29 8

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

HEURISTICA PARA COLOREAR LOS VERTICES DE UNA GRAFICA

T E S I S

Que para obtener el Título de

M A T E M A T I C O

P r e s e n t a

Enrique Campos Náñez

México, D. F.

1989

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

TABLA DE CONTENIDOS

❑ INTRODUCCION

Los mapas más antiguos que se conocen son unas tablillas babilónicas cuya elaboración remonta a más de 3000 años antes de nuestra era. No obstante, fueron los astrónomos, geógrafos, filósofos y matemáticos griegos quienes proporcionaron a la cartografía las primeras bases científicas necesarias para plasmar en un mapa las informaciones recogidas por los viajeros [1].

El problema de los cuatro colores, cuyo origen tiene una relación directa con la elaboración de mapas, fué objeto de estudio durante más de cien años. La fuente de del planteamiento de éste, es una carta de Augustus de Morgan a Sir William Rowan Hamilton con fecha 23 de octubre de 1852 [2]. La afirmación de que "es posible colorear una mapa plano utilizando únicamente cuatro colores, la restricción es que regiones con frontera en común posean colores distintos" fué demostrada por [3], interpretando el problema mediante una gráfica de puntos y líneas, e.d. por cada región (país) se asigna un punto (vértice) y dos puntos serán unidos por una línea (arista) si comparten algún fragmento de su frontera. El problema ahora consiste en asignar el mínimo número de colores a los vértices de la gráfica (a lo más cuatro) de manera que, cualquier pareja de vértices unidas por una arista tengan colores distintos. Desde luego que transformar un mapa a una gráfica no fué la única herramienta usada para demostrar la "Conjetura de Guthier"[2] si no la velocidad, memoria y programación de una computadora.

---

1 GRAN ENCICLOPEDIA LAROUSSE. TOMO II. ED. PLANETA 1980 (pp.747-748).

2 Oysten, Ore. THE FOUR-COLOR PROBLEM, Academic Press, NEY YORK-LONDON, 1967.

3 K. Appel, W. Haken, J. Koch, EVERY PLANAR GRAPH IS FOUR COLORABLE, Illinois Journal of Mathematics, 429-567, 1977.

El planteamiento anterior, ha ejercido desde el nacimiento de la **Teoría de Gráficas**, una fascinación constante. Tal vez debido ha su complejidad, o la aparente simpleza en su planteamiento. Probablemente, este quehacer ha entretenido ha generaciones enteras de estudiosos por su principal característica: es como una barra de jabón; no se le puede apretar por ninguna parte. A pesar de que un algoritmo óptimo no ha sido encontrado, del continuo tratar de acorralar el problema han surgido numerosos resultados en otras áreas, como en el caso planteado por M.Chrobak y M.Slusarek[1], que consiste en establecer un algoritmo para empacar óptimamente, en un rectángulo, y cuya solución equivale a colorear los vértices mediante un algoritmo secuencial [2], aplicado a ciertas familias de gráficas.

La relación entre estos dos campos, la Teoría de Gráficas y las Ciencias de la Computación, siempre ha sido en todos los términos prolífica. La computación, por su parte, utiliza algunos de los algoritmos que se han encontrado en la teoría de gráficas, en particular podemos mencionar los relativos para árboles ya que su aplicación en mecanismos de búsqueda y ordenamiento de datos resulta natural.

Como una de las motivaciones para la realización del presente trabajo, se encuentra precisamente la relación entre la teoría de las gráficas y las ciencias de la computación junto con la necesidad de elaborar material de apoyo para la enseñanza e investigación de la primera.

Es así, como se plantea una heurística para colorear los vértices de una gráfica, ordenandolos previamente mediante el cálculo de grados mínimos en ciertas subgráficas generadoras de la original.

Inicialmente fué necesario construir un paquete computacional que integrara las herramientas más comunes para la generación y edición de gráficas así como diseño de pruebas para el criterio planteado. Estas resuelven algunos de los problemas computacionales para trabajar en esta teoría, generando familias de gráficas conocidas y algunos tipos de gráficas

[1] M.Chrobak y M.Slusarek, Problema 84-23, J.Algorithms 5 (1984) 588.
[2] A.Gyarfás y J. Lehel, Journal of Graph Theory, Vol.12, No.,217-227 (1988)

aleatorias. Una de las rutinas principales consiste en la instrumentación de un editor gráfico, que permite crear, modificar y, si es posible, aplanar una gráfica. Finalmente, se construyeron algunas funciones para calcular el grado de un vértice, el grado mínimo, la conexidad y el complemento de una gráfica.

El resultado fué un paquete cuya estructura y definición se incluye también y ocupa otro de los grandes capítulos del presente. Ahí se analizan las posibles representaciones en memoria de una gráfica, el tipo de estructura de datos necesaria para su instrumentación eficiente, las cualidades de cada una, las diferentes funciones y rutinas utilizadas, así como el lenguaje y compilador adecuados. Conjuntamente con lo anterior se ha elaborado un pequeño manual, en donde se explican las diferentes opciones que éste ofrece y su correcto empleo.

Cabe mencionar como uno de los atributos del presente trabajo la facilidad con que puede ser manipulado por personas sin conocimiento alguno de gráficas ni de computación, permitiendo integrar en pequeña medida a la computadora en la investigación de ciertos problemas teóricos de importancia.

CAPITULO 1

Coloración de Gráficas

En el presente capítulo se dan las definiciones básicas para la comprensión y desarrollo del tema así como de algunos resultados sobre algoritmos para coloración de vértices.

Recomendamos consultar los textos de [1] y [2] para los términos que han sido utilizados en el presente trabajo y que no están definidos aquí.

Asignar colores a los vértices de una gráfica no representa mayor dificultad, hacerlo bajo algún criterio nos conduce en algunas ocasiones a tareas difíciles de llevar a cabo. De esta primera observación se deduce la definición siguiente.

Definición: Sea G una gráfica. Una ***coloración*** para los vértices de G, es una asignación de k enteros positivos llamados colores. Se dice que la coloración es ***propia*** si para cualesquier pareja de vértices adyacentes u,v ($u \neq v$) posea colores distintos. De otra manera diremos que la coloración es ***impropia*** (Figura 1.1)

Definición: Una **k-coloración** de G es una coloración, en la que se utilizan solamente k colores o k enteros positivos.

Definición: Decimos que G es **k-coloreable** si puede ser coloreada propiamente con k colores.

Definición: El número cromático de G denotado por $\chi(G)$, corresponde al *mínimo* entero k, tal que G es **k-coloreable**.

---

[1] Curcó Cobos Ma. del Carmen. Notas de clase. Comunicaciones internas del Departamento de Matemáticas, Facultad de Ciencias, U.N.A.M.

[2] F. Harary. Graph Theory, Addison Wesley, 2ª Edición, 1971.

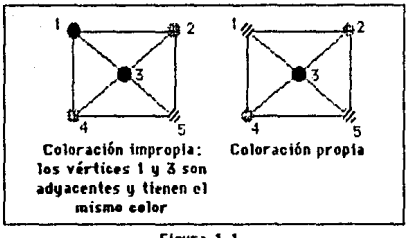
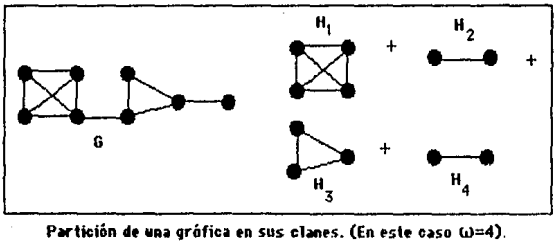


Figura 1.1

Definición: Sea **V** el conjunto de vértices de una gráfica **G**. Un subconjunto **V'⊂V** se dice que es independiente si para cada **u,v∈V', u≠v**, **u** no es adyacente a **v**.

Definición: Una subgráfica **H** es un clan de **G** si para cualesquiera **u,v** vértices de **H**, **u** es adyacente a **v** en **G**.

Definición: $\omega(G) = \max\{|H|\}$, donde H es un clan de G.



Partición de una gráfica en sus clanes. (En este caso ω=4).

Encontrar una $\chi(G)$ coloración para la gráfica **G** resulta ser equivalente a obtener una partición $V_1, V_2, \ldots, V_{\chi(G)}$ de los vértices de **G**, en donde cada v_i con $i=1,2,\ldots,\chi(G)$ es un conjunto independiente (o estable) de cardinalidad máxima. Como un ejemplo de ésto observe la figura 1.2.

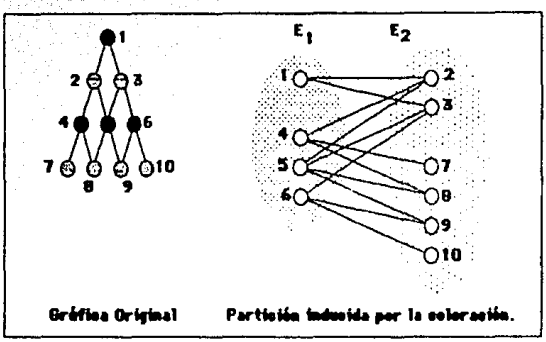


Figura 1.2

Aún que ambos plantemientos son muy simples la naturaleza de estos resulta ser muy intrincada. De hecho, si consideramos la complejidad de los dos problemas alcanzarían la categoría de los más complejos.

Clasificación del Problema de Coloración [3] , [4]

Tomando una clasificación de los problemas -o algorítmos- en base a su complejidad, tenemos una posible partición de éstos, en tres clases principales: P,NP y NP-Completos. (Figura 1.3)

La clase P contiene aquéllos que, una vez planteado un algorítmo para resolverlos, una máquina de Türing (modelo matemático de una computadora) tardará un tiempo equivalente a un polinomio p(n), donde n es el tamaño de la instancia -entrada- de el problema. En este caso se encuentran los siguientes:

- Establecer la conexidad de una gráfica,

- Dados dos vértices de una gráfica G, determinar si existe una trayectoria con extremos en dichos vértices,

- Encontrar un apareamiento de pesos mínimos en una gráfica,

[3] Christos H. Papadimitriou y Keneth Steiglitz, Combinatorial Optimization:Algorithms and Complexity, 342-380, Prentice-Hall.

[4] Sara Baase, Computer Algorithms:Introduction to Design and Analysis,271-272, Addison-Wesley.

o bien,

- Hallar un árbol generador con peso mínimo.

Complementariamente, todos los problemas cuyo tiempo de resolución no puede ser acotado por un polinomio p(n), en el tamaño de la instancia n, pertenecen a la clase NP. Como un ejemplo tenemos el problema de Programación Lineal Entera[3]. Éste consiste en optimizar una función lineal en n variables, sujetas a m restricciones lineales también con la condición adicional de que las soluciones óptimas que se encuentren deberán ser enteras.

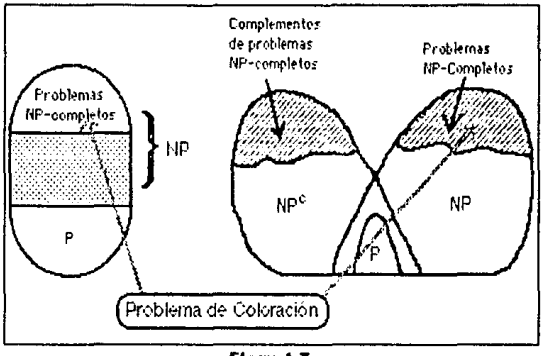


Figura 1.3

Finalmente, y como la más laberíntica, está la clase de los NP-completos. Estos son, en palabras simples, los más difíciles de los problemas de la clase, aquellos que tal vez pueden ser resueltos, en el mejor de los casos, en un tiempo polinomial por una máquina de Türing no determinística. Esto es, la "cabeza" de la máquina tiene una capacidad de "adivinar" o proponer soluciones. Referentes a la teorías de gráficas mencionaremos los siguientes:

- Encontrar el clan más grande,

- El problema del agente viajero,

- Determinar cuando una gráfica es hamiltoniana, y finalmente
- Calcular $\chi(G)$.

Cotas al Número Cromático

Los primeros intentos para acercarse a $\chi(G)$ consistieron en meras aproximaciones, e.d. se obtuvieron algunas cotas para éste. Alguno de los planes consistió en acotarlo mediante la cardinalidad de un clan en G, con mayor número de vértices. Sin embargo, según más tarde se probó, podemos encontrar gráficas con un número cromático tan grande como se desee, en donde el clan más grande tiene dos vértices[6].

Teorema: Para cualquier entero positivo k, existe una gráfica k-cromática, que no contiene triángulos.

Demostración: Para $k=1$ y $k=2$, las gráficas K_1 y K_2 tienen la propiedad requerida. Procederemos por inducción sobre k. Supóngase que hemos construído una gráfica sin triángulos G_k con número cromático $k \geq 2$.

Sean $v_1, v_2, \ldots, v_n$ los vértices de la gráfica G_k. Formamos la siguiente gráfica G_{k+1} como sigue: agregamos $n+1$ vértices nuevos $u_1, u_2, \ldots, u_n, v$, y, para cada $1 \leq i \leq n$, unimos u_i a los vértices adyacentes a v_i y a v. Por ejemplo si G_2 es K_2 entonces G_3 es C_5 (ciclo de 5 vértices) y G_4 la gráfica Grötzsch (ver figura 1.4).

La gráfica G_{k+1} no tiene triángulos, dado que $\{u_1, u_2, \ldots, u_n\}$ es un conjunto independiente en G_{k+1}, por lo que ningún triángulo podrá tener más de un u_i, y si $u_i v_r v_k u_i$ fuera un triángulo en G_{k+1}, $v_i v_r v_k v_i$ sería un triángulo en G_k, lo que contradice la hipótesis.

Probaremos ahora que G_{k+1} es $(k+1)$-cromática. Nótese, primero, que G_{k+1} es ciertamente $(k+1)$-colorable, ya que cualquier k-coloración de G_k puede ser extendida a una $(k+1)$-coloracion de G_{k+1}, coloreando a u_i del mismo color que v_i, $1 \leq i \leq n$, y usando un nuevo color para v. Por lo que nos resta probar que G_{k+1} no es k-coloreable.

Supóngase que G_{K+1} es k-coloreable, y considérese una k-coloración que asigna, sin pérdida de generalidad, el color k a v. Claramente, ningún u_i

6 J.A.Bondy y U.S.R.Murty ,Graph Theory with Applications, 129-131, The Macmillan Press LTD.

puede tener también el color k, ya que todos son adyacentes a v. Por lo tanto son necesarios $k+1$ colores. Por lo que el teorema se sigue □

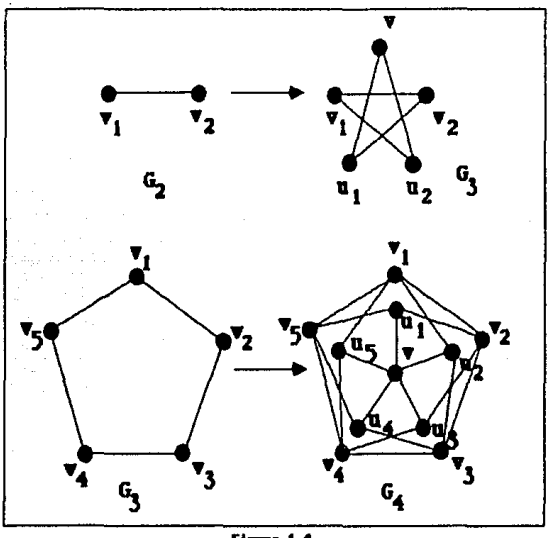


Figura 1.4

Otro tipo de resultados, obtienen cotas al número cromático para gráficas que contengan como gráficas inducidas a otras. Por ejemplo:

Teorema: Si una gráfica G no contiene a $K_{1,3}$ y a $H = (K_2 \cup K_1)+K_2$ como gráficas inducidas, entonces $w(G) \leq \chi(G) \leq w(G)+1$.

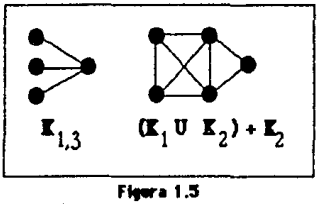


Figura 1.5

O bien el teorema, para gráficas que no contienen al complemento de C_4:

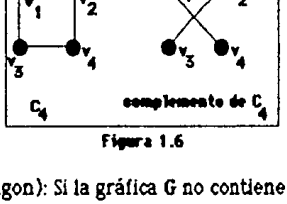


Figura 1.6

Teorema (Stanley Wagon): Si la gráfica G no contiene al complemento de C_4 sin cuerdas como gráfica inducida, entonces $\chi(G) \leq \left\lfloor \frac{w(G)}{2} + 1 \right\rfloor$

Demostración: Sea $w=w(G)$ y A una subgráfica completa de G con $|A|=w$.
Para cada pareja de vértices **a**,**b** V(G), **a**≠**b**, sea C_{ab} el conjunto de vértices cuyos elementos no son adyacentes a **a** ni a **b**. El conjunto C_{ab} es un conjunto independiente, ya que cualquier arista con extremos en C_{ab} formaría, junto con la arista **ab**, el complemento de C_4, lo cual no es posible. De aquí se tiene que $\chi(C) \leq \binom{w}{2}$, si $C=\cup C_{ab}$.

Ahora, sea v en V(G), tal que v no en C ∪ A, entonces v es adyacente a w-1 vértices ya que:
a) Si v es adyacente a todos los vértices de A y A∪{v} es una subgráfica completa de tamaño w+1.
b) Si no fuera adyacente a dos o más vértices de A entonces v estaría en C, lo que contradice nuestra hipótesis.

Definimos el conjunto I_a para cada a en A, como el de aquellos vértices va que no son adyacentes al vértice a y tampoco están en C. Observe que $I_a \cup \{a\}$ es un conjunto independiente, pues si v y w son vértices adyacentes en I_a, entonces $\{v,w\} \cup A-\{a\}$ es una subgráfica completa de tamaño w+1.

De aquí que los vértices que no pertenecen a C pueden ser coloreados por $|A| = w$ colores, y $\chi(G) \leq \binom{w}{2} + w = \left\lfloor \frac{w(G)}{2} + 1 \right\rfloor$, como se desea □

Este tipo de resultados, ofrecen algunas cotas para el número cromático, pero sólo en una familia restringida de gráficas, lo que no ofrece, de ningún modo, una aproximación práctica al problema. Por esto, la mayor parte del desarrollo aplicado en este planteamiento se ha llevado a cabo a través de aproximaciones. Esto es, se construyen algoritmos que ofrezcan una

aproximación "razonable", y se analiza de igual forma el tipo de gráficas donde pueden funcionar.

En el siguiente capítulo se presentan las definiciones formales para algunos de estos algoritmos, los resultados encontrados, así como la heurística -criterio- propuesto y los efectos correspondientes a este nuevo proceso.

CAPITULO 2

Algoritmos de Coloración

Podemos asignar colores a una gráfica de formas muy variadas, desde poner a un vértice el primer color que logre una buena coloración, hasta buscar en los vértices ya coloreados nuevas combinaciones que disminuyan el número de colores utilizados. Las maneras más simples serán aquellas en las que no cambiemos los colores ya asignados. A estos diferentes procesos les llamaremos algoritmos **secuenciales**. A continuación presentamos las definiciones formales correspondientes.

Algoritmos Secuenciales y de PRIMER-COLOR [7]

Definición: Una coloración **SECUENCIAL** (o algoritmo secuencial de coloración) de la gráfica **G** es un proceso mediante el cual la asignación de enteros positivos a los vértices previamente ordenados logra una buena coloración.

Dado el orden $v_1, v_2, ..., v_n$ de los vértices de **G**, para asignar el color a v_i se debe tomar en cuenta unicamente a la subgráfica inducida por el conjunto $\{v_1, v_2, ..., v_i\}$. El entero que corresponde a v_i no cambiará durante el resto del proceso.

Definición: Sea **A** un algoritmo secuencial y consideremos las coloraciones de **G** producidas por **A**, para todas las ordenciones de los vértices de **G**. Llamaremos $\chi_A(G)$ al máximo número de colores utilizados por A en los diferentes ordenes.

Podemos ver este tipo de algoritmos como un juego entre dos personas (**I** y **II**) en el cual **I** da al jugador **II** los vértices uno a uno, para que éste los coloree. Una forma de analizar $\chi_A(G)$ es pensar que el jugador **I** tratará de lograr un orden de los vértices que obligue a **II** a usar un mayor número

[7] A. Gyárfás y J. Lehel, On-Line and First Fit Colorings of Graphs, J. of Graph Theory, Vol.12, No.2, 217-227 (1988).

de colores, y donde **II** no conoce más que los vértices que se le han dado.

De la definición, es claro que $\chi_A(G)$ mide el peor comportamiento de **A** sobre **G**, ya que nos dice el número de colores que se utilizarán en los ordenes donde dicho número . Considerando este hecho $\chi_A(G)$ se analizará para algunas familias de gráficas.

Entre los algoritmos secuenciales el de Primer-Color resulta ser el más simple, dado que asigna el primer color que sea posible a un vértice. A través de este trabajo nos referiremos a él como PC.

Definición: El algoritmo de coloración "PC" es un algoritmo secuencial que recibe los vértices de **G** en un orden $v_1, v_2, ..., v_n$, y les asigna el menor entero positivo posible. Definimos respectivamente a $\chi_{PC}(G)$ como el máximo número de colores que utiliza "PC", dados todos los ordenes de los vértices de **G**.

"PC" produce una secuencia maximal de conjuntos independientes $S_1, S_2, ..., S_k$, donde S_i es un conjunto independiente maximal en la subgráfica inducida por $S_i \cup S_{i+1} \cup ... \cup S_k$, para cada **i**, con $1 \leq i \leq k$. Los conceptos que definiremos enseguida servirán para demostrar algunos resultados sobre $\chi_{PC}(G)$ para distintas familias de gráficas.

Proposición: Si G es una gráfica que no contiene a una trayectoria de longitud 4 (P_4) como subgráfica inducida entonces: $\chi_{PC}(G) = \omega(G)$.
Demostración : Cualquier conjunto independiente maximal de una gráfica que no contiene a P_4 toca todos los clanes maximales de **G**(ver [8]), en un vértice, por lo que el número máximo de conjuntos independientes maximales producidos por la coloración, serán $\omega(G)$. (la figura 2.1 ilustra este hecho).

[8] C.Berge and P.Duchet, Strongly Perfect Graphs. Ann. Discrete Math. 21 (1984) 57-61

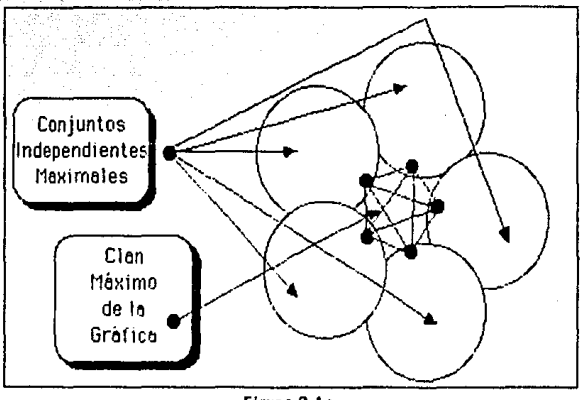
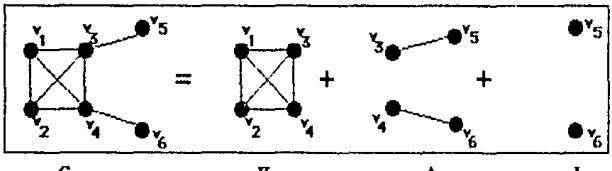


Figura 2.1:
Conjuntos independientes maximales "tocando" al máximo clan de la gráfica.

Definición: Decimos que la gráfica G es una gráfica "dividida" si es la unión de una gráfica completa y un conjunto independiente, por medio de aristas arbitrarias entre ellos figura 2.2.



con A = conjunto de dos aristas y
I = vértices independientes.

Figura 2.2: Gráfica Dividida

Proposición: Si G es una gráfica "dividida" , entonces : $\chi_{PC}(G) \leq \omega(G)+1$.
Demostración:

Por definición de la gráfica G, se tiene que existen K_p una gráfica completa y un conjunto independiente I tales que G es la unión de K_p junto con algunas aristas que tienen un extremo en $V(K_p)$ y el otro en I. De aquí que los conjuntos independientes maximales producidos por PC sean de la siguiente forma:

- Un vértice de K_p junto con vértices de I,
- Vértices de K_p aislados, y finalmente o
- Un sólo conjunto conteniendo vértices de I. En donde cada elemento de I es adyacente a todos los vértices de K_p.

Por lo anterior el máximo número de conjuntos generados es $\omega(G)+1$□

Los teoremas siguientes establecen cotas para $\chi_{PC}(G)$ en ciertas familias de gráficas, mostramos algunos ejemplos de lo tipos de gráficas que ellos mencionan y la cota obtenida para $\chi_{FC}(G)$ en cada caso.

Definición: Dada la gráfica G, decimos que ésta es bipartita si posible encontrar una partición de $V(G)=X \cup Y$, donde X y Y son conjuntos independientes.(Figura 2.3)

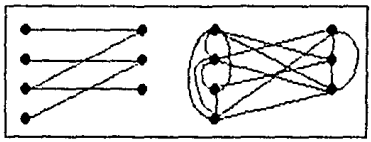


Figura 2.3: Gráfica bipartita y su complemento

Definición: Un conjunto V' de V(G) está saturado por un conjunto F de aristas si cada elemento de V' es extremo de alguna de éstas.

Teorema: Si la gráfica G es complemento de una gráfica bipartita, entonces $\chi_{PC}(G) \leq (3/2)\omega(G)$ y es una buena cota.

Demostración: Sea G una gráfica bipartita y (X, Y) una bipartición. Una partición en clanes maximales de G (y por lo tanto de conj. independientes maximales en G^c) consiste en:

F = conjunto de aristas.

X' = conjunto de vértices no saturados en X.

Y' = conjunto de vértices no saturados en Y.

$|F| + |X'| \leq \omega(G^c)$

$|F| + |Y'| \leq \omega(G^c)$

y finalmente tenemos que,

$|X' \cup Y'| = |X'| + |Y'| \leq \omega(G^c)$, por lo que, $|F|+|X' \cup Y'| \leq (3/2)\omega(G^c)$

$\chi_{PC}(G) \leq (3/2)\omega(G^c)$.

Para ver que es una buena cota, tómese la gráfica bipartita G, con V(G)= A U B U C U D. Y supóngase que lo conjuntos A, B, C y D son ajenos dos a dos, e independientes de tamaño k, y que A U B, B U C y C U D, inducen subgráficas bipartitas completas. En este caso $\chi_{PC}(G)=(3/2)\omega(G^c)$. Ya que $\omega(G^c)=2k$ y $\chi_{PC}(G)=3k$. (ver figura).

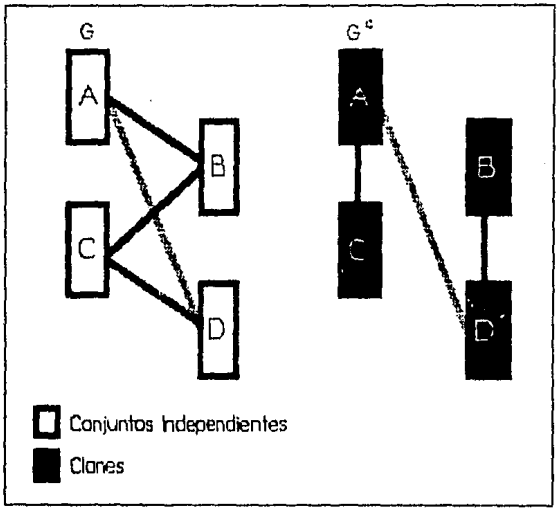


Figura 2.4: Gráfica que cumple con la igualdad en la cota $\chi_{PC}(G) \leq (3/2)\omega(G)$.

Definición: Decimos que G es una gráfica cordal, si no contiene como gráfica inducida un ciclo de longitud k (C_k) con $k \geq 4$.

Teorema: Si G es el complemento de una gráfica cordal entonces:

$$\chi_{PC}(G) \leq 2\omega(G)-1,$$

y existen gráficas que satisfacen la igualdad.

Demostración: Se probará por inducción sobre |V(G)|. El teorema es cierto si G es una gráfica completa. De otra forma sean $C_1, C_2, \ldots, C_k$ una partición en clanes producida por un algoritmo PC en G.

Debemos mostrar que $\omega(G^c) \geq (k+1)/2$. Sea $G^1 = G \setminus V(C_1)$. $C_2, C_3, \ldots, C_k$ es una partición originada por PC en clanes de G^1.

Si G^1 tiene más componentes que G, entonces la afirmación se prueba utilizando la hipótesis de inducción. En otro caso, G contiene un vértice cuyos vecinos inducen un clan, contenido en C_1. Este vértice puede ser agregado a cualquier conjunto independiente máximo de G^1 y de nuevo tenemos que:

$$\omega(G) \geq \omega(G^1) + 1 \geq ((k-1) + 1)/2 + 1 > (k+1)/2.$$

La trayectoria $v_1, v_2, \ldots, v_{3n-1}$ con las aristas $v^3_i\ v^3_{i+2}$ para cada $i=0,1\ldots,n-1$ es un ejemplo de gráfica G, donde $\chi_{PC}(G^c) \leq 2\omega(G^c)-1$ □

Existen, sin embargo, ejemplos donde χ_{PC} diverge. Uno de ellos es el caso de los árboles, que sabemos son 2-coloreables.

Definición: Una gráfica T es un árbol si es una gráfica conexa y no contiene ciclos.

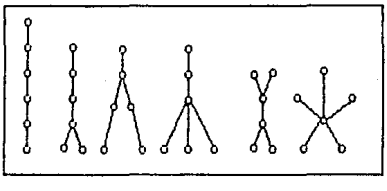


Figura 2.5: Arboles con 6 vértices.

Teorema: Para cada entero positivo **n** existe un árbol T_n tal que $\chi_A(T_n) > n$ para cualquier algoritmo secuencial **A**.

Demostración: En la siguiente demostración, las coloraciones hechas por algoritmos secuenciales serán vistas como un juego entre dos personas, como se describió anteriormente. De acuerdo con este punto de vista, mostraremos una estrategia ganadora para el jugador I, en el sentido de que logre forzar al jugador II a usar el número de colores que se proponga, definiendo árboles ganadores T_n para cada $n=1,2,\ldots$.

Sea T_1 un árbol de un sólo vértice y asúmase que $T_1,\ldots,T_{n-1}$ han sido definidos para lograr $1,\ldots,(n-1)$-coloraciones respectivamente. Construimos T_n como sigue: para cada k, con $1 \leq k \leq n-1$, $|V(T_k)|$, copias distintas de T_k y en cada copia distinguimos un vértice como raíz. El árbol T_n estará formado por la unión de todas estas copias de $T_1,\ldots,T_{n-1}$ con un nuevo vértice **x** unido a cada raíz.

Escogemos la raíz de manera que en cada copia de un mismo árbol tenga un color diferente, forzando de esta manera la existencia de n-1 vértices con distinto color adyacentes a **x**, el color asignado por el jugador II al colorear este último vértice será n □

Estos resultados pueden ser mejorados si fijamos, de alguna manera, el orden en el que el algoritmo PC recibirá los vértices de la gráfica. Así, dando un orden específico para los vértices de una gráfica, y aplicando a éste al algorítmo "PC", generamos un nuevo proceso para la coloración. El planteamiento formal y el análisis de éste es lo que conforma la siguiente sección.

El Ordenamiento por Grados Mínimos

En la sección anterior describimos dos tipos de algorítmos para colorear gráficas: secuenciales y de primer-color. De aquí surgieron preguntas interesantes; ¿Cuál será el comportamiento de cualquiera de éstos al fijar el orden en que se colorean los vértices en el algoritmo PC? y ¿Es posible dar un buen orden para colorearlos?.

Estas interrogantes dieron lugar a una heurística mediante la cual se asigna un orden a los vértices de manera que éste evite "en lo posible" el uso de nuevos colores. El criterio para ordenarlos, está basado en el grado de los vértices, coloreando al final los que tienen el grado mínimo en **G**, buscando así que al observar los vértices ya coloreados, haya entre ellos pocos vecinos. Esta forma de escoger los vértices, visiblemente glotona, mejora algunas de las cotas encontradas para el algoritmo PC

En el párrafo siguiente se describe el procedimiento para ordenar los nodos de una gráfica. Con dicho orden se establece una coloración para los vértices de **G**.

Ordenamiento δ

Sea G una gráfica y $\{v_1, v_2, ..., v_n\}$ los vértices de G.

Construímos $G_0 = G$. Y en general

$G_{i+1} = G_i - v_i$ donde $gr_{G_i}(v_i) = \delta(G_i)$.

Obtenemos el orden asignando el primer lugar al único vértice restante en G_{n-1}, el segundo al vértice que se restó para lograr G_{n-2}, es decir, utilizando el orden inverso al que fueron eliminados.

A esta forma de ordenar los vértices de G le llamaremos un δ-ordenamiento.

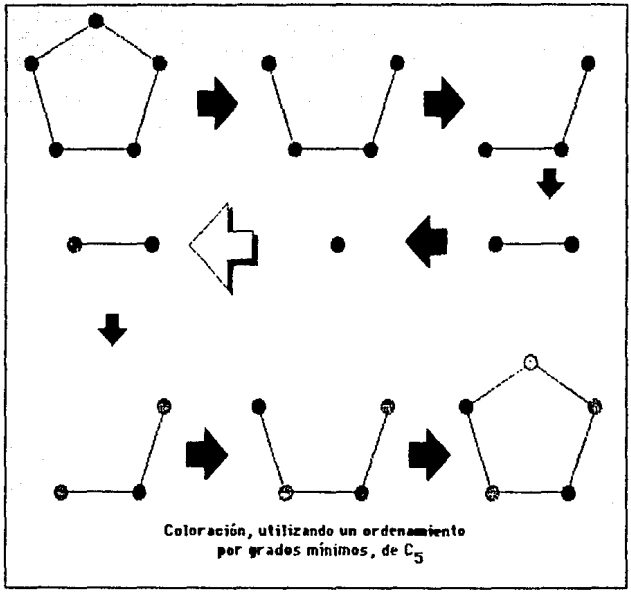


Figura 2.6:
Un ordenamiento δ y la coloración resultante de aplicar PC a este orden..

En la sección siguiente se establecen algunas propiedades sobre un δ-ordenamiento. Como preámbulo se definen conceptos necesarios para el buen entendimiento de éstas.

Definición: Sea **G** una gráfica. El número cromático de **G** respecto a un **δ**-ordenamiento ($\chi_\delta(G)$), como el máximo número de colores utilizados por el ordenamiento sobre todos los ordenes de V(**G**).

Y el número $\delta^* = \max_i \{\delta(G_i)\}$ para $i = 1,2,...,n$ y G_i definidas anteriormente.

Szekeres y Wilf [2] demostraron que para cualquier gráfica **G**, $\chi(G) \leq 1+\max(\delta(G'))$ donde el máximo esta tomado sobre todas las subgráficas G' inducidas de **G**. Un resultado semejante, en el que se establece una cota superior para $\chi_\delta(G)$, esta contenido en la proposición siguiente:

Proposición : Sea **G** una gráfica y $\chi_\delta(G)$ el número cromático de **G** con respecto al ordenamiento δ $\chi_\delta(G) \leq \delta^*+1$.

Demostración:

Sea $u_1,u_2,...,u_n$ el δ-ordenamiento producido en los vértices de **G**.

Al colorear u_i, el algoritmo "PC" contempla únicamente la subgráfica $H=\langle u_1,u_2,...,u_i\rangle$ y asigna a u_i el primer entero que logre una buena coloración de H.

Por definición de δ^*, $gr_H(u_i) \leq \delta^*$. Si se da el caso de que $gr_H(u_i)=\delta^*$ y que los colores 1,2,...,δ^* ya han sido asignados, entonces u_i tomará el color δ^*+1. Como la elección de u_i se hizo de manera arbitraria, el resultado se sigue para toda i=1,...,n □

Lo anterior, nos lleva a resultados inmediatos en algunas familias de gráficas, como por ejemplo, árboles:

Corolario : Dado **T** árbol, $\chi_\delta(T)=2$.

Demostración: Procederemos por inducción sobre el número de vértices del árbol.

En el caso de un sólo vértice se sigue el resultado.

Supongamos que para todo árbol con n vértices se cumple también el resultado.

Sea **T** un árbol con n+1 vértices. Claramente $\delta(T)=1$, y éste grado se alcanza en un vértice extremo de **T**. Por lo que $T_1=T-v$, con $gr(v) = \delta(T)$, es también un árbol. Por lo que coloreamos T1 con a lo más 2 colores, y al colorear el vértice n+1, utilizaremos en el peor de los casos un segundo color □

Corolario : Si **G** es un ciclo, entonces $\chi_\delta(G)=2$ ó $\chi_\delta(G)=3$.

Demostración: Una vez escogido u_1, la subgráfica $H=\langle G-u_1\rangle$ es un árbol, por lo que será coloreado con dos colores únicamente. u_1 es adyacente a dos

[2] F. Harary, Graph Theory. Addison Wesley. 1972 (pp 127-128).

vértices v y w de G. Si el color asignado a éstos es igual, entonces claramente $\chi_\delta(G)=2$. En otro caso $\chi_\delta(G)=3$, cuando v y w tengan colores distintos □

Proposición : Si G es una gráfica dividida, $\chi_\delta(G)=\omega(G)$.

Demostración: Sea G una gráfica dividida.

Por definición existen K_c, S y A, tales que $G=\{K_c \cup S \cup A\}$, con c máximo (por lo que $c=\omega(G)$).

Procederemos por inducción sobre $\omega(G)$. El caso de $\omega(G)=1$ es claro, ya que tenemos una gráfica sin aristas.

Sumpongamos que el resultado es cierto para $\omega(G)=n$, y sea G una gráfica con $\omega(G)=n+1$.

Sea $v_1, v_2, \ldots, v_{n+1}$ un ordenamiento δ, y sea v_i el primer elemento de K_c en ser quitado. La gráfica restante $G_i=G_{i-1}-\{v_i\}$, es tal que:

$$\omega(G_i)=\omega(G)-1,$$

dado que G es también una gráfica dividida. Por lo que, por hipótesis de inducción, $\chi_\delta(G_i)=\omega(G)-1=\omega(G_i)$. Al colorear v_i añadimos un color. Por lo que:

$$\chi_\delta(G_{i-1})=\omega(G).$$

Como los vértices que no han sido coloreados pertenecen a S, y el grado de éstos en G es a lo más c-1, se asigna a cada vértice el color de alguno de los vértices no adyacentes a él en K_c. Por lo tanto:

$$\chi_\delta(G)=\omega(G) \square$$

CAPITULO 3

Estructura del Programa de Cómputo

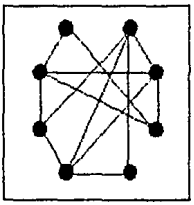
El presente capítulo presenta en cuatro secciones los diferentes aspectos del programa PRAGA (Paquete de Rutinas y Algoritmos para Gráficas), que fué desarrollado para la investigación del problema de coloración y el funcionamiento del ordenamiento planteado en el capítulo anterior.

¿Qué hace el programa PRAGA?

PRAGA es un programa que integra algunas de las herramientas útiles para el trabajo, tanto didáctico como de investigación, en el campo de la Teoría de Gráficas. Este permite llevar a cabo operaciones sobre gráficas como obtener el complemento de éstas, generar gráficas aleatorias, o gráficas específicas, ofreciendo la posibilidad de modificarlas manualmente mediante un editor gráfico y realizar intercambio con la unidad de disco.

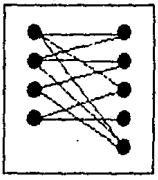
□ Generadores de gráficas:

- Generador de gráficas aleatorias. Este procedimiento recibe como parámetros **p** una probabilidad y **n** el número de aristas que contendrá la gráfica a generarse. p representa, de alguna manera la densidad de la gráfica, esto es, el número de aristas que hay sobre el total posible.



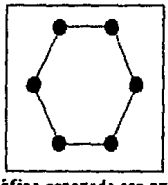
Gráfica generada con parámetros p=5/6 y n=8

• Generador aleatorio de gráficas n-partitas. Los parámetros para esta rutina son: p1 una probabilidad, n el número de partes de la gráfica, v el número de vértices de la gráfica, p2 un vector distribución de probabilidad.



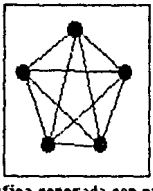
Gráfica generada con p1=0.8, n=2, v=9, p2[1]=0.4 y p2[2] = 0.6.

• Generador de Ciclos: Con único parámetro n el número de vértices del ciclo.



Gráfica generada con n=6.

• Generador de Gráficas completas: Al igual que el anterior con parámetro n = número de vértices.



Gráfica generada con n=5.

• Complemento de una gráfica.

Complemento de C_4.

□ Funciones sobre Gráficas:

- Grado de un vértice.
- Grado mínimo de una gráfica.
- Número de aristas.
- Función booleana que nos indica si existe camino entre dos vértices.

□ Algoritmos

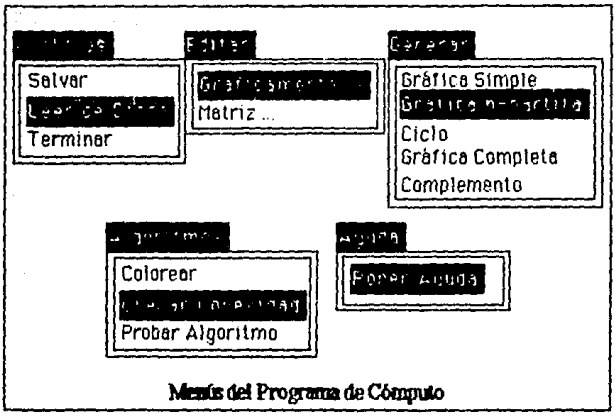
- Conexidad de una gráfica dada.
- Coloración utilizando el ordenamiento mencionado de una gráfica.
- Coloración monitoreada de la gráfica en memoria.

□ Edición manual de una gráfica:

- Capacidad de mover vértices a través de la pantalla.
- Adición de vértices y aristas.
- Facilidad para borrar conjuntos de vértices y aristas.

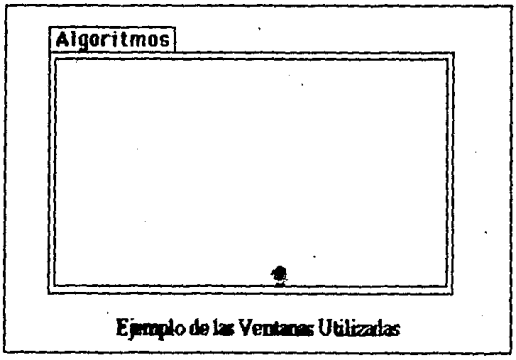
El diseño de un programa que pretende cubrir este tipo de necesidades involucra de alguna manera ciertos problemas de decisión en cuanto a la forma en que manejarán los menús, ventanas y demás opciones que éste contendrá. A continuación se describe de manera corta el tipo de ambientación utilizado para generar el programa final, tomando en cuenta que se buscaba un programa "amigable", que fuera posible usarlo sin conocer nada sobre computación, lo que involucra la construcción de un Ambiente Integrado conteniendo lo siguiente:

- Un sistema de menús "Pull-Down" que hicieran más fáciles de alcanzar las muy variadas opciones del programa. Estos menús clasifican en poco espacio un gran número de opciones sin interferir con el resto de la pantalla. Además, el uso de este tipo de menús, mediante el teclado, es muy sencillo. No es necesario recordar cadenas de claves para llegar a una opción, lo que favorece la "amigabilidad" del programa.



Menús del Programa de Cómputo

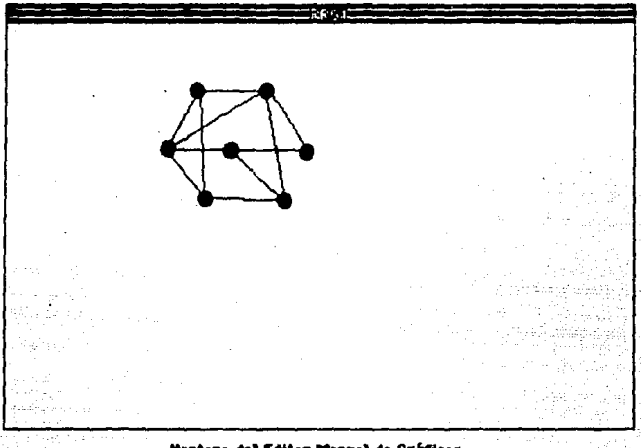
- Un sistema de ventanas, que permitan capturar y ofrecer información, sin destriur el contenido de la pantalla, favoreciendo la velocidad de ejecución del programa, al no modificar continuamente la pantalla.



Ejemplo de las Ventanas Utilizadas

- Un editor manual de la gráfica que se pretende estudiar, el cual permita modificarla moviendo vértices, añadiendo y/o borrando vértices y aristas, etc. Esto permitiría al usuario ver la gráfica que está estudiando, así

como construir sus propios ejemplos manualmente, y realizar algunas operaciones como el aplanamiento de la gráfica (cuando esto es posible).



Ventana del Editor Manual de Gráficas.

• Un menú, basado en íconos (figuras), que indicara las opciones del editor mencionado, al cual llamaremos ***caja de herramientas***. Este permite al usuario ver más claramente en un menú, la opción que desea utilizar.

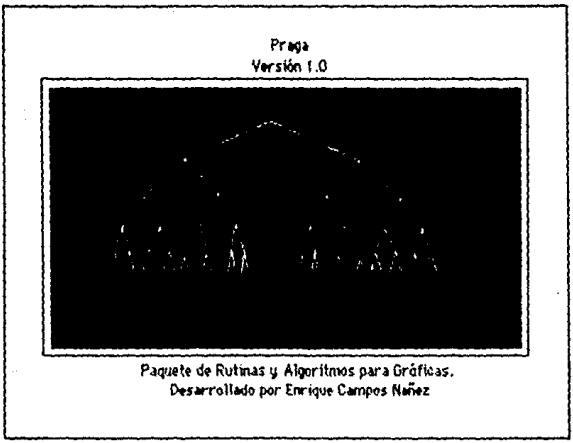
Caja de Herramientas del Editor Gráfico

Operación del Programa

Para utilizar el programa basta cargarlo desde el sistema operativo, después de asegurarse que los siguientes archivos estén en el mismo directorio o disco:

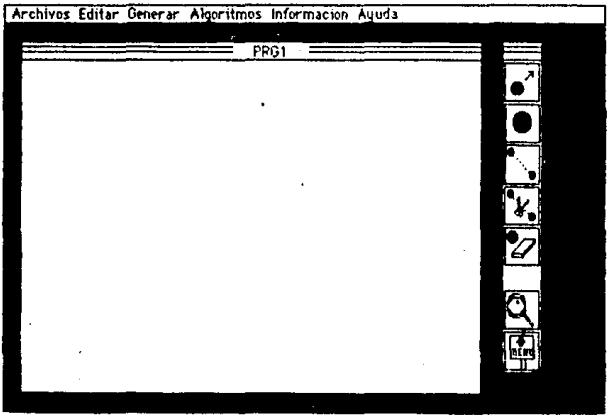
```
PRAGA.CUR          ( Cursores del programa )
PRAGA.ICN          ( Iconos                )
PRAGA1.TXT         ( Texto de información  )
PRAGA2.TXT         ( Texto de ayuda        )
A> praga
```

En seguida aparecerá la pantalla que presenta el programa:

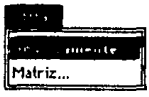


Presentación del Programa

Después de oprimir  , aparecerá la configuración principal del programa.



En esta pantalla se realizarán todas las operaciones del programa. De entrada se tendrá activado el sistema de menús en la opción:



Este diseño de menús presenta las siguientes opciones:

- Archivos:
  - Salvar: Salvar la gráfica en memoria al disco.
  - Leer de Disco: Leer una gráfica del disco previamente grabada.
  - Terminar: Salir al sistema operativo.
- Editar:
  - La gráfica manualmente.
  - La matriz de adyacencias que representa la gráfica.
- Generar:
  - Gráfica Aleatoria.

- Gráfica n-partita.
- Ciclos.
- Gráfica completa.
- Complemento de la gráfica en memoria.

☐ Algoritmos:

- Colorear: Coloración, utilizando el ordenamiento δ.
- Conexidad: Checa si la gráfica en memoria es conexa.
- Probar algoritmo: Colorea paso a paso la gráfica en memoria utilizando un ordenamiento δ.

☐ Información:

- Sobre la versión del programa.
- Sobre las opciones: Información sobre los datos que solicita el programa, generadores.

☐ Ayuda:

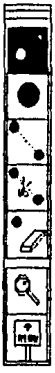
- Poner/Quitar Ayuda: Establece si el operador necesita ayuda para el manejo del editor manual.

Para manejar los menús, se utilizan las flechas en el tablero de la manera siguiente:

-  para cambiar de menú. Por ejemplo, para cambiar del menú Editar al de Generar basta usar la tecla  .
-  para cambiar de sub-menú. Por ejemplo si se quiere cambiar de Gráficamente... a Matriz... basta usar la tecla  .
- Finalmente,  para seleccionar una opción.

Manejo del Editor Manual

El manejo del editor se lleva a cabo mediante la caja de herramientas, en la cual al seleccionar en el sistema de menús la opción de Editar gráficamente..., se marcará una de las ellas.



Para seleccionar una herramienta se utilizan las teclas  y  , y al llegar a la buscada, se presiona  .

Herramientas

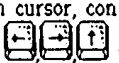
Función: Mover un vértice de lugar.

Utilización: Al seleccionar ésta opción, aparecerá en la pantalla un cursor con forma de mano, se selecciona un vértice llevando el cursor hacia él y oprimiendo  . Después se mueve el vértice, utilizando  y  , hacia su nuevo lugar y presionando  se fija la posición.

Función: Generar un nuevo vértice.

Utilización: Aparecerá en la pantalla el nuevo vértice. Utilice  y  para moverlo a su posición y presione  .

Función: Generar una arista.

Utilización: Aparecerá un cursor, con forma de flecha, en la pantalla de edición. Llévelo utilizando  y  hacia uno de los extremos de la arista nueva y presione  , después mueva el cursor hacia el otro extremo de la arista y presione  .

Función: Borrar una arista.

Utilización: Igual que el procedimiento para generar una nueva arista.

Función: Borrar un conjunto de vértices.

Utilización: Mueva el cursor (mano) que aparecerá en la pantalla y llévelo hacia una esquina del rectángulo que contenga a los vértices que quiera borrar y presione  . Mueva el nuevo cursor (dinamita) hacia la otra esquina y presione  .

Función: Ofrecer información sobre un vértice.

Utilización: Marque el vértice del que desea obtener información moviendo el cursor que aparece mediante las teclas  y y presione  .

Función: Regresar al sistema de menús.

Especificaciones Técnicas

Se discutirán en la siguiente sección algunas de las especificaciones técnicas en cuanto al desarrollo del programa, procurando mostrar la manera en que fueron construidas las diferentes rutinas, así como el proceso de integración. El primer aspecto a discutir será el de la respresentación interna que se utilizó para modelar una gráfica en la computadora, y en la segunda parte se tratarán algunas de las rutinas más importantes.

Representación

Una de las cuestiones más importantes para lograr la eficiencia, en cuanto a espacio de memoria y velocidad, es la representación que tendrá, dentro de la computadora, una gráfica. Existen básicamente dos formas de representar éstas: la Matriz de Adyacencias y por medio de listas ligadas.

Matriz de Adyacencias

Esta representación utiliza una matriz A cuadrada de entradas enteras, cuyos valores a<i,j>, se obtienen como:

$$a\langle i,j\rangle = \begin{cases} 1 \text{ si } v_i \text{ es adyacente a } v_j & (i \neq j) \\ 0 \text{ e o c.} \end{cases}$$

En el caso de gráficas simples, claramente se desperdiciará la mitad del espacio de la matriz puesto que ésta será simétrica (ver figura 3.1). Además, si almacenamos una gráfica poco densa (i.e., el número de aristas es relativamente pequeño), el desperdicio será aún mayor ya que la matriz contendrá ceros en la mayoría de sus entradas. Por otro lado, esta matriz facilita la programación y favorece la rápida ejecución del programa.

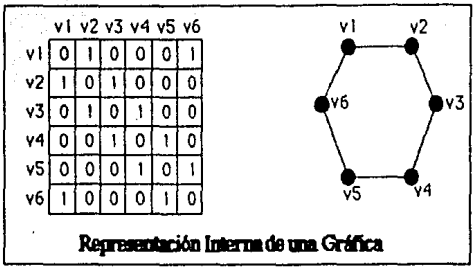


Figura 3.1

Representación por medio de Listas Ligadas

La alternativa para el caso de gráficas poco densas es la representación por medio de listas ligadas, esto es, asignando a cada vértice la lista de los adyacentes a él (ver figura 3.2). Como es fácil observar, el espacio de memoria desperdiciado es menor, puesto que la gráfica en cuestión contiene un número pequeño de aristas. Cuando se trata de una gráfica densa, podemos concluir que el volúmen de memoria utilizado es el mismo; en este caso la eficiencia del programa dependerá del tiempo de ejecución, que es mayor con esta instrumentación. Por otro lado nos induce a utilizar la memoria de la máquina tanto como nos lo permita el compilador, con su implementación del manejo dinámico de memoria.

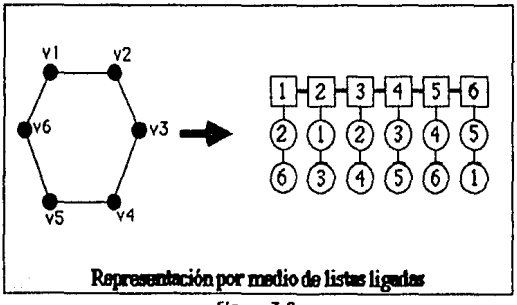


Figura 3.2

Dado que el interés en el desarrollo de la primera versión del paquete no era el de construir gráficas demasiado grandes, se utilizó la primera de las representaciones por medio de la matriz de adyacencias.

RUTINAS PARA EL MANEJO DE MENUS

Para la implementación de las rutinas de manejo de menús fué necesario considerar los siguientes puntos:

- Era necesario que fueran fácilmente manejables.

- Se pensaba utilizar las opciones contenidas en los menús en cualquier momento de la ejecución del programa, por lo que deberían ser interactivas, esto es, que estuvieran siempre presentes en pantalla.

- Requeríamos que las opciones contenidas estuvieran clasificadas por sus diferentes categorías: procedimientos para generar gráficas, los algoritmos aplicables a éstas, rutinas para manejo de archivos, diferentes maneras de editar una gráfica, aquéllas que ofrecen información al usuario y, finalmente, las que ofrecen ayuda al mismo. ''

- Dado que el número de opciones que aparecerían eran alrededor de 18, el desplegar éstas "destruiría" el contenido de la pantalla, por lo que era necesario guardar el contenido de ésta.

Considerando las anteriores necesidades, decidimos utilizar el sistema de menús originalmente programado para el sistema operativo de la computadora Macintosh -semejante al utilizado por la versión 4.0 de Turbo Pascal en su editor -. Este sistema de menús resuelve, de una manera elegante, los problemas de clasificar las opciones por sus características, así como los de restaurar la pantalla después de utilizarlos. Además de ser fácilmente manejable.

Lo descrito anteriormente que instrumentado en una rutina (ver apéndice I) ejecuta lo siguiente:

- Dibuja los menús principales en la parte superior de la pantalla.

- Según las instrucciones del usuario cambia sucesivamente de menús principales, restaurando la parte destruída por el submenú seleccionado anteriormente.

RUTINAS PARA EL MANEJO DE VENTANAS

Con el objeto de lograr un programa completamente interactivo, era necesario construir un sistema de manejo de ventanas, con algunas de las características anteriores, como las de no alterar el contenido de la pantalla. Sin embargo era necesario que tuvieran la capacidad no sólo de ofrecer información, sino también recibirla.

El sistema resultante tiene la capacidad de ejecutarse en cualquier momento del programa. Fué necesario, para esto, utilizar una parte de la memoria principal, con el objeto de guardar ahí el contenido de la pantalla que destruiría la ventana al ser dibujada. De las rutinas que conforman el sistema, dos son de principal importancia.

• **IniciaVentana** : Esta rutina recibe como único parámetro la información de la ventana que se inicia, esto es, su posición y su nombre. Con los anteriores parámetros, ejecuta los siguiente:

- Captura en memoria principal el contenido de la pantalla en el rectángulo que delimita la ventana.

- Borra el rectángulo descrito en el punto anterior.

- Dibuja la ventana y pone el nombre de la ventana a la cabeza.

• **TerminaVentana** : Con los mismos parámetros que IniciaVentana, ésta únicamente restaura el contenido de la pantalla en el rectángulo señalado por el tamaño de la ventana.

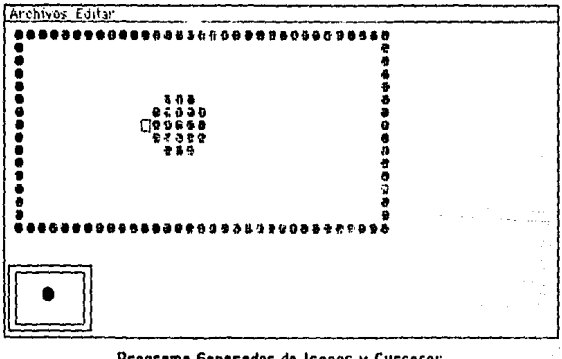
Editor Gráfico

La descripción de esa parte del programa, será necesariamente más detallada, puesto que logra uno de los principales fines en la elaboración del mismo: dibujar a mano gráficas especificas para probar el ordenamiento descrito en el capitulo 2.

El editar manualmente un grafica implicaba problemas muy diversos. Desde generar figuras y cursores que indicaran la operación que se realiza en el momento, hasta utilizar éstos para operar en la gráfica ya fuera moviendo vértices, añadiendo vértices y/o aristas o borrando vértices y/o aristas, además de construir procedimientos para "mover" los cursores generados a través de una ventana. Clasificaremos estas rutinas como sigue: procedimientos para animación en la pantalla, generación de figuras para el programa, ins-trumentación de opciones para la edicion y, finalmente, el sistema de menús particular para el editor.

Generación de Cursores e Iconos

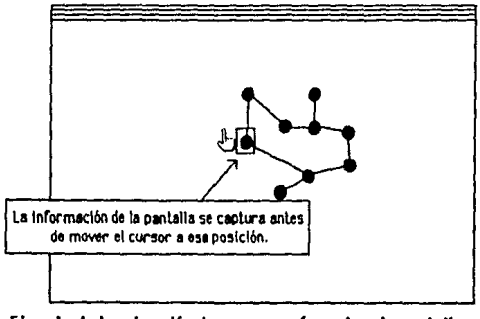
El poder construir o dibujar una gráfica manualmente requirió en primer lugar de generar cursores e íconos para mostrar al usuario gráficamente tanto la opción del editor que maneja como la posicion que tiene en la pantalla.



Programa Generador de Iconos y Cursores

Animación en la pantalla

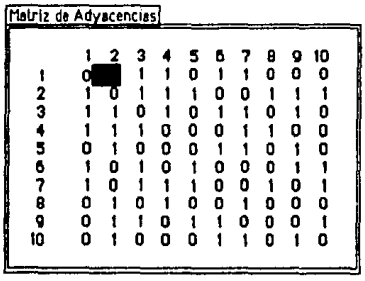
La animación de los cursores utilizados en el editor gráfico se llevo a cabo dibujando el cursor en cuestión, y restaurando más tarde la parte de la pantalla donde éste estuvo. Esto permitió dibujar sin alterar el contenido de la misma, logrando además velocidad.



Ejemplo de la animación de un cursor (mano) en la pantalla.

Editor de la matriz de adyacencias

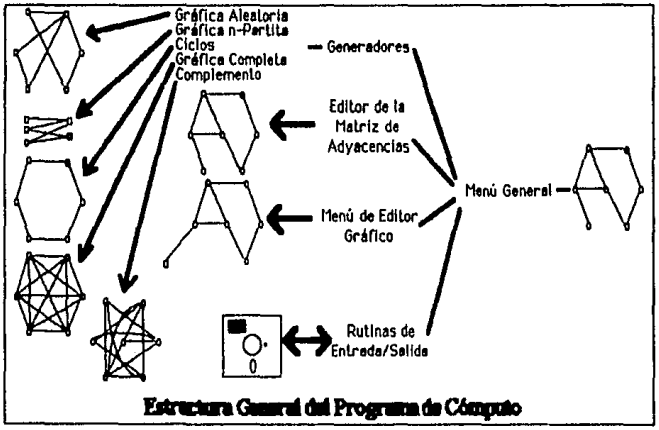
Este editor utiliza el formato de una hoja de cálculo para modificar la representación interna de la gráfica directamente. Esto permite meter ejemplo a la memoria con mayor velocidad. Esta rutina se instrumentó dado que el uso de esta representación es muy común, además que hace al programa más flexible.



Editor de la Matriz de Adyacencias

La Estructura del Programa

El programa de cómputo consta de un conjunto de rutinas que contienen en sí las funciones de generación, edición y coloración. Quedando constante en la memoria únicamente la matriz de adyacencias y las demás variables globales del programa, como banderas para saber si existe gráfica en la memoria, si el usuario necesita ayuda, etc. Además de éstas existen tambien variables para intercambiar información entre el programa principal y las rutinas paralelas. (Figura (3.3))

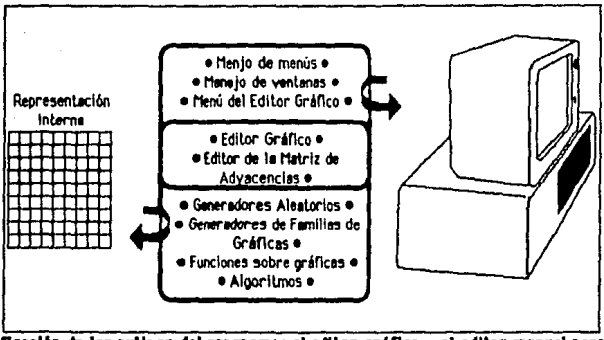


Estructura General del Programa de Cómputo

Podemos establecer una jerarquía de las diferentes rutinas según su finalidad como sigue:

- Rutinas de intercambio de información con el usuario. Entre ellas se encuentran:
  - Manejo de menús, ventanas y editor de la matriz de adyacencias.

•Rutinas Internas. Que incluyen las siguientes: Funciones de grado y conexidad, los distintos generadores y los algoritmos sobre gráficas.



Clasificación de las rutinas del programa: el editor gráfico y el editor manual permiten lograr la intercomunicación con el usuario.

La figura anterior muestra una división principal en las rutinas del programa, cuyo aislamiento da al programa claridad, ya que se manejan por separado las rutinas que afectan directamente a la gráfica en memoria, como son los generadores, los algoritmos, el editor gráfico y el editor de la matriz de adyacencias, y por otra parte las rutinas que no hacen más que ofrecer o pedir información al usuario.

CONCLUSIONES

Uno de los resultados más importantes que se pueden observar en el presente trabajo se refiere más a la parte metodológica que a las matemáticas en sí. La experiencia obtenida con el paquete de cómputo desarrollado nos indica la facilidad con que algunos resultados pueden ser "descubiertos" (no demostrados) al visualizar rápidamente tanto resultados como transformaciones aplicadas. El hecho de crear una nueva manera de acercarse a un problema, una nueva herramienta de trabajo para investigar problemas de teoría de gráficas, nos permite en cierta medida aproximarnos a algunos hechos que con papel y lápiz sería difícil ver.

Por otro lado es justo observar las carencias del programa instrumentado para así indicar nuevas posibilidades en la extensión de este concepto.

El paquete no maneja la memoria dinámicamente, lo que permite en muchos casos el desperdicio ya discutido en el tercer capítulo, por lo que el programar la representación por medio de listas ligadas traería muchas ventajas. Más aun el programar un sistema que permita el manejo de ambas representaciones eligiendo la adecuada con cierto criterio o con la decisión propia del usuario, daría a los procesos mayor eficacia en cuanto a tiempo de ejecución y espacio de memoria.

Otra camino que sería interesante desarrollar es el dar al usuario la capacidad de accesar directamente las rutinas mediante un pequeño lenguaje que ponga a su disposición todas las rutinas y algoritmos programados para así probar sus propias hipótesis. Con la creación de un lenguaje reducido que permita programar algoritmos para gráficas se podrían accesar aún más rápidamente los resultados buscados, probándolos directamente en la computadora y sin la necesidad de programar directamente en un lenguaje más complejo. Esta facilidad, combinada con la posibilidad de editar la gráfica que se trabaja dentro de un mismo paquete, resultará en una mejor herramienta para el desarrollo de teoría de gráficas.

Apéndice A:
PROGRAMA PRINCIPAL

Listado en Turbo Pascal 3.0 del programa principal del paquete PRAGA en su versión 1.0.

Este programa reúne las rutinas construidas, para conformar el paquete en su totalidad. Este paquete contiene:

•**Generador de gráficas**: Rutinas para la generación de gráficas simples, completas, cíclos y n-partitas de manera aleatoria, utilizando para esto el generador de números aleatorios que contiene el Turbo Pascal (Versión 3.0).

•**Algoritmos para coloración**: Implementación de algoritmos para la coloración de vértices y aristas, con diferentes heurísticas.

•**Editor**: El paquete contiene un editor para generar manualmente gráficas conocidas, con las que se desean probar los algoritmos implementados.

•**Procedimientos de Entrada y Salida**: Procedimientos para la salida/entrada a unidad de disco, impresora y pantalla.

Programado en Enero-Junio de 1989, por Enrique Campos Néñez como parte del proyecto de tesis, con la asesoría de Virginia Abrín Batule.

Facultad de Ciencias,
Universidad Nacional Autónoma de México

```
Program Praga( input,output );

{     Declaracion de tipos del Programa      }
Type
  Texto = String[40];        {  Texto : Una frase de 40 caracteres }
  Par   =                    {  Par   : Par ordenado (coordenadas) }
    Record
      x,y : integer;
    end;

Var
  Vacia     : Boolean;         { Var. que dice si ya hay gráfica en memoria   }
  NombreAr  : String[8];       { Nombre externo del archivo de disco          }

{$I Graph.P    }             { Rutinas gráficas de Turbo Pascal  }
{$I RutGraf.Pas}             { Rutinas gráficas construidas      }
{$I Presenta.Pas}            { Presentación del programa         }
{$I Info.Pas}                { Información sobre el programa     }
{$I Tipos.Gra}               { Definición de la estructura de    }
```

```
                              { representación de las gráficas    }

Var
  GrafBaul : File of Grafica;
  dGarro   : Integer;
  HAyuda   : Boolean;
                              { Nombre interno del archivo de disco}

{$I Pila.Pas}                 { Manejo de pilas                    }
{$I RutIO.Man}                { Rutinas de Entrada/Salida          }
{$I FGraf.Man}                { Funciones para gráficas :Parte I   }
{$I FGraf2.Pas}               { Funciones para gráficas :Parte II  }
{$I Menus.Pas}                { Manejador de los menús de pantalla }
{$I Ventanas.Pas}             { Controlador de ventanas            }
{$I Matriz.Pas}               { Editor de la matriz de adyacencias }

Var
  VInfo    : Ventana;
{$I Editor.Man}               { Editor manual de gráficas          }
{$I Color1.Man}               { Coloración de vértices             }


Procedure NuevOpcion( M,S : Integer );
{
        Esta parte del programa ejecutala opción obtenida en el procedimiento anterior
}
var
  GBaul : File of Grafica;
  n,v   : Integer;
  p     : Real;
  p2    : Vector;
  i     : Integer;
begin { Empieza procedimiento ...    }
  Case M of
    1 : case S of
          1 : begin
                with VInfo do begin
                  Nombre := 'Salvar';
                  x1 := 7; x2 := 50;
                  y1 := 7; y2 := 10;
                end;
                IniciaVentana(VInfo);
              GoToXY(1,1); Write('Nombre del archivo : ');
                ReadLn(NombreAr);
                Assign(GBaul,NombreAr+'.GRF');
                ReWrite(GBaul);
                Write(GBaul,LaGraf);
                Close(GBaul);
                TerminaVentana(VInfo);
                IniciaPantalla;
                ReDibuja;
              end;
          2 : begin
                with VInfo do begin
                  Nombre := 'Leer de Disco';
                  x1 := 7; x2 := 50;
                  y1 := 7; y2 := 10;
                end;
```

```
            IniciaVentana(VInfo);
            GoToXY(1,1); Write('Nombre del archivo : ');
            ReadLn(NombreAr);
            Assign(GBaul,NombreAr+'.GRF');
            ReSet(GBaul);
            Read(GBaul,LaGraf);
            Close(GBaul);
            TerminaVentana(VInfo);
            IniciaPantalla;
            ReDibuja;
          end;
      3 : begin
          end;
      else
    end;
2 : case S of
      1 : begin
        editor;
      end;
      2 : begin
        with VInfo do begin
          Nombre := 'Matriz de Adyacencias';
          x1 := 5; x2 := 40;
          y1 := 5; y2 := 20;
        end;
        IniciaVentana(VInfo);
        EditaMatriz;
        TerminaVentana(VInfo);
        ReDibuja;
      end;
      else
    end;
3 : case S of
      1 : begin
            with VInfo do begin
              Nombre := 'Grafica Simple';
              x1 := 7; x2 := 50;
              y1 := 7; y2 := 10;
            end;
            IniciaGrafica;
            IniciaVentana(VInfo);
            GoToXY(1,1); Write('Numero de Vertices : ');
            ReadLn(n);
            GoToXY(1,2); Write('Densidad esperada  : ');
            ReadLn(p);
            Genera(LaGraf,n,p);
            TerminaVentana(VInfo);
            ReDibuja;
          end;
      2 : begin
            with VInfo do begin
              Nombre := 'n-partita';
              x1 := 7; x2 := 50;
              y1 := 7; y2 := 17;
            end;
            IniciaGrafica;
            IniciaVentana(VInfo);
            GoToXY(1,1); Write('Numero de Vertices      : ');
```

```
            ReadLn(v);
            GoToXY(1,2); Write('Densidad esperada      : ');
            ReadLn(p);
            GoToXY(1,3); Write('Numero de Partes (1-7) : ');
            ReadLn(n);
            for i := 1 to n do begin
              GoToXY(1,3+i); Write('Densidad de C',i,' : ');
              ReadLn(p2[i]);
              if i>1 then
                p2[i] := p2[i]+p2[i-1];
            end;
            Gen_EneParti(LaGraf,n,v,p,p2);
            TerminaVentana(VInfo);
            ReDibuja;
          end;
      3 : begin
            with VInfo do begin
              Nombre := 'Ciclo';
              x1 := 7; x2 := 50;
              y1 := 7; y2 := 10;
            end;
            IniciaGrafica;
            IniciaVentana(VInfo);
            GoToXY(1,1); Write('Numero de Vertices : ');
            ReadLn(n);
            GeneraCiclos(n);
            TerminaVentana(VInfo);
            ReDibuja;
          end;
      4 : begin
            with VInfo do begin
              Nombre := 'Grafica Completa';
              x1 := 7; x2 := 50;
              y1 := 7; y2 := 10;
            end;
            IniciaGrafica;
            IniciaVentana(VInfo);
            GoToXY(1,1); Write('Numero de Vertices : ');
            ReadLn(n);
            GeneraK(n);
            TerminaVentana(VInfo);
            ReDibuja;
          end;
      5 : begin
            Complemento;
            ReDibuja;
          end;
    else
  end;
4 : case S of
      1 : begin
            with VInfo do begin
              Nombre := 'Coloracion de G';
              x1 := 5; x2 := 73;
              y1 := 7; y2 := 23;
            end;
            IniciaVentana(VInfo);
            ColoreaPorMIN;
```

```
            Resultados;
            ReadLn;
            TerminaVentana(VInfo);
            ReDibuja;
          end;
      2 : begin
            with VInfo do begin
              Nombre := 'Conexidad';
              x1 := 7; x2 := 50;
              y1 := 7; y2 := 10;
            end;
            IniciaVentana(VInfo);
            GoToXY(1,1);
            if Conexa then
              WriteLn('  La Grafica es CONEXA ')
            else
              WriteLn('  La Grafica no es CONEXA');
            WriteLn;
            Write('Numero de Aristas ',NumeroAristas);
            ReadLn;
            TerminaVentana(VInfo);
          end;
      3 : begin
            with VInfo do begin
              Nombre := 'COLORACION';
              x1 := 50; x2 := 75;
              y1 := 5; y2 := 8;
            end;
            DemoColoreaPorMin;
          end;
    else
  end;
5 : case S of
      1 : begin
            with VInfo do begin
              Nombre := 'Informacion';
              x1 := 5; x2 := 45;
              y1 := 7; y2 := 20;
            end;
            IniciaVentana(VInfo);
            Info1;
            TerminaVentana(VInfo);
          end;
      2 : begin
            with VInfo do begin
              Nombre := 'Informacion';
              x1 := 5; x2 := 70;
              y1 := 7; y2 := 20;
            end;
            IniciaVentana(VInfo);
            Info2;
            TerminaVentana(VInfo);
          end;
    else
  end;
6 : case S of
      1 : begin
        with LaBMenus do begin
```

```
            with Menus[6] do begin
              if MAyuda then begin
                MAyuda := False;
                SMenus[1] := 'Usar Ayuda';
              end
              else begin
                MAyuda := True;
                SMenus[1] := 'Quitar Ayuda';
              end;
            end;
          end;
        end;
      else
      end;
    else
  end;
end;  { ... termina procedimiento    }

{
```

PROGRAMA PRINCIPAL

```
}

BEGIN                          {   Comienza ...                    }
  Presentacion;                {     hace la presentación del programa }
  ReadLn;                      {     espera una tecla del usuario  }
  NombreAr := 'PRG1';          {     inicia el nombre del archivo  }
  MAyuda := False;             {     no se necesita ayuda inicial  }
  Vacia := True;               {     no hay gráfica en memoria     }
  IniciaEditor;                {     inicia el editor manual       }
  IniciaPantalla;              {     dibuja la pantalla del editor }
  IniciaMenu;                  {     inicia los valores de menus   }
  IniciaGrafica;               {     da valores iniciales a la gráfica }
  Men := 2; SMen := 1;         {     el menú y submenú son el 1    }
  Repeat                       {     Repite ...                    }
    BuscaEnSM(Men,Smen);       {      Busca Opción en el menú      }
    HazOpcion(Men,Smen);       {       y la maneja.                }
  Until (Men=1)and(SMen=3);    {     ... hasta que la opción sea salir }
  TextMode;                    {     pone la pantalla en modo de texto }
  ClrScr;                      {     limpia la pantalla            }
END.                           {     ... termina                   }
```

Apéndice B:

RUTINAS GRAFICAS

Rutinas construidas para la generación de gráficas. Incluye :
- **Inicio de la pantalla al modo gráfico. (alta resolución) -**
- **Limpieza de la pantalla gráfica -**
- **Generación de rectángulos -**

```
Procedure Rectangulo(x1,y1,x2,y2,Color : Integer);
{
                    Genera rectángulo con esquinas en (x1,y1) y (x2,y2).
}
  begin
    draw(x1,y1,x2,y1,color);
    draw(x2,y1,x2,y2,color);
    draw(x2,y2,x1,y2,color);
    draw(x1,y2,x1,y1,color);
  end;


Procedure Inicia_Pantalla(Color : Integer);
{
                    Inicia la pantalla al modo gráfico de alta resolución.
}
  begin
    Hires;
    Hirescolor(Color);
  end;


Procedure Limpia;
{
                                Limpia la pantalla.
}
  begin
    FillScreen(0);
  end;
```

Apéndice C:
PRESENTACION

```
Procedure Presentacion;
{
                    Presentación del programa de cómputo.
}

Procedure Arbol( x,y,nivel,altura,long : Integer );
begin
  draw(x,y,x-long,y+altura,0);
  draw(x,y,x+long,y+altura,0);
  if nivel>1 then begin
    Arbol(x-long,y+altura,nivel-1,altura,trunc(long/2));
    Arbol(x+long,y+altura,nivel-1,altura,Trunc(long/2));
  end;
end;


Procedure Presenta;
var
  i    : Integer;
Begin
  Window(4,4,76,23);
  GoToXY(1,1);
  WriteLn('                                  Praga');
  WriteLn('                                 Version 1.0');
  GoToXY(1,18);
  WriteLn('          Paquete de Rutinas y Algoritmos para
      Gráficas.');
  WriteLn('              Desarrollado por Enrique Campos Náñez');
  Window(1,1,80,25);
  Rectangulo((8-1)*8,(6-1)*8+4,72*8,20*8-4,1);
  GraphWindow((9-1)*8,(7-1)*8,71*8,19*8);
  FillScreen(1);
  Arbol(245,10,5,15,100);
End;


begin
  HiRes;
  HiResColor(15);
  FillScreen(0);
  Presenta;
End;
```

Apéndice D:

INFORMACION

Procedimientos para desplegar información acerca del programa y su elaboración.

```
Procedure Info1;
Var
  CHBaul : File of Char;
  Ch     : Char;
begin
  GoToXY(1,1);
  Assign(CHBaul,'Info1.TXT');
  ReSet(CHBaul);
  Repeat
    Read(CHBaul,CH);
    If Not(EOF(CHBaul)) then
      Write(CH);
  Until EOF(CHBaul);
  ReadLn;
end;

Procedure Info2;
Var
  CHBAul : File of CHar;
  CH     : CHar;
begin
  GoToXY(1,1);
  Assign(CHBaul,'Info2.TXT');
  ReSet(CHBaul);
  Repeat
    Read(CHBaul,CH);
    If Not(EOF(CHBaul)) then
      Write(CH);
  Until EOF(CHBaul);
  ReadLn;
end;
```

Apéndice E:

DEFINICION

Definición de la estructura para representación de las gráfica, además de otras definiciones que se usan en el programa como:

-Máximo número de vértices.

-Máximo número de particiones de una gráfica (para efectos de generación aleatoria de gráficas n-partitas).

```
Const
  MaxVert = 50;          {   Número máximo de vértices              }
  MaxPart = 7;           {   Máximo número de partes de una gráfica }

Type
  Marca   = array [1..MaxVert] of Boolean;       { Marcas para los vértices  }
  Vector  = array [1..MaxPart] of Real;          { Vector de entradas reales }
  Matriz  = array [1..MaxVert,1..MaxVert] of Integer; { Matriz cuadrada      }
  Cord    =                                            { Coords. de pantalla }
    Record
      x,y : Integer;
    end;
  Coords  = array [1..MaxVert] of Cord;
  Grafica =
    Record
      NVert    : 1..MaxVert;        {   Número de vértices             }
      Color    : array [1..MaxVert] of Byte;
                                    {   Colores de los vértices        }
      Adya     : Matriz;            {   Matriz de adyacencias          }
      Pos      : Coords;            {   Pos. en coords. cartesianas    }
    End;


Var
  LaGraf : Grafica;                 {   Gráfica que se usará en todo   }
                                    { el programa.                     }
```

Apéndice F:

MANEJO DE PILAS

Implementación de las rutinas para el manejo de pilas. La Pila es una estructura que guarda datos con la siguiente característica: el último dato en ser depositado será el primero en salir. Estas rutinas son necesarias para la realización de la heurística de coloración en base al grado mínimo.

```
Type
  Pila = ARRAY [1..500] of Integer;{  Arreglo que contiene los datos }

Var
  LaPila : Pila;
  Tope   : Integer;    {  Altura en la que se accesa      }

Procedure MetePila( Vert : Integer );
{
                Procedimiento para meter un vértice (número entero) a la pila.
}
begin
  if Tope<MaxVert then begin  { Si Tope es menor a lo max.posible)
    Tope := Tope+1;           {   Incrementa la altura en 1       }
    LaPila[Tope] := Vert; { Asigna al arreglo el valor parámetro)
  end
  else begin                    { ... si no }
    write('La Pila esta llena !!! '); { Escribe en pantalla que se lleno  }
    Halt;                           { Detiene la ejecución del programa }
  end;
end;

Function PilaVacia : Boolean;
{
                        Función que nos dice si la pila esta vacía.
}
begin
  if TOPE <= 0 then             {  Si la altura esta en 0 o menos ...     }
    PilaVacia := True           {    la pila esta vacía                   }
  else                          {  si no ...                              }
    PilaVacia := False;         {    la pila no esta vacía                }
end;

Procedure SacaPila( var
                    Vert : Integer );
{
                Procedimiento para sacar el último elemento metido en la pila.
}
begin
  if Tope>=1 then begin         {  Si la altura es mayor que uno ...      }
    Vert := LaPila[Tope];       {    regresa el valor que hay en esa altura }
```

```
    Tope := Tope-1;              {    baja la altura en 1                    }
  end
  else begin                     { si no ...                                 }
    write('La Pila esta Vacia !!!'); { escribe la pila esta llena            }
    Halt;                          { detiene la ejecución del programa       }
  end;
end;

Procedure IniciePila;
{
                Procedimiento para dar los valores iniciales a la estructura.
}
begin
  Tope := 0;                     { Asigna 0 a la altura }
end;
```

Apéndice G:

RUTINAS DE ENTRADA Y SALIDA

Rutinas de Entrada y Salida (I/O) de datos del programa, comprende :

- **Rutinas para leer de teclado y disco.**
- **Rutinas para salvar información en disco.**
- **Rutinas para manejo de menús.**

```
Procedure LeeTeclas( Var
                       Ch1,Ch2 : Char );
(
                  Procedimiento para leer teclas directamente del teclado.
)
begin
  ch2 := ' ';
  GoToXY(80,24);
  Read(Kbd,Ch1);
  if KeyPressed then
    Read(kbd,Ch2);
end;

Procedure LeeGrafica;
(
                       Procedimiento para leer una grafica de disco.
)
begin
  ClrScr;
  LowVideo;
  GoToXY(1,3);
  writeln('Nombre del archivo (sin extension) :_____ ');
  GoToXY(52,4);
  NormVideo;
  ReadLn(NombreAr);
  Assign(GrafBaul,NombreAr+'.GRF');
  ReSet(GrafBaul);
  Read(GrafBaul,LaGraf);
  Close(GrafBaul);
end;

Procedure EscGrafica;
(
                       Procedimiento para escribir una grafica a disco.
)
begin
```

```
  ClrScr;
  LowVideo;
  GoToXY(1,3);
  writeln('Nombre del archivo (sin extension) : _______ ');
  GoToXY(52,4);
  NormVideo;
  ReadLn(NombreAr);
  Assign(GrafBaul,NombreAr+'.GRF');
  ReWrite(GrafBaul);
  Write(GrafBaul,LaGraf);
  Close(GrafBaul);
end;
```

Apendice H:

FUNCIONES PARA GRAFICAS

Conjunto de funciones para gráficas, necesarias para la implementación de los algoritmos.
Este compendio consta de :

- Grado de un vértice.
- Grado mínimo y máximo.
- Generación de Gráficas Abstractas:
  - a) Simples
  - b) BiPartitas
- Generación de Ciclos y Gráficas Completas
- Complemento de una Gráfica.
- Función para checar conexidad de una gráfica y su número de aristas.

```
Procedure InicioGrafica;
{
                    Procedimiento para darle valores iniciales a una gráfica.
}
var
  i,j : Integer;
begin       (   Empieza el procedimiento ...  )
  with LaGraf do begin
    for i := 1 to MaxVert do begin
      for j := 1 to MaxVert do
        Adya[i,j] := 0;              { Pone ceros en la matriz de adyacencias }
      Pos[i].x := 0;                 { Asigna las coordenadas de los vértices }
      Pos[i].y := 0;                 { al orígen                              }
    end;
    NVert := 0;                      { Asigna 0 al número de vértices         }
  end;
end;        (     ... termina el procedimiento }


Function Grado( Vert : Integer;
                Marc : Marca    ) : Integer;
{
                    La función Grado nos regresa el grado del vértice Vert.
}
var
  i     : Integer;        {   Contador           }
  Suma  : Integer;        {   Variable auxiliar  }
begin         { Empieza el procedimiento ... }
  Suma := 0;                         { El resultado es cero inicialmente }
  with LaGraf do begin
    for i := 1 to NVert do begin
      if (Adya[Vert,i]<>0)and(i<>Vert)and(not(marc[i])) then
        Suma := Suma+1;              { Si hay vértice adyacente y no mar- }
```

```
    end;                              { cada, aumenta en 1 el grado        }
  end;
  Grado := Suma;
end;          { ... termina el procedimiento }


Function GradoMin( var
                     Vert : Integer;
                   Marc   : Marca   ) : Integer;
{
Función entera que nos regresa el grado mínimo de una gráfica, y guarda en Vert el último vértice que
                                encontró con ese grado.
}
var
  GMAux : Integer;     {  Variable auxiliar    }
  i     : Integer;     {  Contador             }
begin
  Vert := 0;
  Repeat
    Vert := Vert+1;               { Busca un vértice que no esté marcado    }
  Until Marc[Vert] = False,
  GMAux := Grado(Vert,Marc);      { Asigna como grado mínimo el gr. de éste }
  for i := 2 to LaGraf.NVert do { Busca para todos los vértices ...        }
    if (Grado(i,marc)<GMAux)and(not(Marc[i])) then begin { si el grado es  }
      GMAux := Grado(i,marc);           { menor ent. asigna el grado       }
      Vert := i;                        { y el vértice                     }
    end;
  GradoMin := GMAux;
end;

Function GradoMax( var
                     Vert : Integer ) : Integer;
{
  Función entera que nos regresa el grado máximo de una gráfica, y guarda en Vert el último vértice
                          que encontró con ese grado.
}
var
  GMAux : Integer;     {  Variable auxiliar    }
  i     : Integer;     {  Contador             }
  Marc  : Marca;
begin
  for i := 1 to LaGraf.NVert do          { Quita las marcas                  }
    Marc[i] := False;
  Vert := 1; GMAux := Grado(Vert,marc); { Asigna el grado del 1er. vértice  }
  for i := 2 to LaGraf.NVert do
    if Grado(i,marc)>GMAux then begin    { Si el gr. de otro vértice es mayor }
      GMAux := Grado(i,marc);            {   asigna el grado                  }
      Vert := i;                         {   y el vértice encontrado.         }
    end;
  GradoMax := GMAux;
end;


Function EVA( v   : Integer;
              var
                j : Integer;
              m   : Marca   ) : Boolean;
{
```

```
                Función que nos dice si Existe Vértice Adyacente a v, que no esté marcado.
)
var
  i   : Integer;                { contador                                    }
  Res : Boolean;                { Auxiliar en el calculo de la respuesta      }
begin
  Res := False;                 { De entrada no existen vértices adyacentes   }
  j := 0;
  for i := 1 to v-1 do
    if (LaGraf.Adya[v,i]<>0)and(not(M[i])) then begin
      j := i; Res := True;      { Si hay un vértice adyacente y no marcado    }
    end;                        { Dar de alta ese vértice                     }
  for i := v+1 to LaGraf.NVert do
    if (LaGraf.Adya[v,i]<>0)and(not(M[i])) then begin
      j := i; Res := True;
    end;
  EVA := Res;
end;


Function Camino( i : Integer ) : Boolean;
{
                Función que nos dice si existe camino entre el vértice 1 y el i.
}
var
  v,j     : Integer;
  M       : Marca;
  Termino : Boolean;
  Res     : Boolean;
begin
  with LaGraf do begin
    IniciaPila;
    for v := 1 to NVert do
      M[v] := False;
    Termino := False;
    v := 1; M[v] := True;
    if Adya[v,i] <> 0 then begin
      Termino := True;
      Res := True;
    end
    else
      Res := False;
    while Termino=False do begin
      if EVA(v,j,m) then begin
        MetePila(v);
        v := j;
        M[v] := True;
        if v=i then begin
          Termino := True;
          Res := True;
        end;
      end
      else begin
        if PilaVacia then begin
          Termino := True;
          Res := False;
        end
```

```
      else
        SacaPila(v);          {   del if PilaVacia }
    end;                      {   del else begin   }
  end;                        {   del while        }
 end;
 Camino := Res;
end;


Function Conexa : Boolean;
{
            Función que checa si la gráfica que está en la memoria es conexa o no.
          Lleva esto a cabo checando la existencia de caminos entre todos los vértices
                                  y el primero de éstos.
}
var
 i      : Integer;
 Result : Boolean;
begin
 Result := True;
 for i := 2 to LaGraf.NVert do begin
  if Not(Camino(i)) then
    Result := False;
 end;
 Conexa := Result;
end;                          {   Termina función }

Procedure Genera( var
                  LaGraf : Grafica;
                       n : Integer;
                       p : Real     );
{
  Procedimiento que genera gráficas aleatoriamente. Tiene como parámetros :
  - n = número de vértices que tendrá la gráfica.
  - p = una probabilidad de que aparezca una arista.
  - LaGraf = Estructura en la que quedará representada la gráfica.
}
var
 i,j : Integer;          {  Contadores                                          }
 DT  : Integer;          {  Auxiliar para el calculo de las coordenadas }
begin
 with LaGraf do begin
  NVert := n;
  for i := 1 to NVert-1 do begin
    for j := i+1 to NVert do begin
      if random<p then begin
        Adyal[i,j] := 1;
        Adyal[j,i] := 1;
      end
      else begin
        Adyal[i,j] := 0;
        Adyal[j,i] := 0;
      end;
    end;
  end;
  DT := 360 div n;
  for i := 1 to NVert do begin
    Pos[i].x := 290+16*(trunc(7*cos(i*DT*3.141592/180)));
```

```
      Pos[i].y := 80+8*(trunc(7*sin(i*DT*3.141592/180)));
    end;
  end;
  Vacio := False;
end;

Procedure Gen_EnePartita( var
                          LaGraf : Grafica;
                              n  : Integer;
                              v  : Integer;
                             p1  : Real;
                             p2  : Vector     );
{
   Procedimiento para generar una grafica bipartita, con los siguientes parámetros :
      - LaGraf : Estructura que contendrá a la gráfica generada.
      - v : número de vértices de la gráfica.
      - p1 : probabilidad de aparición de aristas.
      - p2 : Distribución discreta de probabilidades.
      - n  : Número de particiones de la grafica.
}
var
  i,j,k   : Integer;        {   Contadores    }
  NAleat  : Real;           {   Var. Auxiliar }
  OVector : array [1..MaxVert] of Integer;
  DX      : Integer;
  DY      : Integer;
  pilita  : array [1..7] of Integer;
begin
  for i := 1 to 7 do
    pilita[i] := 0;
  with LaGraf do begin
    NVert := v;
    DX := 3;
    DY := 2;
    for i := 1 to n do begin
      OVector[i] := i;
      pilita[i] := 1;
      Pos[i].x := 290+(i-4)*DX*16;
      Pos[i].y := 80+(pilita[i]-4)*DY*8;
    end;
    for i := n+1 to v do begin
      j := 1;
      NAleat := random;
      while (NAleat>p2[j])and(j<n) do
        j := j+1;
      OVector[i] := j;
      pilita[j] := pilita[j]+1;
      Pos[i].x := 290+(j-4)*DX*16;
      Pos[i].y := 80+(pilita[j]-4)*DY*8;
    end;
    for i := 1 to v do begin
      for j := i+1 to v do begin
        if (random<p1)and(OVector[i]<>OVector[j]) then begin
          Adyal[i,j] := 1;
          Adyal[j,i] := 1;
        end
        else begin
          Adyal[i,j] := 0;
```

```
          Adyalj,i1 := 0;
        end;
      end;
    end;
  end;
  Vacia := False;
end;

Procedure GeneraCiclos( n : Integer );
{
                    Procedimiento para generar ciclos de longitud n;
                      n es un entero de longitud máxima 49.
}
var
  i  : Integer;
  DT : Real;
begin
  IF (N<=MaxVert)and(n>0) then begin
    with LaGraf do begin
      NVert := n;
      DT := 360/n;
      for i := 1 to n-1 do begin
        Adyali,i+1] := 1;
        Adyali+1,i1 := 1;
        Posli].x := 290+16*(Trunc(7*cos(dt*i*3.141592/180)));
        Posli].y := 80+8*(Trunc(7*sin(dt*i*3.141592/180)));
      end;
      Adyal1,n] := 1; Adyaln,1] := 1;
      Posln].x := 290+16*5;
      Posln].y := 80;
    end;
    Vacia := False;
  end
  else LaGraf.NVert := 0;
end;

Procedure GeneraK( n : Integer );
{
                  Procedimiento para generar gráficas completas con n vértices.
}
var
  i,j : Integer;
  DT  : Real;
begin
  if (N<=MaxVert)and(n>0) then begin
    with LaGraf do begin
      NVert := n;
      DT := 360/n;
      for i := 1 to n do begin
        for j := 1 to n do begin
          if i<>j then begin
            Adyali,j] := 1;
            Adyalj,i] := 1;
          end;
        end;
        Posli].x := 290+16*(Trunc(7*cos(dt*i*3.141592/180)));
        Posli].y := 80+8*(Trunc(7*sin(dt*i*3.141592/180)));
      end;
```

```
    end;
    Vacia := False;
  end
  else LaGraf.NVert := 0;
end;


Procedure Complemento;
{
                Procedimiento para generar el complemento de una gráfica.
}
var
  i,j : Integer;
begin
  with LaGraf do begin
    for i := 1 to NVert do
      for j := i+1 to NVert do begin
        Adyai,j] := 1-Adyai,j];
        Adyaj,i] := Adyai,j];
      end;
  end;
end;

Function NumeroAristas : Integer;
{
   Función que cuenta el número de aristas en una gráfica.
}
var
  i,j : Integer;       {  Contadores                                    }
  Res : Integer;       {  Variable auxiliar para almacenar el resultado }
begin
  Res := 0;
  for i := 1 to (LaGraf.NVert-1) do begin
    for j := i+1 to LaGraf.NVert do begin
      if LaGraf.Adyai,j]<>0 then          { Si la entrada de la matriz es  }
        Res := Res+1;                     { distinta de cero, aumenta en 1 }
    end;
  end;
  NumeroAristas := Res;
end;      { Termina rutina que cuenta aristas }
```

Apéndice I:

MANEJO DE MENUS

Rutinas para el manejo de barras de menús.

```
Const
  MaxMenu  = 6;   { Máximo número de opciones de menús  }
  MaxSMen  = 5;   { Máximo número de submenus por menús }
  Longitud = 25;  { Longitud máxima de una opción       }

Type
  SMenu = String[Longitud];
  Menu  = Record
            SMenus   : array [1..MaxSMen] of Smenu;
            NSMen    : Integer;
            Posicion : Integer;
            Nombre   : Smenu;
            LMaxima  : Integer;
          End;
  BMenu = Record
            Menus   : array [1..MaxMenu] of Menu;
            Activos : Byte;
          End;

Var
  Men,SMen : Integer;
  LaBMenus : BMenu;
  BASMenu  : array [1..500] of Byte;


Procedure InicioMenu;
begin
  with LaBMenus do begin
    Activos := 6;
    with Menus[1] do begin
      Nombre := 'Archivos';
      Posicion := 2;
      NSMen := 3;
      SMenus[1] := 'Salvar';
      SMenus[2] := 'Leer de Disco';
      SMenus[3] := 'Terminar';
      LMaxima := 13;
    end;
    with Menus[2] do begin
      nombre := 'Editar';
      posicion := 12;
      SMen := 2;
      SMenus[1] := 'Graficamente ...';
      SMenus[2] := 'Matriz ...';
```

```
      LMaxima := 16;
    end;
    with Menus[3] do begin
      Nombre := 'Generar';
      Posicion := 20;
      NSMen := 5;
      SMenus[1] := 'Grafica Simple';
      SMenus[2] := 'Grafica n-partita';
      SMenus[3] := 'Ciclo';
      SMenus[4] := 'Grafica Completa';
      SMenus[5] := 'Complemento';
      LMaxima := 17;
    end;
    with Menus[4] do begin
      Nombre := 'Algoritmos';
      Posicion := 29;
      NSMen := 3;
      SMenus[1] := 'Colorear';
      SMenus[2] := 'Checar Conexidad';
      SMenus[3] := 'Probar Algoritmo';
      LMaxima := 16;
    end;
    with Menus[5] do begin
      Nombre := 'Informacion';
      Posicion := 41;
      NSMen := 2;
      SMenus[1] := 'Sobre el programa';
      SMenus[2] := 'Sobre las Opciones';
      LMaxima := 18;
    end;
    with Menus[6] do begin
      Nombre := 'Ayuda';
      Posicion := 54;
      NSMen := 1;
      SMenus[1] := 'Usar Ayuda';
      LMaxima := 12;
    end;
  end;
end;


Procedure Marco(x1,y1,x2,y2 : Integer );
{
                    Dibuja un marco con esquinas en (x1,y1) y (x2,y2).
}
begin
  GraphWindow((x1-1)*8,(y1-1)*8,(x2)*8,(y2)*8);
  FillScreen(0);
  GraphWindow(0,0,639,199);
  Rectangulo((x1-1)*8,(y1-1)*8,(x2)*8,(y2)*8,1);
  Rectangulo((x1-1)*8+4,(y1-1)*8+2,(x2)*8-4,(y2)*8-2,1);
end;

Procedure BorreSMenu( M : Integer );
begin
  with LaBMenus do begin
    with Menus[M] do begin
      PutPic(BASMenu,(Posicion-1)*8,(2+NSMen+1)*8);
```

```
    end;
  end;
end;

Procedure DibujaSMenu( M : Integer );
var
  i : Integer;
begin
  with LaBMenus do begin
    with Menus[m] do begin
      GetPic(BASMenu,(Posicion-1)*8,8,(Posicion+LMaxima+4-1)*8,
      (2+NSMen+1)*8);
      if NSMen>0 then begin
        Marco(Posicion,2,Posicion+2+LMaxima,2+NSMen+1);
        for i := 1 to NSMen do begin
          GoToXY(Posicion+2,2+i);
          Write(SMenus[i]);
        end;
      end;
    end;
  end;
end;

Procedure MarcaSMenu( S : Integer;
                      M : Integer );
var
  bas : array [1..500] of byte;
begin
  ColorTable(0,1,2,3);
  with LaBMenus.Menus[m] do begin
    if NSmen>0 then begin
      GetPic(bas,(posicion)*8,(s+2)*8,
                 (posicion+lmaxima+1)*8,(s+2)*8-1);
      ColorTable(1,0,2,3);
      PutPic(bas,(posicion)*8,(s+2)*8-1);
    end;
  end;
  ColorTable(0,1,2,3);
end;

Procedure MarcaMenu( M : Integer );
var
  bas : array [1..500] of byte;
begin
  ColorTable(0,1,2,3);
  with LaBMenus.Menus[m] do begin
    GetPic(bas,(posicion-1)*8,0,(posicion+length(Nombre)-1)*8,7);
    ColorTable(1,0,2,3);
    PutPic(bas,(posicion-1)*8,7);
  end;
  ColorTable(0,1,2,3);
end;

Procedure DibujaMenu( M,S : Integer );
{
                Dibuja el menú principal con las opciones M,S maracadas.
}
var
```

```
  I : Integer;
begin
 with LaBMenus do begin
   for i := 1 to Activos do
     with Menus[i] do begin
       GoToXY(Posicion,1); Write(Nombre);
     end;
    Draw(0,8,639,8,1);
  end;
  MarcaMenu(M);
  DibujaSMenu(M);
  MarcaSMenu(S,M);
end;

Procedure BuscaEnBM( Var
                     M,S : Integer );
{
                  Procedimiento para elejir una opción en el menú principal.
}
Var
  Ch1,Ch2 : Char;
  Termino : Boolean;
begin
  DibujaMenu(M,S);
  Termino := False;
  Repeat
    LeeTeclas(Ch1,Ch2);
    Case Ch1 of
      #27 : begin
            Case Ch2 of
              ' ' : begin
                      MarcaMenu(M);
                      BorraSMenu(M);
                      M := 0; S := 0;
                      Termino := True;
                    end;
              'H' : begin
                      MarcaSMenu(S,M);
                      if S<2 then
                        S := LaBMenus.Menus[m].NSmen
                      else
                        S := S-1;
                        MarcaSMenu(S,M);
                    end;
              'P' : begin
                      MarcaSMenu(S,M);
                      if S>LaBMenus.Menus[m].NSmen-1 then
                        S := 1
                      else
                        S := S+1;
                      MarcaSMenu(S,M);
                    end;
              'M' : begin
                      BorraSmenu(M);
                      MarcaMenu(M);
                      if M>LaBMenus.Activos-1 then
                        M := 1
                      else
```

```
                    M := M+1;
                  DibujaSMenu(M);
                  S := 1;
                  MarcaSMenu(S,M);
                  MarcaMenu(M);
                end;
            'K' : begin
                  BorraSMenu(M);
                  MarcaMenu(M);
                  if M<2 then
                    M := LaBMenus.Activos
                  else
                    M := M-1;
                  DibujaSMenu(m);
                  S := 1;
                  MarcaSMenu(S,M);
                  MarcaMenu(M);
                end;
          else
        end;
      end;
    #13 : begin
          Termino := True;
        end;
  else
 end;
Until Termino;
BorraSmenu(M);
MarcaMenu(M);
GraphWindow(0,0,639,199);
end;
```

Apéndice J:

VENTANAS

Compendio de procedimientos para crear el ambiente de ventanas.

```
Type
  Ventana = Record
              x1,y1,x2,y2 : Integer;
              Nombre      : String[25];
            End;

Var
  BASVen : array [1..16000] of Byte;


Procedure InicioVentana( LaVen : Ventana );
{
                    Inicia la ventana en la pantalla, con los siguientes pasos:
                      • Guarda en memoria el contenido de la pantalla •
                       • Limpia la parte correspondiente a la ventana •
                      • Dibuja la ventana con los parámetros señalados •
}
var
  I : Integer;
begin
  with LaVen do begin
    GetPic(BASVen,(x1-2)*8,(y1-3)*8-1,(x2+1)*8,(y2+1)*8);
    GoToXY(x1-1,y1-2);
    for i := 1 to Length(Nombre)+2 do write(' ');
    GoToXY(x1,y1-2); write(Nombre);
    rectangulo((x1-2)*8,(y1-3)*8-1,(x1+length(Nombre))*8,(y1-)*8,1);
    marco(x1-1,y1-1,x2+1,y2+1);
    window(x1,y1,x2,y2);
    GoToXY(1,1);
  end;
end;

Procedure TerminaVentana( LaVen : Ventana );
{
              Restaura el contenido anterior de la pantalla, ocupado por la ventana.
}
begin
  with LaVen do begin
    PutPic(BASVen,(x1-2)*8,(y2+1)*8);
    window(1,1,80,25);
  end;
end;
```

Apéndice K:

EDITOR DE LA MATRIZ DE ADYACENCIA

Rutinas para la edición directa de la matriz de adyacencias de la gráfica.

```
Procedure EditaMatriz;

Const
  Columnas  = 10;
  Renglones = 10;

Var
  BasMat : array [1..100] of Byte;
  i,j    : Integer;

Procedure LeeTeclas( Var
                       Ch1,Ch2 : Char );
begin
  Ch1 := ' '; Ch2 := ' ';
  GoToXY(1,1);
  Read(Kbd,Ch1);
  if KeyPressed then
    Read(Kbd,Ch2);
end;

Procedure DibujaMatriz( x,y : Integer );
var
  i,j : Integer;
begin
  for i := 1 to Columnas do begin
    GoToXY((i)*3+1,1);
    Write(x+i-1:3);
    for j := 1 to Renglones do begin
      GoToXY((i)*3+3,j+2);
      Write(LaGraf.Adyalx+i-1,y+j-1]);
    end;
  end;
  for j := 1 to Renglones do begin
    GoToXY(1,j+2);
    Write((y+j-1):3);
  end;
end;

Procedure MarcaMatriz( x,y,i,j : Integer );
```

```
begin
  ColorTable(0,1,2,3);
  GetPic(BasMat,(((x-i+1)*3+5)-1)*8,((y-j+1)+5)*8,((x-i+1)*3+7)*8-1,((y-
j+1)+6)*8-1);
  ColorTable(1,0,2,3);
  PutPic(BasMat,(((x-i+1)*3+5)-1)*8,((y-j+1)+6)*8-1);
end

Procedure BorraMatriz( x,y,i,j : Integer );
begin
  ColorTable(0,1,2,3);
  PutPic(BasMat,(((x-i+1)*3+5)-1)*8,((y-j+1)+6)*8-1);
end;


Procedure CapturaMatriz;
var
  i,j : Integer;
  x,y : Integer;
  Ch1 : Char;
  Ch2 : Char;
begin
  i := 1; j := 1;
  x := 2; y := 1;
  DibujaMatriz(i,j);
  MarcaMatriz(x,y,i,j);
  Repeat
    LeeTeclas(Ch1,Ch2);
    case ch1 of
      #27 : case ch2 of
        'M' : begin
          if (x-i>8)and(x<50) then begin
            BorraMatriz(x,y,i,j);
            i := i+1; x := x+1;
            DibujaMatriz(i,j);
            MarcaMatriz(x,y,i,j);
          end
          else if x<50 then begin
            BorraMatriz(x,y,i,j);
            x := x+1;
            MarcaMatriz(x,y,i,j);
          end;
        end;
        'K' : begin
          if (x=i)and(x>1)and((x-1)>y) then begin
            BorraMatriz(x,y,i,j);
            i := i-1; x := x-1;
            DibujaMatriz(i,j);
            MarcaMatriz(x,y,i,j);
          end
          else if (x>1)and((x-1)>y) then begin
            BorraMatriz(x,y,i,j);
            x := x-1;
            MarcaMatriz(x,y,i,j);
          end;
        end;
        'P' : begin
          if (y-j>8)and(y<50)and(x>(y+1)) then begin
```

```
            BorraMatriz(x,y,i,j);
            j := j+1; y := y+1;
            DibujaMatriz(i,j);
            MarcaMatriz(x,y,i,j);
          end
          else if (y<50)and(x>(y+1)) then begin
            BorraMatriz(x,y,i,j);
            y := y+1;
            MarcaMatriz(x,y,i,j);
          end;
        end;
        'H' : begin
          if (y=j)and(y>1) then begin
            BorraMatriz(x,y,i,j);
            j := j-1; y := y-1;
            DibujaMatriz(i,j);
            MarcaMatriz(x,y,i,j);
          end
          else if y>1 then begin
            BorraMatriz(x,y,i,j);
            y := y-1;
            MarcaMatriz(x,y,i,j);
          end;
        end;
        else
      end;
      '1' : begin
        LaGraf.Adyalx,yl := 1;
        LaGraf.Adyaly,xl := 1;
        BorraMatriz(x,y,i,j);
        DibujaMatriz(i,j);
        MarcaMatriz(x,y,i,j);
      end;
      '0' : begin
        LaGraf.Adyalx,yl := 0;
        LaGraf.Adyaly,xl := 0;
        BorraMatriz(x,y,i,j);
        DibujaMatriz(i,j);
        MarcaMatriz(x,y,i,j);
      end;
      else
    end;
  until (Ch1=#27)and(Ch2=' ');
end;

begin
  CapturaMatriz;
  ColorTable(0,1,2,3);
end;
```

Apéndice L:
EDITOR GRAFICO

Implementación de un editor gráfico, con las siguientes características:
- **Capacidad virtual para generar gráficas, creando al mismo tiempo la matriz de adyacencias.**
- **Edición de las gráficas generadas en el programa, ya sean aleatorias, ciclos o completas.**

```
Type

  Icono =
    Record
      x,y : Integer;
      Imagen : array [1..200] of Byte;
      ID     : Integer;
    End;
  Cursor =
    Record
      Imagen : array [1..80] of Byte;
    end;

Var
  manita      : Cursor;
  Vertito     : Cursor;
  aristita    : Cursor;
  candito     : Cursor;
  detonita    : Cursor;
  basura      : Cursor;
  blanco      : Cursor;
  Vert,Aris   : Icono;
  Detona,Mano : Icono;
  Canda,Disco : Icono;
  SisOp       : Icono;
  IconBaul    : File of Icono;
  CursBaul    : File of Cursor;
  Opcion      : Integer;
  CH          : Char;
  i           : Integer;


Procedure CambiaVentana( i : Integer );
{
                Procedimiento para cambiar alternativamente de la ventana de
                                edición a la pantalla completa.
}
begin
  if i = 2 then begin
    GraphWindow(2,27,579,198);
  end
  else begin
    GraphWindow(0,0,639,199);
    ColorTable(0,1,2,3);
```

```
  end;
end;

Procedure Arista( i,j : Integer );
var
  s,c   : Real;
  x1,y1 : Integer;
  x2,y2 : Integer;
begin
  with LaGraf do begin
    Draw(Pos[i].x+8,Pos[i].y-4,Pos[j].x+8,Pos[j].y-4,0);
    Adya[i,j] := 1;
    Adya[j,i] := 1;
  end;
end;

Procedure ReDibuja;
var
  i,j : Integer;
  Num : Integer;
begin
  CambiaVentana(2);
  FillScreen(1);
  Num := LaGraf.NVert;
  ColorTable(1,0,2,3);
  for j := 1 to Num do begin
    PutPic(Vertito,LaGraf.Pos[j].x,LaGraf.Pos[j].y);
  end;
  for j := 1 to Num do begin
    for i := 1 to Num do begin
      if (LaGraf.Adya[i,j] <> 0)and(i<>j) then
        Arista(i,j);
    end;
  end;
  CambiaVentana(1);
end;

Procedure ReDibujaM( M : Marca );
var
  i,j : Integer;
  Num : Integer;
begin
  CambiaVentana(2);
  FillScreen(1);
  Num := LaGraf.NVert;
  ColorTable(1,0,2,3);
  for j := 1 to Num do begin
    if not(M[j]) then
      PutPic(Vertito,LaGraf.Pos[j].x,LaGraf.Pos[j].y);
  end;
  for j := 1 to Num do begin
    for i := 1 to Num do begin
      if (LaGraf.Adya[i,j] <> 0)and(i<>j)and
         (not(m[j]))and(not(m[i])) then
        Arista(i,j);
    end;
  end;
  CambiaVentana(1);
```

```
end;

Procedure Ayuda( Opcion : Integer );
{
                    Ofrece ayuda al usuario sobre la operación del editor.
}
begin
  with VInfo do begin
    x1 := 8; x2 := 43;
    y1 := 6; y2 := 11;
    case Opcion of
      1 : begin
        Nombre := 'Mover Vertice';
      end;
      2 : begin
        Nombre := 'Nuevo Vertice';
      end;
      3 : begin
        Nombre := 'Arista';
      end;
      4 : begin
        Nombre := 'Borrar Arista';
      end;
      5 : begin
        Nombre := 'Borrar Vertice';
      end;
      else
    end;
  end;
  Case Opcion of
    1 : begin
      IniciaVentana(VInfo);
      WriteLn('  Mueva la mano hacia el vertice');
      WriteLn('que desee mover y  presione  en-');
      WriteLn('ter. Mueva el vertice a su nueva');
      WriteLn('posicion y presione <enter>.     ');
      Delay(3000);
      TerminaVentana(VInfo);
    end;
    2 : begin
      IniciaVentana(VInfo);
      WriteLn('  Mueva el nuevo vertice a su    ');
      WriteLn('posicion y presione <enter> para');
      WriteLn('fijarlo.                         ');
      Delay(3000);
      TerminaVentana(VInfo);
    end;
    3 : begin
      IniciaVentana(VInfo);
      WriteLn('  Mueva la flecha hacia uno de   ');
      WriteLn('los extremos de la arista y pre- ');
      WriteLn('sione <enter>. Mueva el  vertice ');
      WriteLn('al 2o extremo y presione <enter>.');
      Delay(3000);
      TerminaVentana(VInfo);
    end;
    4 : begin
      IniciaVentana(VInfo);
```

```
      WriteLn('  Mueva la flecha hacia uno de  ');
      WriteLn('los extremos de la arista a borrar');
      WriteLn('y teclee <enter>.Mueva el vertice ');
      WriteLn('hacia el 2o. extremo y presione   ');
      WriteLn(' <enter>.                         ');
      Delay(3000);
      TerminaVentana(VInfo);
  end;
    5 : begin
      IniciaVentana(VInfo);
      WriteLn('  Marque con la mano y la dinami-');
      WriteLn('ta un rectangulo que contenga los');
      WriteLn('vertices que quiera borrar.       ');
      Delay(3000);
      TerminaVentana(VInfo);
    end;
    else
  end;
end;

Procedure IniciaEditor;
{
                    Procedimiento que dibuja la pantalla de edición.
}
  begin
    assign(IconBaul,'Editor.Icn');
    reset(IconBaul);
    Read(IconBaul,mano);
    Read(IconBaul,Vert);
    Read(IconBaul,Aris);
    Read(IconBaul,canda);
    Read(IconBaul,detona);
    read(Iconbaul,disco);
    read(IconBaul,SisOp);

    assign(CursBaul,'Editor.Cur');
    reset(CursBaul);
    Read(CursBaul,manita);
    Read(CursBaul,Vertito);
    Read(CursBaul,Aristita);
    Read(CursBaul,candito);
    Read(CursBaul,detonita);
    Close(IconBaul); Close(CursBaul);
  end;

Procedure IniciaPantalla;
{
}
begin
    HiRes; HiResColor(15); FillScreen(0);
    rectangulo(1,26,580,199,1);
    rectangulo(1,15,580,199,1);
    draw(1,20,580,20,1);
    draw(1,22,580,22,1);
    CambiaVentana(2);
    FillScreen(1);
    CambiaVentana(1);
    rectangulo(588,26,632,187,1);
```

```
    draw(590,30,630,30,1);
    draw(590,32,630,32,1);
    ColorTable(1,0,2,3);
    PutPic(mano.imagen,590,56);
    PutPic(Vert.imagen,590,76);
    PutPic(Aris.imagen,590,96);
    PutPic(Canda.imagen,590,116);
    PutPic(Detona.imagen,590,136);
    draw(590,140,630,140,1);
    draw(590,142,630,142,1);
    PutPic(Disco.imagen,590,166);
    PutPic(SisOp.imagen,590,186);
    GoToXY(35,3); Write(NombreAr);
  end;        ( Del procedimiento IniciaPantalla )

Procedure Editor;

Procedure PonImagen( i      : Integer;
                     Color : Boolean );
{
                     Procedimiento para poner un dibujo en la pantalla.
}
begin
  if Color then
    ColorTable(0,1,2,3)
  else
    ColorTable(1,0,2,3);
  case i of
    1 : PutPic(mano.imagen,590,56);
    2 : PutPic(Vert.imagen,590,76);
    3 : PutPic(Aris.imagen,590,96);
    4 : PutPic(Canda.imagen,590,116);
    5 : PutPic(Detona.imagen,590,136);
    6 : PutPic(Disco.imagen,590,166);
    7 : PutPic(SisOp.imagen,590,186);
  end;    (   del case )
  ColorTable(0,1,2,3);
end;

Procedure PonCursor( i : Integer;
                     x,y : Integer;
                     Color : Boolean );
begin
  if Color then
    ColorTable(0,1,2,3)
  else
    ColorTable(1,0,2,3);
  case i of
    1 : PutPic(manita,x,y);
    2 : PutPic(Vertito,x,y);
    3 : PutPic(Aristita,x,y);
    4 : PutPic(Candito,x,y);
    5 : PutPic(Detonita,x,y);
    else
  end;
  ColorTable(0,1,2,3);
```

```
end;

Procedure LeeOpcion( var
                       Op : Integer );
{
                Procedimiento que maneja el menú lateral a la ventana de edición.
}
var
  i       : Integer;
  Ch1,Ch2 : Char;
  Termino : Boolean;
begin
  i := 1;
  Termino := False;
  Repeat
                    end;
            'P' : if i<7 then begin
                    PonImagen(i,False);
                    i := i+1;
                  end;
          else
        end;
    else
  end;
  Until Termino;
end;    {  Termina el procedimiento }


Procedure MueveCur( i       : Integer;
                    var
                      x,y   : Integer;
                      x2,y2 : Integer );
begin
  PutPic(Basura,x,y);
  GetPic(Basura,x2,y2-8,x2+16,y2);
  PonCursor(i,x2,y2,False);
  x := x2; y := y2;
end;


Procedure ManejaCursor( i   : Integer;
                        var
                        x,y : Integer );
var
  sx,sy   : Integer;
  ch1,ch2 : Char;
  Termino : Boolean;
begin
  Termino := False;
  Sx := 16; SY := 8;
  GetPic(basura,x,y-8,x+16,y);
  PonCursor(i,x,y,False);
  repeat
    LeeTeclas(Ch1,Ch2);
    case Ch1 of
      #13 : Termino := True;
      #27 : case Ch2 of
              ' ' : begin
```

```
                    PutPic(Basura,x,y);
                    x := 0; y := 0;
                    Termino := True;
                  end;
          'G' : begin
                  if (x>16)and(y>27) then
                    MueveCur(i,x,y,x-sx,y-SY);
                  end;
          'H' : begin
                  if (y>27) then
                    MueveCur(i,x,y,x,y-SY);
                  end;
          'I' : begin
                  if (y>27)and(x<600) then
                    MueveCur(i,x,y,x+sx,y-SY);
                  end;
          'K' : begin
                  if (x>16) then
                    MueveCur(i,x,y,x-sx,y);
                  end;
          'M' : begin
                  if (x<600) then
                    MueveCur(i,x,y,x+sx,y);
                  end;
          'O' : begin
                  if (x>16)and(y<210) then
                    MueveCur(i,x,y,x-sx,y+SY);
                  end;
          'P' : begin
                  if (y<210) then
                    MueveCur(i,x,y,x,y+SY);
                  end;
          'Q' : begin
                  if (y<210)and(x<600) then
                    MueveCur(i,x,y,x+sx,y+SY);
                  end;
        else
      end;
    else
  end;
until Termino;
end;

Procedure BuscaVert( x,y    : Integer;
                     var
                       Vert : Integer );
var
  i : Integer;
begin
  Vert := 0;
  with LaGRaf do begin
    for i := 1 to NVert do
      if (Pos[i].x=x)and(Pos[i].y=y) then
        Vert := i;
  end;
end;
```

```
Procedure BorraArista( i,j : Integer );
var
  s,c   : Real;
  x1,y1 : Integer;
  x2,y2 : Integer;
begin
  with LaGraf do begin
    Adyal[i,j] := 0;
    Adyal[j,i] := 0;
  end;
  CambiaVentana(2);
  ReDibuja;
  CambiaVentana(1);
end;

Procedure NuevoVertice( x,y : Integer );
var
  i : Integer;
begin
  with LaGraf do begin
    NVert := NVert+1;
    Pos[NVert].x := x;
    Pos[NVert].y := y;
    for i := 1 to NVert do begin
      Adyal[i,NVert] := 0;
      Adyal[NVert,i] := 0;
    end;
  end;
end;


Procedure Borra( Vert : Integer );
var
  i,j : Integer;
begin
  with LaGraf do begin
    for j := Vert+1 to NVert do begin
      Pos[j-1].x := Pos[j].x;
      Pos[j-1].y := Pos[j].y;
      for i := 1 to Vert-1 do begin
        Adyal[j-1,i] := Adyal[j,i];
        Adyal[i,j-1] := Adyal[i,j];
      end;
      for i := Vert+1 to NVert do begin
        Adyal[j-1,i-1] := Adyal[j,i];
      end;
    end;
    NVert := NVert-1;
  end;
end;

Procedure InformaVertice( Vertice : Integer );
Var
  LM : Marca;
  i  : Integer;
begin
  CambiaVentana(1);
  for i := 1 to MaxVert do
```

```
    LM[1] := False;
  with VInfo do begin
    Nombre := 'Mas Informacion';
    x1 := 7; x2 := 50;
    y1 := 7; y2 := 10;
  end;
  IniciaVentana(VInfo);
  GoToXY(1,1);
  WriteLn('Vertice Numero : ',Vertice:2);
  WriteLn('Grado          : ',Grado(Vertice,LM):2);
  ReadLn;
  TerminaVentana(VInfo);
end;


Procedure ManejaOpcion( Opcion : Integer );
{
                Procedimiento para manejar las diferentes opciones del menú de la
                                izquierda de la pantalla de edición.
}
var
  x,y           : Integer;
  x2,y2         : Integer;
  Vertice,vert2 : Integer;
  i,xAux,yAux   : Integer;
begin
  case Opcion of
    1 : begin
          x := 290; y := 80;
          ManejaCursor(Opcion,x,y);
          PutPic(Basura,x,y);
          BuscaVert(x,y,vertice);
          if vertice <> 0 then begin
            ManejaCursor(2,x,y);
            LaGraf.Pos[vertice].x := x;
            LaGraf.Pos[vertice].y := y;
            ReDibuja;
          end;
        end;
    2 : begin
          x := 290; y := 80;
          ManejaCursor(Opcion,x,y);
          if (x<>0)or(y<>0) then
            NuevoVertice(x,y);
        end;
    3 : begin
          x := 290; y := 80;
          ManejaCursor(Opcion,x,y);
          PutPic(Basura,x,y);
          if (x<>0)or(y<>0) then begin
            BuscaVert(x,y,vertice);
            if vertice <> 0 then begin
              ManejaCursor(2,x,y);
              PutPic(Basura,x,y);
              if (x<>0)or(y<>0) then begin
                BuscaVert(x,y,vert2);
                if Vert2 <> 0 then begin
```

```
              LaGraf.Adya[vertice,vert2] := 1;
              Arista(Vertice,Vert2);
            end;
          end;
        end;
      end;
    end;
  4 : begin
      x := 290; y := 80;
      ManejaCursor(3,x,y);
      PutPic(Basura,x,y);
      if (x<>0)or(y<>0) then begin
        BuscaVert(x,y,vertice);
        if vertice <> 0 then begin
          ManejaCursor(2,x,y);
          PutPic(Basura,x,y);
          if (x<>0)or(y<>0) then begin
            BuscaVert(x,y,vert2);
            if Vert2 <> 0 then begin
              BorraArista(Vertice,Vert2);
            end;
          end;
        end;
      end;
    end;
  5 : begin
      x := 290; y := 80;
      ManejaCursor(1,x,y);
      PutPic(Basura,x,y);
      if (x<>0)or(y<>0) then begin
        x2 := x; y2 := y;
        ManejaCursor(5,x2,y2);
        PutPic(Basura,x2,y2);
        if x>x2 then begin
          xAux := x;
          x := x2;
          x2  := xAux;
        end;
        if y>y2 then begin
          yAux := y;
          y := y2;
          y2 := yAux;
        end;
        with LaGraf do begin
          for i := 1 to NVert do begin
            if (((Pos[i].x<=x2)and(Pos[i].x>=x))and
              ((Pos[i].y<=y2)and(Pos[i].y>=y))) then
              Borra(i);
          end;
        end;
      end;
      ReDibuja;
    end;
  6 : begin
      x := 290; y := 80;
      ManejaCursor(1,x,y);
      PutPic(Basura,x,y);
      if (x<>0)or(y<>0) then begin
```

```
            BuscaVert(x,y,vertice);
            if Vertice<>0 then
              InformaVertice(Vertice);
          end;
        end;
    else
      CambiaVentana(1);
  end;
end;
```

```
{
```

Programa Principal del Procedimiento

```
}

begin
  Repeat
    LeeOpcion(Opcion);
    if MAyuda then
      Ayuda(Opcion);
    CambiaVentana(2);
    ManejaOpcion(Opcion);
    CambiaVentana(1);
  Until Opcion=7;
  if LaGraf.NVert>=1 then
    Vacia := False;
  CambiaVentana(1);
end;
```

Apéndice M:

ALGORITMO DE COLORACION

Implementación de una heurística para el problema de la coloración de vértices. Consta de los siguientes pasos :

1. ITERACION 1 :
   Quitar el vértice de menor grado, guardarlo en una pila y trabajar con la gráfica restante hasta que ésta conste de 1 solo punto.
2. Colorear el vértice restante del color 1.
3. ITERACION 2 :
   Sacar un vértice de la pila, colorearlo del primer color que no se encuentre entre sus adyacentes. Continuar así hasta que se vacíe la pila.

```
Procedure ColoreaPorMin;
{
        ColoreaPorMin implementa la heurística comentada al principio del programa.
}

Var
  Marcados : Marca;
  i,j      : Integer;
  UnVert   : Integer;
  CVert    : Integer;
  CColor   : Integer;
  Funciona : Boolean;
begin
  DGorro := 0;                      {  Asigna cero a delta'                    }
  IniciaPila;                       {  Pone la altura de la pila en cero       }
  for i := 1 to LaGraf.NVert do
    Marcados[i] := False;           {  Quita las marcas a todos los vértices   }
  CVert := LaGraf.NVert;            {  Asigna a CVert el número de vértices    }
  while CVert>1 do begin            {  Mientras CVert sea menor que 1 ...      }
    i := GradoMin(UnVert,Marcados); {    encuentra un vértice de grado min.   }
    if i>dGorro then                {    si el grado menor encontrado es      }
      dGorro := i;                  {    menor que delta', entonces asigna i  }
    MetePila(UnVert);               {    lo mete en la pila                   }
    Marcados[UnVert] := True;       {    lo marca                             }
    CVert := LaGraf.NVert;
    for i := 1 to LaGraf.NVert do   {    cuenta los vértices marcados         }
      if Marcados[i] then
        CVert := CVert-1;
  end;                              {  ... termina                             }
  for i := 1 to LaGraf.NVert do     {  Encuentra el vértice que no está        }
    if not(Marcados[i]) then        {  marcado                                 }
      UnVert := i;
  LaGraf.Color[UnVert] := 1;        {  Le asigna el color 1                    }
```

```
  CColor :=1;                    { CColor=el número de colores usados    }
  while Tope>=1 do begin         { Mientras la pila tenga elementos ...  }
    SacaPila(UnVert);            {   sacar un elemento                   }
    Marcados[UnVert] := False;   {   quitarle la marca                   }
    j := 0;
    repeat                       {   Encontrar el primer color que       }
      j := j+1;                  {   sea posible asignarle               }
      Funciona := True;
      for i := 1 to LaGraf.NVert do begin
        if (i<>UnVert)and(LaGraf.Adyali,UnVert]<>0)and
        (LaGraf.Color[i]=j)and(not(Marcados[i])) then
          Funciona := False;
      end;
    Until Funciona;
    LaGraf.Color[UnVert] := j;   {   Asignarle el color encontrado       }
  end;
end;

Procedure Resultados;
{
                Programa que despliega los resultados del algoritmo de coloración.
}
Var
  i,DR,DC : Integer;
begin
  LowVideo;
  writeln('                              RESULTADOS');
  writeln('_____________________________________________________________________');
  for i := 1 to ((LaGraf.NVert-1) div 10)+1 do begin
    GoToXY((i-1)*12+1,3);
    Write('No : Color');
  end;
  GoToXY(1,15);
  writeln('_____________________________________________________________________');
  WriteLn('d^ = ',dGorro);
  for i := 1 to LaGraf.NVert do begin
    DR := (i mod 10+1)+4;
    DC := ((i) div 10)*12+3;
    GoToXY(DC,DR);
    Write(i,' : ',LaGraf.Color[i]);
  end;
  ReadLn;
end;


Procedure DemoColoreaPorMin;
{
    Demustra el funcionamiento del ordenamiento, coloreando la gráfica residente en memoria.
}

Var
  Marcados : Marca;
  i,j      : Integer;
  UnVert   : Integer;
  CVert    : Integer;
  CColor   : Integer;
  Funciona : Boolean;
begin
```

```
    dGorro := 0;                    { Asigna cero a delta'                    }
    IniciaPila;                     { Pone la altura de la pila en cero       }
    for i := 1 to LaGraf.NVert do
      Marcados[i] := False;         { Quita las marcas a todos los vértices   }
    CVert := LaGraf.NVert;          { Asigna a CVert el numero de vertices    }
    while CVert>1 do begin          { Mientras CVert sea menor que 1 ...      }
      i := GradoMin(UnVert,Marcados); {   encuentra un vértice de grado min.  }
      if i>dGorro then              {   si el grado menor encontrado es       }
        dGorro := i;                {   menor que delta', ent.se asigna i     }
      MetePila(UnVert);             {   lo mete en la pila                    }
      Marcados[UnVert] := True;     {   lo marca                              }
      CVert := LaGraf.NVert;
      for j := 1 to LaGraf.NVert do {   cuenta los vértices marcados          }
        if Marcados[j] then
          CVert := CVert-1;
      ReDibujaM(Marcados);
      IniciaVentana(VInfo);
      WriteLn('Grado del Vertice : ',i);
      WriteLn('Maximo grado min. : ',dGorro);
      ReadLn;
      TerminaVentana(VInfo);
    end;                            { ... termina                             }
    for i := 1 to LaGraf.NVert do   { Encuentra el vértice que no esté        }
      if not(Marcados[i]) then      { marcado                                 }
        UnVert := i;
    LaGraf.Color[UnVert] := 1;      { Le asigna el color 1                    }
    CColor :=1;                     { CColor=el número de colores usados      }
    while Tope>=1 do begin          { Mientras la pila tenga elementos ...    }
      SacaPila(UnVert);             {   sacar un elemento                     }
      Marcados[UnVert] := False;    {   quitarle la marca                     }
      j := 0;
      repeat                        {   Encontrar el primer color que         }
        j := j+1;                   {   sea posible asignarle                 }
        Funciona := True;
        for i := 1 to LaGraf.NVert do begin
          if (i<>UnVert)and(LaGraf.Adyal[i,UnVert]<>0)and
          (LaGraf.Color[i]=j)and(not(Marcados[i])) then
            Funciona := False;
        end;
      until Funciona;
      LaGraf.Color[UnVert] := j;    {   Asignarle el color encontrado         }
      ReDibujaM(Marcados);
      IniciaVentana(VInfo);
      WriteLn('Color Asignado : ',j);
      ReadLn;
      TerminaVentana(VInfo);
    end;
end;
```