con etiqueta 0 presentes en el coárbol de $G[\overline{S}(v)]$, y el número de nodos con etiqueta 1 presentes en el coárbol de $G[\overline{S}(v)]$, respectivamente. Podemos concluir que se satisface alguno de los pares de desigualdades en cuestión. ■

Ha llegado el momento de dar el algoritmo que construye el coárbol de una cográfica. Para nuestro algoritmo vamos a necesitar los vértices x_i para cualquier $x \in V_G$; vamos a explicar como obtener estos vértices que necesitamos. Sean G una cográfica y σ un ordenamiento LexBFS de V_G, también consideremos a σ^- el ordenamiento LexBFS$^-$ que nos arroja la ejecución LexBFS$^-(G, \sigma)$. Lo que vamos a utilizar son dos arreglos Z y Z^-, tales que para cada $x \in V_G$, $Z[x]$ y $Z^-[x]$ son listas. Por un lado, la lista $Z[x]$ tiene a los vértices $\{x_i : i > 0\}$ y está ordenada respecto a σ. Por otro lado, la lista $Z^-[x]$ tiene a los vértices $\{\overline{x}_i : i > 0\}$ y está ordenada respecto a σ^-. Desde luego, los vértices x_i y $\overline{x}_i$ son tomados en relación a las ejecuciones que resultan en σ y σ^- respectivamente. En cuanto a la construcción del arreglo $Z[x]$, notemos que en el Algoritmo 9 se manipulan a los vértices x_i de forma ordenada para calcular la lista $N[x]$ que tiene las vecindades izquierdas. Por lo que, solo es cuestión de ir añadiendo los vértices x_i al final de la lista $Z[x]$ para que queden σ-ordenados, si es que hay x-celdas, en caso de que no haya x-celdas podemos marcar a $Z[x]$ como nula.

Para calcular $Z^-[x]$, recordemos que también explicamos las modificaciones necesarias para construir el algoritmo LexBFS$^-$-NSP, el cual hace lo propio para tratar con el ordenamiento LexBFS$^-$. Así, lo que nos devuelve la ejecución LexBFS$^-$-NSP(G, σ) es: el ordenamiento σ^-; tres arreglos U, R y M, tales que para todo $x \in V_G$, $R[x] = |\overline{S^N}(x)|$; si $\overline{S^N}(x) = \varnothing$, entonces la entrada $M[x]$ es nula y la lista $U[x]$ es vacía, en cambio para $\overline{S^N}(x) \neq \varnothing$, $M[x]$ tiene al σ^--mínimo de $\overline{S^N}(x)$, o sea a $\overline{x}_1$, y $U[x]$ tiene la lista en orden con las cardinalidades de las x-celdas respecto a σ^-. Posteriormente, en el proceso de verificar NSP para σ^-, hay que calcular las vecindades izquierdas en G de los vértices $\overline{x}_i$ utilizando calculaLvec. Notemos que en este calculo también se manipulan los vértices $\overline{x}_i$, así que basta hacer exactamente lo

mismo que en el otro caso. Es decir, basta con ir guardando los vértices $\overline{x}_i$ en la lista $Z^-[x]$ conforme los vayamos utilizando. Esto funciona, pues haciendo las adiciones al final de la lista, podemos garantizar que la lista $Z^-[x]$ queda σ^--ordenada.

Dado lo anterior, vamos a suponer con toda ligereza, que la ejecución $\mathrm{NSP}(G,\tau)$ nos devuelve, además del par (x,j) que se describe en el Algoritmo 11, el arreglo Z que almacena para cada vértice x la lista $Z[x]=[x_1,\ldots,x_k]$. Del mismo modo, vamos a contar con que $\mathrm{NSP}^-(G,\sigma)$ regresa, además del par (x,j), el arreglo Z^- tal que, para todo $x\in V_G$, $Z^-[x]=[\overline{x}_1,\ldots,\overline{x}_k]$. Nótese que el tiempo llevado a cabo en $\mathrm{NSP}(G,\tau)$, solo aumenta en $|\{x_i\colon x\in V_G, i>0\}|$ pasos extras, o sea en a lo más $|V_G|$ pasos extras según el Corolario 4.3.10; lo mismo aplica para $\mathrm{LexBFS}^-(G,\sigma)$.

Como en el algoritmo de reconocimiento vamos retornar el coárbol de una cográfica, hay que dar una representación sintáctica para los árboles. Para dicha representación vamos a usar los símbolos $\circledast$ y $\circledcirc$ para denotar la coyuntura en nodos con etiqueta 1 o en nodos con etiqueta 0. Es decir, si tenemos dos coárboles T_1 y T_2 con sus respectivas raíces r_1 y r_2, la fórmula $T_1\circledast T_2$ hace referencia al árbol enraizado que obtenemos al colgar, o enraizar, en un nuevo nodo r con etiqueta 1 a los árboles T_1 y T_2. Para que $T_1\circledast T_2$ siga siendo un coárbol, necesitamos garantizar la alternancia de las etiquetas en sus nodos, así que, si r_1 tiene ya etiqueta 1, en lugar de colgar en r al árbol T_1, entenderemos que se suprime a r_1, o sea, solo se cuelgan de r los hijos de r_1 en T_1, los cuales sí tienen etiqueta 0 o son hojas. Desde luego que lo mismo aplica cuando r_2 tiene etiqueta 1. De forma análoga, $T_1\circledcirc T_2$ hace referencia al árbol enaraízado que obtenemos al colgar, o enraizar, en un nuevo nodo r con etiqueta 0 a los árboles T_1 y T_2. Y también, hacemos la salvedad de que, cuando r_1 o r_2 ya sean de etiqueta 0, debemos colgar de r únicamente a los hijos y no a todo el árbol completo. También, vamos a tener la convención de que, para cualquier vértice x, el mismo símbolo x representa al coárbol trivial con vértice x, o sea que enunciados del tipo $x\circledast T_2$ sí tienen sentido. Para clarificar lo anterior, en la Figura 4.5 pusimos varios ejemplos de coárboles donde usamos los constructores $\circledast$ y $\circledcirc$.

Para dar el algoritmo de reconocimiento, vamos a utilizar una subrutina que construye coárboles recursivamente, ésta la podemos ver en el Algoritmo 15.

Una nota del Algoritmo 15 es que, en las formulas que retornan el último `if` y el `else`, se entiende que se van alternando los constructores $\circledast$ y $\circledcirc$ hasta agotar a los vértices x_k y a los $\overline{x}_k$. Como cada celda $S_k(x)$ es $H(T^x_{0k})$ y cada celda $\overline{S}_k(x)$ es $H(T^x_{1k})$, el número de x-celdas respecto a σ, digamos $\kappa(x)$, es el número de nodos 0 de P_x presentes en el coárbol que estamos construyendo, mientras que el número de x-celdas respecto a σ^-, digamos $\eta(x)$, debe ser el número de nodos 1 de P_x en el coárbol que estamos construyendo. Lo anterior lo asegura la Proposición 4.5.15. Esto significa que si entramos en el último `if`, entonces $\kappa(x)-1\leq\eta(x)\leq\kappa(x)$, y

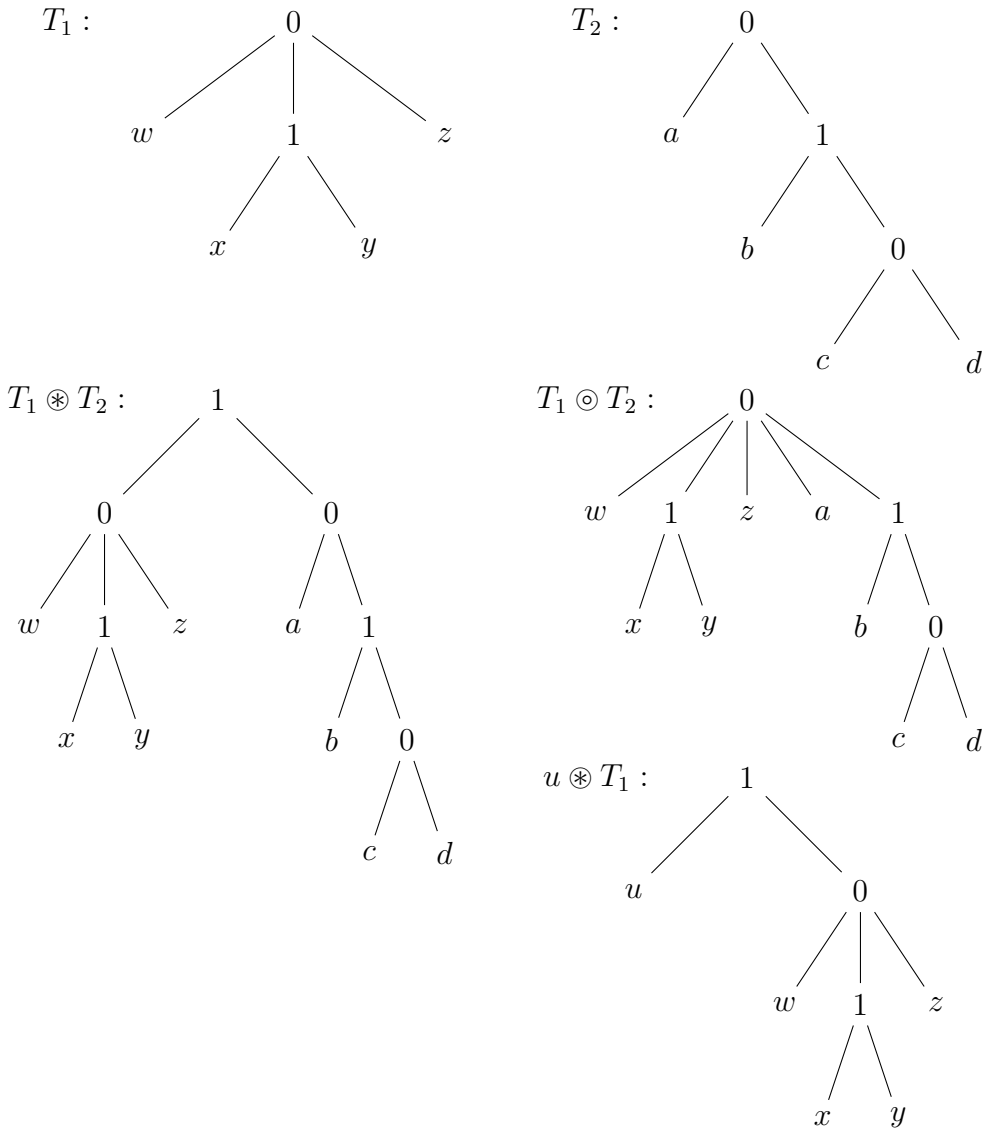


Figura 4.5: Construcción de coárboles utilizando $\circledast$ y $\circledcirc$.

Algoritmo 15: coarbol(G, x, Z, Z^-).

Input: Una cográfica G representada por sus listas de adyacencias; un vértice x de G que coincide con un y_k o con un $\overline{y}_k$, para algunos $y \in V_G$ y $k > 0$, o bien, de tal modo que x es el σ-mínimo de V_G; un arreglo Z tal que para todo $v \in V_G$, $Z[v]$ es nula si $S^N(v) = \varnothing$, y es la lista $[v_1, \ldots, v_{\kappa(v)}]$ en otro caso, donde los vértices $v_1, \ldots, v_{\kappa(v)}$ son los σ-mínimos de las v-celdas respecto a σ, con σ un ordenamiento LexBFS de V_G arbitrario; otro arreglo Z^- tal que para todo $v \in V_G$, $Z^-[v]$ es nula si $\overline{S^N}(v) = \varnothing$, y es la lista $[\overline{v}_1, \ldots, \overline{v}_{\eta(v)}]$, donde los vértices $\overline{v}_1, \ldots, \overline{v}_{\eta(v)}$ son los σ^--mínimos de las v-celdas respecto a σ^-, con σ^- el ordenamiento que devuelve la ejecución LexBFS$^-(G, \sigma)$

Output: Una fórmula utilizando los constructores $\circledast$ y $\circledcirc$ que representa al coárbol de la subgráfica inducida por $\{x\} \cup \bigcup_{i>0} S_i(x) \cup \bigcup_{j>0} \overline{S}_j(x)$

if $Z[x]$ *y* $Z^-[x]$ *son ambas nulas* **then**
  return x;
end
if $Z[x]$ *es nula* **then**
  return $x \circledast \text{coarbol}(G, \overline{x}_1, Z, Z^-)$;
end
if $Z[x]^-$ *es nula* **then**
  return $x \circledcirc \text{coarbol}(G, x_1, Z, Z^-)$;
end
if $x_1\overline{x}_1 \in E_G$ **then**
  return $[\ldots[[[x \circledcirc \text{coarbol}(G, x_1, Z, Z^-)] \circledast \text{coarbol}(G, \overline{x}_1, Z, Z^-)] \circledcirc \text{coarbol}(G, x_2, Z, Z^-)] \circledast \ldots]$;
else
  return $[\ldots[[[x \circledast \text{coarbol}(G, \overline{x}_1, Z, Z^-)] \circledcirc \text{coarbol}(G, x_1, Z, Z^-)] \circledast \text{coarbol}(G, \overline{x}_2, Z, Z^-)] \circledcirc \ldots]$;
end

si entramos en el `else`, entonces $\eta(x) - 1 \leq \kappa(x) \leq \eta(x)$. Por lo tanto, las líneas de nuestro algoritmo sí tienen sentido.

Proposición 4.5.16. *El Algoritmo 15 es correcto. Esto es, dados $x \in V_G$, Z y Z^- como lo especifica el algoritmo, se devuelve, en efecto, el coárbol de la subgráfica inducida por $\{x\} \cup \bigcup_{i>0} S_i(x) \cup \bigcup_{j>0} \overline{S}_j(x) = \{x\} \cup S^N(x) \cup \overline{S^N}(x)$.*

Demostración. Sea T el coárbol de G. Como nuestro algoritmo es recursivo, una demostración inductiva es adecuada. La recursión será fuerte y será sobre el número $n - 1 - \sigma(x)$, donde n es el orden de la cográfica, es decir, comenzaremos del último vértice en el ordenamiento σ hacia el atrás. Sea y el último vértice del ordenamiento σ, esto es $\sigma(y) = n - 1$. Como no hay vértices en G que sean σ-mayores a y, pero y es el vértice σ-mínimo tanto de $S(y)$ como de $\overline{S}(y)$, se sigue que $S(y) = \overline{S}(y) = \{y\}$. O sea que coarbol$(G, y, Z, Z^-)$ devuelve y, pues $S^N(y) = \overline{S^N}(y) = \varnothing$. Así que, el enunciado se satisface para y, porque claramente el árbol trivial que solo tiene a y, coincide con ser el coárbol de gráfica inducida por $\{y\} \cup \bigcup_{i>0} S_i(y) \cup \bigcup_{j>0} \overline{S}_j(y) = \{y\}$. Fijemos $x \in V_G$ y supongamos válido el enunciado para todo $v \in V_G$ que cumpla $\sigma(x) < \sigma(v)$. Si $S^N(x) = \overline{S^N}(x) = \varnothing$, claramente, como en el caso base, el coárbol de la subgráfica inducida por $\{x\} \cup \bigcup_{i>0} S_i(x) \cup \bigcup_{j>0} \overline{S}_j(x) = \{x\}$ es x, pero x, a su vez, es lo que devuelve coarbol(G, x, Z, Z^-).

Algo importante que recordar, es que los argumentos usados en el Teorema 4.5.13 nos permiten derivar que $S(x_i) \subseteq \overline{S}(x)$ y $\overline{S}(\overline{x}_j) \subseteq S(x)$, además, también que $\overline{S}(x_i) \setminus S_i(x) \subseteq \overline{S^A}(x_i)$ y $S(\overline{x}_j) \setminus \overline{S}_j(x) \subseteq S^A(\overline{x}_j)$. Por lo tanto, tomando complementos sobre $\overline{S}(x_i)$ y $S(\overline{x}_j)$ respectivamente, obtenemos $\{x_i\} \cup \overline{S^N}(x_i) \subseteq S_i(x) = S(x_i)$ y $\{\overline{x}_j\} \cup S^N(\overline{x}_j) \subseteq \overline{S}_j(x) = \overline{S}(\overline{x}_j)$. Dado lo anterior, merece la pena mencionar también que $\overline{S}(x_i) \subseteq \overline{S}(x)$ y $S(\overline{x}_j) \subseteq S(x)$, porque $\overline{S}(x_i)$ y $S(\overline{x}_j)$ son las celdas, en σ^- y σ respectivamente, mínimas por contención que incluyen a los sendos módulos $S(x_i)$ y $\overline{S}(\overline{x}_j)$. Ahora, supongamos que $S^N(x) \neq \varnothing$ o $\overline{S^N}(x) \neq \varnothing$. Sean $\kappa(x)$ el número de x-celdas respecto a σ y $\eta(x)$ el número de x-celdas respecto a σ^-. Recalquemos la observación que hicimos antes de comenzar, esto es, que gracias a la Proposición 4.5.15, no puede ocurrir $\kappa(x) + 1 < \eta(x)$ o $\eta(x) + 1 < \kappa(x)$. Por lo que, si $\eta(x) = 0$, se sigue que $\kappa(x) = 1$ y por ende, coarbol(G, x, Z, Z^-) sí regresa el coárbol de $\{x\} \cup \bigcup_{i>0} S_i(x) \cup \bigcup_{j>0} \overline{S}_j(x) = \{x\} \cup S_1(x) = \{x\} \cup S(x_1) = \{x\} \cup \{x_1\} \cup \bigcup_{i>0} S_i(x_1) \cup \bigcup_{j>0} \overline{S}_j(x_1)$. Algo similar ocurre cuando $\kappa(x) = 0$, ya que esto implica $\eta(x)$, o sea que $\{x\} \cup \bigcup_{i>0} S_i(x) \cup \bigcup_{j>0} \overline{S}_j(x) = \{x\} \cup \overline{S}(\overline{x}_1)$.

Finalicemos con el caso en el que $Z[x]$ y $Z^-[x]$ son ambas no vacías. La hipótesis inductiva garantiza que para cualesquiera $i, j > 0$ tales que $S_i(x)$ y $\overline{S}_j(x)$ existen, las ejecuciones coarbol(G, x_i, Z, Z^-) y coarbol$(G, \overline{x}_j, Z, Z^-)$ devuelven respectivamente

los coárboles de las subgráficas inducidas por $\{x_i\} \cup \bigcup_{p>0} S_p(x_i) \cup \bigcup_{q>0} \overline{S}_q(x_i)$ y por $\{\overline{x}_j\} \cup \bigcup_{p>0} S_p(\overline{x}_j) \cup \bigcup_{q>0} \overline{S}_q(\overline{x}_j)$. Si $i, j > 0$ son tales que $S_i(x)$ y $\overline{S}_j(x)$ existen, entonces, usando lo que demostramos en el Teorema 4.5.13 y la Proposición 4.5.15, se cumple que $S_i(x) = S(x_i) = H(T^x_{0i}) = \{x_i\} \cup \bigcup_{p>0} S_p(x_i) \cup \bigcup_{q>0} \overline{S}_q(x_i)$ y $\overline{S}_j(x) = \overline{S}(\overline{x}_j) = H(T^x_{1j}) = \{\overline{x}_j\} \cup \bigcup_{p>0} S_p(\overline{x}_j) \cup \bigcup_{q>0} \overline{S}_q(\overline{x}_j)$. Por lo tanto, coarbol$(G, x, Z, Z^-)$ devuelve el coárbol de la subgráfica inducida por $\{x\} \cup \bigcup_{p>0} S_p(x) \cup \bigcup_{q>0} \overline{S}_q(x)$. Lo anterior, es porque el coárbol de esta última subgráfica lo obtenemos enraizando alternadamente los árboles T^x_{0i} y T^x_{1j} sobre la rama P_x, y por el Teorema 4.5.13, sabemos que los conjuntos de hojas de estos árboles coinciden con $S_i(x)$ y $\overline{S}_j(x)$ respectivamente. También utilizamos el Corolario 4.5.7 para determinar si el padre de x en el coárbol construido, es el primer nodo en P_x con etiqueta 0, o es el primer nodo en P_x con etiqueta 1. ■

Proposición 4.5.17. *Sean G una cográfica, σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento que devuelve LexBFS$^-(G, \sigma)$. Si $x \in V_G$, coarbol(G, x, Z, Z^-) toma tiempo $\mathcal{O}(|S(x)| + |\overline{S}(x)| + \sum_{y \in S^N(x)} d(y) + \sum_{y \in \overline{S^N}(x)} d(y) + d(x))$.*

Demostración. Antes de comenzar, observemos varias cosas para un vértice cualquiera $y \in V_G$. Encontrar a y_1 y $\overline{y}_1$, en caso de que existan, toma tiempo constante; esto porque y_1 es el primer vértice de $Z[y]$, mientras que $\overline{y}_1$ es el primer vértice de $Z^-[y]$. También, notemos que $\overline{y}_1, \ldots, \overline{y}_{|Z^-[y]|}$ pertenecen todos a $N(y)$ por definición. Por lo tanto, de acuerdo con la Proposición 4.5.15, tenemos que alguno de los pares de desigualdades $|Z[y]| \leq |Z^-[y]| \leq |Z[y]| + 1$, o $|Z^-[y]| \leq |Z[y]| \leq |Z^-[y]| + 1$ se satisface, así que tenemos $|Z[y]| + |Z^-[y]| = |\{\overline{y}_1, \ldots, \overline{y}_{|Z^-[y]|}\}| + |\{y_1, \ldots, y_{|Z[y]|}\}| \leq 2|Z^-[y]| + 1 \leq 2d(y) + 1$. Utilizando inducción como en la prueba de correctud, tenemos que, si x es el último vértice de σ, entonces el algoritmo solo verifica que $S^N(x)$ y $\overline{S^N}(x)$ son vacías y devuelve x, lo cual toma tiempo constante. Ahora supongamos que $\sigma(x) < n - 1$. Si $Z[x]$ y $Z^-[x]$ son ambas vacías, ocurre lo mismo que en el caso base, y por ende, la ejecución devuelve a x en tiempo constante.

Algo importante que recordar, es que los argumentos usados en el Teorema 4.5.13 nos permiten derivar que $S(x_i) \subseteq \overline{S}(x)$ y $\overline{S}(\overline{x}_j) \subseteq S(x)$, además, también que $\overline{S}(x_i) \setminus S_i(x) \subseteq \overline{S^A}(x_i)$ y $S(\overline{x}_j) \setminus \overline{S}_j(x) \subseteq S^A(\overline{x}_j)$. Por lo tanto, tomando complementos sobre $\overline{S}(x_i)$ y $S(\overline{x}_j)$ respectivamente, obtenemos $\{x_i\} \cup \overline{S^N}(x_i) \subseteq S_i(x) = S(x_i)$ y $\{\overline{x}_j\} \cup S^N(\overline{x}_j) \subseteq \overline{S}_j(x) = \overline{S}(\overline{x}_j)$. Dado lo anterior, merece la pena mencionar también que $\overline{S}(x_i) \subseteq \overline{S}(x)$ y $S(\overline{x}_j) \subseteq S(x)$, porque $\overline{S}(x_i)$ y $S(\overline{x}_j)$ son las celdas, en σ^- y σ respectivamente, mínimas por contención que incluyen a los sendos módulos $S(x_i)$ y $\overline{S}(\overline{x}_j)$. Si $Z[x]$ es nula, pero $Z^-[x]$ no lo es, la ejecución toma una unidad de tiempo para verificar lo anterior, más los pasos que tome la llamada recursiva a coarbol$(G, \overline{x}_1, Z, Z^-)$. Por hipótesis inductiva, esa última ejecución toma

tiempo $\mathcal{O}(|S(\overline{x}_1)| + |\overline{S}(\overline{x}_1)| + \sum_{y\in S^N(\overline{x}_1)} d(y) + \sum_{y\in\overline{S^N}(\overline{x}_1)d(y)} + d(\overline{x}_1))$, pero, por lo desarrollado en el Teorema 4.5.13, este tiempo está acotado asintóticamente por $\mathcal{O}(|S(x)| + |\overline{S}(x)| + \sum_{y\in\overline{S}(\overline{x}_1)=\overline{S^N}(x)} d(y))$. Algo similar ocurre en el caso en el que $Z^-[x]$ es nula y $Z[x]$ no lo es. Ahora, solo nos queda por ver el caso en el que ni $Z[x]$ ni $Z^-[x]$ son vacías. De nuevo, por la argumentación efectuada en el Teorema 4.5.13 junto con el hecho de que las celdas son ajenas dos a dos, tenemos las siguientes desigualdades: $\sum_{i>0} |S(x_i)| = \sum_{i>0} |S_i(x)| \leq S^N(x)$; $\sum_{i>0} |\overline{S}(x_i)| \leq |\overline{S^A}(x)|$; $\sum_{j>0} |\overline{S}(\overline{x}_j)| = \sum_{j>0} |\overline{S}_j(x)| \leq |\overline{S^N}(x)|$; y $\sum_{j>0} |S(\overline{x}_j)| \leq |S^A(x)|$. Además, también se cumple para cualesquiera $i, j > 0$, que $\sum_{y\in S^N(x_i)} d(y) + \sum_{y\in\overline{S^N}(x_i)} d(y) \leq \sum_{y\in S_i(x)} d(y)$; y $\sum_{y\in S^N(\overline{x}_j)} d(y) + \sum_{y\in\overline{S^N}(\overline{x}_j)} d(y) \leq \sum_{y\in\overline{S}_j(x)} d(y)$. Notemos también que $x_1, \ldots, x_{|Z[x]|}$ son vértices en $S^N(x)$, y que $\overline{x}_1, \ldots, \overline{x}_{|Z^-[x]|}$ son vértices en $\overline{S^N}(x)$. Por lo tanto, usando la hipótesis inductiva, deducimos que el tiempo que lleva a cabo calcular todas las ejecuciones coarbol(G, x_i, Z, Z^-) y coarbol$(G, \overline{x}_j, Z, Z^-)$ es $\mathcal{O}(|S(x)| + |\overline{S}(x)| + \sum_{y\in S^N(x)} d(y) + \sum_{y\in\overline{S^N}(x)} d(y))$. Por otro lado, los constructores $\circledast$ y $\circledcirc$ se utilizan tantas veces como $|Z[x]| + |Z^-[x]| - 1 \leq 2d(x)$. Por lo tanto, el tiempo que utiliza toda la ejecución coarbol(G, x, Z, Z^-), es $\mathcal{O}(|S(x)| + |\overline{S}(x)| + (\sum_{y\in S^N(x)} d(y)) + (\sum_{y\in\overline{S^N}(x)} d(y)) + d(x))$. ■

Corolario 4.5.18. *Sean G una cográfica, σ un ordenamiento LexBFS de V_G y σ^- el ordenamiento que devuelve LexBFS$^-(G, \sigma)$. Si x es el primer vértice de σ, entonces la aplicación del Algoritmo 15 con x, es decir coarbol(G, x, Z, Z^-), construye el coárbol de G en tiempo $\mathcal{O}(n + m)$, donde n y m son el orden y el tamaño de G respectivamente.*

Entonces, nuestro algoritmo de reconocimiento lineal para cográficas queda como se observa en el Algoritmo 16.

De los Corolarios 4.3.17 y 4.5.18, junto con la Proposición 4.4.2, se desprende directamente que el Algoritmo 16 es correcto y es $\mathcal{O}(n + m)$, donde n y m son el orden y el tamaño de G respectivamente.

Algoritmo 16: esCografica(G, τ).

Input: Una gráfica G representada por sus listas de adyacencias y τ un ordenamiento de V_G

Output: Si G es una cográfica, una fórmula utilizando los constructores $\circledast$ y $\circledcirc$ que representa al coárbol de G; si G no es cográfica, los vértices de un P_4 inducido en G

1 /* Recordemos que Z y Z^- son arreglos tales que $Z[x] = [x_1, x_2, \dots]$ y $Z^-[\overline{x}_1, \overline{x}_2, \dots]$ */
2 Ejecutar NSP(G, τ) y guardar el resultado en $(\sigma, (x, r), Z)$;
3 Ejecutar NSP$^-(G, \sigma)$ y guardar el resultado en $(\sigma^-, (y, t), Z^-)$;
4 /* Si σ no satisface NSP */
5 **if** $(x, r) \neq (-1, -1)$ **then**
6 | calcular $Q \leftarrow N_<(\{x_r\}) \cap S(x)$;
7 | calcular $R \leftarrow N_<(\{x_{r+1}\}) \cap S(x)$;
8 | Ejecutar reporta-$P_4(G, \sigma, x, r, x_r, x_{r+1}, Q, R)$ y guardar el resultado en (u_1, u_2, u_3, u_4);
9 | **return** (u_1, u_2, u_3, u_4);
10 **end**
11 /* Si σ^- no satisface NSP */
12 **if** $(y, t) \neq (-1, -1)$ **then**
13 | calcular $Q \leftarrow N_<(\{\overline{y}_t\}) \cap \overline{S}(y)$;
14 | calcular $R \leftarrow N_<(\{\overline{y}_{t+1}\}) \cap \overline{S}(y)$;
15 | Ejecutar reporta-$P_4^-(G, \sigma^-, y, t, \overline{y}_t, \overline{y}_{t+1}, Q, R)$ y guardar el resultado en (u_1, u_2, u_3, u_4);
16 | **return** (u_1, u_2, u_3, u_4);
17 **end**
18 **return** coarbol$(G, \sigma^{-1}(0), Z, Z^-)$;

Ejemplificación del proceso de reconocimiento

En esta parte vamos a revisar someramente, algunos ejemplos del procesamiento del algoritmo de reconocimiento. Vamos a hacer un caso ejecutándolo sobre una cográfica, y otro caso ejecutandolo sobre una gráfica con un P_4 inducido. Las gráficas que usaremos en los ejemplos las podemos ver en la Figura 4.6; la cográfica es G, mientras que la gráfica que tiene un P_4 inducido, y por lo tanto no es cográfica, es H.

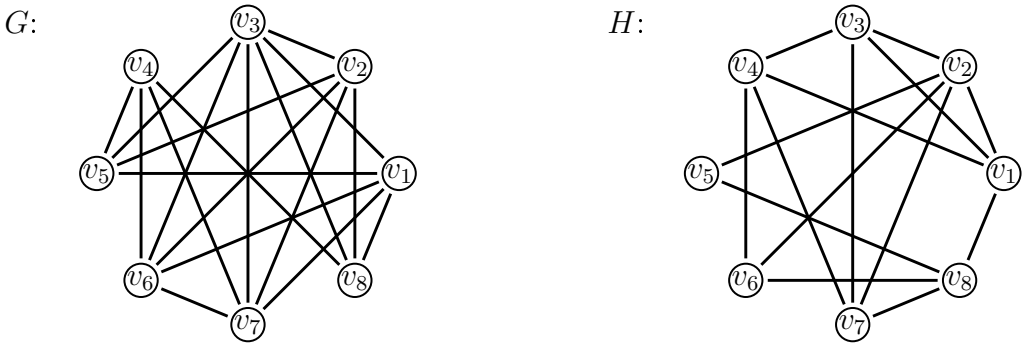


Figura 4.6: Cográfica G y gráfica H con un P_4 inducido.

Primero consideremos la ejecución esCografica(G, τ) utilizando el Algoritmo 16. Vamos a tomar τ como el orden natural de los vértices de G, es decir, $\tau = [v_1, \ldots, v_8]$, suponiendo desde luego que las listas de adyacencias de G aparecen en ese orden.

Lo primero que hacemos es ejecutar NSP(G, τ), o sea, el Algoritmo 11. Este algoritmo, como vimos a lo largo del capítulo previo, combina: la versión lineal de LexBFS (el Algoritmo 8) con los cambios descritos en las páginas 113-118 acoplados, el algoritmo calculaLvec (el Algoritmo 9) trabajando en conjunto con el Algoritmo 10, y por último, las modificaciones mencionadas en la página 152 que permiten calcular

la lista Z. La salida de $\mathrm{NSP}(G,\tau)$ la almacenamos en una tripleta $(\sigma,(x,r),Z)$, veamos explícitamente quiénes son estos objetos. Cabe resaltar que, al principio de la ejecución $\mathrm{NSP}(G,\tau)$, ordenamos las listas de adyacencias de G respecto a τ; una vez hecho esto, las listas quedan como se puede ver en la Figura 4.7.

$$
\begin{array}{lllllll}
v_1\colon & v_3 & v_5 & v_6 & v_7 & v_8 & \\
v_2\colon & v_3 & v_5 & v_6 & v_7 & v_8 & \\
v_3\colon & v_1 & v_2 & v_5 & v_6 & v_7 & v_8 \\
v_4\colon & v_5 & v_6 & v_7 & v_8 & & \\
v_5\colon & v_1 & v_2 & v_3 & v_4 & & \\
v_6\colon & v_1 & v_2 & v_3 & v_4 & v_7 & \\
v_7\colon & v_1 & v_2 & v_3 & v_4 & v_6 & \\
v_8\colon & v_1 & v_2 & v_3 & v_4 & &
\end{array}
$$

Figura 4.7: Listas de adyacencias de G ya τ-ordenadas.

En la Figura 4.8 podemos ver la ejecución de LexBFS tomando como entrada a (G,τ).

$$
\begin{array}{llllll}
L\colon & [\{v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8\}] & & & & \sigma=() \\
L\colon & [\{v_3,v_5,v_6,v_7,v_8\} & \{v_2,v_4\}] & & & \sigma=(v_1) \\
L\colon & [\{v_5,v_6,v_7,v_8\} & \{v_2\} & \{v_4\}] & & \sigma=(v_1,v_3) \\
L\colon & [\{v_6,v_7,v_8\} & \{v_2\} & \{v_4\}] & & \sigma=(v_1,v_3,v_5) \\
L\colon & [\{v_7\} & \{v_8\} & \{v_2\} & \{v_4\}] & \sigma=(v_1,v_3,v_5,v_6) \\
L\colon & [\varnothing & \{v_8\} & \{v_2\} & \{v_4\}] & \sigma=(v_1,v_3,v_5,v_6,v_7) \\
L\colon & [\varnothing & \varnothing & \{v_2\} & \{v_4\}] & \sigma=(v_1,v_3,v_5,v_6,v_7,v_8) \\
L\colon & [\varnothing & \varnothing & \varnothing & \{v_4\}] & \sigma=(v_1,v_3,v_5,v_6,v_7,v_8,v_2) \\
L\colon & [\varnothing & \varnothing & \varnothing & \varnothing] & \sigma=(v_1,v_3,v_5,v_6,v_7,v_8,v_2,v_4)
\end{array}
$$

Figura 4.8: Ejecución del algoritmo LexBFS usando a G y τ como entrada.

De acuerdo con la ejecución de LexBFS con entrada G y τ, pusimos en la Figura 4.9 las celdas y las vecindades izquierdas que necesitamos.

Con un vistazo, podemos verificar que σ satisface NSP, es decir, satisface la propiedad definida en la Definición 4.3.2. Así que, el Corolario 4.3.17 nos dice que

v_1: $S_1(v_1) = \{v_2\}$ $S_2(v_1) = \{v_4\}$
$N_<(v_2) \cap S(v_1) = \{v_3, v_5, v_6, v_7, v_8\}$ $N_<(v_4) \cap S(v_1) = \{v_5, v_6, v_7, v_8\}$
v_3:
v_5: $S_1(v_5) = \{v_6, v_7, v_8\}$
$N_<(v_6) \cap S(v_5) = \varnothing$
v_6: $S_1(v_6) = \{v_8\}$
$N_<(v_8) \cap S(v_6) = \varnothing$
v_7:
v_8:
v_2:
v_4:

Figura 4.9: Celdas $S_i(x)$ y las vecindades izquierdas $N_<(x_i)$.

la pareja (x, r) es la pareja $(-1, -1)$. Por lo demás, podemos ver de las Figuras 4.8 y 4.9 que $\sigma = [v_1, v_3, v_5, v_6, v_7, v_8, v_2, v_4]$ y Z es el arreglo bidimensional tal que

$$Z[v_i] = \begin{cases} (v_2, v_4) & \text{si} \quad i = 1 \\ (v_6) & \text{si} \quad i = 5 \\ (v_8) & \text{si} \quad i = 6 \\ \varnothing & \text{si} \quad i \notin \{1, 5, 6\} \end{cases}$$

Luego, se ejecuta $\text{NSP}^-(G, \sigma)$. Como describimos en la página 129, NSP^- combina LexBFS$^-$-NSP junto con calculaLvec para producir una salida que nos dé: el ordenamiento σ^-, un par ordenado (y, t) con el que vamos a poder saber si σ^- cumple NSP, y, con las modificaciones comentadas en la página 152, también el arreglo Z^-. Al comienzo de la ejecución $\text{NSP}^-(G, \sigma)$ ordenamos las listas de adyacencias de G respecto a σ; una vez hecho esto, las listas quedan como se puede ver en la Figura 4.10.

En la Figura 4.11, podemos ver la ejecución de LexBFS$^-$ usando como entrada a G y σ. Por otro lado. en la Figura 4.12, vemos las celdas respecto a σ^- y las vecindades izquierdas que nos interesan. Recordemos que para $x \in V_G$ e $i > 0$, tenemos que $N_<(\overline{x}_i) \cap \overline{S}(x) = ((\leftarrow, \overline{x}_i)_{<_{\sigma^-}} \setminus \overline{N}_<(\overline{x}_i)) \cap \overline{S}(x)$, donde $(\leftarrow, \overline{x}_i)_{<_{\sigma^-}}$ es el conjunto de vértices σ^--menores a $\overline{x}_i$.

En la Figura 4.12, podemos ver que todas las contenciones que se necesitan para que σ^- cumpla NSP, en efecto se satisfacen. Por lo tanto, el par (y, t) que devuelve $\text{NSP}^-(G, \sigma)$ es el par $(-1, -1)$. Además, $\sigma^- = [v_1, v_2, v_4, v_3, v_5, v_6, v_8, v_7]$ y Z^- es el

$$
\begin{array}{lllllll}
v_1\colon & v_3 & v_5 & v_6 & v_7 & v_8 & \\
v_2\colon & v_3 & v_5 & v_6 & v_7 & v_8 & \\
v_3\colon & v_1 & v_5 & v_6 & v_7 & v_8 & v_2 \\
v_4\colon & v_5 & v_6 & v_7 & v_8 & & \\
v_5\colon & v_1 & v_3 & v_2 & v_4 & & \\
v_6\colon & v_1 & v_3 & v_7 & v_2 & v_4 & \\
v_7\colon & v_1 & v_3 & v_6 & v_2 & v_4 & \\
v_8\colon & v_1 & v_3 & v_2 & v_4 & &
\end{array}
$$

Figura 4.10: Listas de adyacencias de G, pero σ-ordenadas.

$$
\begin{array}{lllll}
L\colon & [\{v_1, v_3, v_5, v_6, v_6, v_8, v_2, v_4\}] & & & \\
 & \sigma^- = () & & & \\
L\colon & [\{v_2, v_4\} & \{v_3, v_5, v_6, v_7, v_8\}] & & \\
 & \sigma^- = (v_1) & & & \\
L\colon & [\{v_4\} & \{v_3, v_5, v_6, v_7, v_8\}] & & \\
 & \sigma^- = (v_1, v_2) & & & \\
L\colon & [\varnothing & \{v_3\} & \{v_5, v_6, v_7, v_8\}] & \\
 & \sigma^- = (v_1, v_2, v_4) & & & \\
L\colon & [\varnothing & \varnothing & \{v_5, v_6, v_7, v_8\}] & \\
 & \sigma^- = (v_1, v_2, v_4, v_3) & & & \\
L\colon & [\varnothing & \varnothing & \{v_6, v_7, v_8\}] & \\
 & \sigma^- = (v_1, v_2, v_4, v_3, v_5) & & & \\
L\colon & [\varnothing & \varnothing & \{v_8\} & \{v_7\}] \\
 & \sigma^- = (v_1, v_2, v_4, v_3, v_5, v_6) & & & \\
L\colon & [\varnothing & \varnothing & \varnothing & \{v_7\}] \\
 & \sigma^- = (v_1, v_2, v_4, v_3, v_5, v_6, v_8) & & & \\
L\colon & [\varnothing & \varnothing & \varnothing & \varnothing] \\
 & \sigma^- = (v_1, v_2, v_4, v_3, v_5, v_6, v_8, v_7) & & &
\end{array}
$$

Figura 4.11: Ejecución del algoritmo LexBFS$^-$ usando a G y σ como entrada.

arreglo bidimensional tal que

$$
Z^-[v_i] = \begin{cases} (v_3, v_5) & \text{si} \quad i = 1 \\ (v_7) & \text{si} \quad i = 6 \\ \varnothing & \text{si} \quad i \notin \{1, 6\} \end{cases}
$$

v_1: $\overline{S}_1(v_1) = \{v_3\}$ $\overline{S}_2(v_1) = \{v_5, v_6, v_8, v_7\}$
$N_<(v_3) \cap \overline{S}(v_1) = \{v_1, v_2\}$ $N_<(v_5) \cap \overline{S}(v_1) = \{v_1, v_2, v_4, v_3\}$
v_2:
v_4:
v_3:
v_5:
v_6: $\overline{S}_1(v_6) = \{v_7\}$
$N_<(v_7) \cap \overline{S}(v_6) = \{v_6\}$
v_8:
v_7:

Figura 4.12: Las celdas $\overline{S}_i(x)$ y las vecindades izquierdas $N_<(\overline{x}_i)$.

Dado que $(x, r) = (y, t) = (-1, -1)$, la ejecución esCografica(G, τ) no entra a ninguno de los bloques `if`, y por ende, devuelve el resultado de la llamada a la rutina coarbol, o sea, el Algoritmo 15. El historial de la ejecución coarbol(G, v_1, Z, Z^-) lo podemos encontrar en la Figura 4.13; omitimos en dicha figura los argumentos Z y Z^-, pues no aportan nada para entender mejor el historial.

1: coarbol(G, v_1)
2: $[[[v_1 \circledcirc \text{coarbol}(G, v_2)] \circledast \text{coarbol}(G, v_3)] \circledcirc \text{coarbol}(G, v_4)] \circledast \text{coarbol}(G, v_5)$
3: $[[[v_1 \circledcirc v_2] \circledast v_3] \circledcirc v_4] \circledast [v_5 \circledcirc \text{coarbol}(G, v_6)]$
4: $[[[v_1 \circledcirc v_2] \circledast v_3] \circledcirc v_4] \circledast [v_5 \circledcirc [[v_6 \circledast \text{coarbol}(G, v_7)] \circledcirc \text{coarbol}(G, v_8)]]$
5: $[[[v_1 \circledcirc v_2] \circledast v_3] \circledcirc v_4] \circledast [v_5 \circledcirc [[v_6 \circledast v_7] \circledcirc v_8]]$

Figura 4.13: Ejecución de coarbol(G, v_1, Z, Z^-).

Por lo tanto, de acuerdo con la Figura 4.13, la expresión que devuelve la ejecución esCografica(G, τ), y por ende, la que define al coárbol de G en la Figura 4.6, es: $[[[v_1 \circledcirc v_2] \circledast v_3] \circledcirc v_4] \circledast [v_5 \circledcirc [[v_6 \circledast v_7] \circledcirc v_8]]$. A esta fórmula le corresponde el coárbol de la Figura 4.14.

Con esto, damos por concluido el caso donde la gráfica que pasamos como entrada al algoritmo sí es una cográfica, y pasamos al caso donde no es cográfica. La primera parte, la de las llamadas a NSP y a NSP$^-$, van a ser muy similares a las del caso anterior; pero vamos a poner todas las ejecuciones completas, justo para observar en

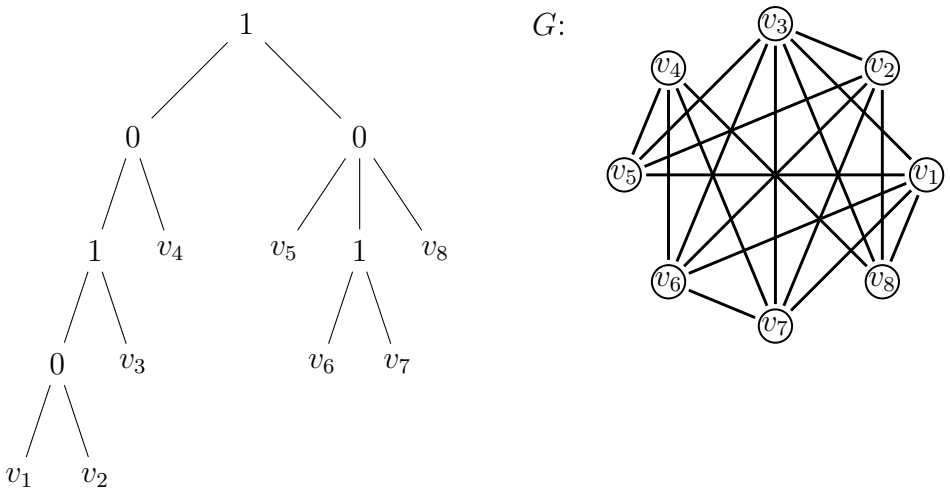


Figura 4.14: Coárbol de la expresión $[[[v_1 \circledcirc v_2] \circledast v_3] \circledcirc v_4] \circledast [v_5 \circledcirc [[v_6 \circledast v_7] \circledcirc v_8]]$, es decir, de G.

donde se rompe la propiedad NSP. Recordemos que vamos a utilizar a la gráfica H de la Figura 4.6.

Comenzamos, como en el caso anterior, con la ejecución esCografica(H, τ). Nótese que vamos a pasar como entrada al mismo ordenamiento $\tau = [v_1, \ldots, v_8]$, pues es el ordenamiento natural de los vértices de H. Una vez que se ejecuta la línea NSP(H, τ), se ordenan las listas de adyacencias de H quedando como están en la Figura 4.15.

$$
\begin{array}{lllllll}
v_1\colon & v_2 & v_3 & v_4 & v_8 & & \\
v_2\colon & v_1 & v_3 & v_5 & v_6 & v_7 & \\
v_3\colon & v_1 & v_2 & v_4 & v_7 & & \\
v_4\colon & v_1 & v_3 & v_6 & v_7 & & \\
v_5\colon & v_2 & v_8 & & & & \\
v_6\colon & v_2 & v_4 & & & & \\
v_7\colon & v_2 & v_3 & v_4 & v_8 & & \\
v_8\colon & v_1 & v_5 & v_7 & & &
\end{array}
$$

Figura 4.15: Listas de adyacencias de H ya τ-ordenadas.

Como ya rememoramos en el caso anterior, la ejecución del Algoritmo 11 en la ejecución NSP(H, τ), conjuga varias subrutinas para poder retribuir como salida una

tripleta $(\sigma, (x, r), Z)$. Las componentes de esta tripleta de salida son: el ordenamiento $\sigma = [v_1, v_2, v_3, v_4, v_8, v_7, v_6, v_5]$ que se genera en la ejecución LexBFS(H, τ), esta ejecución la podemos ver en la Figura 4.16; el par (x, r), que en este caso va a ser $(-1, -1)$, ya que, calculando las celdas y las vecindades izquierdas necesarias como se muestra en las Figuras 4.17 y 4.18, podemos darnos cuenta, usando la observación en 129, de que σ sí satisface NSP. Por último, Z es el arreglo bidimensional que guarda a los σ-mínimos de las celdas, o sea, queda como sigue:

$$Z[v_i] = \begin{cases} (v_7, v_6, v_5) & \text{si} \quad i = 1 \\ (v_4, v_8) & \text{si} \quad i = 2 \\ \varnothing & \text{si} \quad i \notin \{1, 2\} \end{cases}$$

L: $[\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}]$
$\sigma = ()$

L: $[\{v_2, v_3, v_4, v_8\} \quad \{v_5, v_6, v_7\}]$
$\sigma = (v_1)$

L: $[\{v_3\} \quad \{v_4, v_8\} \quad \{v_5, v_6, v_7\}]$
$\sigma = (v_1, v_2)$

L: $[\varnothing \quad \{v_4\} \quad \{v_8\} \quad \{v_7\} \quad \{v_5, v_6\}]$
$\sigma = (v_1, v_2, v_3)$

L: $[\varnothing \quad \varnothing \quad \{v_8\} \quad \{v_7\} \quad \{v_6\} \quad \{v_5\}]$
$\sigma = (v_1, v_2, v_3, v_4)$

L: $[\varnothing \quad \varnothing \quad \varnothing \quad \{v_7\} \quad \{v_6\} \quad \{v_5\}]$
$\sigma = (v_1, v_2, v_3, v_4, v_8)$

L: $[\varnothing \quad \varnothing \quad \varnothing \quad \varnothing \quad \{v_6\} \quad \{v_5\}]$
$\sigma = (v_1, v_2, v_3, v_4, v_8, v_7)$

L: $[\varnothing \quad \varnothing \quad \varnothing \quad \varnothing \quad \varnothing \quad \{v_5\}]$
$\sigma = (v_1, v_2, v_3, v_4, v_8, v_7, v_6)$

L: $[\varnothing \quad \varnothing \quad \varnothing \quad \varnothing \quad \varnothing \quad \varnothing]$
$\sigma = (v_1, v_2, v_3, v_4, v_8, v_7, v_6, v_5)$

Figura 4.16: Ejecución del algoritmo LexBFS usando a H y τ como entrada.

Después, se ejecuta la llamada NSP$^-(H, \sigma)$, la cual, como ya se repasó en el caso previo, devuelve la tripleta $(\sigma^-, (y, t), Z^-)$. En esta llamada se ordenan las listas de adyacencias de H respecto a σ como se puede observar en la Figura 4.19. Muy similar a la llamada de NSP, tenemos que $\sigma^- = [v_1, v_7, v_6, v_5, v_3, v_4, v_8, v_2]$ es el ordenamiento

v_1: $S_1(v_1) = \{v_7\}$ $S_2(v_1) = \{v_6\}$ $S_3(v_1) = \{v_5\}$
v_2: $S_1(v_2) = \{v_4\}$ $S_2(v_2) = \{v_8\}$
v_3:
v_4:
v_8:
v_7:
v_6:
v_5:

Figura 4.17: Celdas $S_i(x)$.

v_1: $N_<(v_7) \cap S(v_1) = \{v_2, v_3, v_4, v_8\}$ $N_<(v_6) \cap S(v_1) = \{v_2, v_4, v_8\}$ $N_<(v_5) \cap S(v_1) = \{v_2, v_8\}$
v_2: $N_<(v_4) \cap S(v_2) = \{v_3\}$ $N_<(v_8) \cap S(v_2) = \varnothing$
v_3:
v_4:
v_8:
v_7:
v_6:
v_5:

Figura 4.18: Vecindades izquierdas $N_<(x_i)$.

que arroja la ejecución de LexBFS$^-$ usando como entrada a H y σ. En la Figura 4.20 podemos ver el historial de esta ejecución. En la Figura 4.21 podemos ver el cálculo de las celdas. Como solo se generan celdas para v_1, vamos a omitir la tabla con el cálculo de las vecindades izquierdas, pues este cálculo solo se realiza para las v_1-celdas.

Resulta entonces, con la información de la Figura 4.21 y notando que $v_8 <_{\sigma^-} v_2$, que el arreglo bidimensional Z^- queda como sigue:

$$Z^-[v_i] = \begin{cases} (v_3, v_4, v_8) & \text{si} \quad i = 1 \\ \varnothing & \text{si} \quad i \neq 1 \end{cases}$$

De acuerdo con las celdas, sus σ^--mínimos en la Figura 4.21, y la observación en 129, vamos a tener las vecindades izquierdas siguientes

v_1: v_2 v_3 v_4 v_8
v_2: v_1 v_3 v_7 v_6 v_5
v_3: v_1 v_2 v_4 v_7
v_4: v_1 v_3 v_7 v_6
v_5: v_2 v_8
v_6: v_2 v_4 v_8
v_7: v_2 v_3 v_4 v_8
v_8: v_1 v_7 v_6 v_5

Figura 4.19: Listas de adyacencias de H, pero σ-ordenadas.

L: $[\{v_1, v_2, v_3, v_4, v_8, v_7, v_6, v_5\}]$
$\sigma^- = ()$
L: $[\{v_7, v_6, v_5\}$ $\{v_2, v_3, v_4, v_8\}]$
$\sigma^- = (v_1)$
L: $[\{v_6, v_5\}$ $\{v_2, v_3, v_4, v_8\}]$
$\sigma^- = (v_1, v_7)$
L: $[\{v_5\}$ $\{v_3\}$ $\{v_2, v_4, v_8\}]$
$\sigma^- = (v_1, v_7, v_6)$
L: $[\varnothing$ $\{v_3\}$ $\{v_4\}$ $\{v_2, v_8\}]$
$\sigma^- = (v_1, v_7, v_6, v_5)$
L: $[\varnothing$ $\varnothing$ $\{v_4\}$ $\{v_8\}$ $\{v_2\}]$
$\sigma^- = (v_1, v_7, v_6, v_5, v_3)$
L: $[\varnothing$ $\varnothing$ $\varnothing$ $\{v_8\}$ $\{v_2\}]$
$\sigma^- = (v_1, v_7, v_6, v_5, v_3, v_4)$
L: $[\varnothing$ $\varnothing$ $\varnothing$ $\varnothing$ $\{v_2\}]$
$\sigma^- = (v_1, v_7, v_6, v_5, v_3, v_4, v_8)$
L: $[\varnothing$ $\varnothing$ $\varnothing$ $\varnothing$ $\varnothing]$
$\sigma^- = (v_1, v_7, v_6, v_5, v_3, v_4, v_8, v_2)$

Figura 4.20: Ejecución del algoritmo LexBFS$^-$ usando a H y σ como entrada.

$$N_<(v_3) \cap \overline{S}(v_1) = \{v_1, v_7\}$$
$$N_<(v_4) \cap \overline{S}(v_1) = \{v_1, v_3, v_6, v_7\}$$
$$N_<(v_8) \cap \overline{S}(v_1) = \{v_1, v_5, v_6, v_7\}$$

v_1: $\overline{S}_1(v_1) = \{v_3\}$ $\overline{S}_2(v_1) = \{v_4\}$ $\overline{S}_3(v_1) = \{v_2, v_8\}$
v_7:
v_6:
v_5:
v_3:
v_4:
v_8:
v_2:

Figura 4.21: Celdas $\overline{S}_i(x)$.

Podemos notar que σ^- no cumple la propiedad NSP con el vértice v_1, pues, si la cumpliera, se debería tener $N_<(v_4) \cap \overline{S}(v_1) = \{v_1, v_3, v_6, v_7\} \subsetneq N_<(v_8) \cap \overline{S}(v_1) = \{v_1, v_5, v_6, v_7\}$, pero esto no ocurre. Por lo que, el par (y, t) que regresa la ejecución $\text{NSP}^-(H, \sigma)$ es $(v_1, 2)$; siendo más específicos, regresa el par $(1, 2)$, ya que v_1 se codifica como 1 respecto a la enumeración natural de las listas de adyacencias en H. Por lo tanto, la ejecución esCografica(H, τ) no entra al primer `if`, porque $(x, r) = (-1, -1)$, pero sí va a entrar al segundo `if`, debido a que $(y, t) \neq (-1, -1)$.

Una vez dentro del segundo `if`, se calculan las listas Q y R utilizando a los vértices $\overline{y}_t = (v_1)_2 = v_4$ y $\overline{y}_{t+1} = (v_1)_{2+1} = v_8$. Es decir, calculamos

$$Q \leftarrow N_<(v_4) \cap \overline{S}(v_1) = [v_1, v_7, v_6, v_3] \text{ y } R \leftarrow N_<(v_8) \cap \overline{S}(v_1) = [v_1, v_7, v_6, v_5]$$

Luego, ejecutamos reporta-$P_4^-(H, \sigma^-, v_1, 2, v_4, v_8, Q, R)$. En esta llamada, siguiendo el pseudo-código de el Algoritmo 14, se calculan, usando la rutina complemento (el Algoritmo 13), las listas $A \leftarrow [v_3, v_4]$ y $B \leftarrow [v_5, v_4]$; estas listas son los complementos en $[v_1, v_8)_{<_{\sigma^-}}$, de R y Q respectivamente. En seguida, calculamos las listas $L_1 \leftarrow [v_4, v_3]$ y $L_2 \leftarrow [v_4, v_5]$ que son las sendas reversas de A y de B. Después, usamos la subrutina contiene (el Algoritmo 10) para hallar el primer elemento de la lista L_1 que no está presente en L_2, que, en este caso es el vértice v_3. Este v_3, resulta ser el σ^--máximo que es menor a v_8, y que además pertenece al conjunto $(\overline{N}_<(v_8) \cap \overline{S}(v_1)) \setminus (\overline{N}_<(v_4) \cap \overline{S}(v_1))$. De nuevo, utilizamos el Algoritmo 10, esta vez para encontrar el primer elemento de B que no esté presente en A, o sea, para hallar v_5 en este caso. Este v_5 es el vértice σ^--mínimo en $\overline{S}(v_1)$ que distingue en $\overline{H}$ a v_4 y v_8, siendo más específicos, que es adyacente en $\overline{H}$ a v_4, pero no lo es a v_8.

Por último, fragmentamos en casos para ver que es lo que devuelve la rutina. El primer caso es que v_1 no esté en $\overline{N}(v_3)$, o sea, en el complemento de $N(v_3)$. Lo

anterior es equivalente a que v_1 y v_3 sean vecinos en H. Como esto sí ocurre, entramos a ese `if`, y por ende, regresamos a la tupla (v_1, v_5, v_3, v_8) como salida de la llamada reporta-$P_4^-(H, \sigma^-, v_1, 2, v_4, v_8, Q, R)$. Claramente, los vértices v_1, v_5, v_3, v_8 inducen un P_4 en la gráfica H de la Figura 4.6, pues la tupla (v_1, v_5, v_3, v_8) es un P_4 inducido en la gráfica $\overline{H}$.

Conclusiones

Cerramos esta obra con un par de observaciones, que evidentemente, tienen como fin resaltar su mucha o poca contribución en el medio. A decir verdad, los únicos capítulos con un aporte fuerte son los que refieren a las cográficas y al algoritmo de reconocimiento, pues, aunque en los capítulos de preliminares haya habido un intento por introducir los temas con el estilo del autor, son temas que claramente se encuentran en casi cualquier texto sobre teoría de gráficas o sobre complejidad computacional.

Sin embargo, en los capítulos restantes, me atrevo a decir que hay contenido más fresco, por decirlo de alguna manera. No conceptos nuevos del todo, puesto que todos los conceptos ya han sido estudiados y utilizados en diversos artículos académicos, por ejemplo, en los que hacemos referencia en la bibliografía; pero sí en el sentido de que la profundidad y el rigor con los que los presentamos conllevan cierto valor. El capítulo de cográficas, en particular la sección del teorema de caracterización, presenta varias demostraciones de mi autoría, sin mencionar que en mi opinión, son demostraciones simples, directas y rigurosas, sin ningún rellano de ambigüedad.

En el capítulo de BFS-Lexicográfico y el algoritmo de reconocimiento, puedo afirmar que se logró un buen puente entre la teoría y la implementación de los algoritmos, ya que en muchas subrutinas se proporciona un pseudo-código claro y riguroso, que puede ser fácilmente traducido a una implementación concreta. Justamenete en esto radica la mayor contribución de este capítulo, en que siempre se tiene en consideración el construir una pauta para una implementación concreta del algoritmo LexBFS y del algoritmo de reconocimiento, sin descuidar la parte teórica de los algoritmos. Como el lector pudo constatar, las pruebas de correctud y los análisis de complejidad para los algoritmos, no siempre son tan directos debido a su complejidad, pero no cejamos en la formalidad y rigor necesarios para poder establecer con una certeza matemática su funcionamiento. A pesar de que el algoritmo de reconocimiento de [2] está definido y explicado en ese artículo, aquí desglosamos todos los puntos acerca de su funcionamiento y su correctud.

Por último, el apéndice donde analizamos dos ejecuciones del algoritmo, también

puede resultar de mucha ayuda para comprender mejor como todas las subrutinas se acoplan en el funcionamiento del algoritmo principal. En conclusión, este trabajo puede resultar de ayuda para cualquier interesado en conocer a detalle el funcionamiento del algoritmo LexBFS o del algoritmo de reconocimiento para cográficas en tiempo lineal; y también para cualquier interesado en llevar a cabo una implementación de estos algoritmos de una manera relativamente simple.

Índice alfabético

Bibliografía

[1] J. A. Bondy y U. S. R. Murty, Graph Theory, Springer-Verlag, 2008.

[2] A. Bretscher, D. Corneil, M. Habib, C. Paul, A Simple Linear Time LexBFS Cograph Recognition Algorithm, SIAM J. Discrete Math. Vol. 22 No. 4 (2008) 1277–1296

[3] C. A. Christen, S. M. Selkow, Some Perfect Coloring Properties of Graphs, Journal of Combinatorial Theory Series B 27 (1979) 49–59.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms 3rd Ed., The MIT Press, Massachusetts Institute of Technology, USA, 2009.

[5] D. G. Corneil, H. Lerchs, L. Stewart. Burlingham, Complement Reducible Graphs, Discrete Applied Mathematics 3 (1981) 163–174.

[6] B. Courcelle, S. Olariu, Upper bounds to the clique width of graphs, Discrete Applied Mathematics 101 (2000) 77–114.

[7] R. Diestel, Graph Theory 5th Ed., Springer-Verlag, 2017.

[8] M. Habib, C. Paul, A simple linear time algorithm for cograph recognition, Discrete Applied Mathematics 145 (2005) 183–197.

[9] M. Sipser, Introduction to the Theory of Computation 2nd Ed., Thomson Course Technology, Boston, USA, 2006.