00324

9

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

"El Juego de Sim "

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

MATEMATICA

P R E S E N T A :

Consuelo Isabel Doddoli de la Macorra

DIVISIÓN DE ESTUDIOS PROFESIONALES

Director de Tesis

Dr. Francisco Larrión Rivero II

FACULTAD DE CIENCIAS
SECCION ESCOLAR

2003

PAGINACION DISCONTINUA

DRA. MARÍA DE LOURDES ESTEVA PERALTA
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito: El Juego de Sim

realizado por Consuelo Isabel Doddoli de la Macorra

con número de cuenta 7694518-6 , quien cubrió los créditos de la carrera de: Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

A t e n t a m e n t e

Director de Tesis Propietario	Dr. Francisco Larrión Riveroll
Propietario	M. en C. Guadalupe Ibargüengoitia González
Propietario	Dr. Javier Paez Cárdenas
Suplente	Dr. Pedro Miramontes Vidal
Suplente	Fis. Marco Antonio Zepeda Zepeda

Consejo Departamental de Matemáticas

M. en C. José Antonio Gómez Ortega
CONSEJO DEPARTAMENTAL

Dedico este trabajo:

A mis hijos Alejandro y Andrea.

A la memoria de mi querido amigo Víctor Mantilla.

Agradecimientos

A Rafael Soto le agradezco haberme planteado el problema que motivo esta tesis, así como, su ayuda para el elaborar el código del programa del Juego Sim.

A Marco Cepeda le agradezco el cuidado que puso en la lectura de este trabaja gracias a la cual se logró mejorar sustancialmente la redacción de esta tesis..

A Francisco Larrión le agradezco la conducción de este trabajo.

A Julieta Fierro y Julia Tagüeña les agradezco su apoyo y constante estímulo, sin ellos, me hubiera tardado más tiempo en ponerle el punto final a esta tesis.

Agradezco a Guadalupe Ibargüengoitia, Javier Paez y Pedro Miramontes quienes amablemente aceptaron participar como miembros del jurado, sus sugerencias y observaciones contribuyeron a mejorar este escrito.

Finalmente agradezco a Juan Tonda su ayuda técnica.

ÍNDICE

ÍNDICE DE ALGORITMOS

INTRODUCCIÓN

Las matemáticas, en su forma elemental, surgieron en épocas remotas. En la búsqueda por resolver problemas concretos, desde las primeras etapas de su desarrollo, el ser humano construyó muy diversos conceptos matemáticos. En las sociedades antiguas encontramos gran cantidad de ejemplos en los que ya se hace uso de conceptos matemáticos, como el de contar y medir, ya fuera para comerciar, construir, navegar, estudiar los cielos, etcétera. Las matemáticas se han caracterizado por haber estado en un continuo desarrollo; sus principios no se han congelado, tienen vida propia. Ello se debe, en gran parte, al hecho de que muchos de estos conceptos, y los resultados obtenidos, tienen su origen en el mundo real y tienen aplicaciones en múltiples ámbitos de la actividad humana, incluyendo las diversas ramas de la ciencia.

Es importante señalar que continuamente, sin pensar en ello, hacemos uso constante de conceptos y resultados matemáticos. Por ejemplo, empleamos la aritmética para calcular nuestro gasto y la geometría elemental para calcular la superficie de un departamento. Las reglas que se utilizan en estos cálculos son sencillas, pero no debemos olvidar que, en algún período de la antigüedad, representaron los logros matemáticos más avanzados de la época. Asimismo, es imposible concebir la tecnología moderna sin la participación de las matemáticas. Un amplio porcentaje de los procesos técnicos que actualmente se llevan a cabo involucran diversos conceptos matemáticos.

Adicionalmente, también es cierto que las diversas ciencias, en mayor o menor grado, hacen uso de la matemática. Por ejemplo, las llamadas ciencias exactas, como la física y la química, utilizan a las matemáticas para el desarrollo de sus teorías. Por otro lado, las ciencias ubicadas en el campo de lo social, como la antropología, sociología, economía, también se apoyan en conceptos matemáticos al desarrollar algunos de sus planteamientos. Podemos así afirmar, que el progreso de varias ciencias ha ido acompañado de su relación con las matemáticas.

Sin lugar a duda, el desarrollo de las matemáticas es una de las grandes aventuras del pensamiento humano. A partir de lo que inicialmente eran unos cuantos resultados en

aritmética y geometría, la matemática ha acumulado, a través de los siglos, una enorme cantidad de conocimientos; ello ha tenido como consecuencia el desarrollo de muy diversas ramas de la propia matemática.

Este trabajo consiste en la discusión y análisis del problema matemático conocido con el nombre del juego de Sim, el cual consiste de seis puntos y dos jugadores. Alternadamente cada jugador, en su turno, une dos puntos dibujando una línea de su color; el primer jugador utiliza el color rojo, el segundo, el azul. El jugador que dibuje primero un triángulo de su propio color, con vértices en algunos de los seis puntos originales, es el que pierde el juego.

Para llevar a cabo este análisis, utilizamos conceptos de la teoría de gráficas y la teoría de juegos. Adicionalmente, construimos un algoritmo, con base en el cual desarrollamos un programa de cómputo que permite jugar el juego mencionado. Para el desarrollo del algoritmo y del programa haremos uso de algunos conceptos básicos de la inteligencia artificial.

En el Capítulo I, por el Teorema de Ramsey, demostramos que en una gráfica completa de orden seis siempre hay un triángulo por lo tanto, en el juego de Sim no es posible el empate, hay un perdedor y un ganador.

En los Capítulos II y III diseñamos un algoritmo que juegue el juego de Sim en la computadora, para ello hacemos uso de algunos conceptos de inteligencia artificial y planteamos el juego como un problema de trazar aristas a través de un espacio de estados, en el que cada estado corresponde a una configuración del juego, empezando en un estado inicial y, mediante el uso de un conjunto de reglas que permiten pasar de un estado a otro, termina en alguno de los estados objetivos. El problema consiste entonces, para cada jugador, en encontrar una ruta a través del espacio de estados que una el estado inicial con alguno de los estados objetivos. Cada estado del juego de Sim es una gráfica coloreada de orden seis.

La generación de estados, en este trabajo, se hace a partir del estado inicial, utilizando los operadores, y después expandimos cada uno de los estados generados y así sucesivamente hasta llegar a un estado objetivo. Para facilitar la compresión de este proceso lo concebimos como si fuera la construcción de un árbol de búsqueda sobre puesto al espacio de estados.

La construcción de este árbol consiste en expandir el nodo raíz y luego todos los nodos generados por éste, y así sucesivamente.

En juego como el de Sim sabemos, por el teorema de Zermelo, que es un juego que está estrictamente determinado, esto es, desde un principio se puede saber el resultado final del juego si los jugadores eligen siempre su mejor opción. Utilizamos este algoritmo para determinar la mejor jugada de los jugadores, así como, el ganador del juego para lo cual definimos primero la función de pagos en los nodos terminales del árbol. Después, se define la máxima ganancia posible del jugador en turno en los nodos no terminales del penúltimo nivel del árbol. Esto se hace tomando en cuenta, para cada uno de estos nodos, las ganancias en los nodos hijos. Y a continuación se hace lo mismo para los nodos del nivel anterior, sólo que ahora se maximiza la ganancia para el otro jugador, pues a él le corresponden los nodos de ese nivel. Continuando de esta manera, se marcha hacia atrás hasta el nodo raíz.

La construcción del árbol tiene la desventaja de que si aparecen varias veces nodos que representan al mismo estado perdemos tiempo al expandirlos todos. La memoria y el tiempo de máquina que se desperdician expandiendo nodos más de una vez puede evitarse "podando" los nodos equivalentes. Así en lugar de construir un árbol de búsqueda construimos la gráfica dirigida.

Decimos que los estados (o los nodos del árbol de juego) A y B son equivalentes cuando las gráficas coloreadas correspondientes son equivalentes, esto es si existe un isomorfismo cromático de A en B.

Para decidir si los estados A y B son equivalentes hacemos lo siguiente: 1) Utilizamos el hecho de que cada estado del juego de Sim es una gráfica coloreada de orden seis para representarlo, en la máquina, a través de su matriz de adyacencia. 2) Generamos todas las posibles permutaciones de los vértices del estado A y B; cada permutación la representamos con la matriz de adyacencia correspondiente. 3) Si el conjunto de matrices correspondientes a las permutaciones de los vértices del estado A es igual al conjunto de matrices formado por las permutaciones de los vértices del estado B entonces los estados A y B son equivalentes.

Para decidir con la computadora si los estados A y B son iguales construimos una función que a cada matriz de adyacencia le asigne un número, de tal forma que, para matrices distintas el número sea distinto y para matrices iguales el número sea el mismo. Entonces, decidir si los estados A y B son iguales es lo mismo que decidir si el número más pequeño del conjunto formado por los números asociados a cada permutación de los vértices del

estado A es igual al número más pequeño del conjunto de números asociados a cada permutación de los vértices del estado B.

Con este número representamos cada nodo de la gráfica del juego en la computadora y a partir del él generamos toda la información necesaria tanto para expandir los nodos como para conocer cada estado del juego.

En realidad, en este trabajo, se aplica el algoritmo de Zermelo a la gráfica dirigida y no al árbol del juego, esto es posible debido a que por cada nodo eliminado quedó en esta gráfica uno que es equivalente. Además, cada nodo de la gráfica dirigida pertenece, al igual que en el árbol, a un solo nivel.

En el Capítulo IV analizamos con el algoritmo de Zermelo el juego de Sim y concluimos que el segundo jugador gana el juego, esto se puede ver en el archivo *ncalific.dat*, con lo que tenemos una prueba computacional.

En este capítulo también se describen tres algoritmos para jugar el juego de Sim:

La primera estrategia, la llamamos Trikis, con la cual el segundo jugador, si en cada momento del juego elige su mejor opción, gana el juego de Sim aún si el primer jugador juega en forma perfecta.

La segunda estrategia, la llamamos Trakis, es para el primer jugador (el perdedor en principio), y sólo permite ganar en el caso de que el segundo jugador (el ganador), se equivoque y no juegue en forma perfecta.

En el archivo *ncalific.dat* se puede observar también que Trakis, a pesar de ser el primer jugador (el perdedor en principio) tiene una probabilidad de 0.9984 de ganar si el segundo jugador juega al azar y sólo cuidando de no hacer un triángulo azul si esto es posible.

La tercera estrategia, la llamamos Trukis, y es tanto para el primer jugador como para el segundo; en ella, el jugador elige al azar una opción que lo conduzca a un nodo que no contenga triángulo; y en el caso de que no exista al menos un nodo sin triángulo, escoge al azar cualquier nodo.

CAPÍTULO I

Teoría de gráficas

I.1.Antecedentes de la teoría de gráficas

La teoría de gráficas es una rama de las matemáticas en la que los problemas se representan y se resuelven justamente utilizando una colección de puntos unidos por aristas, llamadas gráficas.

Con la teoría de gráficas se pueden estudiar problemas de muy diversa índole; ya sean problemas sumamente abstractos o problemas sumamente concretos, como lo son problemas de redes de calles, sistemas de rutas aéreas, redes de comunicación, la red de agua en una ciudad y muchos otros.

El primer problema en el que se usaron conceptos de la teoría de gráficas, es el de los famosos puentes de Königsberg; mismo que fue resuelto por Leonhar Euler en 1736.

En siglo XVIII, en el pueblo de Königsberg (Prusia) había dos islas en el río Pregel, las cuales estaban conectadas entre sí por un puente, a la vez que estaban conectadas mediante otros seis puentes con las riberas, como se muestra en la Fig.I.1.

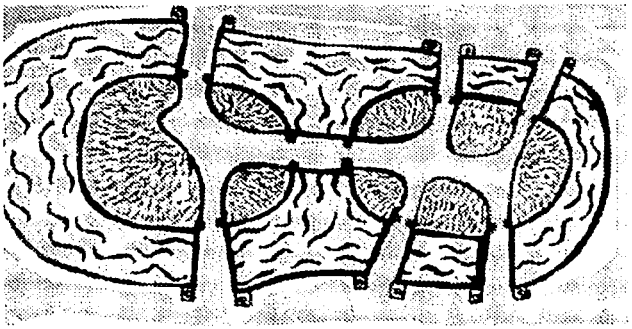


Figura I.1. Puentes de Königsberg.

Los habitantes de Königsberg se preguntaban si era posible cruzar los siete puentes en un paseo continuo, pasando por todos los puentes solamente una vez. Euler abordó este problema, lográndolo resolver. Para ello representó el problema mediante una gráfica; utilizando puntos para representar las áreas de tierra y líneas para representar los puentes que las unían. La Fig.I.2 muestra el problema mediante la gráfica asociada.

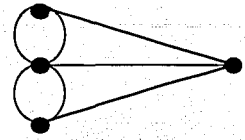


Figura I.2. Gráfica asociada al problema de los puentes de Königsberg.

El problema, en términos matemáticos, se puede plantear de la forma siguiente. ¿Es posible recorrer todas las líneas de la gráfica anterior sin levantar el lápiz del papel y sin pasar dos veces por una misma línea?.
Euler, de manera ingeniosa, demostró que dicho recorrido es imposible. A continuación describimos su razonamiento.

Observemos que cada uno de los cuatro puntos de la Fig. I.2 tiene un número impar de líneas que lo conectan con los demás; en uno de los puntos convergen cinco líneas y en los restantes sólo convergen tres.

Tomemos como punto inicial del recorrido cualquier punto. En términos de puntos de inicio y puntos de terminación del recorrido, hay sólo dos posibilidades; a saber, que el recorrido empiece y termine en el mismo punto o que el punto de inicio y de terminación sean diferentes. En cualquiera de los dos casos habrá, por lo menos, dos puntos que no son principio ni final del recorrido. Así, cada vez que lleguemos a alguno de estos puntos intermedios, tendremos que volver a salir de ellos, por ser precisamente puntos intermedios en el recorrido. Dado que hemos puesto como condición recorrer todas las líneas una sola vez, es obvio que en estos puntos intermedios necesitamos un número par de líneas para poder entrar y salir cuantas veces sea necesario. Por tanto, es imposible hacer el recorrido pasando una sola vez por cada una de las líneas.
Con el problema anterior se dio inicio a la teoría de gráficas.

I.2. Definiciones y conceptos básicos de la teoría de gráficas.

Definición. 1 Una **gráfica simple** es una pareja $G = (V, A)$, donde V es un conjunto no vacío de elementos y A es un conjunto de pares no ordenados de elementos de V. Los elementos de V se llaman **vértices** o **puntos** y las parejas de A se llaman **aristas** o **líneas.**

Según esta definición, una arista $a = \{u, v\}$ es un conjunto formado por dos elementos distintos de V, también se acostumbra denotarla como $a = [u, v]$ o simplemente $a = uv$, pero recordando que $[u, v] = [v, u]$ y $uv = vu$.

Si u y v son vértices de G, entonces diremos que la arista $[u, v]$ **une** al vértice u con el vértice v, y que u y v son sus **extremos**; diremos que una arista **incide** en el vértice v, si tiene al vértice v como uno de sus extremos. Por convención, denotaremos por $V(G)$ al conjunto de vértices de G y por $A(G)$ al conjunto de aristas de G.

De la misma manera en que Euler representó el problema de los puentes de Königsberg mediante un diagrama, es conveniente representar cualquier gráfica por un diagrama. Cada vértice se representa por un punto o un círculo pequeño y cada arista por un segmento de curva que une adecuadamente a los dos extremos.

Debemos aclarar que la gráfica de Euler, representada en la Fig.I.2, no es una gráfica simple, ya que existen pares de aristas diferentes con los mismos extremos y por la Def. 1, en una gráfica simple, dados dos vértices u, v a lo más hay una arista que los une (pues el par $\{u, v\}$ sólo puede estar o no en el conjunto $A(G)$). En realidad, la gráfica de Euler es un ejemplo de lo que se conoce en teoría de las gráficas como una multigráfica, pero en este trabajo sólo utilizaremos gráficas simples. Es muy sencillo construir una gráfica simple a partir de la gráfica de Euler "subdividiendo las aristas múltiples", esto es, introduciendo cuatro nuevos vértices x, y, z y w para reemplazar las aristas múltiples por trayectorias de largo dos. Más formalmente, construyamos la gráfica G, donde el conjunto de vértices V es el conjunto $\{V_1,V_2,V_3,V_4,X,Y,Z,W\}$, (los elementos v_i representan las áreas de tierra) y el conjunto de aristas A es $\{[V_1,X], [X,V_2], [V_1,Y], [Y,V_2], [V_2,Z], [Z,V_3], [V_2,W], [W,V_3],$

$[V_1,V_4]$, $[V_2,V_4]$, $[V_3,V_4]\}$, cuyos elementos representan los puentes. Esta situación queda representada en la Fig.I.3. Es claro que nuestra nueva gráfica G es también, al igual que la de Euler, un modelo útil para el problema de los puentes de Königsberg (puede pensarse que los cuatro vértices agregados representan bancas a la mitad de los puentes). También es claro que el razonamiento que utilizó Euler no se altera por las bancas que hemos puesto a la mitad de algunos puentes. A partir de ahora, y a lo largo de este trabajo, siempre que se diga **gráfica** nos referiremos a **gráfica simple.**

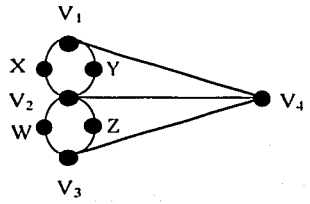


Figura I.3. Representación de los puentes de Königsberg.

Se dice que una gráfica es finita si el número de vértices es finito. Nosotros trabajaremos con gráficas finitas.

A continuación agregamos algunas definiciones que nos serán útiles en el análisis de las gráficas.

Definición. 2 Cualesquiera dos vértices u, v de una gráfica G se dice que son **adyacentes** si están unidos por una arista.

Por ejemplo, en la gráfica de los puentes de Königsberg (Fig. I.3), V_1 y X son adyacentes, como V_1 y V_4, mientras que V_1 y V_3 no son adyacentes.

Definición. 3 Se llama **orden** de G al número de vértices de G. El número de aristas de G es llamado **tamaño** de G.

En el caso de los puentes de Königsberg se tiene una gráfica con ocho vértices y once aristas; esto es, una gráfica de orden ocho y tamaño once.

Definición. 4 Sea v un vértice de G, el número de aristas que inciden en v se llama el **grado** de v en G.

En el problema de los puentes de Königsberg, los vértices V_1, V_3 y V_4 son de grado 3 y el vértice V_2 es de grado 5, mientras que X, Y, Z, W son de grado dos.

Definición. 5 Se dice que una gráfica es **completa** si cualesquiera dos vértices en ella son adyacentes; esto es, todos los vértices están unidos entre sí. Una gráfica G completa con p vértices se denota como $G = K_p$.
En la Fig.I.4 ilustramos algunos ejemplos de gráficas completas.

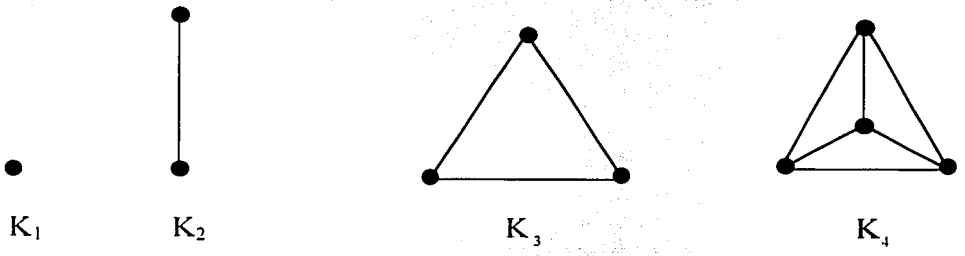


Figura I.4. Gráficas completas.

En matemáticas es importante determinar cuando dos objetos, en algún sentido, son iguales. En particular, en teoría de gráficas es importante determinar las condiciones bajo las cuales dos gráficas serán consideradas como iguales; para ello, introduciremos el concepto de gráficas isomorfas.

Definición. 6 Sean G_1 y G_2 dos gráficas. Sea ϕ: $V(G_1) \rightarrow V(G_2)$ una función biunívoca de $V(G_1)$ en $V(G_2)$. Diremos que ϕ es un **isomorfismo** de G_1 en G_2 si para cualesquiera vértices u_1, v_1 en G_1, éstos son adyacentes si y sólo si los vértices $\phi(u_1)$ y $\phi(v_1)$ son adyacentes en G_2. Adicionalmente, diremos que dos gráficas G_1 y G_2 son **isomorfas** si y sólo si existe un isomorfismo de G_1 en G_2. En la Fig. I.5 se muestran dos gráficas isomorfas G_1 y G_2.

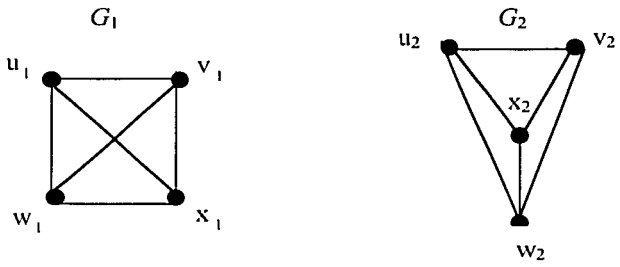


Figura I.5. Gráficas isomorfas.

La relación de isomorfía entre gráficas tiene las siguientes propiedades:

1.- (Reflexividad): G es isomorfa a G para toda G.

Demostración: La función identidad $I : V(G) \rightarrow V(G)$ es un isomorfismo de G en G.

2.- (Simetría): Si G es isomorfa a H entonces H es isomorfa a G.

Demostración: Si ϕ: $V(G) \rightarrow V(H)$ es un isomorfismo de G en H, entonces ϕ^{-1}: $V(H) \rightarrow V(G)$ es un isomorfismo de H en G.

3.- (Transitividad): Si G es isomorfa a H y H es isomorfa a K, entonces G es isomorfa a K.

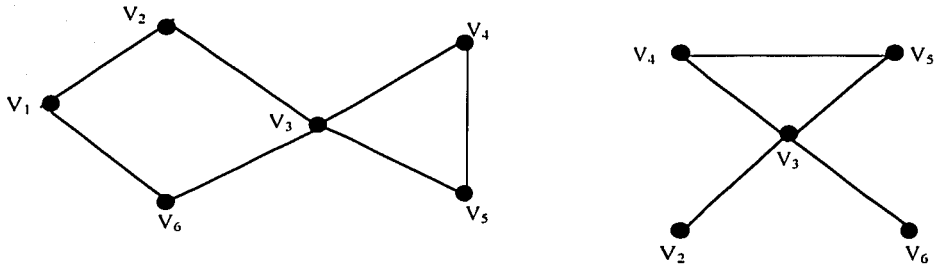
Demostración: Si ϕ: $V(G) \rightarrow V(H)$ es un isomorfismo de G en H, y ψ:$V(H) \rightarrow V(K)$ es un isomorfismo de H en K, entonces la composición $\psi \cup \phi$:$V(G) \rightarrow V(K)$ es un isomorfismo de G en K.

Cualquier relación que sea reflexiva, simétrica y transitiva se llama **relación de equivalencia**. Por lo anterior, la relación de isomorfía entre gráficas es una relación de equivalencia.

Definición. 7 Se dice que la gráfica H está contenida en la gráfica G si $V(H)$ es un subconjunto de $V(G)$ y $A(H)$ es un subconjunto de $A(G)$. En tal caso, se dice que la gráfica H es una **subgráfica** de G.

Si una gráfica F es isomorfa a una subgráfica H de G, entonces también se dice que F es una subgráfica de G.

En la Fig.I.6 se muestran las gráficas G y H; la gráfica H es subgráfica de G.



Gráfica G Gráfica H

Figura I.6. H subgráfica de G.

Definición. 8 Dada una gráfica G, se llama **complemento de G,** denotada $\bar{G}$, a la gráfica que tiene el mismo conjunto de vértices que G, con la propiedad de que cualesquiera dos vértices de $\bar{G}$ son adyacentes si y sólo si los mismos dos vértices no son adyacentes en G.

En la Fig.I.7 se ilustra el concepto de gráfica complemento.

Gráfica G Gráfica $\bar{G}$

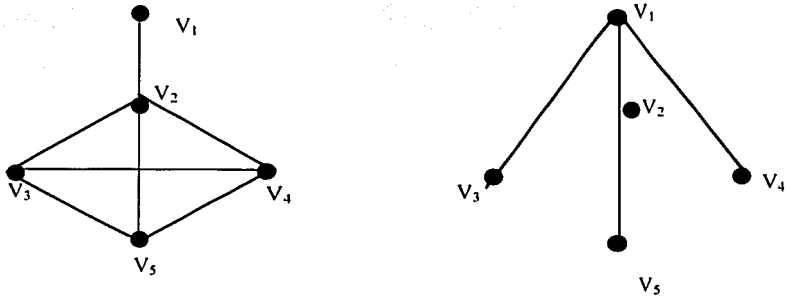


Figura I.7. Gráfica G y su gráfica complemento.

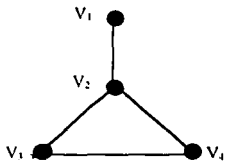
Un resultado inmediato es que si "unimos" las gráficas G y $\bar{G}$ de orden p, obtenemos la gráfica completa K_p.

Como hemos visto, una gráfica cualquiera está determinada por su conjunto de vértices y el conocimiento de qué pares de aristas son adyacentes. Esta misma información puede ser representada en una matriz. Esto tiene la ventaja de que podemos representar una gráfica en la computadora por medio de una matriz y trabajar con ella con una gran variedad de cálculos.

Definición 9 Sea G una gráfica de orden p cuyos vértices son $v_1, v_2, \ldots, v_p$. La **matriz de adyacencia** $A = A(G) = [a_{ij}]$ es una matriz pxp en donde la entrada $a_{ij} = 1$ si los vértices v_i y v_j son adyacentes, y $a_{ij} = 0$ si no lo son.

Esto es, la matriz A es una matriz compuesta de 0 y 1 donde la diagonal principal de A consiste únicamente de 0, ($a_{ii} = 0$ para i = 1, 2, ..., p), además A es una matriz simétrica, esto es $a_{ij} = a_{ji}$ pues decir que v_i y v_j son adyacentes es lo mismo que decir que v_j y v_i son adyacentes

En la Fig. I.8 se muestra una gráfica G y su matriz de adyacencia.



$A =$

0	1	0	0
1	0	1	1
0	1	0	1
0	1	1	0

Figura I.8. Gráfica G y su matriz de adyacencia

En algunos problemas donde se utiliza la teoría de gráficas es conveniente colorear los vértices, las aristas o ambos. En este trabajo utilizaremos la coloración de aristas.

Definición 10.-Una **coloración de aristas** en una gráfica G es la asignación de un color a cada una de las aristas de G, o más formalmente, una función $\psi : A(G) \rightarrow C$ donde C es un conjunto de colores $\{1, 2, \ldots, k\}$. Una **gráfica coloreada** es una gráfica G junto con una coloración de aristas.

En la Fig. I.9 se muestra una gráfica coloreada G

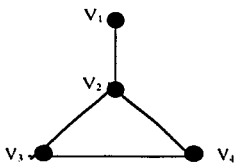


Figura I.9. Gráfica coloreada G.

Definición 11.- Dos gráficas coloreadas G_1 y G_2 son **equivalentes** si existe un isomorfismo entre ellas que respete los colores, esto es, una función $\omega: V(G_1) \rightarrow V(G_2)$ biunívoca de $V(G_1)$ en $V(G_2)$ tal que para cualesquiera vértices u_1, v_1 en G_1, éstos son adyacentes si y sólo si los vértices $\omega(u_1)$ y $\omega(v_1)$ son adyacentes en G_2, y además la arista $[v_1,v_2]$ es de color c en G_1 si y sólo si, la arista $[\omega(v_1), \omega(v_2)]$ es del mismo color c en G_2. Un isomorfismo que respete los colores lo llamaremos **isomorfismo cromático**.
En la Fig. I.10 se muestran dos gráficas coloreadas isomorfas G_1 y G_2.

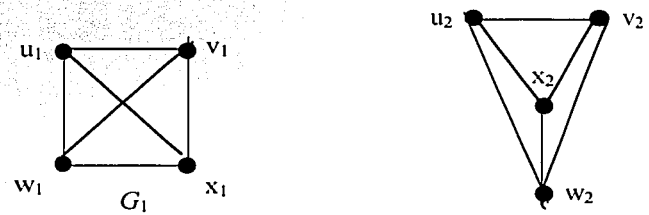


Figura I.10. Gráficas coloreadas isomorfas.

La relación de "ser equivalentes" que acabamos de definir entre gráficas coloreadas es en efecto una relación de equivalencia. Además de las tres propiedades de la isomorfía que probamos después de la Def. 6 sólo necesitamos observar que el isomorfismo identidad en una gráfica coloreada respeta colores, que si un isomorfismo respeta colores su inverso también es cromático, y que la composición de dos isomorfismos cromáticos también lo es.

Por ser de utilidad en nuestros desarrollos posteriores, recordaremos aquí algunos conceptos y propiedades generales acerca de las relaciones de equivalencia.

Si ~ es una relación de equivalencia en un conjunto A y a es un elemento de A, la **clase**

de equivalencia de a es el conjunto $[a]$ de todos los elementos de A que están relacionados con a, en símbolos $[a] = \{ b \in A : a \sim b \}$.

Gracias a la propiedad reflexiva, $a \in [a]$ para todo $a \in A$, así que las clases de equivalencia son todas no vacías y la unión de todas ellas es A.

Si $a \sim b$, entonces $[a] = [b]$. En efecto, si $a \sim b$ entonces por transitividad y simetría los elementos relacionados con a son los mismos que los relacionados con b.
Si a no está relacionado con b, entonces $[a] \cap [b] = \varnothing$. En efecto, si tuviéramos un elemento x en la intersección de $[a]$ con $[b]$, sería $x \sim a$ y $x \sim b$ y entonces $a \sim b$, contrario a la hipótesis. Por las dos propiedades anteriores, esta propiedad nos da que $a \sim b$ si y sólo si $[a] = [b]$.

Resumiendo lo anterior, podemos decir que la colección de todas las clases de equivalencia $P = \{[a] : a \in A\}$ es una **partición** de A.

En el caso en el que A es el conjunto de todas las gráficas coloreadas con vértices 1, 2, ..., n, veremos ahora que la clase de equivalencia de una gráfica coloreada G puede describirse muy fácilmente en términos de G y del conjunto de todas las permutaciones de los números 1, 2, ..., n. Dada una permutación ϕ de estos números, la **permutada** de G mediante ϕ es la gráfica coloreada G_ϕ con vértices 1, 2, ..., n y una arista $[\phi(i), \phi(j)]$ (de color c) por cada arista $[i, j]$ (de color c) de G. Claramente G_ϕ es equivalente a G, pues la permutación ϕ es un isomorfismo cromático de G en G_ϕ. Recíprocamente, si H es cualquier gráfica coloreada equivalente a G, por definición existe un isomorfismo cromático ϕ de G en H, y es claro que H coincide con la permutada G_ϕ para esta ϕ. En suma, la clase de equivalencia $[G]$ de una gráfica coloreada G es el conjunto de todas las permutadas de G. Aplicando lo visto más arriba, tenemos que dos gráficas coloreadas G y H son equivalentes si y sólo si sus conjuntos de permutadas $[G]$ y $[H]$ son iguales, y que G y H no son equivalentes si y sólo si sus conjuntos de permutadas $[G]$ y $[H]$ son ajenos.

I.3. Introducción a los números de Ramsey

Recordemos que el juego de Sim se desarrolla entre dos jugadores que tiran alternadamente. El juego consiste de seis puntos. Cada jugador debe dibujar una línea de un punto a otro; el primer jugador dibuja líneas rojas y el segundo azules. El jugador que primero trace un triángulo de su propio color, pierde.

Una pregunta importante que nos hacemos sobre el juego es: ¿siempre hay un ganador y un perdedor? Es decir, ¿necesariamente alguno de los jugadores traza un triángulo de su propio color?

En términos de teoría de gráficas, la pregunta anterior se puede formular de la siguiente manera: ¿qué condiciones se requieren para asegurar que en una gráfica completa de orden seis, donde los vértices se unen utilizando aristas de dos colores, se forme un triángulo de un solo color?.

Para dar una respuesta a esta pregunta, utilizaremos el concepto de número de Ramsey, el cual lo vamos a introducir a través de un problema que ilustra el significado de dicho concepto.

Problema de los Anfitriones Excéntricos.

Un señor y su esposa están planeando realizar una cena, pero cada uno de ellos tiene cierta peculiaridad. Al señor le gusta que todos los invitados que estén sentados en una mesa se conozcan entre sí, porque cree que eso crea armonía durante la cena. Por otra parte, su esposa piensa que no debe haber conocidos mutuos sentados en una misma mesa, ya que considera que un aspecto importante de la cena es el tener la oportunidad de hacer nuevas amistades. Para salvar el problema deciden que dos de las mesas que habrá en la cena, denominadas mesa A y mesa B tengan las siguientes características: en la mesa A, la del esposo, se sentarán únicamente invitados que se conocen todos entre sí, mientras que en la mesa B no habrá dos personas que se conozcan mutuamente.

Supongamos que la mesa A es para m personas y la B es para n. ¿Cuál es el número mínimo de personas que deben asistir a la cena, para asegurar que al menos una de las dos mesas se llene?.

Para empezar a analizar el problema, notemos que en el caso más sencillo en el que las dos

mesas son para igual número de personas. Con $m = n = 2$, basta con invitar a dos personas para asegurar que al menos una de las dos mesas se llene; ya que las únicas dos posibilidades que existen es que estas dos personas se conozcan o no se conozcan entre sí; esto es, se llena la mesa A, o se llena la mesa B.

Abordemos el caso en que $m = n = 3$ ¿Cuál es el número mínimo de invitados que deben de asistir a la cena, de tal forma que garanticemos que al menos una mesa quede llena?.

Obviamente, si invitamos a una o a dos personas, independientemente de si se conozcan o no, no se llena ninguna de las mesas. Por tanto, el número mínimo de invitados es estrictamente mayor que dos.

Supongamos que el número de invitados es tres. Dado que existe la posibilidad de que dos de los invitados se conozcan entre si y el tercero no conozca a ninguno de los anteriores, o conozca sólo a uno de ellos, no se asegura, por tanto, que invitando a tres personas se llene al menos una de las mesas.

Supongamos ahora que el número de invitados es cuatro. Dado que existe la posibilidad, por ejemplo, que dos de los invitados se conozcan entre sí y los otros dos también se conozcan entre sí, pero ninguno de los dos primeros conoce a ninguno de los dos segundos, no se asegura que invitando a cuatro personas se llene al menos una de las mesas.

Supongamos que el número de invitados es cinco. Existe la posibilidad de que cada individuo conozca sólo a otros dos, digamos el primero al segundo, el segundo al tercero, el tercero al cuarto, el cuarto al quinto y el quinto al primero. Como los dos invitados que cada uno conoce no se conocen entre sí, no hay tres que se conozcan mutuamente, no se llena en este caso la mesa A. La mesa B tampoco se llena ya que los dos invitados que cada uno desconoce se conocen entre sí y por tanto tampoco hay tres que se desconozcan mutuamente.

Supongamos ahora que el número de invitados es 6. Este caso es un poco más complicado y para abordarlo haremos uso de la teoría de gráficas. Así, si el problema de los Anfitriones Excéntricos lo representamos mediante una gráfica G de orden p, en la que los vértices representan a las personas invitadas (p personas invitadas) y dos vértices son adyacentes si las personas correspondientes se conocen entre sí; entonces, el caso de seis invitados puede

representarse con una gráfica de orden 6.

En este caso de 6 invitados el número de situaciones posibles se multiplica, veamos algunas de ellas. A fin de ilustrar cómo la teoría de gráficas nos puede ayudar a analizar este problema empezaremos tomando la situación en la que de los seis invitados, tres se conocen mutuamente, por lo que cumplen las condiciones para sentarse en la mesa A. Esta situación la podemos expresar por medio de la gráfica G de la Fig. I.11.

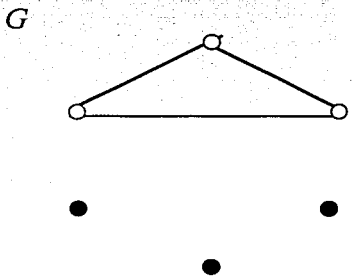


Figura I.11.Seis invitados a la cena y tres de ellos se conocen mutuamente.

De la gráfica G podemos decir lo siguiente:

1.- El orden de G es seis, pues son seis los invitados.

2.- El triángulo formado por los tres vértices blancos y las aristas rojas es una subgráfica de G, que corresponde a los tres invitados que se conocen entre sí; tal gráfica la llamaremos F_1. Observemos que F_1 es una gráfica completa de orden tres (K_3), ya que los tres vértices que la forman están unidos entre sí.

Supongamos ahora que son cuatro los invitados que se conocen entre sí; esta situación puede ser representada con la gráfica de la Fig. I.12.

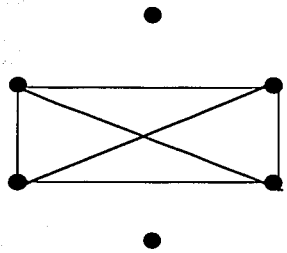


Figura I.12.Seis invitados a la cena y cuatro se conocen entre sí.

Esta gráfica contiene varias subgráficas; una de ellas, la formada por las aristas azules, representa tres invitados que se conocen mutuamente, por lo que cumplen las condiciones para sentarse en la mesa A.

Otra posibilidad es que de los seis invitados a la cena, no haya tres de ellos que se conozcan mutuamente; esto es, no hay tres invitados que puedan sentarse en la mesa A. Aquí hay muchas situaciones posibles; por ejemplo, puede suceder que cada persona conozca sólo a dos más. La gráfica G de la Fig. I.13 representa una de estas situaciones.

G

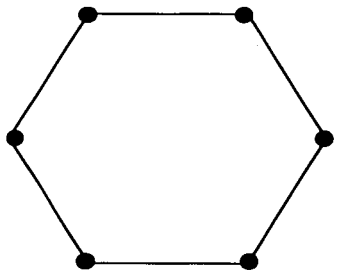


Figura I.13. Seis invitados a la cena y cada uno conoce sólo a dos más.

Consideremos ahora el complemento de la gráfica G, denotada por $\overline{G}$, (Fig. I.14).

$\overline{G}$

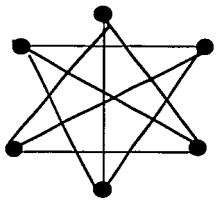


Figura I.14. Invitados que no se conocen.

En esta gráfica, por ser el complemento de G, dos vértices unidos representan la situación en la que las dos personas no se conocen.

La gráfica $\bar{G}$ contiene varias subgráficas; una de ellas, el triángulo formado por las aristas azules, representa a tres personas que no se conocen entre sí, por lo que cumplen las condiciones para sentarse en la mesa B. A esta subgráfica formada por las aristas azules la llamaremos F_2.

Otra situación posible es que de los seis invitados, dos personas que no se conozcan entre sí conozcan cada una sólo a otras dos personas, las cuales a su vez no se conozcan entre sí. Esta situación queda representada por la gráfica G de la Fig. I.15.

G

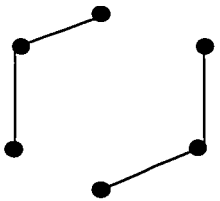


Figura I.15.Seis invitados a la cena donde dos personas conocen sólo a otras dos.

Consideremos de nuevo la gráfica complemento de G, ilustrada en la Fig. I.16.

$\overline{G}$

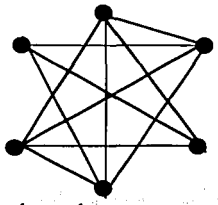


Figura I.16. Invitados a la cena que no se conocen.

La gráfica $\bar{G}$, al igual que en los casos anteriores, contiene varias subgráficas; una de ellas es el triángulo formado por las aristas rojas que representa a tres invitados que no se conocen mutuamente. Por lo tanto, existen tres invitados que cumplen las condiciones para sentarse en la mesa B.

Podemos observar que en todos los casos mencionados existe una subgráfica, de G o de $\bar{G}$, completa de orden tres; esto es, se forma un triángulo ya sea en G o en $\bar{G}$.

Así, el problema de encontrar el número mínimo de invitados, de tal forma que al menos una de las dos mesas, con tres lugares cada una, se llene, se resuelve encontrando el menor número p (número de invitados) tal que para cualquier gráfica G de orden p, se cumple al menos una de las dos condiciones siguientes:

1) G contiene una subgráfica completa de orden tres (K_3),

o

2) $\bar{G}$ contiene una subgráfica completa de orden tres (K_3).

Al número p se le conoce como **número de Ramsey** para (K_3, K_3). Como ya vimos, para los casos de menos de 6 invitados no se asegura que alguna de estas condiciones se cumpla, así que ya sabemos que el número de Ramsey para (K_3, K_3) tiene que ser mayor que cinco.

En el caso analizado anteriormente tanto la mesa A como la B fueron para tres invitados; en general, en el Problema de los Anfitriones Excéntricos, la mesa A tiene m lugares y la mesa B tiene n lugares.

Podemos definir de manera general, el número de Ramsey de la siguiente manera:

Definición. 12 Sean F_1 y F_2 dos gráficas cualesquiera. Definimos el **número de Ramsey** de estas dos gráficas, denotado por $r(F_1, F_2)$, como el entero menor p tal que para cada gráfica G de orden p sucede al menos alguna de las siguientes dos condiciones:

1) G contiene a F_1 como una subgráfica,

o

2) $\bar{G}$ contiene a F_2 como una subgráfica.

Así, el problema de los Anfitriones Excéntricos, en el caso general de que la mesa A sea para m personas y la mesa B para n personas, es equivalente a encontrar el número de Ramsey para las gráficas completas K_m y K_n.

I.4.- Propiedades de los números de Ramsey.

Actualmente los matemáticos trabajan en encontrar números de Ramsey. Hasta ahora, para el caso en que $m \geq 3$ y $n \geq 3$ se conocen muy pocos de estos números; en algunos casos se conoce sólo el intervalo en el que se encuentra el número. Así, actualmente el problema de los Anfitriones Excéntricos continúa siendo un problema abierto, ya que no está resuelto para el caso general.

En la Tabla 1 se muestran los números de Ramsey conocidos.

	m							
n	3	4	5	6	7	8	9	10
3	6	9	14	18	23	28	36	40 a 43
4	9	18	25	35 a 41				
5			49	58 a 87				
6				102 a 165				
7					205 a 540			

Tabla 1. Números de Ramsey.

Una propiedad sencilla, pero importante, de los números de Ramsey es que su valor no depende del orden en que se tomen las gráficas F_1 y F_2. Esta propiedad nos lleva a formular el siguiente teorema.

Teorema 1. Si F_1 y F_2 son dos gráficas cualesquiera, entonces

$$r(F_1, F_2) = r(F_2, F_1).$$

Demostración:

Sean $r(F_1, F_2) = p$ y $r(F_2, F_1) = p_0$ vamos a demostrar que $p = p_0$.

Sea H una gráfica cualquiera de orden p.

Como $r(F_1, F_2) = p$ y $\overline{H}$ es una gráfica de orden p, entonces sucede alguna de las dos siguientes cosas,

1) $\overline{H}$ contiene a F_1 como subgráfica,

o

2) $\overline{\overline{H}} = H$ contiene a F_2 como subgráfica.

Esto es, para cada gráfica H de orden p, aseguramos que

1) H contiene a F_2 como subgráfica,

o

2) $\overline{H}$ contiene a F_1 como subgráfica.

Dado que por hipótesis $r(F_2, F_1) = p_0$, por la minimalidad de los números de Ramsey se sigue que $p \geq p_0$.

Sea ahora G una gráfica cualquiera de orden p_0. De manera análoga, la gráfica $\overline{G}$ contiene a F_2 como una subgráfica o $\overline{G}$ contiene a F_1 como una subgráfica.

Como $r(F_2, F_1) = p$, podemos asegurar que $p_0 \geq p$.

Considerando las desigualdades encontradas, concluimos que $p = p_0$.

Hasta ahora sólo hemos definido los números de Ramsey, no hemos dicho todavía cómo encontrarlos. A continuación expondremos una manera alternativa de definir los números de Ramsey, considerando aristas coloreadas; esta definición nos ayudará a encontrar algunos números de Ramsey para algunos casos particulares.

Tomemos de nuevo el ejemplo de los Anfitriones Excéntricos, para el caso en que la mesa A y la mesa B tengan tres lugares cada una y sean seis los invitados a la cena.

Uno de los casos que ya analizamos fue la situación representada por la gráfica G de la Fig.I.17; en esta misma figura ilustramos la gráfica $\bar{G}$.

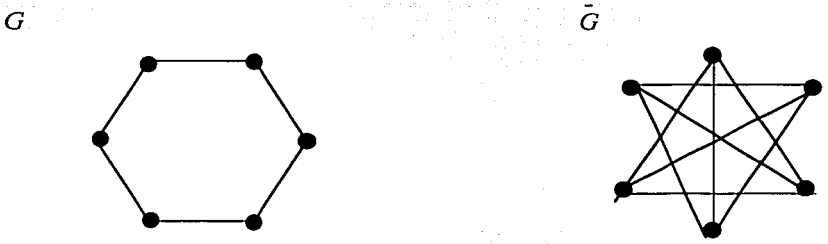


Figura I.17. Gráfica G y su gráfica complemento.

El conjunto de aristas de la gráfica completa de orden seis, K_6, es la unión ajena de los conjuntos de aristas A(G) y A($\bar{G}$). En otras palabras, cada una de las aristas de K_6, pertenece a A(G) o a A($\bar{G}$) pero no a ambas. Supongamos que cada arista de K_6 que pertenece a A(G) la pintamos de rojo y cada arista de A($\bar{G}$) la pintamos de azul (Fig. I.18).

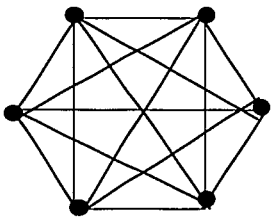


Figura I.18. Gráfica G y su gráfica complemento.

Así, el problema de encontrar el número mínimo de invitados, de tal forma que alguna de las dos mesas con tres lugares cada una (la A o la B) se llene, se transforma en el problema de encontrar el menor número p tal que para cada coloración de las aristas de K_p con los colores rojo y azul se tiene que K_p contiene como subgráfica una K_3 roja o una K_3 azul.

En general, una definición equivalente de los números de Ramsey es la siguiente:

Definición. 13 Sean F_1 y F_2 dos gráficas cualesquiera. Definimos el **número de Ramsey** de estas dos gráficas, denotado por $r(F_1, F_2)$, como el menor entero p tal que si cada arista de K_p es coloreada de rojo o azul, entonces o existe una subgráfica F_1 de K_p roja o una subgráfica F_2 de K_p azul.

Encontrando números de Ramsey.

Sean F_1 y F_2 dos gráficas. Es fácil ver que si al menos una de ellas es de orden 1, entonces $r(F_1, F_2) = 1$, ya que al menos una de ellas está formada por sólo un vértice y entonces K_1 la contiene como subgráfica.

Teorema 2 Sean K_2 y K_3, dos gráficas completas de orden dos y tres, respectivamente. Aseguramos que $r(K_3, K_2) = 3$.

Demostración:

Primero vamos a comprobar que $r(K_3, K_2) > 2$.

La gráfica K_2 está formada por dos vértices y una arista; si a la arista la coloreamos de rojo, podemos ver que esta gráfica no contiene una subgráfica K_3 roja, o una K_2 azul. Por lo que no es cierto que para cualquier gráfica completa de orden 2 que coloreemos ya sea de rojo o de azul, exista una K_3 roja, o una K_2 azul. Por tanto, $r(K_3, K_2) > 2$.

Analicemos ahora el caso de $p = 3$. Si todas las aristas de K_3 son azules, entonces contiene una subgráfica K_2 azul; si una arista de K_3 es coloreada de rojo, existe una subgráfica K_2 azul; lo mismo sucede si dos aristas de K_3 son coloreadas de rojo; por último, si todas las aristas son coloreadas de rojo existe una K_3, roja. Por lo tanto $r(K_3, K_2) = 3$ (ver Fig.I.19).

Figura I.19. Gráficas K_3.

Teorema 3. Sean K_3, una gráfica completa de orden tres. Aseguramos que $r(K_3, K_3) = 6$.

Demostración:

Primero vamos a demostrar que $r(K_3, K_3) > 5$.

Supongamos la gráfica G de orden 5 y $\bar{G}$ su complemento de la Fig. I.20.

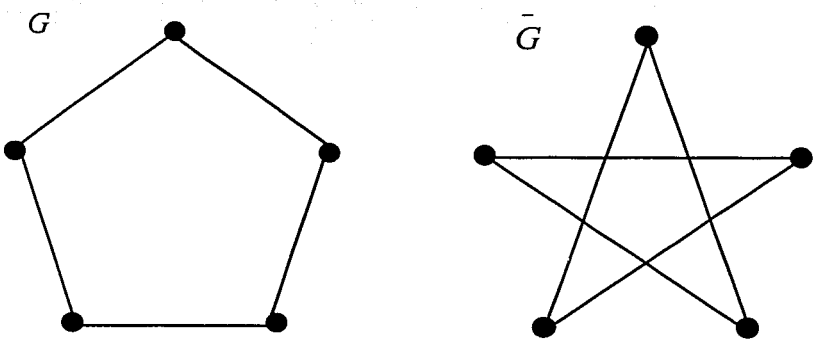


Figura I.20. Gráfica G y su gráfica complemento.

Como podemos ver en la Fig. I.20, la gráfica G no contiene una subgráfica K_3 roja y $\bar{G}$ no contiene una subgráfica azul K_3, por lo que $r(K_3, K_3) > 5$.

Para demostrar que $r(K_3, K_3) = 6$, es suficiente con demostrar que si las aristas de K_6 son coloreadas de rojo, entonces existe una K_3 roja, o una K_3 azul.

Sea v_1 un vértice cualquiera de K_6. Como en v_1 inciden 5 aristas, al menos 3 son del mismo color, podemos suponer que son rojas sin pérdida de generalidad. Supongamos que dichas aristas son $[v_1, v_2]$, $[v_1, v_3]$, y $[v_1, v_4]$.

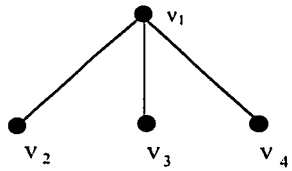


Figura I.21. Vértice v_1 de K_6.

Si cualquiera de las aristas $[v_2, v_3]$, $[v_3, v_4]$, $[v_2, v_4]$ es roja, tendríamos una K_3 roja. Si estas tres aristas son azules, forman una K_3 azul. Por lo tanto, $r(K_3, K_3) = 6$.

Con esto hemos demostrado que seis es el menor número de vértices tal que cualquier bicoloración de la gráfica completa contiene un triángulo de un mismo color (rojo o azul).

Con el teorema que acabamos de demostrar podemos, ahora sí, dar respuesta a la pregunta planteada al inicio del capítulo, ¿podemos asegurar que siempre hay un ganador y un perdedor en el juego de Sim? La respuesta a esta pregunta es afirmativa; siempre, independientemente de cómo juegue cada uno de los jugadores, se forma necesariamente un triángulo rojo o uno azul; ya que en una gráfica completa de orden seis, si coloreamos las aristas de rojo o azul siempre se forma un triángulo con alguno de estos colores. En conclusión, en el juego de Sim no puede haber empate.

CAPÍTULO II

Inteligencia artificial.

En el capítulo anterior planteamos el juego de Sim y demostramos que en este juego siempre hay un ganador. El propósito de este capítulo es desarrollar una forma de representar este juego a través de una gráfica dirigida, que se llamará *la gráfica del juego de Sim*. Para ello utilizaremos algunos conceptos de inteligencia artificial y de teoría de gráficas.

II.1 Conceptos Básicos de la Inteligencia Artificial y Planteamiento del Problema.

En inteligencia artificial, el primer paso para diseñar un programa que resuelva un problema específico, es crear una descripción formal y manejable del propio problema. Para lograr esta descripción debemos hacer lo siguiente:

1.-Definir un espacio de estados que contenga todas las configuraciones posibles.
2.- Identificar uno o más estados que describan situaciones en los que comience el proceso de resolución del problema; llamados **estados iniciales.**
3.- Especificar uno o más estados que pudieran ser soluciones aceptables del problema; denominados **estados objetivo.**
4.- Especificar un conjunto de reglas que describen las acciones disponibles para pasar de un estado a otro; llamadas **operadores.**

De acuerdo con los conceptos anteriores, el juego de Sim puede entonces quedar formalmente descrito de la manera siguiente:
Estado: Cualquier configuración que contenga de 0 a 15 aristas con extremos en dos de los seis vértices originales de la gráfica. Las aristas deben estar coloreadas con (a lo más) los colores rojo y azul, el número de aristas rojas debe ser igual al número de aristas azules o a éste más uno.

Estado inicial: Es la configuración en la que únicamente hay seis vértices y no ha sido trazada ninguna arista por alguno de los jugadores. En este juego, el estado inicial es único.

Estado objetivo: Cualquier configuración del juego en la que exista un triángulo azul o rojo, cuyos vértices sean tres de los seis vértices originales de la gráfica.

Operadores: Trazar una arista roja (si el estado tiene un número par de aristas) o azul (en otro caso) de un vértice a otro, siempre y cuando estos vértices no hayan sido previamente unidos.

Así, el juego de Sim queda planteado como un problema de trazar aristas a través de un **espacio de estados**, en el que cada estado corresponde a una configuración del juego. Por tanto, el juego de Sim empieza a partir del estado inicial y, mediante el uso de un conjunto de reglas que permiten pasar de un estado a otro, termina en alguno de los estados objetivos.

El problema consiste entonces, para cada jugador, en encontrar una ruta a través del espacio de estados que una el estado inicial con alguno de los estados objetivos en el que el jugador gane.

Hasta aquí hemos logrado tener una representación formal del juego de Sim. Pasaremos ahora a la búsqueda de soluciones; esto es, encontrar las rutas en el espacio de estados que conecten el estado inicial con alguno de los estados objetivo.

II.2 Generación de estados

En el juego de Sim la generación de estados empieza necesariamente a partir del estado inicial, representado gráficamente en la Fig.II.1.

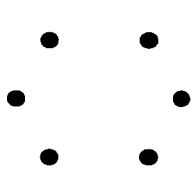
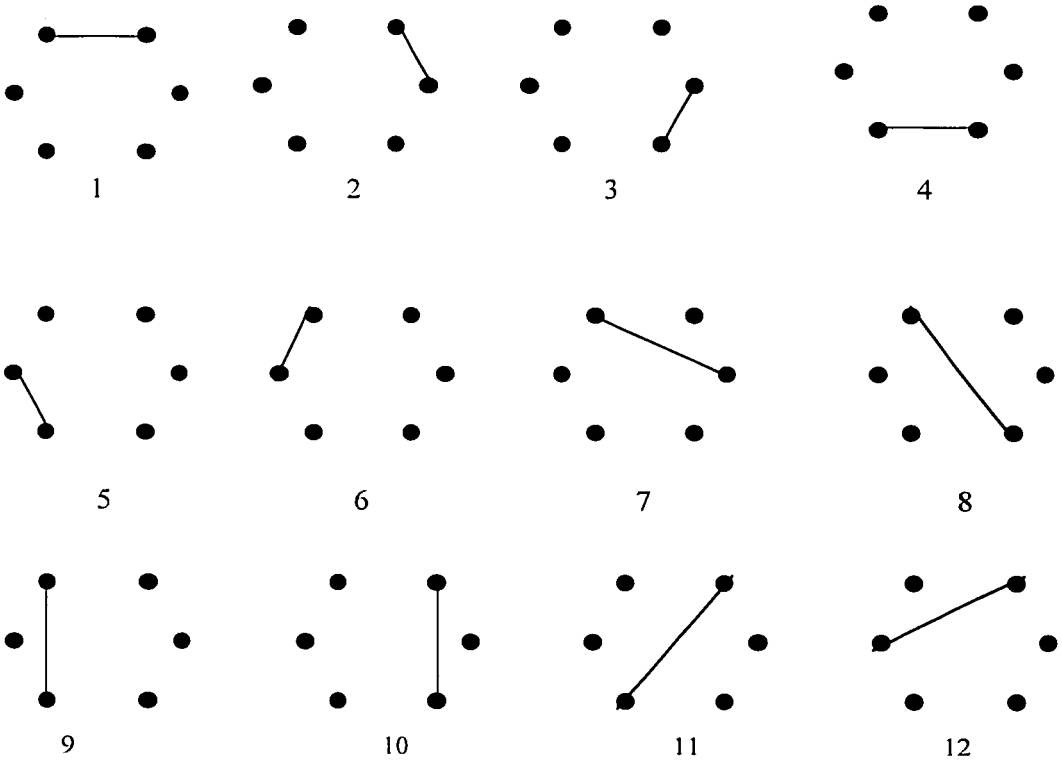
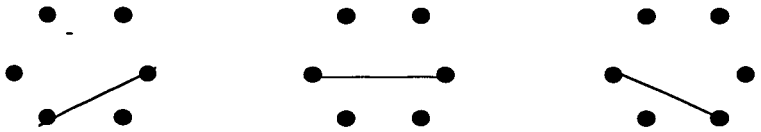


Figura II.1. Estado inicial.

Puesto que el estado inicial no es un estado objetivo, es necesario generar otros estados, utilizando los operadores. Mediante este proceso se generan 15 nuevos estados, los cuales están representados en la Fig.II.2. Al proceso de generación de nuevos estados a partir de un estado dado le llamaremos **expansión** del estado dado.





13 14 15

Figura II.2. Espacios generados a partir del estado inicial.

Puesto que al expandir el estado inicial se generan 15 estados, el siguiente paso consiste en elegir cuál es el estado que nos conviene seguir expandiendo.

La búsqueda consiste justamente en escoger sólo un estado, haciendo a un lado los demás; los cuales serán considerados posteriormente en caso de no llegar a un estado objetivo a través de la primera opción. Supongamos que en el siguiente nivel de expansión escogemos el primer estado descrito en la Fig.II.2. Verificamos si este estado seleccionado es un estado objetivo, que no lo es; en tal situación procedemos a expandir dicho estado aplicando los operadores; obteniendo así 14 nuevos estados.

Para continuar con el proceso de expansión hay que decidir ahora entre expandir uno de estos nuevos 14 estados o regresar y elegir otro de los primeros 15 estados.

La elección del estado que se desea expandir se decide en función de una **estrategia de búsqueda.**

Para facilitar la comprensión del proceso de búsqueda, conviene concebirlo como si fuera la construcción de un árbol de búsqueda sobrepuesto en el espacio de estados.

Antes de continuar describiendo este proceso, pasaremos a establecer algunas definiciones que nos serán de gran utilidad.

Definición. 14 Un **árbol de búsqueda** es una gráfica que empieza en un **nodo raíz** que representa un estado inicial, cada nodo subsiguiente representa un nuevo estado generado y cada arista un operador.

Definición. 15 Los **nodos hoja** son los nodos que no tienen ningún sucesor en el árbol, ya sea porque todavía no se han expandido o porque ya lo fueron pero generan un conjunto vacío de sucesores.

Definición. 16. Al nodo que se expande se le denomina **nodo padre** y a los nodos generados se les llama **nodos hijos** o descendientes.

Todas las estrategias de búsqueda pueden concebirse como un recorrido sobre una estructura en forma de árbol en el que cada nodo representa un estado del problema y cada arista de este árbol representa una operación entre estados sucesivos. El proceso de búsqueda consiste en encontrar uno o más caminos que conectan al estado inicial (nodo raíz) con uno o más estados objetivos.
Existen, en general, diversas estrategias de búsqueda; su diferencia radica en la forma que tiene cada una de ellas de expandir el árbol. Por ejemplo, algunas veces es conveniente expandir, a partir del estado inicial, todos los nodos del siguiente nivel, y así sucesivamente hasta llegar a un estado objetivo. En otras ocasiones es mejor expandir una sola rama del árbol, hasta que se decide terminar la búsqueda por esa dirección; a continuación, se regresa a un nodo creado con anterioridad y se prosigue con el proceso de búsqueda. Éstas son sólo dos formas de estrategia de búsqueda, pero obviamente existen muchas más.

La elección de la estrategia de búsqueda depende de las características del problema. Por ejemplo, en el caso de un programa que juegue ajedrez, la construcción total del árbol es prácticamente imposible, ya que al expandir el estado inicial, existen alrededor de 30 jugadas. Por cada una de ellas existen otras 30 jugadas del adversario. Es decir la primera jugada de ambos produce $30^2 = 900$ diferentes jugadas posibles. La segunda jugada de ambos hará que surjan $30^2 \times 30^2$. Al final de un juego típico de 40 jugadas tendríamos que considerar aproximadamente 30^{80} ($> 1.47 \times 10^{118}$) diferentes jugadas posibles (para simplificar el cálculo, se está considerando que, en promedio, hay 30 posibles jugadas en cada momento del juego). El tiempo de máquina y consumo de memoria necesario para recorrer un árbol de este tamaño son enormes, por lo que en problemas como éste no es conveniente construir todo el árbol.

En el juego de Sim, el árbol de búsqueda resulta ser demasiado grande para hacerlo con lápiz y papel; sin embargo, es posible construirlo en la computadora. Si pensáramos en un juego análogo con un mayor número de vértices, el árbol sería demasiado grande, aún para hacerlo en una computadora; en tal situación, estaríamos obligados a utilizar una estrategia de búsqueda que no involucrara la construcción de todo el árbol.

En este trabajo usamos la estrategia de búsqueda denominada **Estrategia de Búsqueda a lo Ancho,** consistente en primero expandir el nodo raíz y luego todos los nodos generados por éste, y así sucesivamente. Esto es, todos los nodos que estén en el nivel *d* del árbol de búsqueda se expanden antes de que sean expandidos los nodos que están en el nivel *d+1*.

A continuación vamos a ilustrar la forma en que se construye el árbol de búsqueda del juego de Sim. Tomaremos, a manera de ejemplo, el caso del juego con sólo cuatro puntos. La construcción que se realiza para el juego de cuatro puntos, puede generalizarse a cualquier número de puntos. Conviene mencionar que el juego con seis puntos tiene 3,554,627,472,076 (tres billones quinientos cincuenta y cuatro mil seiscientos veintisiete millones cuatrocientos setenta y dos mil setenta y seis) estados posibles (si bien no todos pertenecen al árbol del juego, pues algunos son "descendientes" de estados objetivo, esto nos da idea de lo grande que es el árbol en el caso de 6 puntos).

II.3 Construcción del árbol búsqueda para el juego de Sim.

El juego con cuatro puntos que utilizaremos para ilustrar la construcción del árbol de búsqueda del juego de Sim consiste en lo siguiente: hay cuatro vértices de un cuadrado dibujados en una hoja de papel. Los jugadores tiran alternadamente y cada jugada consiste en trazar una arista con extremos en dos de los cuatro vértices del cuadrado. El primer jugador siempre pinta líneas rojas y el segundo azules. No se permite volver a unir dos puntos previamente unidos. Pierde el juego aquel jugador que dibuje un triángulo de su color.

El árbol de este juego tiene, como estado inicial, el que se muestra en la Fig. II.3, consistente de cuatro puntos y ninguna arista.

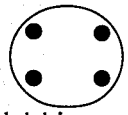


Figura II.3. Estado inicial del juego con cuatro puntos.

Para construir el primer nivel del árbol debemos expandir el nodo raíz; así, obtenemos 6 nuevos nodos, cada uno de los cuales es una primera posible jugada del primer jugador. Dibujamos cada nuevo nodo generado y lo unimos al nodo padre, que en este caso es el nodo raíz, con una arista de color del jugador en turno, en este caso roja (Fig. II.4).

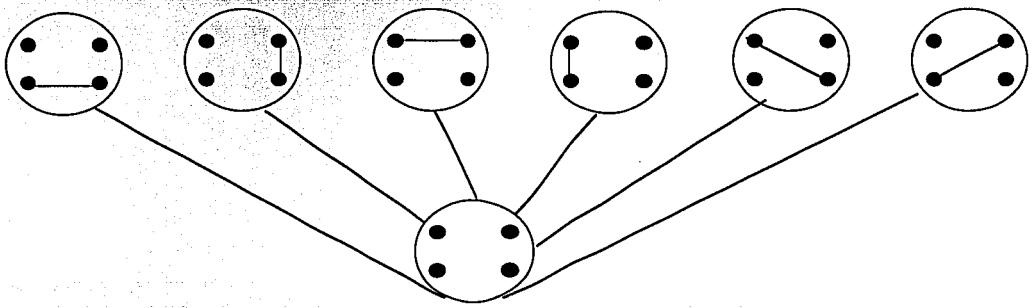


Figura II.4. Expansión del nodo raíz.

Para obtener el segundo nivel, es necesario expandir cada uno de los nodos del primer nivel; cada uno de estos nodos tiene cinco nodos descendientes; por lo tanto, el segundo nivel del árbol consiste de $6 \times 5 = 30$ nodos; cada uno de ellos corresponde a una posible jugada del segundo jugador. Dibujamos los 30 nodos y nuevamente los unimos a su respectivo nodo padre, esta vez con una arista de color azul.

Para obtener el tercer nivel, expandimos cada nodo del segundo nivel; por cada uno de éstos se generan 4 nuevos nodos; este nivel contiene $6 \times 5 \times 4 = 120$ nodos. Nuevamente dibujamos cada uno de ellos y los unimos, en este caso, con una arista roja al nodo padre correspondiente.

Continuamos repitiendo este procedimiento hasta que ya no sea posible dibujar una arista que una a dos vértices del cuadrado; esto es, hasta que los cuatro puntos estén unidos todos entre sí.

En el juego de 4 vértices hay 1957 estados posibles, de ellos hay 6! = 720 estados en el último nivel (con las seis aristas trazadas). Recordemos (Teorema 3) que en el juego con seis vértices siempre hay un ganador y un perdedor; no hay posibilidad de empate. En el caso de 4 vértices sí existe la posibilidad de empate.

II.4 Algoritmo para expandir un estado.

Para construir el algoritmo que expanda cualquier nodo del árbol, necesitamos representar primero los nodos en la computadora.

Cada nodo del árbol del juego lo representaremos a través de un vector (cuya dimensión es igual al número de aristas $\binom{n}{2}$ del juego) al que llamaremos *lineam*, de la siguiente forma.

En nuestro ejemplo de 4 puntos, tomamos la gráfica completa de orden cuatro y a cada una de las aristas le asignamos un número fijo que servirá de identificador, como se muestra en la Fig. II.5.

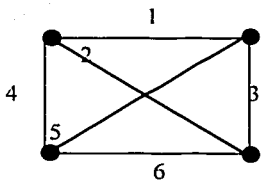


Figura II.5. Numeración de aristas.

Cada una de las entradas del vector *lineam* corresponderá a cada una de las aristas.
Así, si la arista 1 fue trazada por el primer jugador, colocamos un 1 en la primera enterada

del vector; si la arista 1 la trazó el segundo jugador, colocamos un 2 en la primera entrada; en caso de que la arista 1 no haya sido trazada aún, colocamos un 0 en este lugar.

Esto lo hacemos para cada una de las aristas del estado que vamos a representar en la computadora. En general, si la arista i-ésima no ha sido trazada, fue trazada por el primer jugador o fue trazada por el segundo jugador, la entrada i-ésima del vector tendrá un 0, un 1 o un 2 según sea el caso.

Ejemplo:
Supongamos el estado ilustrado en la Fig. II.6.

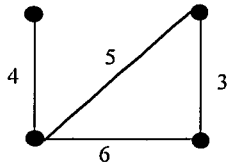


Figura II.6. Estado del juego de cuatro puntos.

El vector *lineam* que representa el estado de la Fig. II.6 es la siguiente:

0	0	1	2	1	2
1	2	3	4	5	6

Una vez resuelto el problema de representar mediante un vector un estado arbitrario del juego en la computadora, pasaremos a desarrollar el algoritmo para expandir los estados subsiguientes.

Para ilustrar el algoritmo, utilizaremos el estado ilustrado en la Fig.II.6 como el estado a ser expandido. Este algoritmo lo llamaremos *generahijas*.

El algoritmo *generahijas* lee el vector *lineam*, el cual representa el estado que deseamos expandir. Utilizamos una variable auxiliar, llamada *nj*, la cual representa el número del jugador en turno; si $nj = 1$, el turno es del primer jugador; si $nj = 2$ el turno es del segundo

jugador. Este algoritmo produce como salida a la matriz *lineah*; en la cual están representados todos los nodos generados a partir del nodo padre. En el caso del juego de 4 puntos, la matriz *lineah* contendrá 6 columnas; cada renglón describe uno de los estados generados. El número de renglones utilizados de la matriz será igual al total de estados generados. Utilizaremos la variable auxiliar *ndeh* para numerar los estados generados; dicha variable estará inicializada en cero.

Este algoritmo consiste en las tres etapas siguientes:

En la primera etapa se identifican las aristas que aún no han sido trazadas por ningún jugador en el estado que va a ser expandido; esto es, se localizan los ceros que existen en el vector *lineam*. Utilizaremos la variable i, con i = 1,...,6, para identificar la aristas. Obviamente, en caso de que no exista ningún cero en el vector *lineam*, significa que el estado ya no puede ser expandido.

Para localizar la primera arista que no ha sido trazada recorremos el vector, de izquierda a derecha, hasta encontrar el primer cero; esto es, hasta que *lineam*(i) = 0. Siguiendo el ejemplo de la Fig.II.6, tenemos que:
lineam(1) = 0.

La segunda etapa genera un nuevo nodo. Ya que se va a generar un nodo nuevo, la variable *ndeh* se incrementa en uno. Este nodo nuevo lo generamos de la siguiente manera: Primero copiamos el vector *lineam* en la matriz *lineah*(*ndeh*, j), con j = 1,...,6; en el ejemplo que estamos utilizando, *ndeh* = 1, por lo que *lineah*(1, j) con j = 1,..,6 que corresponde al primer renglón de la matriz *lineah* es:

0	0	1	2	1	2

A continuación trazamos la arista aún no trazada identificada en la etapa 1. Para ello colocamos el valor de *nj* en la entrada i-ésima del renglón *ndeh*-ésimo de la matriz *linieah*; esto es *lineah*(*ndeh*, i) = *nj*. En el ejemplo que estamos ilustrando suponiendo que el turno corresponde al primer jugador, el primer renglón de la matriz *lineah* es el siguiente:

1	0	1	2	1	2

En la tercera etapa, el valor de la variable i es incrementado en uno y se repiten las etapas primera y segunda del proceso. El proceso finaliza una vez que se hayan recorrido todas las entradas del vector *lineah*.

En ejemplo que estamos siguiendo, la matriz *lineah* generada al final del proceso es la siguiente:

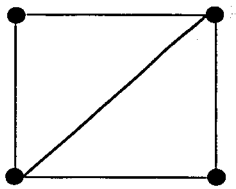
1	0	1	2	1	2
0	1	1	2	1	2

Dado que el número de nuevos estados generados fue dos, tenemos que *ndeh* = 2; esto significa que sólo utilizaremos dos renglones de dicha matriz.

La interpretación de las entradas, en general, en la matriz *lineah* es la siguiente: si *lineah*(*ndeh*, j) = 0, significa que la arista no ha sido trazada; si *lineah*(*ndeh*, j) = 1, significa que la arista fue trazada por el primer jugador; finalmente, si *lineah*(*ndeh*, j) = 2, significa que la arista fue trazada por el segundo jugador. Con esto tenemos los elementos necesarios generar y representar los nuevos estados.

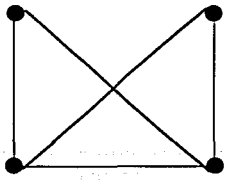
En particular, en el ejemplo que hemos discutido, los estados que representa la matriz *lineah* están ilustrados en la Fig.II.7.

Primer estado.



ndeh=1

Segundo estado



ndeh=2

Figura II.7. Estados representados en la matriz *lineah*

II. 5 Estados "repetidos".

En la sección anterior, describimos como construir el árbol de búsqueda del juego de Sim, realizando la expansión de cada nodo. Este proceso tiene la desventaja de que si aparecen varias veces nodos que representan esencialmente el mismo estado, perdemos tiempo al expandirlos todos, ya que dichos nodos ya se encontraron y expandieron con anterioridad. El esfuerzo que se desperdicia cuando se genera un mismo nodo "más de una vez" puede evitarse "podando" los nodos "repetidos". Así, en lugar de construir un árbol de búsqueda se construye una **gráfica dirigida;** la gráfica dirigida se diferenciará del árbol en el hecho de que varios caminos pueden confluir a un mismo nodo.

Dado que en nuestro caso la forma de "podar" el árbol de búsqueda del juego de Sim es no generar estados que sean iguales a otros que hayan sido anteriormente generados, necesitamos identificar cuándo dos estados son iguales.

De aquí en adelante consideraremos que nuestras gráficas coloreadas siempre tienen por vértices a los números 1,2,...,n, donde n es el número de vértices en el juego de Sim.
Antes de dar una definición formal de cuándo dos estados son "iguales", ilustraremos algunas propiedades que tienen las gráficas asociadas a dichos estados. Para ello, a manera de ejemplo, utilizaremos el juego de cuatro puntos.

Al expandir el estado inicial del juego de cuatro puntos, obtenemos los estados que se muestran en la Fig. II.8.

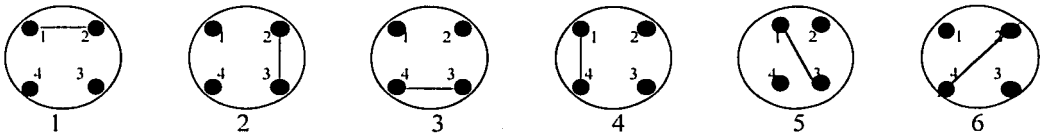


Figura II.8. Estados expandidos a partir del nodo inicial.

Cada uno de los estados anteriores es una gráfica coloreada de orden cuatro, formada únicamente por una arista roja que une a dos vértices. Se observa que para efectos del juego no hay ninguna diferencia esencial entre estos seis estados. Más formalmente, esto se debe a lo siguiente: En cada una de estas gráficas podemos encontrar una permutación de los vértices de tal manera que obtengamos siempre el mismo estado (digamos el primero). Notemos que si en la gráfica del estado 2 permutamos el vértice 1 con el 3, obtenemos la gráfica del estado 1. En otras palabras, podemos considerar que la gráfica el estado 2 es la misma que la del estado 1, pero en la cual se ha cambiado la numeración de los vértices. Asimismo, si en la gráfica del estado 3 permutamos el vértice 1 con 4 y el vértice 2 con el 3, obtenemos nuevamente la gráfica del estado 1. De manera análoga, para las gráficas de los estados 4, 5 y 6 podemos encontrar permutaciones de los vértices de tal forma que obtengamos la gráfica del estado 1.

Como primera idea, entonces, podemos considerar en general que dos estados son "iguales" si existe una permutación de los vértices de las gráficas coloreadas asociadas a estos estados tal que puede obtenerse una gráfica a partir de la otra. Como ya se observó para el ejemplo del nivel 1, desde el punto de vista del juego no hay diferencia entre dos estados equivalentes A y B con gráficas coloreadas G_1 y G_2: no importa la sucesión de jugadas que hayan conducido a estos estados, ni importa cuáles sean las aristas particulares rojas y azules de G_1 y G_2, lo importante para el jugador es la configuración de aristas rojas y azules a la que se enfrenta, así que no importa si estamos en el estado A o en el B para efectos de decidir cuál es la jugada que nos conviene hacer. Más formalmente, estamos considerando que dos estados A y B son "iguales" si existe una permutación f de los vértices de la gráfica G_1 de tal forma que obtengamos la gráfica G_2 simplemente poniendo,

por cada arista roja $[i, j]$ de G_1, una arista roja $[f(i), f(j)]$ en G_2 y similarmente para las aristas azules.

Por otro lado, en el Capítulo I, quedó establecido que dos gráficas coloreadas G_1 y G_2 son equivalentes si y sólo si existe un isomorfismo cromático de G_1 en G_2, esto es una función biyectiva f de los vértices de G_1 en los vértices de G_2 tal que cumpla las siguientes condiciones:

1) Dos vértices cualesquiera v_1, v_2 de la gráfica G_1 son adyacentes si y sólo si los vértices $f(v_1)$, $f(v_2)$ son adyacentes en la gráfica G_2.

2) Cualquier arista $[v_1, v_2]$ es de color c en G_1 si y sólo si la arista $[f(v_1), f(v_2)]$ es del mismo color c en G_2.

En consecuencia, nuestra noción de estados "iguales" coincide plenamente con la equivalencia (isomorfía cromática) de las gráficas coloreadas correspondientes. Por esto, a partir de ahora diremos que dos estados (o dos nodos del árbol de juego) son equivalentes cuando las gráficas coloreadas correspondientes sean equivalentes.

Entonces, como se verá en la siguiente sección construiremos la gráfica dirigida del juego de Sim poniendo sólo un nodo por cada clase de equivalencia de nodos del árbol de juego.

II.6 La gráfica dirigida del juego de Sim

Para ilustrar la construcción de la gráfica dirigida, retomemos el ejemplo del juego con cuatro vértices. Como se vio en la sección anterior, al expandir el estado inicial se obtienen seis estados, todos equivalentes entre sí. De ahí que, en la gráfica dirigida del juego, sólo colocamos uno de estos seis estados. Por ejemplo, coloquemos el estado 3. Esta situación queda ilustrada en la Fig. II.9. Nótese que la dirección de la gráfica es ascendente: el nodo inicial (abajo) es padre del nodo superior.

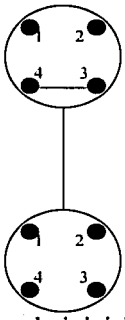


Figura II.9. Expansión del nodo inicial en la gráfica dirigida.

En el siguiente paso tenemos que expandir el nodo hoja de esta gráfica dirigida. El segundo jugador tiene las posibilidades que se muestran en la Fig. II.10.

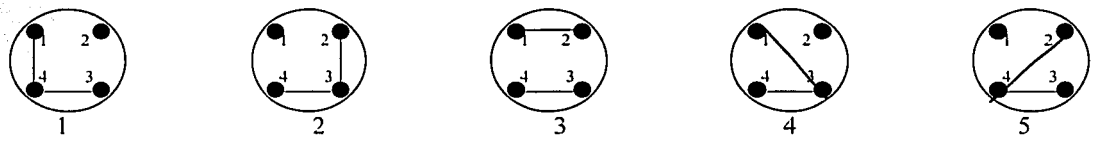


Figura II.10. Opciones del segundo jugador.

Podemos observar que los estados 1, 2, 4 y 5 que se muestran en la Fig. II.10 consisten de una gráfica coloreada donde hay una arista roja y una azul que inciden en un mismo vértice. En cada uno de estos estados, también podemos encontrar una permutación de los vértices de tal manera que el vértice que une a las dos aristas, la roja y la azul, sea siempre el mismo. Por ejemplo, en el estado 2 si permutamos el vértice 1 con el 2 y el 4 con el 3 obtenemos el primer estado. Entonces, los estados 1, 2, 4 y 5 son equivalentes entre sí.
Sin embargo, en el estado 3 tenemos una gráfica con una arista azul y otra roja, que no tienen un vértice en común. Para esta gráfica no existe ninguna permutación de los vértices que nos permita obtener alguno de los otros estados. Por lo tanto, el estado 3 no es equivalente a ninguno de los otros cuatro.

Así, el segundo nivel de la gráfica dirigida del juego está constituido únicamente por dos estados; a saber, el estado 3 y cualquiera de los estados 1, 2, 4 o 5 (usaremos el 1). En la Fig. II.11 se muestra la gráfica dirigida del juego hasta el segundo nivel.

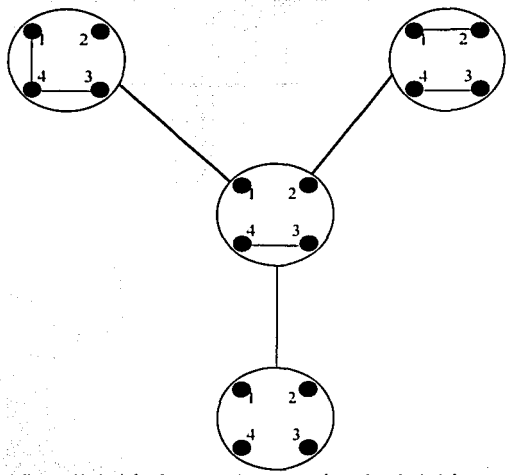


Figura II.11. Gráfica dirigida hasta el segundo nivel del juego de cuatro puntos.

Para continuar con el proceso de construcción de la gráfica dirigida, necesitamos expandir los dos nuevos nodos hoja. En cada uno de ellos, el primer jugador tiene las posibilidades que se muestran en la Fig. II.12.

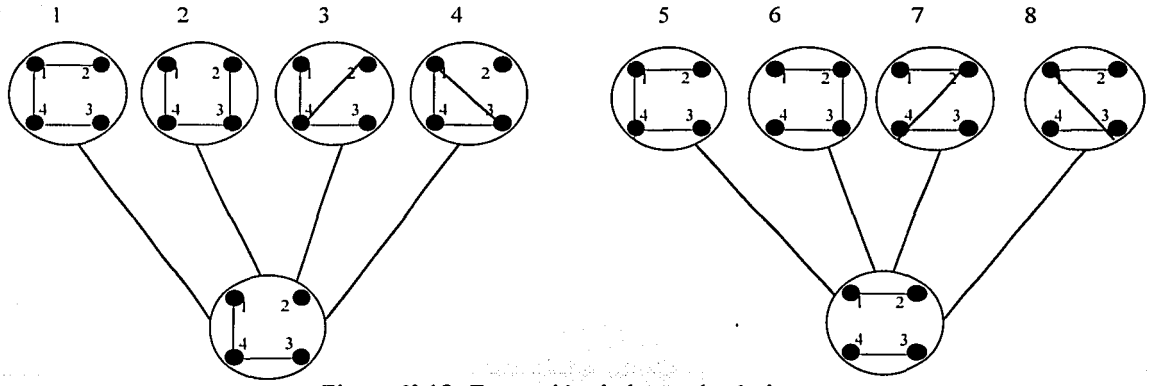


Figura II.12. Expansión de los nodos hojas.

Podemos ver que el estado 1 de la Fig.II.12 consiste de una gráfica coloreada donde una arista azul une a dos vértices, en los cuales a su vez incide una arista roja, las cuales no están unidas entre sí. El estado 2 consiste de una gráfica en la que en uno de sus vértices inciden dos aristas rojas, y existe una arista azul que une el vértice ajeno a las aristas rojas con uno de los vértices donde incide sólo una arista roja. En este caso, tenemos que los estados 1 y 2 no son equivalentes, pues claramente no hay ningún isomorfismo cromático entre las gráficas correspondientes, ya que en el estado 1 no existe un vértice donde incidan dos aristas rojas.

El estado 3 consiste de una gráfica coloreada en la que en uno de sus vértices inciden dos aristas rojas y una azul; en este caso tampoco es posible encontrar un isomorfismo cromático con alguno de los dos estados anteriores, ya que en los estados 1 y 2 no existe un vértice en el que incidan tres aristas.

El estado 4 consiste de una gráfica coloreada en la que hay dos aristas rojas que inciden en un mismo vértice, y las aristas rojas a su vez inciden en dos vértices unidos por una arista azul. Este estado tampoco es equivalente a ninguno de los anteriores, ya que en él hay un vértice aislado y en los otros no.

En los estados 5, 6, 7 y 8 tenemos una gráfica coloreada en la que siempre existe un vértice donde inciden dos aristas rojas, y sólo una de las aristas rojas, a su vez, incide en un vértice donde también lo hace una arista azul. Todos estos estados son equivalentes entre sí y con el estado 2. Veamos, para cada caso, un isomorfismo cromático con el estado 2.

En el caso del estado 5, una permutación f que nos permite obtener el estado 2 es la siguiente: f envía el vértice 1 al 4, el vértice 2 al 1, el vértice 3 al 2 y el 4 al 3. Esto lo podemos expresar en la siguiente manera:

$$f = \begin{pmatrix} 1234 \\ 4123 \end{pmatrix}.$$

Veamos con detalle que f es un isomorfismo cromático de la gráfica coloreada 5 a la 2: Las aristas rojas de la gráfica 5 son [1,4] y [4,3] y [f(1), f(4)] = [4,3] y [f(4), f(3)] = [3,2] son las aristas rojas de la gráfica 2. Por otra parte, la única arista azul de la gráfica 5 es [1,2], y [f(1), f(2)] = [4,1] es la única arista azul de la gráfica 2. En consecuencia, las gráficas 2 y 5 son equivalentes y sólo pondremos una de ellas (la 2) en la gráfica dirigida del juego.

Para los estados 6, 7 y 8 sirven los isomorfismos cromáticos con el estado 2 dados por las tablas siguientes:

$$f = \begin{pmatrix} 1234 \\ 1432 \end{pmatrix}.$$

$$f = \begin{pmatrix} 1234 \\ 1423 \end{pmatrix}.$$

$$f = \begin{pmatrix} 1234 \\ 4132 \end{pmatrix}.$$

Por lo tanto, en la gráfica dirigida del juego agregamos los estados 1, 3, 4 que no son equivalentes entre sí y sólo uno (el 2) de los estados 2, 5, 6, 7 y 8. Observemos que los cuatro hijos 1, 2, 3 y 4 del nodo izquierdo del segundo nivel permanecen en nuestra gráfica dirigida que por tanto contiene una subgráfica como la parte izquierda de la Fig. II.13. Sin embargo, los hijos 5, 6, 7 y 8 del nodo derecho del segundo nivel desaparecen en la gráfica dirigida pues son equivalentes al nodo 2, esto se señala poniendo una arista roja ascendente, en la gráfica dirigida, del nodo derecho del segundo nivel al nodo 2. Esta situación se muestra en la Fig. II.13.

En general, la gráfica dirigida D del juego se construye usando las convenciones que aquí hemos introducido: como nodos de D, ponemos uno y sólo uno de cada clase de equivalencia de estados, y entre dos nodos A y B ponemos una arista ascendente (del color del jugador en turno) de A en B si existen nodos A' y B' del árbol del juego, equivalentes a A y B respectivamente, tales que B' es hijo de A'. En particular, es por esto que la gráfica dirigida no necesariamente es un árbol, como también puede verse en la Fig. II.13.

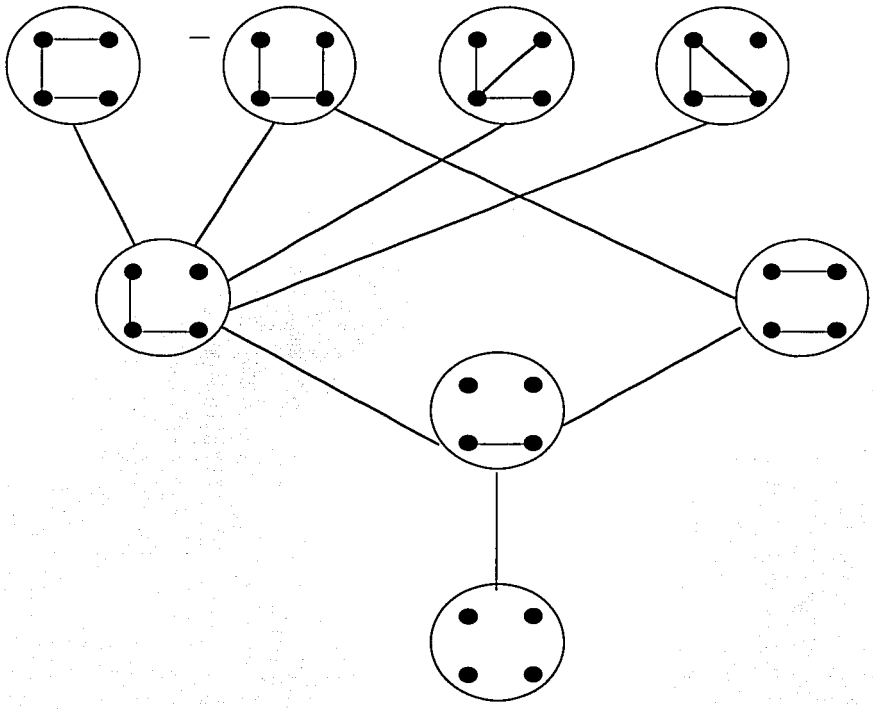


Figura II.13. Gráfica dirigida hasta el cuarto nivel.

II.7 Localización de estados equivalentes

Para desarrollar el algoritmo que nos permita decidir en la computadora si dos estados son equivalentes, desarrollaremos primero una nueva forma de representar cada uno de los estados.

En este algoritmo, representaremos un estado por medio de una matriz cuadrada, denominada matriz *estado*, cuya dimensión es igual al número de vértices del juego; en el caso que estamos usando como ejemplo, la matriz será de dimensión cuatro. La matriz *estado* toma en cuenta la coloración y es una variante de la matriz de adyacencia (Def. 9, Capítulo I). En esta matriz la entrada (i, j) representa la arista que une el vértice i con el

vértice j, así que colocaremos un cero en la entrada (i, j) en caso de que los vértices i, j no estén unidos; el número 1 en la entrada (i, j) indicará que la arista la trazó el primer jugador (color rojo), mientras que el número 2 indicará que la trazó el segundo jugador (azul).
A fin de ilustrar esta manera de representar un estado, tomemos, por ejemplo, el estado G_1 que se muestra en la Fig. II.14.

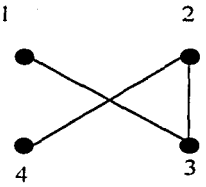


Figura II.14. Estado G_2.

La matriz *estado* que representa al estado de la Fig. II.14 tiene las siguientes características: Es una matriz de cuatro por cuatro. Las entradas (1,2), (1,4) y (3,4) son iguales a cero. Las entradas (2,3) y (3,4) son iguales a 1 y la entrada (1,3) es igual a 2.
La arista que une al vértice i con el vértice j, es la misma que une al vértice j con el vértice i; por lo tanto, la entradas (i, j) y (j, i) toman el mismo valor (0, 1 ó 2); esto es, la matriz *estado*, al igual que la de adyacencia, es una matriz simétrica. En las entradas de la diagonal de la matriz, esto es, en las entradas de la forma (i, i) siempre habrá ceros ya que los puntos de la gráfica no están conectados consigo mismos.
La matriz que representa al estado G_1 de la Fig. II.14 es la siguiente:

0	0	2	0
0	0	1	1
2	1	0	0
0	1	0	0

Una vez establecida la nueva forma en que vamos a representar los estados, en la computadora, a partir de la matriz *estado*, pasamos a analizar el caso de dos estados equivalentes. Para ello, junto con el estado G_1 de la Fig. II.14, tomemos el estado G_2 ilustrado en la Fig. II.15 y la matriz que lo representa.

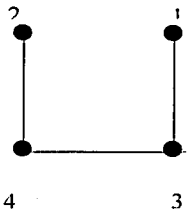
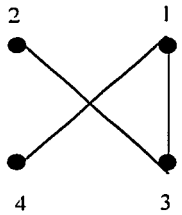


Figura II.15. Estado G_2

1	0	0	1	0
2	0	0	0	2
3	1	0	0	1
4	0	2	1	0
	1	2	3	4

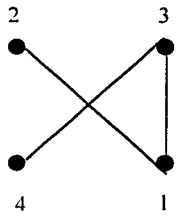
Los estados G_1 y G_2 de las Figuras II.14 y II.15 respectivamente son equivalentes, un isomorfismo cromático ϕ de G_1 en G_2 consiste en enviar los vértices 1, 2, 3 y 4 a los vértices 2, 3, 4 y 1 respectivamente. Sin embargo, como las gráficas coloreadas son distintas, las matrices que representan a estos estados son necesariamente distintas.

Para entender cómo se obtiene la matriz de G_2 a partir de la de G_1 y la permutación ϕ que transforma G_1 en G_2, realizaremos una transición más gradual de G_1 a G_2.
En cada paso intermedio, la nueva gráfica coloreada se obtendrá de la anterior mediante el intercambio de dos vértices. Simultáneamente, iremos construyendo la matriz asociada a cada nuevo estado obtenido.
En el primer paso, intercambiamos los vértices 1 y 2 de G_1:



0	0	1	1
0	0	2	0
1	2	0	0
1	0	0	0

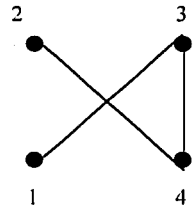
En el siguiente paso intercambiamos los vértices 1 y 3 de la gráfica anterior:



—

0	2	1	0
2	0	0	0
1	0	0	1
0	0	1	0

Y finalmente intercambiamos los vértices 1 y 4:



0	0	1	0
0	0	0	2
1	0	0	1
0	2	1	0

Esta última gráfica es G_2. Notemos que a cada paso; esto es, en cada permutación del vértice i por el vértice j, la nueva matriz obtenida es el resultado de intercambiar el renglón i por el renglón j y después intercambiar la columna i por la columna j. Esto que se verifica fácilmente cuando el isomorfismo consiste de sólo intercambiar dos vértices ocurre en general: ahora podemos verificar de una sola vez que la matriz de G_2 se obtiene de la de G_1 permutando los renglones mediante la permutación ϕ y a continuación permutando las columnas también mediante la permutación ϕ. Sin embargo, en nuestro algoritmo no permutaremos los renglones y las columnas de las matrices de los estados, sino que calcularemos las matrices de los permutados de los estados.

Como vimos en el Capítulo I, la clase de equivalencia de un estado A es simplemente el conjunto [A] de sus permutados. En particular, los estados A y B son equivalentes, si y sólo si, el conjunto formado por las matrices de los permutados de A es igual al conjunto formado por las matrices de los permutados de B.

Para decidir con la computadora si los estados A y B son iguales vamos a construir una función que a cada matriz *estado* le asigne un número, al que llamaremos "número característico provisional", (denotado por *ncp*), de tal forma que, para matrices distintas el número sea distinto y para matrices iguales el número sea el mismo.

Entonces, como decidir si los estados A y B son iguales es lo mismo que decidir si los conjuntos de matrices de los permutados de A es igual al conjunto de las matrices de los permutados de B, bastará con decidir si los conjuntos de números característicos provisionales de las matrices de los permutados de A y B son iguales o ajenos. Más aún, si consideramos para un estado A el mínimo de los números característicos provisionales de las matrices de los permutados de A (a éste le llamaremos el número característico del estado A), es claro que los estados A y B son iguales si y sólo si sus números característicos son iguales. En la siguiente sección construiremos el número característico provisional asociado con una matriz *estado*. Para ahorrar espacio en la computadora, y dado que la matriz *estado* es simétrica y con ceros en la diagonal, utilizaremos sólo la parte inferior de la misma; esto es, la arista que une al vértice i con el vértice j la representaremos en *estado*(i, j), con i > j.

II.8 Construcción del número característico provisional.

Para la construcción de esta función utilizaremos la matriz *estado*, así como, la numeración que dimos a cada arista en el algoritmo *generahijas* en la sección 2.6. La numeración tanto de las aristas como de los vértices se muestra en la Fig. II.16.

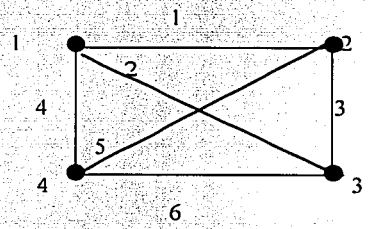


Figura II.16. Numeración de aristas del juego de cuatro puntos.

De la Fig. II.16 podemos observar la siguiente relación que existe entre el vector *lineam* y la matriz *estado*: La arista señalada con el número 1, en la Fig. II.16, une al vértice 1 con el 2, por lo tanto, esta arista siempre va a estar representada en *estado*(2, 1); la arista señalada con el número 2, une al vértice 1 con el 3, por lo que, siempre estará representada en *estado*(3,1), etc.

En el caso general del juego de n puntos, la entrada (i, j) de la matriz *estado* (con i > j) corresponderá a la entrada $\binom{i-1}{2} + j$ del vector *lineam*. Esto corresponde a numerar las aristas de la gráfica completa de n vértices en el orden: [2, 1], [3, 1], [3, 2], [4, 1],..., [4, 3], [5, 1],..., [n, n-1].

Para facilitar la construcción del número característico provisional, además, de representar las jugadas del primero y segundo jugador, señalaremos en la matriz *estado* el lugar que le corresponde a cada arista, con el número de ésta entre paréntesis.

Consideremos el estado representado en la Fig. II.17 y su matriz asociada:

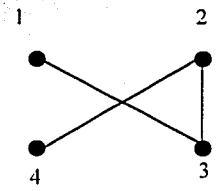


Figura II.17. Estado del juego de cuatro puntos.

0			
(1) 0	0		
(2) 2	(3) 1	0	
(4) 0	(5) 1	(6) 0	0

El número característico provisional será construido en base 2, aunque en la máquina estará guardado en una variable entera de doble precisión. Para asignarlo a la matriz del estado de la Fig. II.17, hacemos lo siguiente:

Recorremos la matriz, buscando las jugadas del primer jugador, esto es, los 1.

Si en el lugar señalado con (1) hay un 1, el primer dígito, de derecha a izquierda, del número característico provisional, *ncp*, es 1, ($ncp = 2^0$), si no, es 0. Si en (2) hay un 1, el segundo dígito es 1, ($ncp = ncp + 2^1$), si no es 0. Realizamos el procedimiento anterior para cada una de las entradas de la matriz *estado* señaladas con el número (1) hasta el número (6). Con esto tenemos la primera mitad de los dígitos del número característico provisional, en el ejemplo que estamos siguiendo, tendremos ya los primeros seis dígitos.

Procedemos de la misma manera para la segunda mitad de los dígitos. Sólo que ahora, buscaremos las aristas trazadas por del segundo jugador, esto es, las entradas de la matriz donde haya un 2. Otra vez, empezamos por la entrada señalada con el número (1); si aquí hay un 2, el primer dígito de la segunda mitad del número será 1; en el caso del juego de cuatro puntos el séptimo dígito será 1, ($ncp := ncp + 64*2^0$), si no, el dígito será 0. Si en el lugar (2), hay un 2, el siguiente dígito será 1, ($ncp = ncp + 64*2^1$), si no 0. Y así para cada una de las entradas de la matriz *estado* señaladas con el número (1), hasta el número (6).
En el caso del estado representado por la Fig. II.17, las aristas 3 y 5 fueron trazadas por el primer jugador, la 2 por el segundo y las aristas 1, 4 y 6, no han sido trazadas:

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	0	1	0	1	0	0
6	5	4	3	2	1	6	5	4	3	2	1
jugadas del segundo tirador						jugadas del primer tirador					

El 000010010100 (en base dos) es el número que asignamos a la matriz del estado, en notación decimal, $ncp = 2^2 + 2^4 + 2^7 = 128 + 16 + 4 = 148$.

Para cada una de las posibles permutaciones de los vértices de un estado, asignaremos una matriz, y a cada matriz un número característico provisional.
Ahora asignaremos a cada estado su número característico, que no es más que el mínimo de los números característicos provisionales de las matrices de sus permutados.
Entonces, como ya se aclaró en la sección anterior, los estados A y B son iguales si y sólo si sus números característicos son iguales.

II.9 Algoritmo para construir el número característico provisional.

En el algoritmo para construir el número característico provisional de una matriz *estado*, al que llamaremos *asignancp*, necesitamos representar en la computadora las entradas de la matriz *estado* que le corresponden a cada arista (En el ejemplo anterior, los números señalados entre paréntesis).

Para esto usaremos una matriz cuadrada denominada *localización*, cuya dimensión es igual al número de vértices del juego. En esta matriz guardaremos las posiciones en el vector *lineam* que corresponden a las aristas de la gráfica completa (o a las entradas de la matriz *estado*). Como ya se dijo antes, sólo usaremos la parte inferior de la matriz. En el caso de 4 puntos, esta matriz la definimos de la siguiente manera:

localizacion(2,1) = 1
localizacion(3,1) = 2
localizacion(3,2) = 3
localizacion(4,1) = 4
localizacion(4,2) = 5
localizacion(4,3) = 6.

En el caso general del juego de n puntos, gracias a lo visto antes, será:

$$localizacion(i, j) = \binom{i-1}{2} + j.$$

También utilizaremos dos vectores, cuya dimensión es igual al número de vértices del juego, a los que llamaremos *lir y lic*. En el vector *lir* representaremos el renglón de la matriz *estado* que le corresponde a cada arista de un estado cualquiera (o a cada posición en el vector *lineam*). De la misma manera en el vector *lic* representaremos la columna de la matriz *estado* que le corresponde a cada arista del juego. Por ejemplo, la arista 1 que va del vértice 1 al 2 está representada en *estado*(2,1), por lo que, *lir*(1) = 2 y *lic*(1) = 1, etc.

lir(1) = 2 la arista 1 se localiza el renglón 2 de *estado*.
lir(2) = 3. la arista 2 se localiza el renglón 3 de *estado*.
lir(3) = 3. la arista 3 se localiza el renglón 3 de *estado*.
lir(4) = 4. la arista 4 se localiza el renglón 4 de *estado*.
lir(5) = 4. la arista 5 se localiza el renglón 4 de *estado*.
lir(6) = 4. la arista 6 se localiza el renglón 4 de *estado*.
lic(1) = 1 la arista 1 se localiza en la columna 1 de *estado*.
lic(2) = 1 la arista 2 se localiza en la columna 1 de *estado*.

lic(3) = 2 la arista 3 se localiza en la columna 2 de *estado.*
lic(4) = 1 la arista 4 se localiza en la columna 1 de *estado.*
lic(5) = 2 la arista 5 se localiza en la columna 2 de *estado.*
lic(6) = 3 la arista 6 se localiza en la columna 6 de *estado.*

En el algoritmo *asignancp*, usaremos las siguientes variables:
nl = número de líneas de la gráfica completa del juego.
qncp = número característico provisional correspondiente a una matriz *estado*.
qnci = 2^{nl} lugar que le corresponde al primer dígito de la segunda mitad del número característico provisional.
(La *q* al principio es para que sea de doble precisión).

El algoritmo *asignancp* comienza asignando cero a la variable *qncp* y ejecuta a continuación las siguientes dos etapas:

En la primera etapa construimos la primera mitad del número característico provisional, para ello buscamos en la matriz *estado* las aristas trazadas por el primer jugador. Para ello localizamos los números 1 en las entradas de la matriz *estado*, señaladas con el número (1) hasta el número (6), de la siguiente manera:

Para i = 1 **hasta** *nl*
nrenglon = *lir*(*i*)
ncolumna = *lic*(*i*)
Si *estado* (*nrenglon*, *ncolumna*) = 1 **entonces**
qncp = *qncp* + $2^{(i-1)}$

En la segunda etapa construimos la segunda mitad del número característico provisional. Para lo cual buscamos en la matriz *estado* las aristas trazadas por el segundo jugador. Esto es, localizamos los números 2 en las entradas señaladas con el número (1) hasta el (6) de la siguiente manera:

Para i = 1 **hasta** *nl*:
nrenglon = *lir*(i)
ncolumna = *lic*(*i*)

Si *estado* (*nrenglon*, *ncolumna*) = 2 **entonces**

$qncp = qncp + qnci * 2^{(i-1)}$

Claramente, al terminar *asignancp* el número característico provisional de la matriz *estado* queda almacenado en la variable *qncp*.

II.10 Algoritmo para asignar el *número característico* a un estado.

El algoritmo *varitamagica*, para asignar el número característico a un estado A utiliza, a su vez, otros dos algoritmos además de *asignancp*. Primero explicaremos el algoritmo *varitamagica* y después cada uno de los dos nuevos algoritmos usados en él.

En el algoritmo *varitamagica* necesitamos lo siguiente:

1.- La variable *nv* donde representamos el número de vértices del juego.

2.- Un vector de dimensión *nv* al que llamamos *npermutacion*. En él representaremos cada permutación. Por ejemplo, en el juego de cuatro puntos cuando aún no hemos hecho ninguna permutación, *npermutación*(1) = 1, *npermutación*(2) = 2, *npermutación*(3) = 3, *npermutación*(4) = 4; si permutamos el vértice 1 con el 2, entonces *npermutación*(1) = 2, *npermutación*(2) = 1, *npermutación*(3) = 3, *npermutación*(4) = 4.

3.-La matriz *estado* de cada permutación la representamos en una matriz cuadrada llamada *estadop*, ya que la matriz *estado* donde no hemos hecho ninguna permutación la utilizaremos en el resto del programa.

4.- La variable *qncp* donde representamos el número característico provisional.

5.- La variable *qncmin* para representar el menor de los números característicos provisionales

6.- La variable *qnc* para representar el número característico.

7.- La variable *indicp* que nos indicará si existe otra permutación de los vértices.

Este algoritmo consiste de tres etapas. Para ilustrarlo vamos asignar el número característico al estado A representado en la Fig. II.18.

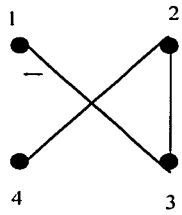


Figura II.18. Estado A

En la primera etapa generamos el número característico provisional del estado A, donde aún no hemos hecho ninguna permutación, para esto hacemos lo siguiente:

a)- Representamos cada vértice del estado donde aún nos hemos hecho ninguna permutación en los vectores *npermutacion*,

Para i = 1 **hasta** *nv*.

npermutacion(i) = i

En el ejemplo *npermutación*(1) = 1, *npermutación*(2) = 2, *npermutación*(3) = 3, *npermutación*(4) = 4

b)- Utilizando el vector *npermutacion*, con el algoritmo *generaestadop* generamos la matriz *estadop* correspondiente a los vértices del inciso anterior (este algoritmo se explicará más adelante).

En el ejemplo la matriz *estadop* es la siguiente:

0			
0	0		
2	1	0	
0	1	0	0

c)- Con el algoritmo *asignancp* asignamos el número característico provisional, *qncp* a la matriz *estadop* generada en inciso anterior

En el ejemplo, este número es $000010010100 = 2^2 + 2^4 + 2^7 = 148$.

d)- Como nos interesa el menor de los números característicos provisionales, cada vez que generamos uno, compararemos si es menor que los generados anteriormente.

Como hasta ahora sólo hemos calculado el primer número característico provisional entonces:

qncmin = *qncp*

En el ejemplo *qncmin* = *qncp* = 148.

En la segunda etapa generamos los estados permutados de A y calculamos el número característico provisional a la matriz correspondiente.

a) Con el algoritmo *siguientepermutacion* generamos el nuevo vector *npermutacion* si es que aún hay tal, en este caso le damos a *indicp* el valor 1, y si ya han sido usadas todas las permutaciones, le damos a *indicp* el valor 0.

En el ejemplo, *npermutacion*(1) = 1, *npermutacion*(2) = 2, *npermutacion*(3) = 4, *npermutacion*(4) = 3.

b) En el caso de que en inciso anterior haya existido una permutación más de los vértices del estado A asignamos con el algoritmo *generaestadop* la matriz *estadop* correspondiente al nuevo estado permutado A'. Esto es:

Si *indicp* = 1 **entonces**

Gosub *generaestadop*

En el ejemplo, la matriz *estadop* correspondiente ahora al estado A' es la siguiente:

0			
0	0		
0	1	0	
2	1	0	0

c) Con el algoritmo *asignancp* asignamos el número característico provisional, *qncp*, a la matriz del inciso anterior.
En el ejemplo, el número característico provisional correspondiente a la matriz *estadop* es: $qncp = 001000010100 = 2^2 + 2^4 + 2^9 = 532$.

d) Comparamos si el número característico provisional que calculamos en el inciso anterior con el menor calculado hasta aquí, es decir,
si $qncmin > qncp$ **entonces** $qncmin = qncp$

En el ejemplo, el número característico provisional del estado A' es mayor que *qncmin*, por lo tanto *qncmin* queda con el valor que tenía.

e) Repetimos la segunda etapa del algoritmo hasta que no exista otra permutación del estado A, esto es, hasta $indicp = 0$.

La tercera etapa consiste en asignar el más pequeño de los números característicos provisionales a la variable *qnc*:

$qnc = qncmin$.

Al terminar el algoritmo *varitamagica* el número característico del estado A está representado en la variable *qnc*.

A continuación explicaremos los dos algoritmos (el que genera las permutaciones y el que genera la matriz *estadop* correspondiente al estado permutado), que utilizamos en el algoritmo anterior.

II.11 Algoritmo para generar la matriz *estadop* correspondiente a una permutación.

El algoritmo, *generaestadop*, que asigna la matriz, *estadop*, a cada nuevo estado A' utiliza la numeración que se le dio a las aristas de la gráfica completa en el algoritmo *generahijas* en la Sección 4 de este capítulo. Se usarán los vectores *lir* y *lic* de la Sección 9.

Para ilustrar este algoritmo tomaremos como ejemplo el estado A de la Fig. II.19 y la matriz que lo representa.

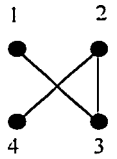


Figura II.19 estado A

0			
0	0		
2	1	0	
0	1	0	0

Para obtener el estado A', ilustrado en la Fig. II.20, permutamos el vértice 4 con el 3.

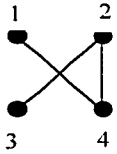


Figura II.20 estado A'

Recordemos, que este algoritmo forma parte del programa para asignar el número característico a un estado A. En él, los vértices permutados los representamos en la matriz *npermutacion*. Por lo tanto, los vértices del estado A' del ejemplo están representados como sigue:

npermutacion(1) = 1
npermutacion(2) = 2

npermutacion(3) = 4

npermutacion(4) = 3.

En el algoritmo *generaestadop*, para generar la matriz *estadop* correspondiente al estado A' se utilizan:

1.- El vector *npermutacion* que contiene la permutación que se aplicará al estado A.

2.- Las matrices *lir* y *lic* para localizar el renglón y la columna de la matriz *estado* donde está representada cada arista. (Estas matrices fueron definidas en la Sección 9)

3.- Las variables *nrenglon y ncolumna* donde representaremos los vértices (renglones y columnas de la matriz *estado*) de cada arista del estado A.

4.- Las variables *nrenglonp* y *ncolumnap* donde representaremos provisionalmente los valores que toman *nrenglon y ncolumna* después de aplicar la permutación *npermutacion* de los vértices.

5. La variable *nl* que representa el número de aristas del juego.

6. La variable *ntemp* donde representaremos provisionalmente el valor de cada arista en la matriz *estado*.

El algoritmo *generaestadop* es el siguiente:

Se repiten las siguientes etapas una vez por cada línea **i** de la gráfica completa del juego.

Para i = 1 **hasta** *nl*

a) Asignamos en las variables *nrenglon* y *ncolumna* el número de renglón y columna de la matriz *estado* que corresponde a la arista **i** del estado A.

nrenglon = *lir*(i)

ncolumna = *lic*(i)

En el ejemplo, *nrenglon* = *lir*(1) = 2 y *ncolumna* = *lic*(1) = 1.

b) Asignamos a la variable *ntemp* el valor (0, 1 ó 2) con el que está representada la arista **i** en *estado*.

ntemp = *estado*(*nrenglon, ncolumna*).

En el ejemplo la arista 1 está representada en *estado*(2, 1) por un 0. Entonces *ntemp* = 0.

c) Utilizando la permutación *npermutacion*, asignamos en las variables *nrenglonp* y *ncolumnap* los números que corresponden a los vértices de la arista **i**, después de la permutación.

nrenglonp = *npermutacion*(*nrenglon*)

ncolumnap = *npermutacion*(*ncolumna*)

En el ejemplo estos vértices quedaron fijos por lo que *nrenglonp* = *npermutacion*(1) = 1 y *ncolumnap* = *npermutacion*(2) = 2.

d) Para utilizar la parte inferior de la matriz necesitamos asegurarnos de guardar *ntemp* en *estadop*(i, j) con i < j, por esto:

Si *nrenglonp* < *ncolumnap* **entonces**

nntemp = *nrenglonp*

nrenglonp = *ncolumnap*

ncolumnap = *nntemp*

En el ejemplo, *nrenglonp* < *ncolumnap* entonces

nntemp = *nrenglonp* = 1

nrenglonp = *ncolumnap* = 2

ncolumnap = *nntemp* = 1

e) Copiamos el número que le corresponde a la arista **i** en *estado* en la matriz *estadop*, (que tenemos representado en *ntemp*) a su nuevo lugar en la matriz correspondiente al estado A' con los vértices permutados.

estadop(nrenglonp, ncolumnap) = *ntemp*

En el ejemplo, *estadop*(2, 1) = 0.

II.12 Algoritmo que permuta los vértices de un estado A.

El algoritmo *siguientepermutacion*, que realiza las permutaciones de los vértices de un estado A utiliza una propiedad de las permutaciones que se refiere a la forma en que una cadena de símbolos puede ser ordenada.

Por ejemplo, los números naturales expresados en base 10. Aquí los símbolos son los dígitos 0, 1, 2,..., 9 y las cadenas son los números que se pueden formar con ellos. Como los dígitos están ordenados del 0 sigue el 1, del 1 el 2, etc., esto genera el orden a que estamos acostumbrados en los números.

Si consideramos las permutaciones de tres dígitos 1, 2, 3 podemos ver el orden como si las permutaciones fueran números de tres dígitos y las ordenamos en orden creciente: 123, 132, 213, 231, 312, 321. Observemos que estas permutaciones, interpretadas como números naturales, son todos los números de tres cifras **distintas** que se pueden formar con los dígitos 1, 2 y 3.

En general se puede definir el orden de las cadenas de símbolos comparando las cadenas, símbolo a símbolo, de izquierda a derecha hasta encontrar el primer lugar donde las cadenas difieran. Una cadena va antes que la otra si en ese lugar tiene un símbolo que va antes que el símbolo que tiene la otra.

El algoritmo *siguientepermutacion* recibe una permutación y regresa la siguiente en el orden descrito antes. Si ya no hay una permutación siguiente, entonces lo indica con la variable *indicp* = 0.

El algoritmo que permuta los vértices de un estado empieza a partir del estado original, donde no se ha hecho ninguna permutación. Por la manera que numeramos los vértices de un estado, la permutación original corresponde a la cadena 1, 2, ..., n. Esta cadena es el número más pequeño que podemos formar con estos "dígitos". La forma en que generaremos las permutaciones es en orden creciente, es decir a partir del número más pequeño formado por los dígitos 1, 2,..., n hasta el más grande formado por estos mismos dígitos.

El algoritmo *siguientepermutacion* consiste en varias etapas. La primera etapa consiste en localizar la parte de la permutación que va a cambiar, las siguientes tienen que ver con cómo efectuar el cambio.

Primera etapa.- Localizar la parte que va a cambiar.
Van a cambiar los dígitos desde un cierto lugar a la derecha y los números a la izquierda quedan igual. Por ejemplo, en el caso de 4 dígitos, a la permutación 1234 claramente sigue la permutación 1243: el 34 cambia por el 43, éstos son los dos últimos dígitos de la parte de la derecha que cambian, el 1 y el 2 no cambiaron: ésta es la parte de la izquierda que no cambia. Aquí lo que hay que localizar es el segundo dígito de derecha a izquierda, el 3, y a partir de este dígito a la derecha es donde van a ocurrir los cambios. Como segundo ejemplo consideraremos la permutación 2437651.

La regla que sigue la rutina en general es la siguiente:
Recorre la cadena (la permutación) de derecha a izquierda, hasta encontrar el primer número que tenga a su derecha (en cualquier parte de la derecha) un número mayor que él (al menos uno ya que pueden ser varios). En el ejemplo de la permutación 1234, el 4 no tiene nada a la derecha, así que no es el que buscamos (el último nunca va a ser, por esta razón). El que sigue de derecha a izquierda es el 3, como el 3 tiene a su derecha el 4, y el 4 es mayor que el 3 entonces el 3 es el que buscamos ya que es el primer número, de derecha a izquierda, que tiene números mayores que él a su derecha. En la permutación 2437651, el número buscado es el 3.

Segunda etapa.- Localizar el menor de los números mayores que el encontrado en la etapa 1 y que están a la derecha de éste.
En el caso de la permutación 1234 el único número mayor que el 3 y a su derecha es el 4, de manera que este es el que buscamos. En el segundo ejemplo hay tres números mayores que el 3 a su derecha: el 7, el 6 y el 5, de modo que el 5 es el buscado.

Tercera etapa.- Poner el número localizado en la segunda etapa en lugar del localizado en la primera etapa.
En el caso de la permutación 1234 hay que poner el 4 (el encontrado en la segunda etapa) de manera que la siguiente permutación va a empezar con 124. En el segundo ejemplo, la siguiente permutación va a empezar con 245.

Cuarta etapa- Poner los números restantes a continuación de los que ya tenemos, en orden creciente. En ese orden para que la permutación sea la más pequeña posible con el inicio ya determinado y sea la que sigue a la original, sin omitir ninguna intermedia.
En el caso de la permutación 1234 ya tenemos 124, el único número que falta es el 3. De manera que la siguiente permutación es 1243. En el segundo ejemplo ya sabemos que la siguiente permutación empieza con 245, los restantes son 3, 7, 6 y 1, así que la siguiente permutación es 2451367.

Se detecta la última permutación posible cuando no podemos localizar el número que buscamos en la primera etapa, cuando no hay ningún número con números mayores que él a la derecha. Esto ocurre cuando todos los números están en orden decreciente 4321 es la última permutación de 1234.

Es importante notar que el algoritmo *siguientepermutacion* está generando la permutación que sigue, sin omitir ninguna.

La permutación que estamos generando es mayor que la que recibimos, donde "mayor" significa que aparece después en el orden que hemos considerado. Esto resulta de que, en la segunda etapa, estamos sustituyendo el número localizado en la primera etapa por un número mayor que él y que todos los números a la izquierda de éste permanecen sin cambio.

Además, de todas las posibles permutaciones mayores que la que recibimos estamos generando la más pequeña. Esto se debe a los siguientes razonamientos, que ilustraremos con la permutación p = 2437651.

En primer lugar, veremos que si una permutación no es la mayor de todas, debe tener algún dígito que tiene otro mayor a su derecha. Consideremos una permutación q mayor que p, por ejemplo q = 2513456 > p = 2437651. En general, si q > p, en el primer lugar de izquierda a derecha en el que p y q difieren (diremos que en este lugar está la **primera diferencia** entre p y q) el dígito d de q es mayor que el de p (en el ejemplo, 5 > 4). Como d no aparece en p ni en el lugar de la primera diferencia ni antes, entonces d debe estar a la

derecha (en el ejemplo, 5 está a la derecha de 4 en p). En particular, si los dígitos de una permutación decrecen todos de izquierda a derecha, no habrá una mayor.
Fijemos ahora una permutación que no sea la mayor (en nuestro ejemplo, p = 2437651). Queremos ver cuál es la permutación s que sigue inmediatamente a p en nuestro orden; claramente s es la menor de las permutaciones mayores que p.

Consideremos ahora dos permutaciones q y r, ambas mayores que p, pero tales que la primera diferencia de q y p está más a la derecha que la de r y p. Entonces q < r.

Por ejemplo, si q = 2465371 > p = 2437651 y r = 2537164 > p = 2437651, y claramente r > q. Para la permutación s (la siguiente de p) esto nos dice que la primera diferencia entre s y p debe estar lo más a la derecha posible.

Supongamos ahora que las dos permutaciones q y r, ambas mayores que p, tienen sus primeras diferencias con p en el mismo lugar, por ejemplo q = 2457316 > p = 2437651 y r = 2461537. Claramente, en este caso, la menor de q y r es aquélla que tenga un dígito menor en este lugar de la primera diferencia. En nuestro ejemplo, como 5 < 6, entonces q < r. Esto nos dice que si en el lugar de la primera diferencia de s y p, la permutación p tiene el dígito d, entonces s debe tener en ese lugar el menor dígito posible que sea mayor que d, y éste es claramente el menor de los dígitos mayores que d que aparecen en p a la derecha de d. En el ejemplo con p = 2437651, el 3 es el dígito más a la derecha que tiene uno mayor a su derecha, así que s debe empezar por 24, y en el tercer lugar debe tener el menor de los dígitos mayores que 3 que estén a su derecha, que es el 5 (nótese que el 4 no, pues está a la izquierda). Entonces s debe comenzar con 245.
Finalmente, ya que sabemos cuáles dígitos debe tener s hasta su primera diferencia con p, necesitamos encontrar la menor de las permutaciones que tengan ese inicio dado, y ésta claramente es la que tenga a continuación los dígitos restantes en orden creciente. En el ejemplo ya sabíamos que s debe comenzar con 245, los dígitos restantes son 3, 7, 6 y 1, así que s = 2451367.

En el algoritmo *siguientepermutacion* utilizamos los elementos siguientes:

1.- La variable *indicp* con la que indicamos si existe otra permutación.

2.- La variable *im* representa la posición, en la cadena, del número que vamos a comparar.

3.-La variable *inb* donde representaremos la posición, en la cadena, del número que buscamos.

4.- La variable *nb* donde representaremos el número que buscamos.

5.- El vector *npermutacion* donde representaremos cada permutación.

6.- El vector *npt* donde representaremos los números a la derecha de *nb*.

7.- La variable *nm* en la que se representará el primer número de derecha a izquierda que tenga un número mayor que él a su derecha.

El algoritmo *siguientepermutacion* es el siguiente:

Inicializamos las variables

indicp = 1 indica que existe otra permutación.

im = *nv* − 1 El número con el cual vamos a empezar a comparar se encuentra en la penúltima posición, de derecha a izquierda, de la cadena.

En la primera etapa del algoritmo *siguientepermutacion*, localizamos el primer número *nm*, de derecha a izquierda, que tenga a su derecha un número mayor que él.

Almacenamos también en *nb* el menor de los números mayores que *nm* que estén a su derecha. Esta primera etapa consiste del ciclo siguiente:

ciclop: *nm* = *npermutacion*(*im*).

inb = 0 ya que todavía no hemos encontrado ningún número mayor que *im*.

nb = *nv* + 1 de entrada, un valor mayor que cualquiera de la cadena.

Comparamos si el número que está en la posición *im*, es menor que cualquiera de los números que se encuentran a la derecha de él.

Para i = *im*+1 **hasta** *nv*.

Si *npermutacion*(**i**) > *nm* **y** *npermutacion*(**i**) < *nb* **entonces**

inb = **i** (*npermutacion*(**i**) es un número mayor y a la derecha que *im*.)

nb = *npermutacion*(**i**).

Notemos que en la variable *nb* queda representado el número más pequeño de los mayores que se encuentran a la derecha de *nm*.

Si no existe un número mayor a la derecha de *im*, esto es,

Si *inb* = 0 **entonces**

a) Si ya se terminó de recorrer la cadena, en cuyo caso no existe otra permutación de los números, hemos terminado el algoritmo, esto es:

Si *im* = 1 **entonces** *indicp* = 0 **Fin**.

b) Si aún existen números por comparar, *im* va a ser la siguiente posición a la izquierda y repetimos ***ciclop***.

im = *im* – 1

vc a *ciclop*

En la segunda etapa colocamos el número encontrado en la primera etapa, *nb*, en la posición *im* de la cadena. También representamos los números que están a la derecha de *im* en el vector *npt*.

Para i = *im* **hasta** *nv*

npt(**i**) = *npermutacion*(**i**)

npermutacion(*im*) = *nb*.

npt(*inb*) = *nv* + 1.

En la tercera etapa colocamos, en la cadena, los dígitos restantes en orden decreciente.

Para i = *im* + 1 **hasta** *nv*

ns = *nv* + 1

Para j = *im* hasta *nv*

Si *npt*(**j**) < *ns* **entonces**

ns = *npt*(**j**)

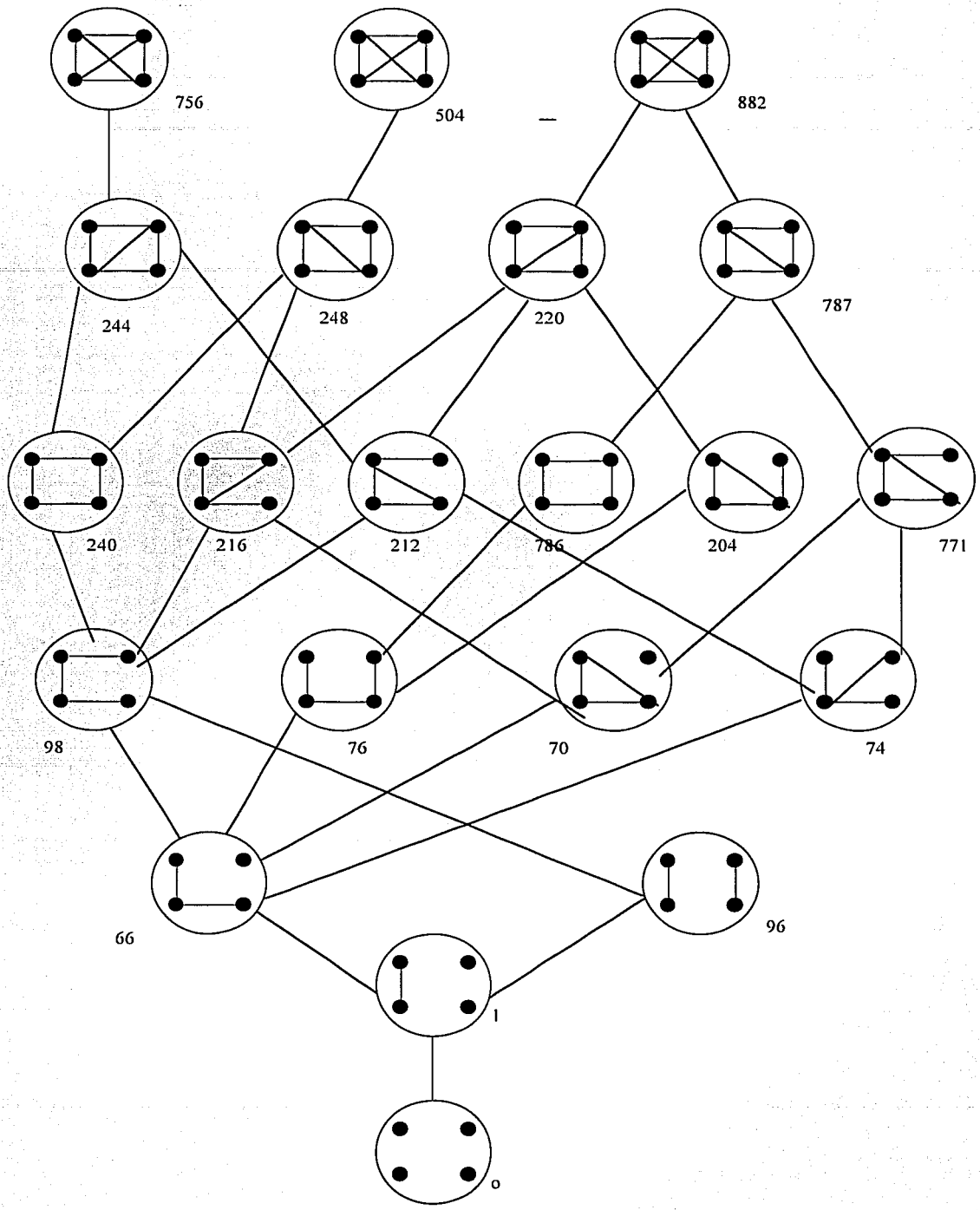
ins = **j**

npermutacion(**i**) = *ns*

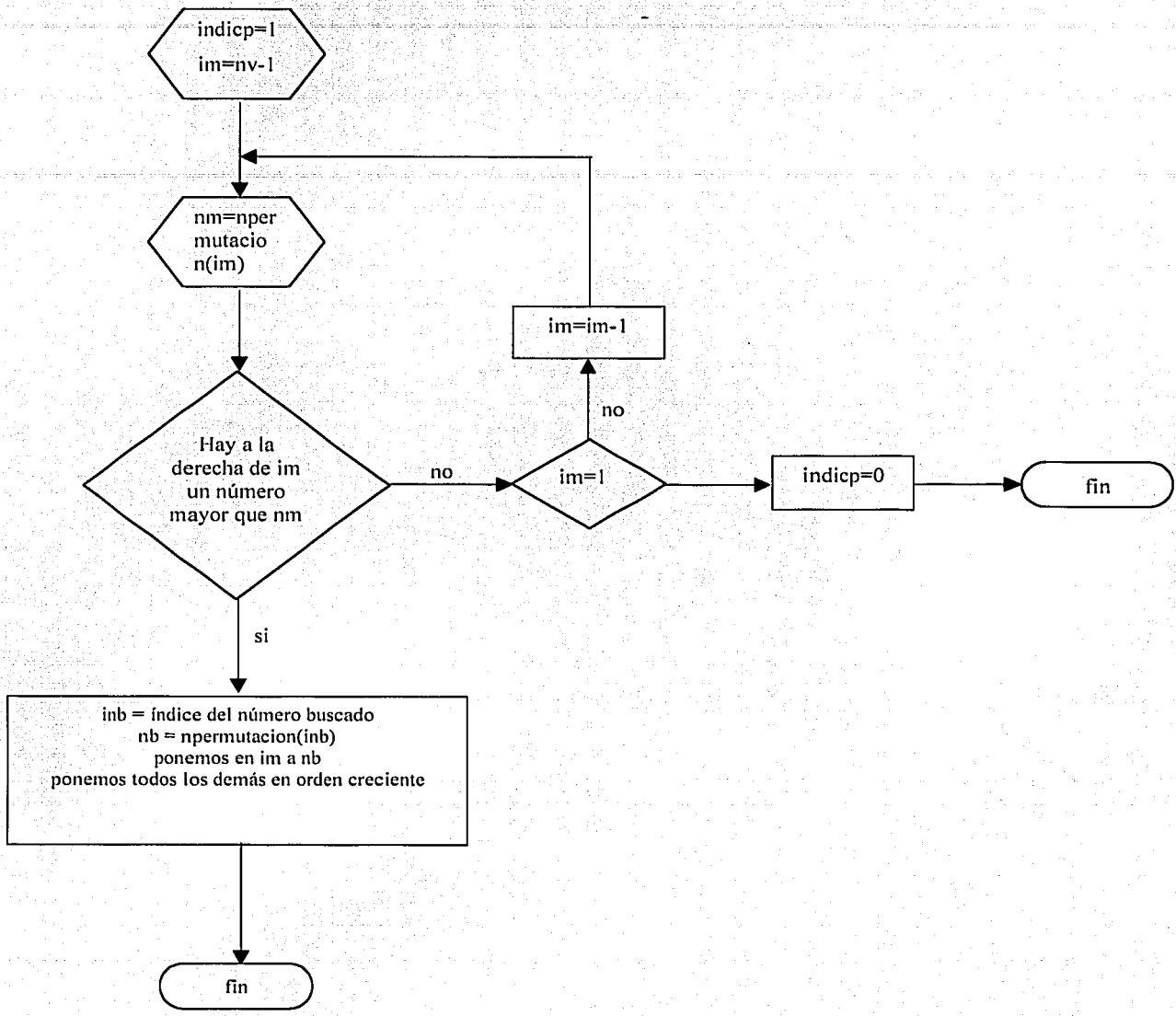
npt(*ins*) = *nv* + 1

Con esto termina el algoritmo *siguientepermutacion.*

Con los algoritmos que desarrollado en este capítulo "podamos" el árbol del juego de Sim para formar así la gráfica dirigida. En el ejemplo de cuatro puntos esta gráfica se muestra en la figura siguiente.


756
504
882
244
248
220
787
240
216
212
786
204
771
98
76
70
74
66
96
1
o

El diagrama de flujo del algoritmo que realiza las permutaciones es el siguiente:



CAPÍTULO III

Gráfica dirigida del juego de Sim

El propósito de este capítulo es desarrollar los algoritmos necesarios para representar en la computadora la gráfica del juego de Sim.

III.1 Representación de la gráfica dirigida.

Para desarrollar el algoritmo que construye la gráfica dirigida del juego de Sim, necesitamos primero una forma de almacenar los nodos de la gráfica en la computadora.
Para esto, utilizaremos un vector, al que llamaremos *qconfigs*. En cada entrada de este vector representaremos, a través del número característico, los nodos que forman esta gráfica, es decir, *qconfigs* será una arreglo de números, donde cada uno de ellos representa un nodo.
Para saber a qué nivel de la gráfica pertenece cada uno de estos nodos, utilizaremos otro vector, de dimensión igual al número de líneas de la gráfica completa del juego más dos (*nl* +2), al que denominamos *iniconfigs*, en él señalaremos la entrada de *qconfigs* (vector de nodos) donde empieza cada nivel de la gráfica dirigida del juego de Sim.
Por ejemplo, en el juego de cuatro puntos los vectores *qconfigs* e *iniconfigs* son los siguientes:

qconfigs:

0	1	66	96	98	76	70	74	240	216	212	786	204	771	244	248	220
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

787	756	504	882	
18	19	20	21	22

iniconfigs:

Con el vector *iniconfigs* sabremos donde empieza y termina cada nivel de la gráfica,
iniconfigs(1) = 1, el primer nivel de la gráfica empieza en *qconfigs*(1),
iniconfigs(2) = 2, el segundo nivel de la gráfica empieza en *qconfigs*(2),
iniconfigs(3) = 3, el tercer nivel de la gráfica empieza en *qconfigs*(3),
iniconfigs(4) = 5, el cuarto nivel de la gráfica empieza en *qconfigs*(5),
iniconfigs(7) = 19, el séptimo (último) nivel empieza en *qconfigs*(19),
iniconfigs(8) = 22, este nivel no existe, sólo lo utilizamos para saber cuántos nodos tiene el último nivel (en este caso 3).

Como vimos en el Capítulo II, el primer nivel de la gráfica dirigida corresponde al nodo raíz (el nodo donde aún no se ha realizado ninguna jugada), el número característico de este nodo siempre es igual a cero, por lo que *qconfigs*(1) = 0.
De la misma manera *iniconfigs*(1) = 1 e *iniconfigs*(2) = 2, ya que el primer nivel de la gráfica siempre empieza en *qconfigs*(1) y el segundo en *qconfigs*(2).
A partir del nodo raíz, *qconfigs*(1) = 0, vamos a generar el segundo nivel de la gráfica, y así sucesivamente hasta llegar al último nivel.

III.2 Algoritmo para construir la gráfica dirigida del juego de Sim.

El algoritmo que construye la gráfica dirigida del juego de Sim recibe el nombre, en el programa, de *hazarbol*. Este algoritmo está formado de varios algoritmos, primero explicaremos en qué consiste *hazarbol* y después lo haremos para cada uno de los algoritmos que lo componen.

Para ilustrarlo, utilizaremos nuevamente el ejemplo del juego de cuatro puntos. Supongamos que hemos construido la gráfica de este juego hasta el tercer nivel, que el turno es del primer jugador, *nj* = 1, y que el nodo que expandiremos es el que tiene el número característico igual a 66.

La gráfica hasta el tercer nivel y los números característicos de los nodos que la forman están representados en la Fig. III.1.

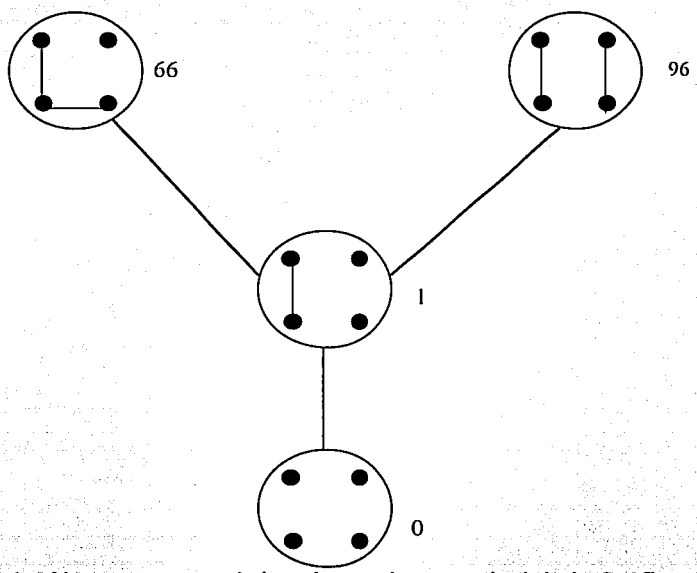


Figura III.1. Números característicos hasta el tercer nivel de la Gráfica dirigida

Los vectores *qconfigs* e *iniconfigs* hasta el tercer nivel son las siguientes:

qconfigs

0	1	66	96

iniconfigs

1	2	3	5

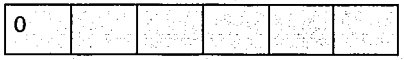
En esta sección construiremos el algoritmo *hazarbol*, el cual consiste de cinco etapas, también daremos el diagrama de flujo de este algoritmo. Los algoritmos que utiliza *hazarbol* se explican en las siguientes secciones.

La primera etapa del algoritmo *hazarbol* inicializa las siguientes variables:

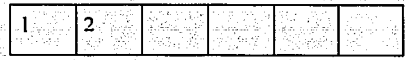
qconfigs(1) = 0	Número característico del nodo raíz.
iniconfigs(1) = 1	Lugar de *qconfigs* donde empieza el primer nivel de la gráfica.
iniconfigs(2) = 2	Lugar de *qconfigs* donde empieza el segundo nivel de la gráfica.
iconfigu = 1	Variable que índica último lugar, hasta el momento, de *qconfigs* donde hay registrado un número característico.
nivel = 1	Variable que índica el último nivel construido de la gráfica.
nj = 1	Variable que representa el número de jugador en turno.
njb = 2	Variable que representa el jugador que no tira.

Con los valores iniciales de los vectores *qconfigs* e *iniconfigs* hemos construido el primer nivel de la gráfica, hasta aquí, estos dos vectores son los siguientes:

qconfigs:



iniconfigs:



En la segunda etapa del algoritmo se construye un nuevo nivel de la gráfica. Esto consiste únicamente en actualizar una serie de variables, que indican cuál es el último nivel de la gráfica que se construyó y el que se comenzará a construir. Está etapa consiste de un bloque al que llamaremos *nivelnuevo*.

nivelnuevo:

iconfig1 = *iniconfigs*(*nivel*)	Primer índice del nivel ya terminado
iconfig2 = *iniconfigs*(*nivel* +1) – 1	Último índice del nivel terminado
iconfigt = *iconfig1*	Índice del nodo padre
iconfigi = *iconfig2* + 1	Índice del primer nodo del siguiente nivel.

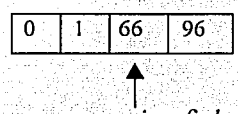
En el ejemplo del juego con cuatro puntos, supusimos que hemos construido hasta el tercer nivel de la gráfica. Por lo tanto, las variables anteriores son las siguientes:

iconfig1 = *iniconfigs*(*nivel*).

iconfig1 = *iniconfigs*(3) = 3.

El primer índice del nivel ya terminado, es decir el índice del primer nodo del tercer nivel, lo localizamos en *iniconfigs*(3)=3.

qconfigs:

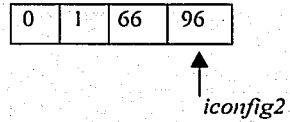


iconfig2 = *iniconfigs*(*nivel* + 1) – 1.

iconfigs2 = *iniconfigs*(3 + 1) – 1 = 4.

El último índice del nivel que está terminado se encuentra un índice menos donde empezará el siguiente nivel. Entonces el nivel 4 empezará en iniconfigs(4), el último nodo del nivel 3, está representado en iniconfigs(4) –1 = 4

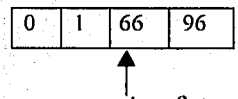
qconfigs:



iconfigt = *iconfig1*

iconfigt = 3, El índice del nodo que vamos a expandir para construir el cuarto nivel.

qconfigs:

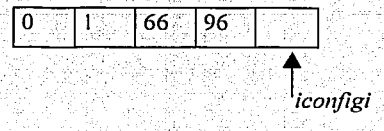


iconfigi = *iconfig2* + 1.

iconfigi = 4 + 1 = 5.

El índice donde colocaremos el primer nodo del siguiente nivel, será uno más de donde termina el tercer nivel.

qconfigs:



La tercera etapa consiste en expandir el nodo padre. Cada nodo lo vamos a expandir a partir de su número característico; esta etapa consiste de siete pasos, los cuales están agrupados en un bloque al que llamaremos *confignueva*.

confignueva:

a) Tomamos el número característico del nodo que deseamos expandir,

qnc = *qconfigs(iconfigi)*

En el ejemplo, para construir el cuarto nivel de la gráfica, tomamos el primer nodo del tercer nivel. *qnc* = *iconfigs*(*iconfigi*) = *iconfig*(3) = 66.

b) Con el algoritmo *nc2edot*, a partir del número característico del nodo padre, generamos la matriz *estado* que le corresponde a este nodo. (Este algoritmo se explicará más adelante).

gosub *nc2edot*

En el ejemplo el nodo padre es:

nc = *configs(iconfigi)* = *iconfigs*(3) = 66.

La matriz *estado* que le corresponde a este número es:

2			
1	0		
0	0	0	

c) Con el algoritmo, *buscatriangulo*, (este algoritmo se explicará más adelante) revisamos si el nodo padre es un estado objetivo del jugador en turno, esto es, si en la matriz *estado* está representado un triángulo del color del jugador contrario al jugador en turno. Para buscar triángulos del color contrario al jugador en turno es necesario cambiar la variable *nj*:

Si *nj* = 1 **entonces** *nj* = 2 **si no** *nj* = 1.
Gosub *buscatriangulo*

La variable *nj* toma el valor que tenía originalmente.

Si *nj* = 1 **entonces** *nj* = 2 **si no** *nj* = 1.

d) En caso de que en la matriz **estado** del inciso anterior esté representado un estado objetivo para el jugador *nj* lo señalamos en el vector *ncalificacion*, con 32000 ó 0 si el triángulo es de color rojo o azul (Esto, al igual que la variable *ncalt*, se explicará en el Capítulo IV.) Además, en este caso, *buscatriangulo* regresa la variable *ntri* con un valor mayor que cero, y ésta será la señal de que el nodo en turno es un estado objetivo y no tenemos que expandirlo. En este caso buscamos el siguiente nodo de este nivel (pasando a la cuarta etapa).

Si *nj*=1 **entonces** *ncalt* = 32000 **si no** *ncalt*=0.
Si *ntri* > 0 **entonces** *ncalificacion*(*iconfigt*) = *ncalt*
Goto siguienteconfig

En el ejemplo, este nodo no corresponde a un estado objetivo, ya que no contiene un triángulo, por lo que pasamos al siguiente punto.

e) Con el algoritmo *edo2lineam*, a partir de la matriz *estado*, del inciso c), generamos el vector *lineam* que corresponde al nodo padre. (Este algoritmo se explicará más adelante).

Gosub *edo12lineam*

En el ejemplo, el vector *lineam* que corresponde a *estado* es:

2	1	0	0	0	0

f) Una vez generado el vector *lineam*, con el algoritmo *generahijas* expandimos el nodo padre.

Gosub *generahijas*

En el ejemplo, la matriz *lineah* (donde representamos los nodos generados), es la siguiente:

2	1	1	0	0	0
2	1	0	1	0	0
2	1	0	0	1	0
2	1	0	0	0	1

Y $nh = 4$ (número de nodos generados).

g) Con el algoritmo *sacanumerosdiferentes*, calculamos el número característico de cada nodo y descartamos las repeticiones, es decir, aquellos nodos cuyo número característico es igual a otro ya generado. Los números característicos diferentes los representamos en un vector *ncdh.* (este algoritmo se explicará más adelante).

Gosub *sacanumerosdiferentes*

En el ejemplo que estamos siguiendo, ningún nodo es equivalente a otro, *ncdh* es el siguiente vector:

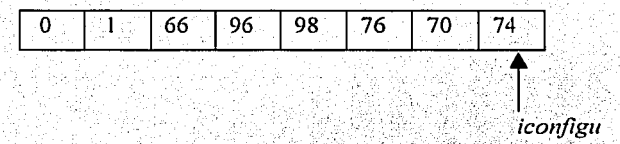
98	76	70	74

h) En *ncdh* están representados los nodos generados, de un nodo padre, que no son equivalentes entre sí. De estos nodos, con el algoritmo *agregancsnuevos*, sólo agregaremos a la gráfica del juego (al vector *qconfigs*) aquellos que no existan previamente en ella. (Este algoritmo se explicará más adelante).

Gosub *agregancsnuevos*

En el ejemplo que hemos seguido ninguno de los nodos representados en *ncdh* existen en *qconfigs*, por lo que agregamos todos e *iconfigu* = 8, pues ya se han usado ocho entradas de *qconfigs*.

qconfigs



Con el proceso anterior sólo hemos expandimos un nodo. La cuarta etapa, el bloque *siguienteconfig*, consiste en verificar si existe otro nodo en el nivel que estamos expandiendo. Para esto, revisamos si el índice del nodo en turno (*iconfigt*) es menor que el índice del último nodo del nivel que estamos expandiendo (*iconfig2)*. Si lo es, quiere decir que todavía hay más nodos por expandir, por lo que tomamos el siguiente nodo como nodo padre, *iconfigt* = *iconfigt* + 1, y repetimos el bloque **confignueva.**

siguienteconfig:
Si *iconfigt* < *iconfig2* **entonces**
iconfigt = *iconfigt* + 1
gosub confignueva

Si en el nivel donde nos encontramos, ya no existe otro nodo que expandir, actualizamos el vector *iniconfigs*.

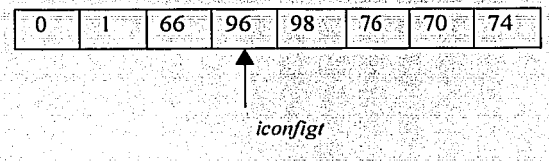
iniconfigs(*nivel* + 2) = *iconfigu* + 1

En el ejemplo,

iconfigt = 3.

iconfig2 = 4.

Como *iconfigt* < *iconfig2*, significa que existe otro nodo por expandir en el nivel 3, por lo que *iconfigt* = 4 y repetimos el procedimiento el bloque **confignueva.**



En la quinta etapa revisamos si existe otro nivel de la gráfica por construir, esto es, si *nivel* es igual al número de aristas del juego (*nl*), en este caso, habremos terminado de construir la gráfica del juego.

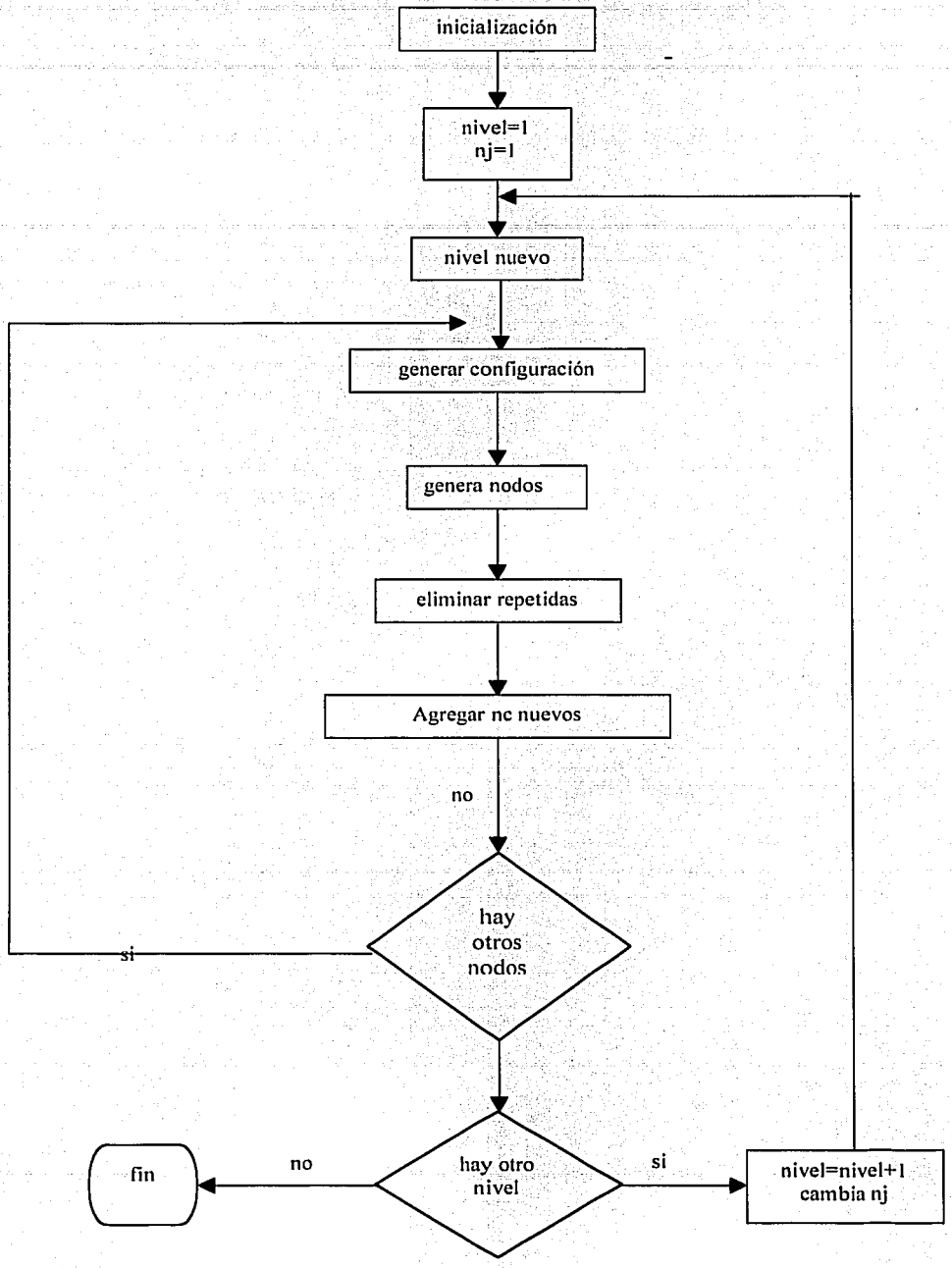
Si *nivel* = *nl* **entonces return.**

En el caso contrario incrementamos la variable *nivel* en 1 y cambiamos el número del jugador en turno, es decir, si *nj* = 1, ahora *nj* = 2 y viceversa, y repetimos el proceso a partir del bloque **nivelnuevo.**

nivel = *nivel* + 1
njt = *nj*: *nj* = *njb*: *njb*: *njt*
go to nivelnuevo.

Con el algoritmo anterior hemos construido la gráfica dirigida del juego de Sim.

DIAGRAMA DE FLUJO PARA CONSTRUIR LA GRÁFICA DIRIGIDA DEL JUEGO DE SIM



A continuación vamos a explicar cada uno de los programas que forman el algoritmo *hazarbol*.

III.3 Cómo generar la matriz *estado* a partir del número característico.

Por la forma en que construimos el número característico de un estado, sabemos que la primera mitad de los dígitos, de derecha a izquierda, de este número, corresponden a las aristas trazadas por el primer jugador. Así mismo, en la segunda mitad los dígitos corresponden a las aristas trazadas por el segundo jugador.

Para construir la matriz *estado* a partir del número característico, primero buscaremos las aristas trazadas por el primer jugador. Para esto, revisaremos si el primer dígito en base 2, de derecha a izquierda, del número característico es uno, si lo es, entonces en el lugar de *estado*, que le corresponde a la línea 1, colocaremos un 1, el caso contrario significa que la arista 1 no ha sido trazada por el primer jugador, por lo que colocamos un cero en la entrada que corresponde a la arista 1 en la matriz *estado*. Este procedimiento lo repetimos tantas veces como número de líneas tenga la gráfica completa del juego y con esto habremos representado en *estado* las aristas trazadas por el primer jugador.

Las aristas trazadas por el segundo jugador las localizamos a partir de la segunda mitad, de derecha a izquierda, de los dígitos del número característico. Para lo cual verificaremos si el primer dígito de la segunda mitad es 1, si lo es, entonces colocamos un dos en el lugar de *estado* que le corresponde a la arista 1, en el caso contrario colocamos un cero. Otra vez, repetimos este procedimiento tantas veces como número de líneas tenga la gráfica completa del juego y con esto habremos representado en *estado* las jugadas realizadas por el segundo jugador.

En el algoritmo, *nc2edo*, que genera la matriz *estado*, a partir del número característico, para determinar los dígitos binarios iguales a uno usaremos la función lógica AND. Esta función asigna los siguientes valores:

1 AND 1 = 1

1 AND 0 = 0

0 AND 1 = 0

0 AND 0 = 0

La forma como utilizaremos la función AND es la siguiente:

Para verificar si el dígito *i* del número característico es 1 aplicaremos la función AND a *nc* y 2^i. Recordemos que el número característico de un estado lo construimos en base dos. Si *nc* AND $2^i = 1$, sabremos que la arista $i + 1$ fue trazada por el jugador en turno (*nj*), y si *nc* AND $2^i = 0$, la arista $i + 1$ no fue trazada por el jugador *nj*.

Por ejemplo, supongamos que le aplicamos la función AND al siguiente número característico

nc = 100000001000

AND 2^0 = 000000000001

= 000000000000

Como el único 1 en 2^0 es el primer dígito, de derecha a izquierda, entonces *nc* AND 2^0 es igual a 1, si y sólo si el primer dígito de nc es 1 (ya que es el único caso en se daría la situación de 1 AND 1).

En el algoritmo *nc2edo* utilizamos los siguientes elementos:

1.-La variable *nl* para representar el número de aristas de la gráfica completa del juego.

2.-Los vectores *lir* y *lic* definidos en el Capítulo II sección 9.

El algoritmo *nc2edo*, consiste de dos etapas. En la primera ponemos ceros en cada entrada de *estado* y buscamos las aristas trazadas por el primer jugador. Para esto aplicamos la función AND a *nc*.

Para i = 1 **hasta** *nl*

nr = *nc* AND 2^{i-1}

Si *nr* ≠ 0 **entonces**

estado(*lir*(**i**), *lic*(**i**)) = 1.

En la segunda etapa, buscamos las aristas trazadas por el segundo jugador en los dígitos, de la segunda mitad del número característico provisional. El primer dígito de esta segunda mitad se encuentra en la posición 2^{nl}.

Para i = 1 **hasta** *nl*

$nr = nc$ AND (2^{nl+i-1})

Si $nr \neq 0$ **entonces**

estado(*lir*(**i**), *lic*(**i**)) = 2

Con este algoritmo hemos generado la matriz *estado* a partir del *número característico* de un nodo

III.4 Cómo encontrar un estado objetivo

En el algoritmo, *buscatriangulo*, para identificar si un estado es objetivo, es decir, si en un estado hay un triángulo rojo o azul utilizamos la matriz *estado*.
Recordemos que la matriz *estado* es una matriz simétrica, donde representamos la arista que une al vértice i con el vértice j en *estado*(i, j), con 1 ó 2, según el jugador que haya trazado la arista [i, j]. En el algoritmo *buscatriangulo*, también señalaremos con 1 ó 2 los lugares simétricos de la matriz, así, si *estado*(i, j) = 1, entonces *estado*(j, i) = 1.

Para ilustrar las condiciones que debe cumplir un estado objetivo en la matriz *estado*, consideremos la Figura III.2 y su matriz asociada

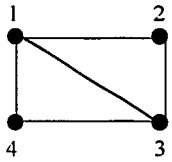


Fig. III.2. Estado del juego de cuatro puntos.

0	1	1	2
1	0	1	0
1	1	0	2
2	0	2	0

Existe una arista del primer jugador que une al vértice 1 y al vértice 2, representada en *estado*(1, 2) = 1, para que esta línea forme un triángulo debe existir una arista roja del vértice 2 a un vértice cualquiera k ≠ 1, por lo que en este caso *estado*(2, k) = 1. A su vez, debe haber una tercera arista roja que una el vértice k con el vértice 1, representada en *estado* de (k, 1) = 1.

En el ejemplo *estado* (1, 2) = 1, *estado* (2, 3) = 1 y *estado*(3, 1) = 1, en este caso k = 3, entonces podemos determinar que la matriz *estado* de la Fig. III.2 es un estado objetivo, ya que en ella hay representado un triángulo que une a los vértices 1, 2, 3.
En general, las tres condiciones que acabamos de establecer son representadas en la matriz *estado* de la siguiente manera, donde i, j y k son tres vértices distintos:

1.- *estado*(i,j) = *nj*

2.- *estado*(j,k) = *nj*

3.- *estado*(k,i) = *nj*.

En el algoritmo *buscatriangulo*, con el que determinamos si una matriz **estado** es estado objetivo con triángulo del jugador *nj*, (*nj* = 1 ó 2), necesitamos:

1.-La matriz *estado* asociada al nodo de la gráfica donde se desea determinar si existe un estado objetivo.

2.-La variable *nj* para identificar al jugador

3.-La matriz *triangulos* donde representaremos los vértices de los triángulos encontrados.

4.- Una variable *ntri* con la cual contaremos el número de triángulos.

5.- La variable *nv* que indica el número de vértices del juego.

El algoritmo que determina si un estado es objetivo es el siguiente:

Completamos la matriz *estado*, copiando el valor de *estado*(i, j) a *estado*(j, i) para todo par $0 < i < j < nv + 1$

Como aún no hemos encontrado ningún triángulo *ntri* = 0.

Recorremos los renglones y columnas de la matriz *estado* hasta que se encuentre una arista trazada por el jugador *nj*. Recordemos que la diagonal principal de esta matriz siempre es cero, por lo que empezamos en la segunda columna del primer renglón. Con la variable i recorreremos los renglones, y con la variable j recorremos las columnas.

Para i = 1 **hasta** *nv*

Para j = i+1 **hasta** *nv*

Si *estado*(i, j) = *nj*, **entonces**

Revisamos que se cumpla la segunda condición, si la arista [i, k] ha sido trazada por el jugador *nj*. Por lo que

Para k = j + 1 **hasta** *nv*

Si *estado*(i, k) = *nj* **entonces**

Revisamos que se cumpla la tercera condición, si la arista [k, j] ha sido trazada por el jugador *nj*.

Si *estado*(k, j) = *nj* **entonces**

Hay un triángulo por lo que

ntri = *ntri*+1.

Representamos en la matriz *triangulos* los vértices del triángulo encontrado:

triangulos(*ntri*, 1) = i

triangulos(*ntri*, 2) = j

triangulos(*ntri*, 3) = k

Con este algoritmo podemos determinar si en un estado existe un triángulo rojo azul.

III.5 Cómo pasar de la matriz *estado* a *lineam*.

Para generar el vector *lineam* de un nodo a partir de su matriz *estado*, lo que haremos es localizar en *estado* cada arista de la gráfica completa del juego, y representarla en el lugar que le corresponde en *linea*.

Por ejemplo: para la siguiente matriz *estado*

0			
0	0		
1	0	2	

Como *estado*(1,1) no existe, ya que no hay ninguna arista del vértice 1 al 1, empezamos localizando la arista que esta representada en *estado*(2, 1) = 0, es decir la arista que va del vértice 2 al 1, en este caso, la arista 1. Entonces *lineam*(1) = 0.

Como sólo estamos copiando *estado*(i, j) para i < j, pasamos al siguiente renglón, es decir a la arista representada en *estado*(3, 1) = 0. La arista que va del vértice 3 al 1, es la número dos. Por lo que *lineam*(2) = 0. Este procedimiento lo repetimos hasta llenar el vector *lineam*. En el ejemplo, la gráfica tiene cuatro vértices y *lineam* es el siguiente:

0	0	0	1	0	2

El algoritmo que genera el vector *lineam* a partir de la matriz *estado* lo llamaremos *edo2lineam*. En este algoritmo necesitamos las siguientes variables:

1. *nv* número de vértices de la gráfica del juego.
2. *nr* para recorrer renglones de *estado*. Recorreremos a partir del renglón 2 hasta nv
3. *nc* para recorrer las columnas de *estado*. El recorrido de las columnas será desde 1 hasta *nr*−1, ya que leeremos sólo *estado*(i, j) con i > j.
4. *nl* donde asignaremos el número de la arista localizada.

El algoritmo *edo2lineam* es el siguiente:

Localizamos la arista que está representada en cada entrada debajo de la diagonal de *estado* y copiamos su valor en *lineam*. Para lo cual empezamos en el segundo renglón

Para $nr = 2$ **hasta** nv

Para $nc = 1$ **hasta** $nr - 1$

$nl = localizacion(nr, nc)$

$lineam(nl) = estado(nr, nc)$

Con este algoritmo hemos generado el vector *lineam* a partir de la matriz *estado*,

III.6 Cómo eliminar nodos repetidos.

Una vez que los nodos descendientes de un nodo padre están representados en la matriz *lineah* con el algoritmo *sacanumerosdiferentes* generamos el número característico de cada uno y eliminamos aquellos nodos que son equivalentes. Esto lo hacemos de la siguiente manera:

Consideremos la matriz *lineah* siguiente:

1	0	0	1	0	2
0	1	0	1	0	2
0	0	1	1	0	2
0	0	0	1	1	2

Cada nodo descendiente, esto es, cada renglón de *lineah*

1. Lo copiamos en una matriz *linea*. Por ejemplo, el primer nodo descendiente representado en *lineah*(1).

linea(1)

1	0	0	1	0	2

2. Generamos la matriz *estado*, correspondiente a *linea*(1).

1			
0	0		
1	0	2	

3. Una vez generada la matriz *estado* que corresponde al primer nodo descendiente, con el algoritmo *varitamagica* calculamos el *número característico* de esta matriz. Para la matriz anterior este número es 98.

Los números característicos de los nodos que no son equivalentes, los representaremos en un vector al que llamaremos *ncdh*

4. Si el número característico generado en el punto anterior ya está representado en la matriz *ncdh* repetimos el proceso anterior para el siguiente nodo descendiente. En el caso contrario, agregamos el *número característico* al vector *ncdh*.

En el ejemplo el vector *ncdh* es el siguiente:

98	76	70	74

Y *nndh* = 4.

Algoritmo *sacanumerosdiferentes*

En el programa para descartar los nodos equivalentes necesitamos:

1.- *ncdh* vector donde representaremos los nodos que no son equivalentes.

2.- *nncd* variable para contar los nodos representados en *ncdh*.

3.- *nl* número de líneas del juego

4.- *nbdif* indica si el número característico ya está en *ncdh*

5.- *nh* esta variable indica el número de nodos generados, la cual fue creada en el algoritmo que expandió el nodo.

El algoritmo es el siguiente:

Como aún no hay ningún nodo en *ncdh*. Entonces

nncd = 0

Para cada uno de los renglones y columnas de *lineah*, esto es:

Para *nht* = 1 **hasta** *nh*

Para ii = 1 **hasta** *nl*

La copiamos en el vector *linea*

linea(ii) = *lineah*(*nht, ii)*

Generamos la matriz *estado* correspondiente al vector *linea*(*nht*).

GOSUB *linea2edo*

Calculamos el número característico de la matriz *estado* que acabamos de generar.

GOSUB *varitamagica*

Revisamos que *nc* no este en *ncdh*. Para esto, comparamos *nc* con cada uno de los números que están en *ncdh*.

nbdif = 0 (por si *nc* no está en *ncdh*)

Para ii = 1 **hasta** *nncd*

Si *nc* = *ncdh*(ii) **entonces** *nbdif* = 1

Si *nbdif* = 0 significa que *nc* es diferente a los números que están en *ncdh*. En este caso incrementamos en 1 la variable *nncd* (ya que habrá un número más en *ncdh)*, y agregamos *nc* a *ncdh*.

Si *nbdif* = 0 **entonces**

nncd = *nncd* + 1

ncdh(nncd) = *nc*.

Cuando hayamos terminado el procedimiento anterior habremos representados en *ncdh* los nodos que no son equivalentes, generados de un nodo padre. El siguiente paso será agregarlos a la gráfica dirigida del juego.

III.7 Cómo agregar nuevos nodos a la gráfica.

Para agregar a la gráfica del juego los nuevos nodos que están representados en *ncdh*, primero revisaremos que éstos no estén ya en la gráfica dirigida del juego. Por lo que cada nodo que está representado en *ncdh* lo comparamos con cada nodo del nivel que estamos construyendo

En el ejemplo tenemos que:

ncdh

98	76	70	74

iconfigs

0	1	66	96

Para revisar que cada uno de los nodos que están representados en *ncdh* no han sido generados antes hacemos lo siguiente:

1.-*nc* es igual al primer nodo representado en *ncdh.*
En el ejemplo *nc* = 98.

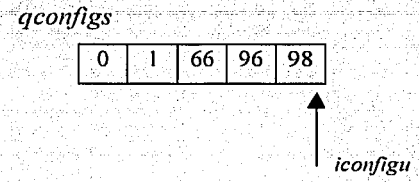
2.-Revisamos que este número no esté ya representado en la gráfica del juego, para esto, revisamos a partir del primer, hasta el último nodo del nivel que estamos construyendo, es decir desde *iconfigi* (índice del primer nodo) hasta *iconfigu* (índice del último nodo colocado en *qconfigs*).

En el ejemplo *iconfigi* = 5 e *iconfigu* = 4

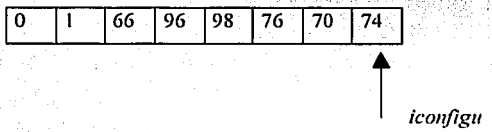
3.-Si ya está, tomamos el siguiente número del vector *ncdh* y repetimos el procedimiento anterior tantas veces como nodos representados haya en *ncdh,* es decir tantas veces como el valor de *nncd.*

Si no está, incrementamos la variable *iconfigu*, y agregamos *nc* a *qconfigs* en la entrada *iconfigu*.

En el caso del ejemplo que estamos siguiendo, *nc* no está en *iconfigs*, por lo que hacemos *iconfigu* = 5, e *qconfigs* queda así:



4.- Tomamos el siguiente número en *ncdh* y repetimos el procedimiento anterior tantas veces como el número *nncd* de nodos representados en *ncdh*.
En el ejemplo, ninguno de los nodos generados están en *qconfigs*, por lo que agregamos todos. Cuando hayamos terminado el proceso anterior *qconfigs* será el siguiente vector:



Algoritmo *agregancsnuevos* para agregar nuevos nodos a la gráfica.

En este algoritmo necesitamos:

1.- El vector *ncdh* donde están representados los nodos candidatos a agregarse a la gráfica dirigida del juego de Sim.

Las variables:

2.- *nncdh* que indica el número de nodos representados en *ncdh*.

3 - *iconfigi* el primer índice del nodo del nivel que estamos construyendo.

4.- *iconfigu* el índice del último nodo representado en la gráfica.

El algoritmo es el siguiente:

Para cada nodo representado en *ncdh* hacemos lo siguiente

Para i = 1 **hasta** *nncd*

nc = *ncdh*(i)

Comparamos *nc* con cada nodo del nivel que estamos formado

nbdif = 0 (por si no está)

Para j = *iconfigi* **hasta** *iconfigu*

Si *nc* = *iconfigs*(j) **entonces** *nbdif* = 1.

Si *nc* no está representado en *iconfigs*, incrementamos *iconfigu* y agregamos *nc* a *qconfigs*.

Si *nbdif* = 0 **entonces**

iconfigu = *iconfigu* + 1

iconfigs(*iconfigu*) = *nc*.

CAPÍTULO IV

Teoría de juegos.

En los capítulos anteriores, encontramos una forma de representar en la computadora la gráfica dirigida del juego de Sim. En este capítulo veremos cómo deben comportarse los jugadores, para lo cual, analizaremos el juego de Sim desde el punto de vista de la teoría de juegos.

IV.1 Conceptos preliminares.

Desde el punto de vista de la teoría de juegos hay seis elementos esenciales en el juego de Sim.

1.- Es un juego entre dos participantes, a esto se le llama juego **bipersonal.**

2.- Es un juego **antagónico** ya que los jugadores tienen intereses opuestos con respecto al resultado del juego; por consiguiente, en cada jugada, el jugador tiende a asegurarse el beneficio máximo y tiende a que sus contrincantes pierdan al máximo.

3.- El juego es **finito**, ya que cada jugador tiene un número finito de movimientos posibles y el juego termina en un número finito de jugadas.

4.- No hay sorpresas; en cada momento los jugadores están completamente informados sobre la historia del juego, por lo que, se le llama juego de **información perfecta.**

5.- Es un juego en donde lo que gana un jugador lo pierde el otro, a este tipo de juegos se les llama de **suma cero.**

6.- Cada jugador elige su jugada en una forma determinista, pues no hay movimientos al azar.

Definición. 17 Una **estrategia** es una descripción completa de cómo un jugador se comportará bajo cada una de las posibles circunstancias del juego.

Definición. 18 Una **estrategia ganadora** es una estrategia que si se elige garantiza el triunfo independientemente de lo que haga el oponente

El siguiente teorema fue demostrado por Zermelo en 1912, en lo que fue el primer artículo sobre teoría de juegos, en él se basará nuestro análisis del juego de Sim.

Teorema 4 Si en cada momento de un juego bipersonal, finito, de suma cero y con información perfecta, los jugadores eligen su mejor opción, el resultado del juego es siempre el mismo: gana el primer jugador, gana el segundo jugador o el juego termina en empate. A esta propiedad se le llama **determinación estricta.**

En la siguiente sección daremos los argumentos que en esencia prueban este teorema adaptándolos para el juego de Sim.

El teorema de Zermelo establece que desde un principio se puede saber el resultado final del juego si los jugadores eligen siempre su mejor opción. Atendiendo a esto, parecería que jugar este tipo de juegos no tiene sentido. ¿Para qué jugar un juego cuyo resultado está determinado de antemano? Lo que sucede es que en la práctica es difícil y a veces casi imposible saber cuál es la mejor opción de un jugador en cada momento del juego.

Para conocer cuál es la mejor opción de los jugadores en cada momento del juego de Sim, así como su resultado final veremos ahora una forma en que se puede analizar un juego bipersonal, antagónico, de suma cero y de información perfecta.

IV.2 Análisis del juego

Para el análisis del juego de Sim utilizaremos el árbol que construimos en el Capítulo II. Cualquier trayectoria desde el nodo inicial a un nodo terminal define una partida del juego. Hay así tantas partidas como nodos terminales tiene el árbol.
Una partida se desarrolla de la siguiente manera: el jugador al que se le haya asignado el nodo inicial, v_0, elige una arista que incida en v_0 (una opción), la que conduce a otro nodo,

que a su vez está asignado al otro jugador, quien elige una opción dentro de las disponibles y así sucesivamente hasta llegar a un nodo terminal.
En teoría de juegos existen varias formas de representar un juego. En este trabajo usaremos la representación en forma extensiva de un juego.

Definición. 19 La representación en **forma extensiva** de un juego de suma cero, entre dos jugadores con información perfecta y sin movimientos al azar consta de lo siguiente:

1) Un árbol Γ con un nodo raíz al que llamaremos nodo raíz de Γ.
2) Una asignación de los nodos no finales entre los jugadores. Es decir se debe señalar qué nodos corresponden a cada jugador. Todos los nodos no finales deben ser asignados y ningún nodo puede corresponder a más de un jugador. En otras palabras, definimos una partición de los nodos no finales de Γ en dos conjuntos s_1 y s_2 donde $S = s_1 \cup s_2$ es el conjunto de momentos del juego.
3) Una asignación de opciones a cada jugador, estas opciones corresponden a las aristas que parten de cada nodo no terminal.
4) En cada nodo final se asignará una ganancia para cada jugador mediante una función f llamada **función de pagos** (o de **ganancias**) la cual asigna un vector de dos dimensiones a cada nodo final.

En el caso del juego de Sim el árbol lo hemos construido en el capítulo II, donde el conjunto s_1 está formado por los nodos que corresponden al primer jugador, esto es, los nodos que forman los niveles impares del árbol (el nodo raíz, los nodos del tercer nivel, quinto, etc.) y el conjunto s_2 está formado por los nodos que le corresponden al segundo jugador, estos son los nodos que forman los niveles pares del árbol (los nodos del segundo, cuarto, sexto nivel, etc.). Las opciones del primer jugador son las aristas marcadas con rojo y las del segundo jugador las aristas marcadas con azul.

El pago, en los nodos terminales, para cada jugador es 1, 0, ó -1 según gane, empate o pierda respectivamente. Por lo que cada nodo final del árbol tendrá una ganancia para cada jugador. Por ejemplo, en un nodo donde gane el jugador rojo, esto es, en los nodos que exista un triángulo azul, la ganancia para el primer jugador es 1 y para el jugador azul -1,

denotaremos por (1,-1) el vector de ganancia en este caso. En un nodo donde ningún jugador gane, es decir, en aquellos nodos finales donde no exista un triángulo de un solo color, la ganancia para ambos será (0, 0). Y si en un nodo gana el jugador azul, la ganancia para éste será 1 y para rojo -1, denotada por (-1, 1).

Es importante notar, que en el juego de Sim con seis puntos, en los nodos finales no es posible el empate, ya que, por el teorema de Ramsey (Teorema 3 del Capítulo 1) sabemos que en una gráfica completa con seis puntos siempre hay un triángulo rojo o azul.
La forma en que analizaremos y encontraremos una estrategia del juego de Sim es usando el algoritmo de Zermelo, el cual consiste en lo siguiente: 1.- Se define primero la función de pagos en los nodos terminales. 2.- Se define la máxima ganancia posible del jugador en turno en los nodos no terminales del penúltimo nivel. Esto se hace tomando en cuenta, para cada uno de estos nodos, las ganancias en los nodos hijos. 3.- A continuación se hace lo mismo para los nodos del nivel anterior, sólo que ahora se maximiza la ganancia para el otro jugador, pues a él le corresponden los nodos de ese nivel. Continuando de esta manera, se marcha hacia atrás hasta el nodo raíz. Por esta razón a esta técnica se le conoce también como inducción hacia atrás.

En el juego de Sim para determinar la opción que maximiza las ganancias del jugador situado en los nodos que preceden a los finales, utilizamos las tres reglas siguientes.

Regla 1.- Si entre los nodos que descienden de un mismo nodo padre hay al menos una opción que conduzca al jugador situado en ese nodo a un nodo donde su ganancia sea 1, entonces la ganancia de este jugador, en el nodo padre, también es 1.
Por ejemplo, en la Fig. IV.1 se muestra una situación donde al menos existe una opción donde la ganancia del primer jugador (el que está situado en el nodo padre) es uno.

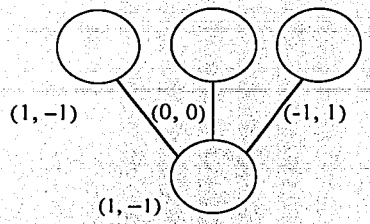


Figura IV.1. Ganancia del jugador en el nodo padre igual a 1.

Regla 2.- Si en los nodos que descienden de un mismo nodo padre, no hay al menos una opción que conduzca al jugador situado en ese nodo a un nodo donde la ganancia sea 1, pero hay al menos una opción que lo conduzca a un nodo donde la ganancia es 0, entonces la máxima ganancia posible del jugador situado en el nodo padre es 0.

Por ejemplo, en la Fig. IV.2 la máxima ganancia para el jugador situado en el nodo padre (el primero), la obtiene en cualquiera de los nodos donde la ganancia de los dos jugadores es 0.

.

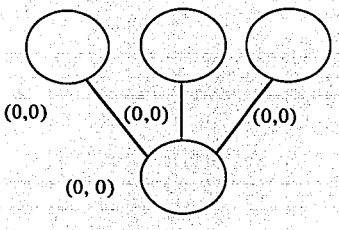


Figura IV.2.Ganancia del jugador en el nodo padre igual a 0.

Regla 3.- Si en los nodos que descienden de un mismo nodo padre, no hay al menos un nodo donde la ganancia del jugador situado en el nodo padre sea 1 o 0, entonces la máxima ganancia posible del jugador situado en el nodo padre, es −1

Por ejemplo, en la Fig. IV.3 todas las opciones para el jugador situado en el nodo padre lo conducen a nodos donde la máxima ganancia posible es −1.

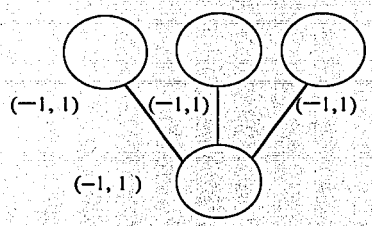


Figura IV.3. Ganancia del jugador en el nodo padre igual a −1.

Aplicando la primera (en el orden dado) que sea posible de las tres reglas anteriores a los nodos que forman el penúltimo nivel del árbol, maximizamos la ganancia del jugador situado en estos nodos. A continuación marcamos cada nodo con la pareja que en la entrada correspondiente al jugador tiene la máxima ganancia m obtenida y, en la otra entrada, su inverso $-m$. Esto último es porque $-m$ es la máxima ganancia a la que podría aspirar el contrincante si el juego llega a ese nodo (recordar que el juego es de suma cero).
Repetimos el procedimiento anterior para maximizar la ganancia del jugador situado en los nodos que forman el antepenúltimo nivel y así sucesivamente hasta llegar al nodo raíz. Es decir, tenemos un algoritmo recursivo que empieza en los nodos que preceden a los finales y retrocede hasta llegar al nodo raíz.

Con el procedimiento anterior, además de conocer la mejor opción de los jugadores en cualquier momento del juego, sabemos el resultado final del juego, ya que si el vector correspondiente al nodo raíz es (1,−1), significa que el primer jugador gana el juego (si en cada momento elige su mejor opción). Si este vector es (−1,1) entonces el segundo jugador gana el juego. Y en el caso del vector (0, 0) el juego termina en empate.

Ahora bien, en el Capítulo II, expusimos las razones por las cuales no es conveniente representar en la computadora el árbol del juego. También vimos que una alternativa es "podar" el árbol y de esta manera representar el juego a través de una gráfica dirigida. En ese mismo capítulo, propusimos una forma específica de "podar" el árbol del juego de Sim y construimos la gráfica dirigida que sólo tiene un nodo de cada clase de nodos equivalentes. Recordemos que la gráfica dirigida ya no es un árbol ya que un nodo puede ser hijo de dos o más nodos padres. De todas formas, en la gráfica dirigida tenemos toda la

información necesaria, pues por cada nodo eliminado quedó en esta gráfica uno que es equivalente. Además, cada nodo de la gráfica dirigida pertenece, al igual que en el árbol, a un solo nivel (de hecho, el nivel de un nodo de la gráfica es simplemente el número de líneas trazadas más uno). Podemos entonces aplicar el método de Zermelo directamente sobre la gráfica dirigida: se aplica primero la función de pagos en todos los nodos terminales, después se define en cada nodo no terminal del penúltimo nivel la ganancia del jugador en turno, después se hace lo mismo para cada uno de los niveles anteriores hasta llegar al nodo raíz.

En la Fig. IV.4 mostramos la gráfica dirigida del juego de Sim con cuatro puntos: Señalamos en cada nodo el color del jugador que obtiene la máxima ganancia en ese nodo, rojo si la ganancia del primer jugador es 1, azul si la ganancia del segundo jugador es 1 y verde en aquellos nodos donde la ganancia de ambos jugadores es 0.

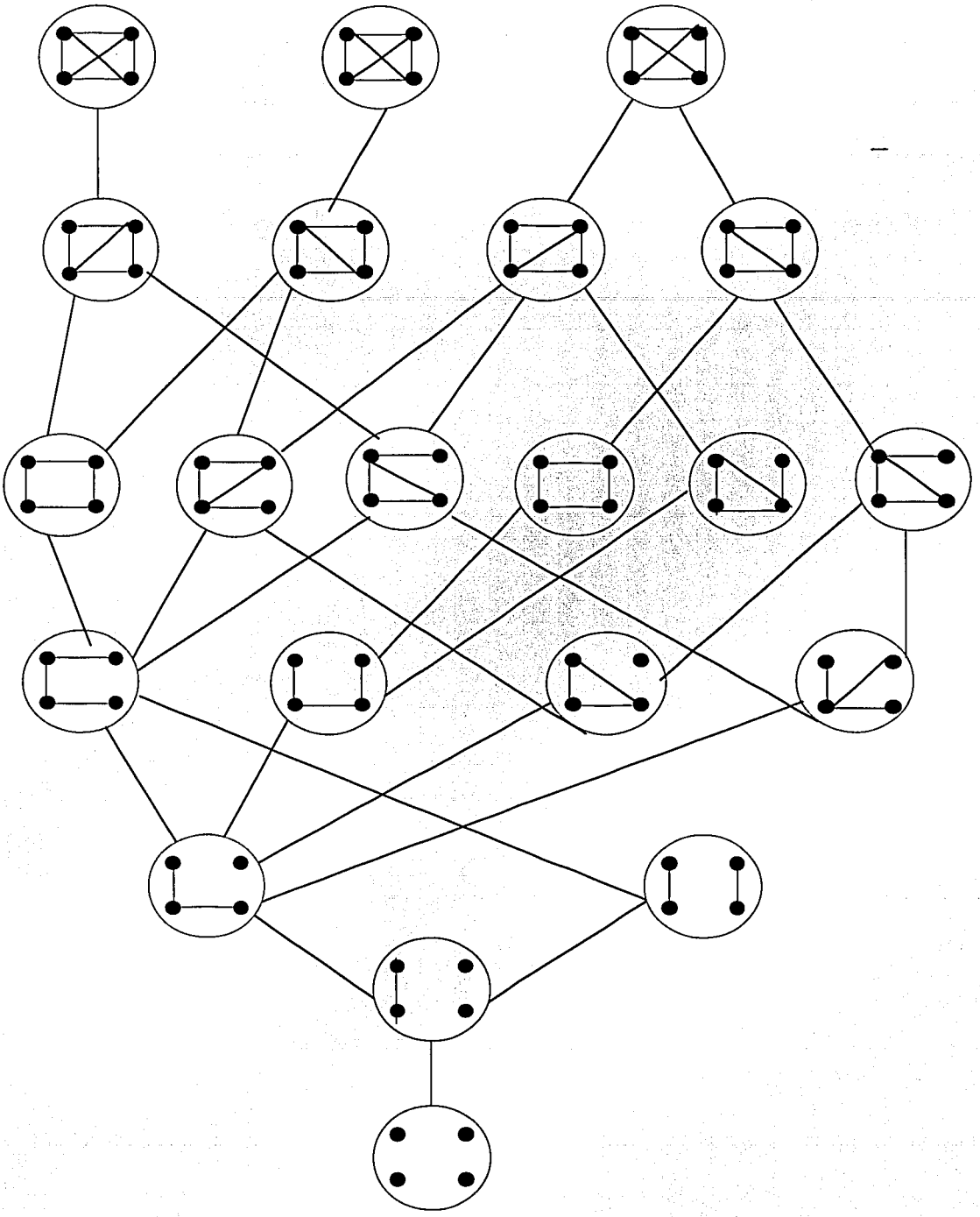


Figura IV.4. Gráfica dirigida del juego de cuatro puntos

Con el análisis anterior del juego de cuatro puntos, sabemos que el resultado es empate, siempre y cuando, ambos jugadores elijan su mejor opción en cada momento el juego.

En el juego de seis puntos sabemos, por el teorema de Ramsey, el empate no es posible: gana el primer jugador, o gana el segundo. El análisis para el caso de seis puntos lo haremos más adelante.

IV.3 Algoritmo de Zermelo para el juego de Sim.

A continuación presentaremos el algoritmo *califica* que utilizaremos para analizar el juego de Sim.
Para este algoritmo se utilizarán los vectores *qconfigs* e *iniconfigs*. Recordemos que la forma en que representamos cada nodo de la gráfica en la computadora es a través de su número característico en el vector *qconfigs*. Y que en *iniconfigs* representamos el índice en *qconfigs* del primer nodo de cada nivel de la gráfica.

Para representar las ganancias de los jugadores utilizaremos el vector *ncalificacion*: los nodos donde la máxima ganancia del primer jugador es 1 los representamos, en *ncalificacion*, con el número 1. Los nodos donde la máxima ganancia del segundo jugador es 1, los representaremos con el número 2. Y los nodos donde la máxima ganancia de ambos jugadores es 0, los representamos con el número 3.

Recordemos que, en el algoritmo que construye la gráfica del juego de Sim (Capítulo III) cada vez que generamos un nodo descendiente verificamos si en él existía un triángulo de color rojo o azul. De ser así, lo señalamos en el vector *ncalificacion*, por lo que, el algoritmo *califica* sólo revisará aquellos nodos que no tengan triángulo.

El algoritmo *califica* consiste en dos etapas. En la primera, aplicamos la función de pagos en los nodos del último nivel de la gráfica (en el juego con seis vértices esta etapa no es necesaria, ya que, en estos estados siempre existe al menos un triángulo y por lo tanto fueron calificados cuando se construyó la gráfica). En la segunda maximizamos las

ganancias de los jugadores en cada uno de los nodos que preceden a los finales utilizando las tres reglas que definimos en la Sección 2.

La primera etapa de *califica* consiste en lo siguiente:

1.- Localizamos el índice del primer y último nodo del último nivel de la gráfica del juego de Sim. Para lo cual, localizamos el primer y último índice, del último nivel de la gráfica, en el vector *qconfigs*.

iconfig1 = *iniconfigs*(*nivelmax*)
iconfig2 = *iniconfigs*(*nivelmax* + 1) –1

Por ejemplo, en el juego de cuatro puntos, la gráfica consiste de 7 niveles. El índice en *qconfigs* del primer nodo del último nivel se localiza en *iniconfigs*(7), y el índice del último nodo en *iniconfigs*(8) –1. Como *iniconfigs*(7) = 19 e *iniconfigs*(8) = 22. El último nivel de la gráfica está formado por los nodos que se encuentran representados por su número característico entre *qconfigs*(19) y *qconfigs*(21).

2.- Para cada uno de los nodos del último nivel de la gráfica hacemos lo siguiente:

a) Almacenamos en *iconfigt* el número característico que le corresponde al nodo en turno

Para *iconfigt* = *iconfig1* **hasta** *iconfig2*

b) Con el algoritmo *nc2edo* a partir del número característico del nodo en turno generamos la matriz *estado*.

En el ejemplo, el primer nodo, en este caso el nodo en turno, es *iniconfigs*(19) = 756. A partir del número característico, 756, generamos la matriz *estado* correspondiente a este número.

0	0	0	0
1	0	0	0
2	2	0	0
1	1	2	0

c) Con el algoritmo *buscatriangulo* revisamos si en la matriz *estado*, existe un triángulo rojo o azul.

En el ejemplo, *estado*(2, 1) = 1, *estado*(4, 1) = 1, *estado*(4, 2) = 1, por lo tanto, el nodo en turno contiene un triángulo rojo.

d) Si en la matriz *estado* hay un triángulo azul, el nodo en turno es una opción donde la ganancia del primer jugador es 1. Esto lo representamos en el vector *ncalificacion* con el número 1. Si en la matriz *estado* hay un triángulo rojo, el nodo en turno es una opción donde la ganancia del segundo jugador es 1, esto lo representamos en *ncalificacion* con el número 2. Si en la matriz *estado* no hay un triángulo, la ganancia de ambos jugadores en el nodo en turno es 0, esto lo representamos en el vector *ncalificacion* por un 3. El índice del nodo en turno en *ncalificacion* es el mismo que le corresponde a este nodo en *qconfigs*.

En el ejemplo, en la matriz *estado* hay un triángulo rojo, por lo que *ncalificacion*(19) = 2. Notemos que el índice que corresponde al nodo en turno en *qconfigs* es 19: Como este nodo es una opción donde el segundo jugador obtiene la máxima ganancia, entonces *ncalificacion*(19) = 2.

Al terminar este proceso tendremos representado en el vector *ncalificacion*, la ganancia de los jugadores en cada nodo del último nivel de la gráfica.

Por ejemplo, en el juego de cuatro puntos, la ganancia de los jugadores en los nodos del último nivel de la gráfica, está representada en el vector *ncalificacion* de la siguiente manera:

	2	1	3	
18	19	20	21	22

El procedimiento anterior corresponde a la primera etapa del algoritmo, es decir, hemos aplicado la función de pagos al último nivel de la gráfica dirigida del juego de Sim.

A continuación damos el procedimiento de la segunda etapa del algoritmo, esto es, el algoritmo que maximiza la ganancia del jugador situado en los nodos de los niveles que preceden al último nivel de la gráfica del juego. El número de este nivel lo representamos en la variable *nivel*.

1 Al iniciar, localizamos los nodos que forman el penúltimo nivel (*nivel* = *nivelmaximo* - 1) de la gráfica dirigida

iconfig1 = *iniconfigs*(*nivel*)	índice del primer nodo de *nivel*
iconfig2 = *iniconfigs*(*nivel* +1) – 1	índice del último nodo de *nivel*
iconfigls = *iniconfigs*(*nivel* +1)	índice del siguiente nivel de la gráfica.

Por ejemplo, en el juego de 4 puntos, la variable *nivel* = 6. Por lo que el índice del primer nodo del penúltimo nivel se localiza en *iniconfigs*(6) = 15 = *iconfig1*; el índice del último nodo de este nivel se encuentra en *iniconfigs*(7) – 1 = 19 – 1 = 18 = *iconfig2* y primer nodo del siguiente nivel se encuentra en *iniconfigs*(6 +1) = 19 = *iconfigls*

2 Maximizamos las ganancias del jugador situado en los nodos de este nivel, para esto, es necesario conocer el número del jugador que está situado ellos. Recordemos que los nodos de los niveles impares le corresponden al primer jugador y los niveles pares al segundo. Por lo que, el número del jugador (*nj*) al que corresponden los nodos del nivel en turno es

$$nj = ((nivel + 1) \bmod 2) + 1$$

El otro jugador se representará en la variable *njb*

Si *nj* = 1 **entonces** *njb* = 2 **si no** *njb* = 1

En el ejemplo, el número de jugador en turno es: *nj* = ((6 +1) mod 2) + 1 = 1+1 = 2

3 Para cada uno de los nodos, que no hayan sido calificados antes, maximizamos las ganancias del jugador en turno para lo cual hacemos lo siguiente:

Para *iconfigt* = *iconfig1* **hasta** *iconfig2*

qnc = *qconfigs*(*iconfigt*)

En el ejemplo, inicialmente *iconfigt* =15 y *qnc* = 244

a) Con el algoritmo *nc2edo* generamos la matriz *estado*, a partir del número característico del nodo en turno.

1			
0	2		
1	1	2	

b) Con el algoritmo *edo2lineam* generamos a partir de la matriz *estado* el vector *lineam*.

En el ejemplo, el vector *lineam* que corresponde a la matriz *estado* es:

1	0	2	1	1	2
1	2	3	4	5	6

Recordemos que, para aplicar las tres reglas con las que maximizamos la ganancia del jugador en turno, es necesario conocer las ganancias de los jugadores en los nodos descendientes del nodo en turno. Para esto hacemos lo siguiente:

c) Con el algoritmo *generahijas* generamos los descendientes del nodo en turno, los cuales los representamos en la matriz *lineah*. Es importante notar que estos nodos son los del nivel inmediato superior, donde la ganancia del jugador situado en estos nodos ya esta definida.

En el ejemplo, el vector *lineah* es:

1	2	2	1	1	2

Y la variable, *nhd* = 1, representa el número de descendientes del nodo en turno.

d) Con el algoritmo *sacanumerosdiferentes* descartamos los nodos descendientes repetidos y representamos sus números característicos en el vector *qncdh*, además en la variable *nncd* contamos el número de nodos descendientes que están representados en *qncdh*.

En el ejemplo, sólo hay un descendiente, así que el vector *qncdh* es el siguiente:

756

e)- Maximizamos las ganancias del jugador situado en el nodo en turno, para esto utilizamos las siguientes variables:

a) *nte* = 0
ntj = 0
nto = 0

b) Cada nodo descendiente (generado en el inciso d) lo localizamos en *qconfigs*

Para *iht* = 1 **hasta** *nncd*

qnch = *qncdh*(*iht*)

ict = *iconfig1s*

compara:

Si *qnch* **es igual a** *qconfigs*(*ict*) **entonces**

En el ejemplo, *qconfigs*(19) = 756.

c) En la variable *ncht* registramos el número que le corresponde a este nodo en *ncalificacion*

En el ejemplo, *ncht* = *ncalificacion* (19) = 2.

d) Si *ncht* = 3, significa que en el nodo descendiente la ganancia del jugador en turno es 0. Esto lo representamos incrementando en uno la variable *nte*.

Si *ncht* = 3 **entonces** *nte* = *nte* + 1

Si *ncht* = *nj*, significa que en el nodo descendiente la ganancia del jugador en turno es 1. Esto lo representamos incrementando en uno la variable *ntj*.

Si *ncht* = *nj* **entonces** *ntj* = *ntj* + 1

Si *ncht* es igual al número contrario del jugador en turno, significa que la ganancia del jugador es -1, esto lo representamos incrementando en uno la variable *nto*.

Si *ncht* = *njb* **entonces** *nto* = *nto* + 1

Siguiente iht

En el ejemplo *ntj* = 1, *nte* = 0 y *nto* = 0.

e) Repetimos el proceso de maximizar la ganancia del jugador, en el siguiente nodo descendiente, esto es, el siguiente nodo representado en *ncdh*.

En el ejemplo sólo hay un único nodo descendiente que es el representado por el número característico 756.

f) Para aplicar las tres reglas que maximizan la ganancia del jugador situado en el nodo en turno hacemos lo siguiente:
Si *ntj* es mayor que cero significa que entre los descendientes del nodo en turno hay al menos, una opción donde la ganancia del jugador en turno es 1. Esto lo representamos en *ncalificacion* con el número del jugador en turno.

Si *ntj* > 0 **entonces** *ncalificacion*(*iconfigt*) = *nj*

g) Si *ntj* no es mayor que cero, y *nte* si lo es, significa que entre los descendientes no hay una opción donde la ganancia del jugador en turno sea 1, pero hay por lo menos una donde la ganancia es 0. Esto lo representamos en *ncalificacion* con el número 3. Recordemos que el índice en *ncalificacion* es el mismo que el del nodo en turno en *qconfigs*.

Si *nte* > 0 **entonces** *ncalificacion*(*iconfigt*) = 3

h) Si *ntj* y *nte* no son mayores que cero, significa que no hay por lo menos una opción donde la ganancia del jugador en turno sea 1 ó 0, esto es, en todas las opciones la ganancia es −1. Esto lo representamos en *ncalificacion* con el número del jugador contrario.

ncalificacion(*iconfigt*) = *njb*

En el ejemplo *ntj* = 1, y el número del jugador en turno es 2, por lo que *ncalificacion*(19) = 2.

.Con lo anterior hemos maximizado la ganancia del jugador en turno en un solo nodo. Para cada nodo del nivel donde estamos situados repetimos el procedimiento anterior (todo el punto 3) hasta que *iconfigt* sea *iconfig2*.

Siguiente *iconfigt*

4.-Con lo anterior hemos maximizado las ganancias del jugador, *nj*, en los nodos del penúltimo nivel de la gráfica. Para hacerlo en los niveles que preceden a este hacemos la variable *nivel* = *nivel* − 1, y repetimos el procedimiento anterior a partir del punto 1. Este procedimiento termina cuando *nivel* = 1.

nivel = *nivel* −*1*
Ve a 1.

Con el algoritmo anterior, hemos representado en *ncalificacion* la máxima ganancia de los jugadores, en los nodos que les pertenecen.
En el caso del juego de cuatro puntos, el vector *ncalificacion* es la siguiente:

3	3	3	3	3	3	3	3	1	1	3	3	3	3	2	1	3	3	2	1	3
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Observemos, que *ncalificacion*(1) = 3; esto significa que la máxima ganancia del jugador situado en el nodo raíz es cero. Por lo tanto, si cada jugador realiza su mejor opción en cada momento del juego, éste termina en empate. Esta es una forma de saber, de antemano, el resultado final del juego de cuatro puntos. En este caso, en el juego de cuatro puntos, este resultado lo pudimos ver en la gráfica dirigida del juego. Sin embargo para el juego de seis puntos es imposible hacerlo así, por lo que la manera en que sabremos el resultado del juego es con el vector *ncalificacion* (ver la Sección 5, antes del algoritmo *sacatodoslosnumeros*).

IV.4 Cómo jugar.

Con el análisis del juego de Sim que hicimos en la Sección 1 de este capítulo conocemos la opción que le conviene elegir a los dos jugadores en cada momento del juego.
A continuación desarrollaremos un algoritmo, al que llamaremos *trikis*, para que la computadora juegue eligiendo la mejor opción en cada momento del juego.

En el algoritmo *trikis* utilizamos lo siguiente:

1.-El vector *ncalificacion*, donde representamos la ganancia de los jugadores en cada nodo del juego.

Recordemos que los nodos donde la ganancia del primer jugador es 1 están representados en el vector *ncalificacion* por el número 1. Los nodos donde la ganancia del segundo es 1, están representados por el número 2 y aquellos donde la ganancia de ambos jugadores es 0, están representados por 3.

2.- El vector *qconfigs* donde están representados los nodos de la gráfica por su número característico.

El algoritmo *trikis* consiste de las etapas siguientes:

1.- Conocer qué jugador que va a efectuar la jugada, así como en qué nivel de la gráfica se encuentra el juego en el momento de realizar la jugada. Para esto, utilizaremos el algoritmo *quientira2*, el cual se explicará más adelante.

En el algoritmo *quientira2* se generan dos variables: *njugador*, la cual representa el número del jugador en turno y *nlt*, que representa el número de aristas trazadas que tiene el estado en cada momento del juego. Con esta variable podremos saber en qué nivel de la gráfica se encuentra el juego, ya que éste es igual al número de líneas del nodo más uno.

Por ejemplo, supongamos que en el juego de cuatro puntos, en la rutina *quientira2*, obtenemos los siguientes valores: *njugador* = 2 y *nlt* = 3; por lo tanto, el turno es del segundo jugador, quien se encuentra situado en el cuarto nivel de la gráfica.

2.- Para que el jugador situado en el nodo en turno elija su mejor opción, es necesario generar los descendientes de este nodo. Para lo cual hacemos lo siguiente:

2.1 Con el algoritmo *edo2lineam* a partir de la matriz *estado* generamos el vector *lineam* del nodo en turno.

En el ejemplo el vector *lineam* del nodo en turno es la siguiente:

0	1	2	0	1	0

2.2. A partir del vector *lineam* generamos sólo los nodos descendientes sin triángulo del nodo en turno. En el caso de que todos los nodos contengan triángulo, generamos todos.

Observemos que los nodos descendientes con triángulo son una opción donde la ganancia del jugador en turno es -1. Es por esta razón que nos interesan los nodos descendientes sin triángulo. Para generarlos utilizaremos la rutina *generahijas2*, (la cual se explicará más adelante). Esta rutina genera la matriz *lineah* donde están representados los nodos descendientes del nodo en turno y el vector *ljugada* donde se representa el número de la arista que trazó el jugador en turno, para generar el nodo descendiente. Además se genera la variable, *ndeh*, donde se representa el número de nodos descendientes.

En el ejemplo, la matriz donde se representan los nodos descendientes del nodo en turno es la siguiente:

lineah:

2	1	2	0	1	0
0	1	2	2	1	0
0	1	2	0	1	2

ljugada

1
4
6

El vector *ljugada* índica que en el primer nodo descendiente el jugador en turno trazó la arista marcada con el número 1, en el segundo nodo descendiente el jugador trazó la arista marcada con el número 4 y en el tercer nodo descendiente trazó la arista marcada con el número 6.

La variable, *ndeh* = 3 ya que se generaron tres nodos descendientes.

3.- De los nodos generados en el punto anterior, se eligen aquellos donde el jugador en turno obtiene la máxima ganancia posible. Para esto, es necesario conocer la ganancia del jugador en cada uno de estos nodos.

Recordemos que esta ganancia está representada en el vector *ncalificacion* y que el índice de un nodo X en este vector es el mismo que le corresponde al nodo X en el vector *qconfigs*. Por lo que, si localizamos el índice en *qconfigs* de cada nodo descendiente, podremos conocer la ganancia del jugador en estos nodos.

Para localizar en el vector *qconfigs* los nodos descendientes generados en el punto anterior hacemos lo siguiente:

a) Generar el número característico de cada nodo.

b) Buscar cada número característico en el vector *qconfigs*.

Para generar el número característico de cada nodo hacemos lo siguiente:

Para *i* = 1 **hasta** *ndeh*

3.1 Copiamos el nodo descendiente *i*, representado en el vector *lineah* en el vector *linea*.

En el ejemplo, *i* = 1, corresponde al nodo representado en el primer renglón de la matriz *lineah*, por lo que el vector *linea* es el siguiente:

2	1	2	0	1	0

3.2 Con el algoritmo *linea2edo* generamos la matriz *estado* a partir del vector *linea*.

En el ejemplo, la matriz *estado* es la siguiente:

2			
1	2		
0	1	0	

3.3 Con el algoritmo *varitamagica* calculamos el número característico del nodo *i*.

En el ejemplo, el número característico del nodo en turno es 96

3.4 En el vector *qnch*, representamos el número característico, (generado en el punto anterior), del nodo *i*.

En el ejemplo, $i = 1$, por lo que $qnch(1) = 204$. Notemos que el índice de *qnch* corresponde al número de *i*.

Al terminar el punto tres tendremos en el vector *qnch* los números característicos de los nodos descendientes.

En el ejemplo, el vector *qnch* es el siguiente:

204
786
204

4.- Para localizar el índice de cada número característico del vector *qnch*, hacemos lo siguiente:

Para *inh* = 1 **hasta** *ndeh*

4.1. Buscamos el número característico *inh* en el vector *qconfigs*, para esto hacemos lo siguiente:

qnc = *qnch*(*inh*)

iconfigt = *iconfig1* (donde comienza el siguiente nivel)

En el ejemplo, *qnc* = *qnch*(1) = 204.

4.2. Comparamos *qnc* con cada uno de los números característicos que forman el siguiente nivel de la gráfica, (estos números están en el vector *qconfigs*), hasta encontrar *qnc*.

ss1:

Si *qnc* ≠ *qconfigs*(*iconfigl*)

entonces *iconfigl* = *iconfigl* + 1:

Ve a ss1.

El índice de *qnc* en *qconfigs* queda en la variable *iconfigl*.

En el ejemplo, los números característicos que forman el quinto nivel son:

qconfigs(9) = 240, *qconfigs*(10) = 216, *qconfigs*(11) = 212,
qconfigs(12) = 786, *qconfigs*(13) = 204, *qconfigs*(14) = 771.

Como *nc* = *qconfigs*(13) = 204, entonces queda *iconfigl* = 13.

4.3. Localizamos en el vector *ncalificacion* la ganancia del jugador en el nodo *inh*. Esta ganancia la representamos en el vector *lcalificacionh*.
En el ejemplo, *ncalificacion*(13) = 3; por lo que *lcalificacionh*(1) = 3.

Al terminar el proceso anterior, en el vector *lcalificacionh* tendremos representada la ganancia del jugador, en cada uno de los nodos descendientes generados en el punto 2.
En el ejemplo, el vector *lcalificacionh* es el siguiente:

3
3
3

5.- Una vez que conocemos las ganancias del jugador en turno en las opciones que tiene para elegir su jugada vamos a buscar aquéllas donde la ganancia del jugador sea la máxima

posible. Para esto, vamos a aplicar una de las tres reglas que definimos en la sección 2 de este capítulo. Esto lo hacemos de la siguiente manera:

5.1 Las opciones donde la ganancia del jugador en turno es 1, están representadas en el vector *lcalificacionh* con el número del jugador en turno; por lo que, para elegir una opción con la máxima ganancia recorremos el vector *lcalificacionh* en busca del número del jugador en turno:

a) Con la variable *nhg* contamos el número de opciones donde la ganancia del jugador es 1. Empezamos con *nhg* = 0.

b) Recorremos el vector *lcalificacionh*, de la siguiente manera:

Para *inh* = 1 **hasta** *ndeh*

Si *lcalificacionh*(*inh*) = *nj* **entonces**:

nhg = *nhg* + 1 (ya que hay una opción más).

Representamos el índice del nodo con ganancia máxima en *lcalificacionh* en el vector *indicem*.

indicem(*nhg*) = *inh*.

b) Si al menos en uno de los nodos la ganancia del jugador en turno fue 1, pasamos al punto 6, en el caso contrario pasamos al 5.2.

5.2 En el caso que no exista ninguna opción donde la ganancia del jugador sea 1, buscamos una opción donde la ganancia sea 0, por lo que buscamos en el vector *lcalificacionh* las ganancias representadas por el número 3. Esto lo hacemos así:

a) *nhg* = 0.

b) **Para** *inh* = 1 **hasta** *ndeh*

Si *lcalificacionh*(*inh*) = 3 **entonces**:

nhg = *nhg* + 1

indicem(*nhg*) = *inh*.

c) Si al menos existe un nodo donde la ganancia del jugador en turno es 0 pasamos al punto 6, en el caso contrario, pasamos al 5.3.

5.3 En el caso de que no exista por lo menos una opción donde la ganancia del jugador sea 1 ó 0, en todas las opciones disponibles del jugador la ganancia es -1. Por lo que en este caso:

Representamos el índice en *lcalificacion* de las opciones donde la ganancia del jugador es -1, en el vector *indicem*. Esto lo hacemos de la siguiente manera:

Para *inh* = 1 **hasta** *ndeh*

indicem(*inh*) = *inh*

nhg = *ndeh*

En el ejemplo, el vector *lcalificacion* contiene únicamente al número 3, así que todas las opciones disponibles del jugador en turno lo conducen a un nodo donde su ganancia es 0. La variable *nhg* = 3. Y el vector *indicem* es la siguiente:

1
2
3

6.- Tenemos *nhg* opciones donde el jugador obtiene la mayor ganancia posible. De estas escogemos una al azar. Para esto hacemos lo siguiente:

iijugada = int(*nhg* * RND) +1.

En el ejemplo, supongamos que *iijugada* = 3.

7.- Buscamos en el vector *indicem* la opción que corresponde a *iijugada* y la guardamos en la variable *ijugada*.

ijugada = *indicem*(*iiugada*)

En el ejemplo *indicem*(3) = 3. Por lo que *ijugada* = 3.

8.-Finalmente la opción que va a jugar el jugador en turno, es la que se encuentra en *ljugada*(*ijugada*).

nlajugar = *ljugada*(*ijugada*)

En el ejemplo, *nlajugar* = *ljugada*(3) = 6

9.- Los vértices de la arista que va a dibujar el jugador en turno son:

v1l = *lir*(*nlajugar*)

v2l = *lic*(*nlajugar*)

Las matrices *lir* y *lic* las definimos en el Capítulo II.

En el ejemplo, los vértices de la arista son:

v1l = *lir*(6) = 4

v2l = *lic*(6) = 3.

La arista que va a trazar el jugador en turno es la que une los vértices 4 y 3.

10.- Una vez que conocemos los vértices de la arista que va a trazar el jugador en turno, la representamos en la matriz *nestado*.

nestado(*v1l*, *v2l*) = *njugador*

nestado(*v2l*, *v1l*) = *njugador*.

En el ejemplo *nestado*(4, 3) = 2

La matriz *nestado* es la siguiente:

0			
1	2		
0	1	2	

11.- Dibuja la arista en la pantalla con el color del jugador en turno.

Con el algoritmo *generahijas2* hemos elegido una opción donde el jugador en turno obtiene la máxima ganancia posible. Notemos que el algoritmo está hecho sin importar si el jugador en turno es el primero o el segundo.

A continuación explicaremos los programas que usa el algoritmo *generahijas2*.

Algoritmo *quientira2* para saber el número del jugador en turno

Este algoritmo genera las variables *nj* y *nlt*; con la primera determinamos el número del jugador en turno y con la segunda el número de aristas trazadas hasta el momento actual del juego.
En este algoritmo, se utiliza las variable *nl1* y *nl2* para contar las líneas trazadas por el primer y segundo jugador respectivamente.
El conocer el número de líneas en el estado actual del juego nos permite determinar el número del nivel de la gráfica donde se encuentra ubicado el juego en determinado momento. Cada nivel de la gráfica del juego es igual al número de aristas trazadas hasta ese momento más uno.

El algoritmo es el siguiente:
1.-Se inicializa el número de aristas trazadas por los jugadores.

nl1 = 0
nl2 = 0

2.-Cuenta en la matriz *estado* las aristas trazadas por cada jugador. Para lo cual recorre esta matriz contando las aristas representadas por el número 1 y 2.

Para i = 2 **hasta** *nv* (el número de vértices del juego)
Para j = 1 **hasta** i-1 (recordemos que en esta matriz sólo se usa la parte inferior.)
Si *estado* (i, j) = 1 **entonces** *nl1* = *nl1*+1.
Si *estado* (i, j) = 2 **entonces** *nl2* = *nl2* +1

3.- Si el número de líneas trazadas por el primer jugador es igual al número de líneas trazadas por el segundo jugador, el jugador en turno es el primero, de lo contrario, el turno es del segundo jugador.

Si $nl1 = nl2$ **entonces** $nj = 1$ **si no** $nj = 2$

4.- El número total de líneas trazadas es igual a la suma de las líneas trazadas por ambos jugadores. Por lo que:

$nll = nl1 + nl2$.

Algoritmo *generahijas2* para generar los descendientes del nodo padre sin triángulo.

Para que un jugador elija su mejor opción, es necesario construir un algoritmo que genere los nodos descendientes, sin triángulo, del nodo en turno. Es importante observar que en las opciones que conducen al jugador a un nodo con triángulo, la ganancia de éste es -1.
En el Capítulo II explicamos la rutina *generahijas*, la cual genera todos los descendientes de un nodo padre, para esto, numeramos las aristas de la gráfica completa y las representamos en el vector *lineam*.

Por ejemplo, en el juego de cuatro puntos la numeración de la gráfica completa se muestra en la Fig. IV.5.

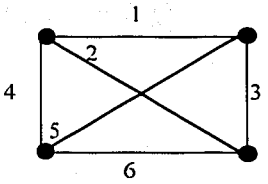


Figura IV.5.Numeración de aristas.

Recordemos que un estado cualquiera del juego lo representamos en el vector *lineam* de la manera siguiente: si la arista x la trazó el primer jugador, colocamos un 1 en el lugar x de *lineam*. Si la arista x la trazó el segundo jugador, colocamos un 2 en el lugar x de *lineam*. Si la arista x no ha sido trazada aún colocamos un 0 en el lugar x de *lineam*.

En la rutina *generahijas2* generamos los nodos descendientes, sin triángulo, del nodo padre, en ella, representaremos un estado de la misma forma que lo hicimos en el algoritmo *generahijas*.

Para ilustrar la rutina *generahijas2*, seguiremos este ejemplo: supongamos que el turno es del segundo jugador y que el vector donde está representado el nodo padre, esto es, el vector *lineam*, es el siguiente:

0	1	2	0	1	0

En la rutina *generahijas2* se hace lo siguiente:

1.-En la variable *ndeh* representamos el número de nodos descendientes. Como aún no hemos generado ninguno, *ndeh* = 0.

2.-Generamos los nodos descendientes del nodo padre. Para lo cual buscamos en el vector *lineam* las aristas que no han sido trazadas, es decir, los lugares del vector *lineam* donde hay un cero.

Para i1 = 1 **hasta** *nl*

Si *lineam*(i1) ≠ 0 entonces la arista i1 ya ha sido trazada, por lo que pasamos a la siguiente entrada de *lineam*.

Si *lineam*(i1) = 0, significa que la arista i1 no ha sido trazada. En este caso seguimos los siguientes pasos: (En el ejemplo i1 = 1, *lineam*(1) = 0)

a) Copiamos el vector *lineam* en el vector *linea*

En el ejemplo, el vector *linea* es:

0	1	2	0	1	0

b) Generamos el nodo descendiente en el vector *linea*, esto es, colocamos el número del jugador en turno en el lugar donde encontramos un cero
linea(i1) = *nj*

En el ejemplo que estamos siguiendo, *linea*(1) = 2, por lo que el vector *linea*, es el siguiente:

2	1	2	0	1	0

c) Con la rutina *linea2edo* generamos la matriz *estado* a partir del vector *linea*.

En el ejemplo, la matriz *estado* es la siguiente

2			
1	2		
0	1	0	

d) Con la rutina *buscatriangulo* revisamos si en la matriz *estado* del inciso anterior está representado un triángulo. En el caso de que así sea, ignoraremos este nodo descendiente y continuamos el proceso (vamos a **Siguiente** i1).
En el ejemplo que estamos siguiendo, en la matriz no está representado un triángulo.

e) En el caso de que en la matriz *estado* del inciso c no este representado un triángulo, hacemos lo siguiente:
Se incrementa la variable *ndeh* en uno, ya que existe un nodo descendiente sin triángulo.

$$ndeh = ndeh + 1$$

En nuestro ejemplo, *ndeh* = 1.

f) En la matriz *lineah* vamos a representar todos los nodos descendientes sin triángulo. Para esto, primero copiamos el vector donde está representado el nodo padre, esto es, copiamos el vector *lineam* en la matriz *lineah*. Esta última matriz es de dos dimensiones, la primera corresponde al número de descendientes generados y la segunda a la descripción del nodo descendiente. La matriz *lineah* la generamos de la siguiente manera:

Para i2 = 1 **hasta** *nl*

lineah (*ndeh*, i2) = *lineam*(i2)

En el ejemplo *lineah*(1, 1...6) es lo siguiente:

0	1	2	0	1	0

g) Generamos el nodo descendiente *ndeh*, esto lo hacemos colocando el número del jugador en turno en el lugar de la matriz *lineam* donde en el punto 2 encontramos un 0, por lo que

lineah (*ndeh*, i1) = *nj*.

Además representamos en el vector *ljugada* el número de la arista que hemos trazado en el nodo descendiente que acabamos de generar, entonces

ljugada(*ndeh*) = i1.

En el ejemplo, *ndeh* y i1 son iguales a 1 por lo que, *linea*(1, 1) = 2. Y *lineah* es la siguiente:

2	1	2	0	1	0

ljugada

1

h) Continuamos la generación de nodos descendientes:

Siguiente i1

Al terminar el proceso anterior habremos representado, en la matriz *lineah*, en caso de existir, los descendientes del nodo padre que no contienen triángulo.

En el ejemplo, al terminar el proceso anterior, el número de descendientes sin triángulo, *ndeh*, es tres y la matrices *lineah* y *ljugada* son las siguientes:

lineah:

2	1	2	0	1	0
0	1	2	2	1	0
0	1	2	0	1	2

ljugada

1
4
6

3.- Si entre los nodos descendientes, que generamos en el punto 2, existe al menos un nodo descendiente, sin triángulo, terminamos el procedimiento.
En el caso contrario, generamos todos los descendientes del nodo padre, para esto hacemos lo siguiente:

a) Para cada arista no trazada en el vector *lineam*, esto, es para cada lugar en *lineah* donde exista un cero, colocamos el número del jugador en turno, esto lo hacemos de la siguiente manera:

Para i1=1

Si *lineam*(i1) = 0 **entonces**

Contamos el nodo descendiente

ndeh = *ndeh* +1

Copiamos el nodo padre

Para i2 = 1 **hasta** *nl*

lineah(*ndeh*, i2) = *lineam*(i2)

Siguiente i2

Generamos el nodo descendiente trazando la arista i1

lineah (*ndeh*, i1) = *nj*

La arista trazada en el nodo descendiente la representamos en *ljugada*

ljugada (*ndeh*) = i1

Siguiente i1

En esta sección hemos desarrollado la estrategia *Trikis* la cual elige la mejor opción en cada momento del juego. En particular cuando se juega con esta estrategia como segundo jugador está garantizado el triunfo en el juego.

IV.5 Variantes del juego de Sim.

La primera estrategia que describimos (la de Trikis), es una estrategia con la cual el segundo jugador, si en cada momento del juego elige su mejor opción, gana el juego de Sim aún si el primer jugador juega en forma perfecta. A continuación daremos dos estrategias más para jugar el juego de Sim sólo con seis vértices.

La segunda estrategia, a la que llamaremos Trakis, es una estrategia para el primer jugador, y tiene relevancia en el caso en que el segundo jugador (el ganador en principio) no juega en forma perfecta. En efecto, aunque ya sabemos que el segundo jugador ganará si en cada momento del juego elige su mejor opción, es bien posible que un error del segundo jugador lleve la partida a una posición en la que el primer jugador (el perdedor en principio) pueda ganar. Conviene entonces que el primer jugador trate de maximizar la probabilidad de que el segundo cometa uno de estos errores. Esto es, desarrollaremos una estrategia para el primer jugador donde se utilizarán probabilidades. Un enfoque sencillo consiste en suponer que el segundo jugador juega al azar, fijándose únicamente en no hacer un triángulo azul

mientras esto sea posible (algo muy similar a lo que hace un jugador con poca experiencia). La estrategia Trakis intentará, bajo esta suposición, escoger su mejor opción pero además exponiendo al segundo jugador a situaciones en las que pueda equivocarse más fácilmente. Para ilustrar lo anterior, consideremos la Fig. IV.6.

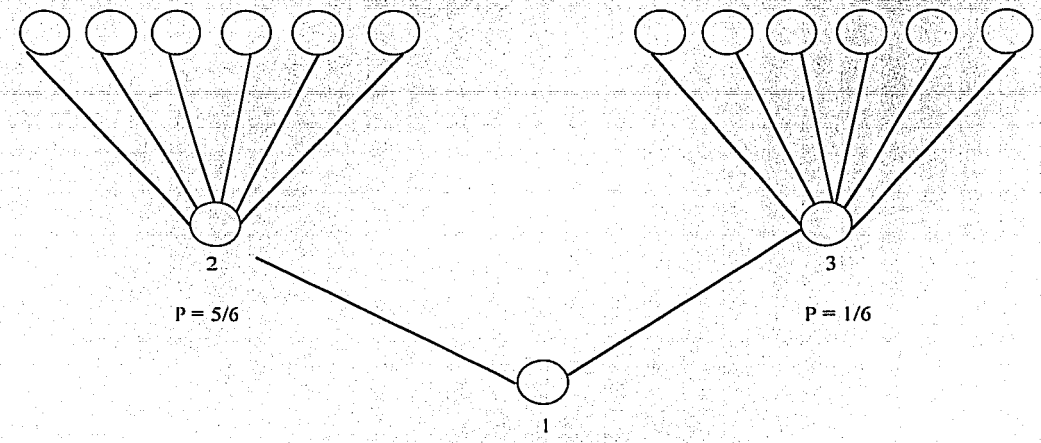


Figura IV.6. Dos posibles opciones para el primer jugador.

Supongamos que la Fig. IV.6 es una rama del árbol del juego de Sim. En los nodos marcados con rojo la ganancia de los jugadores es (1,-1) y en los marcados con azul es (-1,1). Cuando el primer jugador está situado en el nodo 1, tiene dos opciones para elegir. Cualquiera de estas opciones lo conduce a un nodo donde su ganancia es –1, esto es, un nodo donde el primer jugador pierde si el segundo juega perfectamente. Sin embargo, la mejor opción, para el primer jugador, es aquella que lo conduce al nodo 2 ya que en él existen 5 nodos descendientes donde la ganancia del primer jugador es 1 y sólo un nodo donde la ganancia de este jugador es –1, por lo tanto, la probabilidad en el nodo 2 de que el segundo jugador no elija su mejor opción es 5/6. Por otro lado, en los nodos descendientes del nodo 3 existen 5 nodos donde la ganancia del segundo jugador es 1 y sólo un nodo donde la ganancia del primer jugador es 1. Es decir, la probabilidad de que en el nodo 3 el segundo jugador se equivoque y no elija su mejor opción es 1/6. Por lo tanto, en el nodo 2 la probabilidad de que el segundo jugador se equivoque al elegir su opción es mayor que en el nodo 3.

En el algoritmo que juega con la estrategia de *Trakis* utilizaremos una variante del vector *ncalificacion* en donde representaremos la probabilidad, en cada nodo, de que gane el primer jugador (el perdedor en principio) cuando el segundo jugador juegue al azar, únicamente tratando de evitar triángulos de su color. Por razones de eficiencia computacional convendrá cambiar la escala y considerar las probabilidades como números enteros entre 0 y 32,000 en lugar de reales entre 0 y 1.

El nuevo vector *ncalificacion* lo construimos de la siguiente manera:

1.- En cada nodo del último nivel de la gráfica, el primer jugador gana o pierde con certeza, así que en este nivel las probabilidades son sólo 0 y 32000.
a) A los nodos donde la ganancia del primer jugador es 1 les asignamos el número 32000.
b) A los nodos donde el primer jugador obtiene una ganancia de –1 les asignamos el número 0.

2.- En cada nodo de un nivel que no sea el último calculamos la probabilidad de que el primer jugador gane de la siguiente manera:

a) Si el nodo padre le corresponde al primer jugador entonces la probabilidad de que en este nodo gane el primer jugador es igual al máximo de los números asignados a los nodos descendientes. En efecto, si el primer jugador elige siempre la mejor opción entonces elegirá la que le da la máxima probabilidad de ganar.

Por ejemplo, supongamos la rama de la gráfica que se muestra en la Fig. IV.7, donde el color de cada nodo indica el jugador que obtiene la máxima ganancia en él.

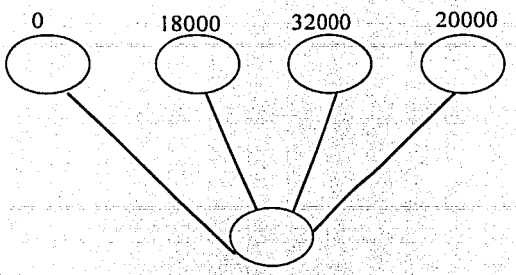


Figura IV.7. Máxima ganancia del primer jugador.

b) Si el nodo padre le corresponde al segundo jugador, de acuerdo con nuestra suposición sobre su modo de jugar, la probabilidad de que el primer jugador gane en este nodo es igual al promedio de los números asignados a los nodos descendientes que no tengan triángulo, cuando estos existan. Si todos los descendientes tienen triángulo entonces la probabilidad será 32000 lo que significa que ganará con certeza el jugador 1.

Por ejemplo, supongamos la rama de la gráfica que se muestra en la Fig. IV.8, donde el color de cada nodo indica el jugador que obtiene la máxima ganancia en él.

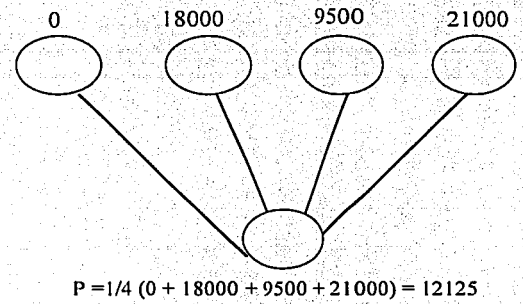


Figura IV.8. Ganancia del segundo jugador.

A continuación vamos a construir el nuevo vector *ncalificacion* para la estrategia de Trakis. Este algoritmo, que llamaremos *califica*, es sólo para el caso de seis vértices. Recordemos que cuando construimos la gráfica del juego (algoritmo hazarbol, Capítulo III) cada nodo que contuviera un triángulo lo calificamos en ese momento: si el triángulo es rojo se representó con el número 32000, y con 0 en el caso contrario. Como sabemos que en el último nivel todos los nodos contienen un triángulo (teorema de Ramsey) por lo tanto ya están calificados. En el algoritmo *califica* empezaremos en el penúltimo nivel de la gráfica. Este algoritmo lo explicaremos a grandes rasgos ya que es muy semejante al que desarrollamos en la sección 3 de este capítulo.

1.- El número del nivel donde vamos a calificar lo representamos en la variable *nivel*

Para *nivel* = *nivelmax*-1 **hasta llegar a** 1

2.- Localizamos el índice del primer y último nodo del nivel donde está situado el juego, para esto usamos los vectores *qconfigs* e *iniconfigs*.

iconfig1 = *iniconfigs*(*nivel*)	índice del primer nodo de *nivel*
iconfig2 = *iniconfigs*(*nivel* +1) − 1	índice del último nodo de *nivel*
iconfigls = *iniconfigs*(*nivel* +1)	índice del siguiente nivel de la gráfica.

3.- Determinamos el número del jugador en turno.

nj = ((nivel +1)mod2)+1.
Si *nj* = 1 **entonces** *njb* = 2 **si no** *njb* = 1

4.- Para cada nodo del nivel donde está situado el juego hacemos lo siguiente:

Para *iconfigt* = *iconfig1* **hasta** *iconfig2*

Sólo se califican los nodos que aún no han sido calificados:

Si *ncalificacion*(*iconfigt*) = -1 **entonces**

qnc = *qconfigs*(*iconfigt*)

a) Con el algoritmo *nc2edo* generamos la matriz *estado* a partir del número característico del nodo en turno.
b) Con el algoritmo *edo2lineam* a partir de la matriz *estado*, generamos el vector *lineam*.
c) Con el algoritmo *generahijas* generamos los descendientes del nodo en turno y los representamos en el vector *lineam*.
d) Con el algoritmo *sacatodoslosnumeros* (este algoritmo se explicará más adelante) generamos los números característicos de los nodos del inciso anterior que no contengan triángulo. Estos números los representamos en un vector *qncdh*, y los contamos con la variable *nncd*. En el caso que todos tengan triángulo generamos el número característico de todos los nodos descendientes, y los representamos en el mismo vector.

5.- Representamos en el vector *ncalificacion* la probabilidad que tiene el primer jugador de ganar en el nodo en turno. Recordemos que la forma de representar esta probabilidad depende del jugador al que le pertenezca este nodo, por lo que este punto tiene dos etapas:

5.1.- Si el nodo en turno corresponde al primer jugador, entonces, la probabilidad que tiene este jugador de ganar está representada por el mayor de los números de los nodos descendientes.

Para representar esta probabilidad en el vector *ncalificacion* hacemos lo siguiente:

Si *nj* =1 **entonces**

ncalmax = 0

Para cada número característico representado en el vector *qncdh*, hacemos lo siguiente:
(Recordemos que estos números son los nodos descendientes del vector en turno)

Para *iht* = 1 **hasta** *nncd*

qnch = *qncdh*(*iht*)

ict = *iconfigls*

a) Buscamos en el vector *ncalificacion*, la probabilidad de que el primer jugador gane en el nodo *iht*.

Compara:

Si *qnch* **es igual** *qconfigs*(*ict*) **entonces**

b) La probabilidad que tiene el primer jugador de ganar en el nodo *iht* la representamos en la variable *ncht*.

ncht = *ncalificacion*(*ict*)

c) Para asignar al nodo padre la probabilidad mayor de los descendientes comparamos *ncht* con *ncalmax*, en el caso de que esta última variable sea menor, entonces *ncalmax* = *ncht*.

Siguiente *iht*

Al terminar el proceso del punto 5.1, tendremos en *ncalmax* la mayor probabilidad, en los nodos descendientes, de que gane el primer jugador.

ncalificacion(*iconfigt*) = *ncalmax*

5.2.- Si el turno corresponde al segundo jugador, entonces la probabilidad de que el primer jugador gane la representamos con el promedio de los números que están asignados en los nodos descendientes que no contengan triángulo, cuando existan. Si todos los nodos descendientes contienen triángulo entonces la calificación será 32000. Para esto hacemos lo siguiente:

Si *nj* = 2 **entonces**

qncalprom = 0

Para cada descendiente del nodo en turno:

Para *iht* = 1 **hasta** *nncd*

qnch = *qncdh*(*iht*)

ict = *iconfigls*

a) Buscamos en el vector *ncalificacion*, la probabilidad de ganar del primer jugador en el nodo *iht*.

Compara2:

Si *qnch* **es igual** *qconfigs*(*ict*) **entonces**

b) El número del inciso anterior lo representamos en la variable *ncht*.

ncht = *ncalificacion*(*ict*)

c) Para calcular el promedio *qncalprom* = *qncalprom* + *ncht*.

Siguiente *iht*

d) Finalmente calculamos (la parte entera de) el promedio, por lo que,

qncalprom = **INT** (*qncalmax* / *nncd*).

e) Con el número *qncalprom* representamos en el vector *ncalificacion* la probabilidad de ganar, en el nodo padre, el primer jugador cuando el segundo no juega en forma perfecta.

6.- Con el proceso anterior hemos representado en el vector *ncalificacion*, la probabilidad, en el nodo en turno, que el primer jugado gane suponiendo que el segundo jugador no juega en forma perfecta. Este proceso lo repetimos, a partir del punto 4, tantas veces como nodos formen el nivel de la gráfica donde estamos situados.

Siguiente *iconfigt*

7.- Para calcular, en los nodos que preceden al nivel donde estamos situados, la probabilidad de que el primer jugador gane repetimos el proceso anterior, ahora con la variable *nivel* = *nivel* - 1 a partir del punto 1. Este proceso termina cuando *nivel* = 1.
Siguiente *nivel*

Con este algoritmo hemos construido el vector *ncalificacion*, cabe aclarar que este es el que se utiliza en el programa del juego de Sim, tanto en la estrategia de Trakis como en la de Trikis. En esta última, como se explicó en la sección 3 de este capítulo, el vector *ncalificacion* que se utiliza sólo tiene, para cada nodo, una calificación de 1 ó 2 (si gana el primer jugador o el segundo). Como el vector *ncalificacion* que hemos construido para la estrategia de Trakis contiene más información, en vez de construir en el programa los dos vectores *ncalificacion* sólo se construye el de Trakis, y Trikis lo utiliza de la siguiente manera: Si un nodo tiene calificación de 32000, entonces Trikis lo toma como 1, y si la calificación es menor que 32000 lo toma como 2.
El vector *ncalificacion* se incluye en el archivo *ncalific.dat*, en donde se puede ver que *ncalificacion*(1) = 31950 (= 2 para Trikis) con lo que tenemos una prueba computacional de que el segundo jugador gana el juego de Sim con seis puntos. Observemos también que Trakis, a pesar de ser el primer jugador (el perdedor en principio) tiene una muy grande probabilidad de ganar si el segundo jugador juega al azar y sólo cuidando de no hacer un triángulo azul si es posible.

Algoritmo *sacatodoslosnumeros*

A continuación explicaremos el algoritmo *sacatodoslosnumeros*, que se utilizó en el algoritmo *califica*.
Una vez que los nodos descendientes de un nodo padre están representados en la matriz *lineah* con el algoritmo *sacatodoslosnumeros* generamos los números característicos de aquellos nodos que no contengan triángulo (si existen), y si todos tienen triángulo, generamos todos los números característicos. Esto lo hacemos de la siguiente manera:

1.- La variable *nncd* = 0 ya que no hemos generado ningún número característico aún.

2.- Cada nodo descendiente, esto es, cada renglón de la matriz *lineah* lo copiamos en un vector *linea*.

Para *nht* = 1 **hasta** *ndeh*
Para *ii* = 1 **hasta** *nl*
linea(ii) = *lineah*(*nht*, *ii*)
Siguiente ii

3.- Con el algoritmo *linea2edo* generamos la matriz *estado*, correspondiente al vector *linea* del punto anterior.

4.- Una vez generada la matriz *estado*, que corresponde al nodo descendiente *nht*, con el algoritmo *buscatriangulo* revisamos si este nodo contiene un triángulo.

5.- Si el nodo del punto anterior no contiene triángulo
Si *ntri* = 0 **entonces**
Con el algoritmo *varitamagica* se genera el número característico del nodo en turno
nncd =*nncd* + 1 (Contamos el número de nodos sin triángulos)
qncdh(*nncd*) = *qnc*

Si por lo menos hubo un nodo que no contiene triángulo terminamos, en el caso contrario generamos los números característicos de todos los nodos descendientes y los representamos en el vector *qncdh*.
Si *nncd* > 0 **entonces fin**

6.- En el caso de que todos los nodos descendientes contengan triángulo:
Para *nht* = 1 **hasta** *ndeh* (Copiamos cada renglón de *lineah* en *linea*)
Para *ii* = 1 **hasta** *nl*
linea(ii) = *lineah*(*nht*, *ii*)
Siguiente *ii*
7.- Con el algoritmo *linea2edo* generamos la matriz *estado* del nodo en turno

8.- Con el algoritmo *varitamagica* generamos el número característico del nodo en turno.
9.- *nncd* = *nncd* +1 incrementamos el número de nodos representados en el vector *qncdh*
10.- *qncdh*(*nncd*) = *qnc* Representamos en el vector *qncdh* el número característico generado antes.

Siguiente *nht*

Algoritmo para utilizar la estrategia de *Trakis*

A continuación vamos a construir el algoritmo para jugar utilizando la estrategia de Trakis. Recordemos que ésta es una estrategia para el primer jugador, (el perdedor en principio), y sólo permite ganar en el caso de que el segundo jugador (el ganador), se equivoque y no juegue en forma perfecta.
En este algoritmo utilizamos los vectores *qconfigs* e *iniconfigs*, en ellos están representados los nodos de la gráfica por su número característico y el índice en *qconfigs* del nodo donde empieza cada nivel de la gráfica. También utilizaremos el vector *ncalificación*, donde hemos representado la probabilidad de que gane el primer jugador con un número entero entre 0 y 32000.

El algoritmo para utilizar la estrategia de Trakis es el siguiente:
1.- Identificamos el número del jugador que va a realizar la jugada y el número del nivel de la gráfica donde se encuentra situado el juego. Para esto, utilizamos la rutina *quientira2*, donde se generan las variables *njugador* y *nlt*.

2.- Si el turno es del segundo jugador, la máquina juega con la estrategia de Trikis.

Si *nj* = 2 **entonces Ve a** *trikis*,

En el caso de que el turno sea del primer jugador hacemos lo siguiente:

Si *nj* = 1 **entonces**

njugador = *nj*

nivels = *nlt* + 2

3.- Localizamos el índice en *iconfigs* del primer y último nodo donde se encuentran situados los descendientes del nodo en turno.

$$nivels = nlt + 2$$

$$iconfig1 = iniconfigs(nivels)$$

$$iconfig2 = iniconfigs(nivels + 1) - 1$$

4.- Para que el primer jugador elija su mejor opción es necesario conocer las probabilidades que tiene de ganar en los nodos descendientes. Para lo cual, seguimos los siguientes pasos:

4.1 Con el algoritmo *edo2lineam* a partir de la matriz *estado* generamos el vector *lineam*.

4.2 Con el algoritmo *generahijas2* generamos los descendientes sin triángulo del nodo en turno. En el caso de que no existan nodos descendientes sin triángulo, generamos todos. Recordemos que esta rutina genera la matriz *lineah* y el vector *ljugada*. En la primera están representados los nodos descendientes del nodo en turno, en el segundo, está representado el número de la arista que trazó el jugador, para generar el nodo descendiente. Además se genera la variable, *ndeh*, donde se representa el número de nodos descendientes.

Para localizar la probabilidad en los nodos descendientes en *ncalificacion* es necesario identificar su índice en *iconfigs* para lo cual hacemos lo siguiente.

a) Generar el número característico de cada nodo.
b) Buscar cada número característico en el vector *iconfigs*.

5.- A continuación generamos el número característico de cada nodo

Para *inh* = 1 **hasta** *ndeh*

5.1 Copiamos al vector *linea* el nodo descendiente *inh*, representado en la matriz *lineah*.

5.2 Con el algoritmo *lineam2edo* generamos la matriz *estado* a partir de el vector *lineam*.

5.3 Con el algoritmo *varitamagica* generamos el número característico del nodo *inh*.

5.4 En la matriz *qnch*, representamos el número característico, (generado en el punto anterior), del nodo *inh*.

qnch(*inh*) = *qnc*

5.5 Incrementamos *inh* en uno y repetimos el punto 5, tantas veces como nodos descendientes hayamos generado en el punto 4. Al terminar el proceso anterior, tendremos en la matriz *qnch* los números característicos de los nodos descendientes del nodo en turno.

Siguiente *inh*

6.- Para localizar las opciones donde el primer jugador tiene la mayor probabilidad de ganar, hacemos lo siguiente:

ncalifmax = 0

Para *inh* = 1 **hasta** *ndeh*

6.1 Buscamos el número característico *inh* en el vector *iconfigs*, para esto hacemos los siguiente:

qnc = *qnch*(*inh*)

iconfigt = *iconfig1* (índice del primer nodo del siguiente nivel)

6.2 Comparamos *qnc* con cada uno de los números característicos que forman el siguiente nivel de la gráfica donde está situado el juego.

ss2

Si *qnc* ≠ *qconfigs*(*iconfigt*)

entonces *iconfigt* = *iconfigt* +1

Ve a ss2

si *qnc* = *qconfigs*(*iconfigt*) **entonces** (pasamos a 6.3)

6.3 Con el índice del vector *iconfigs* donde se encuentra *qnc* localizamos en el vector *ncalificacion*, la probabilidad de ganar del primer jugador en el nodo *inh*. Este número lo representamos en la variable *ncalifi*.

ncalifi = *ncalificacion*(*iconfit*)

6.4 La calificación del jugador en el nodo *inh* la representamos en el vector *lcalificacion*.

lcalificacion(*inh*) = *ncalifi*

6.5 Comparamos la calificación del jugador en el nodo *inh* con *ncalifmax*, es decir

Si *ncalifi* > *ncalifmax* **entonces** *ncalifmax* = *ncalif*.

Siguiente *inh*

7.-Una vez que conocemos las probabilidades del primer jugador, en las opciones que tiene disponibles para elegir su jugada, vamos a buscar aquellas donde la probabilidad sea la máxima posible.

nhg = 0

7.1 Buscamos las probabilidades representadas con el número mayor, para esto hacemos lo siguiente:

Para *inh* = 1 **hasta** *ndeh*

Si *lcalificacion*(*inh*) = *ncalifmax* **entonces**

Es una opción donde el jugador tiene la máxima probabilidad posible de ganar, por lo que

nhg = *nhg*+1,

Representamos el índice de *inh* en *lcalificacionr* en el vector *indicem*

indicem(*nhg*) = *inh*.

Siguiente *inh*

Al terminar tenemos en la variable *nhg* el número de opciones donde el primer jugador tiene la mayor probabilidad de ganar y en el vector *indicem* los índices en *lcalificacion* de estas opciones.

8.- Se elige un número al azar entre 1 y *nhg*

iijugada = INT(*nhg**RND) + 1

9.- Tomamos la opción representada en el vector *indicem* con este número

ijugada = *indicem*(*iijugada*)

10.-El número de la arista que va a trazar el jugador en la opción elegida está representada en el vector *ljugada*.

nlajugar = *ljugada*(*ijugada*)

11.-Localizamos los vértices de la arista que va a trazar el jugador, para esto utilizamos los vectores *lir*, *lic* (ver Capítulo II, Sección 9).

v1t = *lir*(*nlajugar*)

v2t = *lic*(*nlajugar*)

12.- Se representa en la matriz *estado* la arista que traza el jugador en la opción elegida.

estado(*v1t*, *v2t*) = *njugador*

13.- Dibuja la arista (roja) en la pantalla.

Con esto finalizamos el desarrollo de la segunda estrategia (Trakis) para el juego de Sim.

La tercera estrategia la llamamos Trukis, y es tanto para el primer jugador como para el segundo; en ella, el jugador elige al azar una opción que lo conduzca a un nodo que no contenga triángulo; y en el caso de que no exista al menos un nodo sin triángulo, escoge al azar cualquier nodo. De hecho, esta es la forma en que supusimos que jugaba el segundo jugador en la estrategia de Trakis.

Algoritmo para utilizar la estrategia de Trukis

A continuación daremos el algoritmo Trukis para que la máquina juegue con esta estrategia. El algoritmo es el siguiente:

1.- Identificamos el número del jugador que va a efectuar la jugada y el número del nivel de la gráfica donde se encuentra situado el juego. Para esto, utilizamos la rutina *quientira2*, cuyo algoritmo explicamos en la Sección 4 de este capítulo.

2.- Con el algoritmo *edo2lineam* generamos a partir de la matriz *estado*, del nodo en turno, el vector *lineam*.

3.- A partir del vector *lineam* generamos los descendientes sin triángulo, del nodo en turno. En el caso de que no existan nodos descendientes sin triángulo, generamos todos los nodos. Para esto utilizamos la rutina *generahijas2*. Recordemos que esta rutina genera la matrices *lineah* y *ljugada*. En la primera están representados los nodos descendientes del nodo en turno, en la segunda, está representado el índice en *lineah* de la arista que trazó el jugador en turno para generar el nodo descendiente. Además se genera la variable, *ndeh*, donde se representa el número de nodos descendientes.

4.- Se elige un número al azar entre 1 y el número de nodos descendientes generados en el punto anterior.

$ijugada = \text{INT}(ndeh*\text{RND})+1$

5.- El número de la arista que va a trazar el jugador en turno en la opción elegida está representado en el vector *ljugada*.

$nlajugar = ljugada(ijugada)$

6.- Se localizan los vértices de la arista que va a trazar el jugador, para esto utilizamos los vectores *lir*, *lic*.

$v1t = lir(nlajugar)$

$v2t = lic(nlajugar)$

7.- Se representa en la matriz *estado* la arista que traza el jugador en la opción elegida.

$estado(v1t, v2t) = njugador$

8.- Se traza en la pantalla la arista del color del jugador en turno.

Con esto hemos terminado los algoritmos de las tres estrategias con las que juega la máquina el juego de Sim.

CONCLUSIONES

En este trabajo hemos analizado el problema matemático conocido con el nombre del juego de Sim. Así mismo, hemos construimos un algoritmo y desarrollamos un programa de cómputo, en el lenguaje Q-basic, para jugar este juego.

Para plantear el juego en términos matemáticos se utilizaron algunos conceptos de la teoría de gráficas, en particular demostramos el teorema de Ramsey a través del cual sabemos que en el juego de Sim no es posible el empate, es decir, siempre hay un perdedor y un ganador.

Para desarrollar el algoritmo que juega el juego de Sim en la computadora hicimos uso de algunos conceptos de inteligencia artificial y planteamos el juego como un problema de trazar aristas a través de un espacio de estados, donde cada estado del juego de Sim es una gráfica coloreada de orden seis.

La generación de estados la concebimos como si fuera la construcción de un árbol de búsqueda sobrepuesto al espacio de estados. La forma en que evitamos que se repitan nodos que representan un mismo estado es "podando" los nodos equivalentes y así, en lugar de construir un árbol de búsqueda, construimos la gráfica dirigida del juego de Sim. Para esto, se definió la equivalencia entre los estados A y B como un isomorfismo cromático de la gráfica que representa al estado A en la gráfica que representa al estado B.

Con la teoría de juegos sabemos que un juego con las características del juego de Sim está estrictamente determinado; esto es, desde un principio se puede saber el resultado final del juego, si los jugadores eligen siempre su mejor opción. Se aplicó el algoritmo de Zermelo a la gráfica dirigida del juego para determinar que si el segundo jugador juega en forma perfecta entonces este es el jugador que gana el juego. Se dio una prueba computacional de esto en el archivo *ncalific.dat* en disquete que se anexa a este trabajo.

Se desarrollaron algoritmos de tres estrategias para jugar el juego de Sim.

La primera estrategia, la llamamos Trikis, con la cual el segundo jugador, si en cada momento del juego elige su mejor opción, gana el juego de Sim aún si el primer jugador juega en forma perfecta.

La segunda estrategia, la llamamos Trakis, es para el primer jugador (el perdedor en principio), y permite ganar a este jugador sólo en el caso de que el segundo jugador, se equivoque y no juegue en forma perfecta.

La tercera estrategia, la llamamos Trukis, y es tanto para el primer jugador como para el segundo; en ella, el jugador elige al azar una opción que lo conduzca a un nodo que no contenga triángulo; en el caso de que exista.

Este trabajo es un ejemplo de cómo en un juego, en apariencia fácil, confluyen distintas áreas de las matemáticas y de la computación en su planteamiento y análisis y permiten extraer conclusiones importantes para resolver el problema original.

APÉNDICE I
GUÍA PARA EL USUARIO

En la pantalla del juego de Sim encontrarás las siguientes opciones:

F1 – Ayuda

F2 – Elegir Jugadores

F3 – Elegir Velocidad

F4 – Jugar

F5 – Salir del programa

Para entrar a la pantalla de Ayuda basta con oprimir la tecla F1 o "A". En ella encontrarás las reglas del Juego de Sim, así como una breve descripción de la forma de jugar de los tres jugadores programados.

Para elegir los jugadores oprime la tecla F2 o "E". En esta pantalla podrás elegir el primer jugador con las teclas "1", "2", "3" y "4", así como el segundo jugador con las teclas "A", "B", "C", y "D".

Para elegir la velocidad del juego oprime las teclas F4 o "V". En la pantalla aparecerán las siguientes velocidades Baja, Media y Alta. Elige la velocidad deseada con la tecla "B", "M" y "A" respectivamente.

Para iniciar el juego oprime la tecla F4 o "S".

Para salir del programa oprime la tecla "Esc" o F5

```
REM ************************************************************
REM *
REM                          APENDICE 2
REM *
REM *                    PROGRAMA DEL JUEGO DE SIM
REM*
REM ************************************************************
DEFINT I-N
DEFDBL Q
DEFLNG Z

nv = 6                                  ' número de vértices
nl = nv * (nv - 1) / 2                  ' número de líneas
qnci = 2 ^ nl                           ' factor para recorrer nl lugares
nivelmax = nl + 1: nlm2 = nl + 2
nconfigsmax = 4000
ntrimax = nv * (nv - 1) * (nv - 2) / 6
DIM ntriangulos(1 TO ntrimax, 1 TO 3)
DIM qconfigs(1 TO nconfigsmax)          ' Guarda el nc que representa cada
nodo.
DIM iniconfigs(1 TO nlm2)               ' Índice donde comienza cada nivel en
                                        ' qconfigs().
DIM lir(1 TO nl), lic(1 TO nl) ' localización inversa de renglones y columnas.
DIM nestado(1 TO nv, 1 TO nv), nestadop(1 TO nv, 1 TO nv)
DIM nestadot(1 TO nv, 1 TO nv)
DIM npermutacion(1 TO nv), npt(1 TO nv)
DIM lineah(1 TO nl, 1 TO nl)
DIM localizacion(1 TO nv, 1 TO nv)  ' localización de cada línea en estado().
DIM linea(1 TO nl)
DIM lineam(1 TO nl)
DIM ncalificacion(1 TO nconfigsmax) ' Calificación de cada configuración
DIM ljugada(1 TO nl)                ' Jugadas posibles sin triángulo.
DIM qnch(1 TO nl)
DIM lcalificacionh(1 TO nl)
DIM indicem(1 TO nl)            ' Guarda el índice de las jugadas buenas
                                ' en la lista de todas las hijas.
DIM qncdh(1 TO nl)              ' Número característico de las hijas.
DIM zcolor(1 TO 6, 1 TO 2)      ' Colores para resaltar líneas.
GOSUB deflocalizacion2


' Activar solo una de los siguientes dos módulos:

' Para Llenar arreglos con la gráfica calificada    (Se corre una sola vez)
'GOSUB arbolcalificado    'Llena arreglos de la gráfica
'PRINT
'PRINT nntt
'SOUND 1000, 2
'GOSUB guardadatos
'SOUND 1000, 2
'END

' Para jugar. Leyendo los datos de la gráfica, previamente guardados.
GOSUB leedatos
RANDOMIZE TIMER
pi = 4 * ATN(1)
npxi = 0: npxf = 639: npyi = 0: npyf = 479   ' define el área de la
pantalla
```

```
vxi = -80: vxf = 80: vyi = 0: vyf = 100      ' valores representados
DIM xv(1 TO nv), yv(1 TO nv)              ' Coordenadas de los vértices.
DIM lineac(1 TO nl)                ' Como línea para pintar configuración.
DIM ncolor(0 TO 2)                 ' Colores de los jugadores (0=no jugado).
DIM nombrej$(1 TO 4)
DIM nombrev$(1 TO 3)
DIM ttespera(1 TO 3)             ' Tiempo de espera para velocidad.

nombrej$(1) = "Humano"
nombrej$(2) = "Trikis"
nombrej$(3) = "Trakis"
nombrej$(4) = "Trukis"
nombrev$(1) = "Baja"
nombrev$(2) = "Media"
nombrev$(3) = "Alta"
ttespera(1) = 1
ttespera(2) = .5
ttespera(3) = .2

REM ************ Parámetros para parpadeo *********************
tprendido = .3
tapagado = .2
t12 = 44000     ' 12 horas aprox.
ncc = 14        ' color de cursor parpadeante.

REM *******  Define jugadores iniciales ************************

jugador1 = 3        ' 1 = humano,  2 = Trikis,  3 = Trakis,  4 = Trukis
jugador2 = 1        ' 1 = humano,  2 = Trikis,  3 = Trakis,  4 = Trukis
ivelocidad = 3
njg1 = 0            ' Juegos ganados acumulados, para puntaje.
njg2 = 0

SCREEN 12
CLS
GOSUB escalas
GOSUB definecolores

REM ************************************************************
REM ********* Termina inicialización ***************************
REM ************************************************************

menucentral:
GOSUB limpiabuffer  ' elimina teclas no atendidos en buffer.
GOSUB pontablero2   ' pinta toda la pantalla
esperam:
GOSUB tecla
IF k1 = 2 THEN                                          ' Teclas largas.
   IF kt = 59 THEN GOSUB ayuda: GOTO menucentral
   IF kt = 60 THEN GOSUB elegirj: GOTO menucentral      ' jugador
   IF kt = 61 THEN GOSUB elegirv: GOTO menucentral      ' velocidad
   IF kt = 62 THEN GOSUB juega1vs2d: GOTO menucentral   ' juega
   IF kt = 63 THEN GOTO salir
   GOTO esperam
END IF
IF k$ = "A" OR k$ = "a" THEN GOSUB ayuda: GOTO menucentral
IF k$ = "J" OR k$ = "j" THEN GOSUB juega1vs2d: GOTO menucentral
IF k$ = "E" OR k$ = "e" THEN GOSUB elegirj: GOTO menucentral
```

```
IF k$ = "V" OR k$ = "v" THEN GOSUB elegirv: GOTO menucentral
IF k$ = "S" OR k$ = "s" THEN GOTO salir
IF kt = 27 THEN GOTO salir                    ' escape
GOTO esperam
salir:                ' antes de salir pregunta si o no
CLS
FOR i = 1 TO 5
 LINE (180 + i, 210 + i)-(460 - i, 280 - i), 6, B
NEXT i
COLOR 3
LOCATE 16, 28
PRINT "Abandonar el juego? (S/N)"
COLOR 10
LOCATE 16, 50
PRINT "S"
LOCATE 16, 52
PRINT "N"
sal:
GOSUB tecla
IF k$ = "S" OR k$ = "s" THEN CLS : END
IF k$ = "N" OR k$ = "n" THEN GOTO menucentral
IF kt = 27 THEN GOTO menucentral
GOTO sal

REM ************************************************************************
REM ************************************************************************
REM ************************  RUTINAS  *******************************
REM ************************************************************************
REM ************************************************************************

elegirj:              ' Rutina para elegir jugadores
njg1 = 0              ' Inicialmente pone los marcadores a cero
njg2 = 0

GOSUB ponpuntaje
LINE (7, 44)-(314, 424), 0, BF
COLOR 3
LOCATE 6, 11
PRINT "Elija los jugadores";
LOCATE 7, 11
PRINT "y termine con <Esc>";
LOCATE 24, 13
PRINT "<Esc> = Terminar";
COLOR ncolor(1)
LOCATE 10, 7: PRINT "Jugador 1";
COLOR ncolor(2)
LOCATE 10, 26: PRINT "Jugador 2";
LINE (46, 164)-(121, 165), ncolor(1), BF
LINE (198, 164)-(274, 165), ncolor(2), BF

ej0:
COLOR 8
LOCATE 13, 9: PRINT "- Humano           - Humano";
LOCATE 15, 9: PRINT "- Trikis           - Trikis";
LOCATE 17, 9: PRINT "- Trakis           - Trakis";
LOCATE 19, 9: PRINT "- Trukis           - Trukis";

COLOR 10
```

```
LOCATE 13, 7: PRINT "1";
LOCATE 15, 7: PRINT "2";
LOCATE 17, 7: PRINT "3";
LOCATE 19, 7: PRINT "4";
LOCATE 13, 26: PRINT "A";
LOCATE 15, 26: PRINT "B";
LOCATE 17, 26: PRINT "C";
LOCATE 19, 26: PRINT "D";
LOCATE 24, 13
PRINT "<Esc>";

COLOR ncolor(1)
LOCATE 11 + 2 * jugador1, 11
PRINT nombrej$(jugador1);
LOCATE 25, 45
PRINT nombrej$(jugador1); ":";

COLOR ncolor(2)
LOCATE 11 + 2 * jugador2, 30
PRINT nombrej$(jugador2);
LOCATE 25, 65
PRINT nombrej$(jugador2); ":";

ej1:
GOSUB tecla
IF k$ = "1" THEN jugador1 = 1: GOTO ej0
IF k$ = "2" THEN jugador1 = 2: GOTO ej0
IF k$ = "3" THEN jugador1 = 3: GOTO ej0
IF k$ = "4" THEN jugador1 = 4: GOTO ej0
IF k$ = "a" OR k$ = "A" THEN jugador2 = 1: GOTO ej0
IF k$ = "b" OR k$ = "B" THEN jugador2 = 2: GOTO ej0
IF k$ = "c" OR k$ = "C" THEN jugador2 = 3: GOTO ej0
IF k$ = "d" OR k$ = "D" THEN jugador2 = 4: GOTO ej0
IF k1 = 1 AND kt = 27 THEN RETURN
GOTO ej1

REM ************************************************************************
REM ************************************************************************

elegirv:                                ' Elegir velocidad
LINE (7, 44)-(314, 424), 0, BF
COLOR 3
LOCATE 9, 7
PRINT "Elija la velocidad deseada.";
LOCATE 13, 17: PRINT "Baja";
LOCATE 15, 17: PRINT "Media";
LOCATE 17, 17: PRINT "Alta";
COLOR 10
LOCATE 13, 17: PRINT "B";
LOCATE 15, 17: PRINT "M";
LOCATE 17, 17: PRINT "A";
ev2:
GOSUB tecla
IF k$ = "B" OR k$ = "b" THEN ivelocidad = 1: RETURN
IF k$ = "M" OR k$ = "m" THEN ivelocidad = 2: RETURN
IF k$ = "A" OR k$ = "a" THEN ivelocidad = 3: RETURN
GOTO ev2
```

```
REM ************************************************************************
REM ************************************************************************
ayuda:
CLS
LOCATE 4, 1
PRINT "     En el Juego de Sim cada jugador, en su turno, escoge una de
las 15 "
LOCATE 6, 1
PRINT "    líneas y las dibuja de su color. El primer jugador utiliza el
rojo  "
LOCATE 8, 1
PRINT "    y el segundo azul. Pierde el jugador que forme un triángulo de
su    "
LOCATE 10, 1
PRINT "    color. Para colorear una línea, basta indicar los números de
los "
LOCATE 12, 1
PRINT "    puntos que une. Hay tres jugadores programados: Trikis, Trakis
y Trukis. "
LOCATE 14, 1
PRINT "    Trikis, como primer jugador, gana sólo que el otro jugador se
equivoque. "
LOCATE 16, 1
PRINT "    Como segundo jugador Trikis siempre gana. "
LOCATE 18, 1
PRINT "    Trakis, como primer jugador, elige una opción donde el otro
jugador tiene"
LOCATE 20, 1
PRINT "    la mayor probabilidad de equivocarse. Como segundo jugador
Trakis juega  "
LOCATE 22, 1
PRINT "    como Trikis."
LOCATE 24, 1
PRINT "    Trukis, siempre elige al azar una jugada en la que no se forme
un triángulo"
LOCATE 26, 1
PRINT "    de su color, si ello es posible."
LOCATE 28, 1
PRINT "     Para salir de ayuda oprima la tecla Esc."

GOSUB tecla
RETURN

REM ************************************************************************
REM ************************************************************************

guardadatos:            ' Guarda qconfigs() que contiene los números
                        ' característicos de todas las configuraciones;
                        ' iniconfigs() que contiene los índices donde
                        ' comienzan los datos de cada nivel y
ncalificacion()
                        ' que contiene la calificación de cada configuración.

OPEN "C:\datos6p.txt" FOR OUTPUT AS #1
FOR i = 1 TO 3730       ' Número total de configuraciones más uno
                        ' qconfigs(3730)=0 configuración no existente
 WRITE #1, qconfigs(i)
NEXT i
```

```
FOR i = 1 TO 17           ' Número de tiradas más dos
 WRITE #1, iniconfigs(i)
NEXT i
FOR i = 1 TO 3729         ' Número total de configuraciones
 WRITE #1, ncalificacion(i)
NEXT i
CLOSE #1
RETURN

REM ************************************************************************
REM ************************************************************************

leedatos:       ' ver comentarios en guardadatos
OPEN "C:\QB400\datos6p.txt" FOR INPUT AS #1
'OPEN "C:\datos6p.txt" FOR INPUT AS #1
FOR i = 1 TO 3730
 INPUT #1, qt
 qconfigs(i) = qt
NEXT i
FOR i = 1 TO 17
 INPUT #1, inict
 iniconfigs(i) = inict
NEXT i
FOR i = 1 TO 3729
 INPUT #1, ncalt
 ncalificacion(i) = ncalt
NEXT i
CLOSE #1
RETURN

REM ************************************************************************
REM ************************************************************************

pintaconfiguracion2:      ' pinta un círculo en cada vértice y dibuja las
                          ' líneas jugadas.
FOR i = 1 TO nv
th = i * 2 * pi / nv
xv(i) = xcc + rc * COS(th)
yv(i) = ycc + rc * SIN(th)
npx = bx + sx * xv(i)
npy = by + sy * yv(i)
CIRCLE (npx, npy), 2, 14     ' Círculos en los vértices
NEXT i
FOR i = 1 TO nl
njugador = lineac(i)         ' Se fija en quien jugó cada línea.
IF njugador <> 0 THEN        ' dibuja únicamente las líneas jugadas.
   v1t = lic(i)
   v2t = lir(i)
   npx1 = bx + sx * xv(v1t)
   npy1 = by + sy * yv(v1t)
   npx2 = bx + sx * xv(v2t)
   npy2 = by + sy * yv(v2t)

   LINE (npx1, npy1)-(npx2, npy2), ncolor(njugador)
END IF
NEXT i
npx = bx + sx * xcc
npy = by + sy * ycc
```

```
'LOCATE npy / 16 + 5, npx / 8 + 5
'PRINT qnc
RETURN

REM ************************************************************************
REM ************************************************************************

pintalinea1:                    ' pinta la línea del vértice v1t al vértice
v2t
                                ' con el color 1. Para resaltar una línea o un
                                ' triángulo se dibuja siempre del color 1 y se
                                ' redefine con el comando PALETTE.
npx1 = bx + sx * xv(v1t)
npy1 = by + sy * yv(v1t)
npx2 = bx + sx * xv(v2t)
npy2 = by + sy * yv(v2t)
LINE (npx1, npy1)-(npx2, npy2), 1
RETURN

REM ************************************************************************
REM ************************************************************************

pintalinea:                     ' pinta la línea del vértice v1t al vértice
v2t
                                ' con el color de njugador.
npx1 = bx + sx * xv(v1t)
npy1 = by + sy * yv(v1t)
npx2 = bx + sx * xv(v2t)
npy2 = by + sy * yv(v2t)
LINE (npx1, npy1)-(npx2, npy2), ncolor(njugador)
RETURN

REM ************************************************************************
REM ************************************************************************

pintalineas:                    ' pinta la línea del vértice v1t al vértice
v2t
                                ' con el color de njugador y parpadea.
                                ' Redefine COLOR 1.
npx1 = bx + sx * xv(v1t)
npy1 = by + sy * yv(v1t)
npx2 = bx + sx * xv(v2t)
npy2 = by + sy * yv(v2t)
PALETTE 1, zcolor(1, njugador)
LINE (npx1, npy1)-(npx2, npy2), 1
SOUND 2200, .3

FOR ico = 1 TO 6
PALETTE 1, zcolor(ico, njugador)
t1 = TIMER + .12
pl1:
tt = TIMER
IF tt < t1 AND (t1 - tt) < t12 THEN GOTO pl1
NEXT ico
FOR ico = 6 TO 1 STEP -1
PALETTE 1, zcolor(ico, njugador)
t1 = TIMER + .12
pl2:
```

```
tt = TIMER
IF tt < t1 AND (t1 - tt) < t12 THEN GOTO p12
NEXT ico
LINE (npx1, npy1)-(npx2, npy2), ncolor(njugador)

IF (jugador1 = 1 AND jugador2 <> 1) OR (jugador1 <> 1 AND jugador2 = 1)
THEN
LOCATE 14, 14: PRINT "           "
END IF
RETURN

REM ************************************************************************
REM ************************************************************************

ilumina3:                          ' Hace parpadear todos los triángulos
                                   ' hasta que se oprime una tecla.
PALETTE 1, zcolor(1, njugador)
GOSUB pinta3     ' Pinta todos los triángulos con el color 1.
ico = 1
iico = 1
il1:
GOSUB espera2
IF ico = 6 AND iico = 1 THEN iico = -1
IF ico = 1 AND iico = -1 THEN iico = 1
ico = ico + iico
PALETTE 1, zcolor(ico, njugador)
k$ = INKEY$
IF k$ = "" THEN GOTO il1
RETURN

REM ************************************************************************
REM ************************************************************************

pinta3:                ' Pinta todos los triángulos con el color 1.
FOR i3 = 1 TO ntri
 v1t = ntriangulos(i3, 1): v2t = ntriangulos(i3, 2): GOSUB pintalinea1
 v1t = ntriangulos(i3, 2): v2t = ntriangulos(i3, 3): GOSUB pintalinea1
 v1t = ntriangulos(i3, 3): v2t = ntriangulos(i3, 1): GOSUB pintalinea1
NEXT i3
RETURN

REM ************************************************************************
REM ************************************************************************

tirahumano3:                       ' Rutina que permite a humano tirar.
LINE (7, 44)-(314, 424), 0, BF
COLOR 3
LOCATE 10, 5
PRINT "Para jugar, teclea los números";
LOCATE 11, 5
PRINT "de los vértices que definen la";
LOCATE 12, 5
PRINT "línea que deseas trazar.";
LOCATE 15, 11
PRINT "(1, 2, 3, 4, 5, 6)";
LOCATE 18, 16
PRINT "Rendirse";
```

```
COLOR 10                ' Resalta las teclas que se pueden oprimir
LOCATE 15, 12           ' con el color que así lo índica.
PRINT "1";
LOCATE 15, 15
PRINT "2";
LOCATE 15, 18
PRINT "3";
LOCATE 15, 21
PRINT "4";
LOCATE 15, 24
PRINT "5";
LOCATE 15, 27
PRINT "6";
LOCATE 18, 16
PRINT "R";

ktmax = nv + 48
nbsalida = 0        '( 1=Salió con Esc    0=Salió normal   2=Salió con R)
GOSUB quientira2: njugador = nj    ' para saber quien juega.
LOCATE 24, 10
PRINT "                      "
COLOR 3
LOCATE 24, 10
PRINT "línea de:   a:"
nrenc = 24: ncolc = 20: GOSUB parpadea    ' regresa en kt la tecla
oprimida.
IF kt = 27 THEN nbsalida = 1: RETURN
IF kt = 82 OR kt = 114 THEN nbsalida = 2: RETURN
IF kt < 49 OR kt > ktmax THEN GOTO tirahumano3
v1t = kt - 48     ' Hace v1t el número de vértice 1.
th3:
nrenc = 24: ncolc = 25: GOSUB parpadea
IF kt = 27 THEN nbsalida = 1: RETURN
IF kt = 82 OR kt = 114 THEN nbsalida = 2: RETURN
IF kt < 49 OR kt > ktmax THEN GOTO th3
v2t = kt - 48       ' N£mero del vértice 2.
IF v1t = v2t THEN GOTO th3
IF nestado(v1t, v2t) <> 0 THEN GOSUB trompetilla: GOTO tirahumano3
nestado(v1t, v2t) = njugador
nestado(v2t, v1t) = njugador
GOSUB pintalineas               'Dibuja la línea y la hace parpadear.
'LOCATE 24, 10
'PRINT "                      "
LINE (7, 44)-(314, 424), 0, BF
COLOR 3
LOCATE 14, 14
IF jugador1 <> 1 OR jugador2 <> 1 THEN PRINT "Pensando..."
RETURN

REM ************************************************************************
REM ************************************************************************

juega1vs2d:          ' Juega jugador1 contra jugador2 definitivo
IF njg1 >= 32000 OR njg2 >= 32000 THEN GOSUB trompetilla: RETURN
GOSUB pontablero2    ' pinta toda la pantalla
otro4d:
GOSUB pontablero3    ' Pinta únicamente lo que corresponde a un juego
nuevo
```

```
IF jugador1 <> 1 AND jugador2 <> 1 THEN GOSUB portada2
otro5d:
COLOR ncolor(1)
LOCATE 29, 19
PRINT nombrej$(jugador1);
nbsalida = 0
ON jugador1 GOSUB tirahumano3, trikis, trakis, Trukis
IF nbsalida = 1 THEN RETURN          ' Salió con Esc
IF nbsalida = 2 THEN GOTO ganod          ' Salió con R
GOSUB edo2edot
GOSUB buscatriangulo          ' busca triángulos del jugador nj en
nestadot()
IF ntri > 0 THEN GOTO ganod
COLOR ncolor(2)                ' Turno del jugador 2.
LOCATE 29, 19
PRINT nombrej$(jugador2);
nbsalida = 0
ON jugador2 GOSUB tirahumano3, trikis, trakis, Trukis
IF nbsalida = 1 THEN RETURN            ' Salió con Esc
IF nbsalida = 2 THEN GOTO ganod          ' Salió con R
GOSUB edo2edot
GOSUB buscatriangulo          ' busca triángulos del jugador nj en
nestadot()
IF ntri = 0 THEN GOTO otro5d  ' Si no hay triángulo va a que juegue el 1.
GOTO ganod

ganod:                          ' alguien gano
IF nj = 1 THEN njc = 2 ELSE njc = 1
GOSUB fanfarrias
IF njc = 1 THEN njg1 = njg1 + 1 ELSE njg2 = njg2 + 1
GOSUB ponpuntaje
k$ = INKEY$
IF k$ <> "" THEN
  IF ASC(k$) = 27 THEN RETURN        ' se oprimió esc.
END IF
IF jugador1 = 1 OR jugador2 = 1 THEN    ' esta jugando humano.
  GOSUB portada3
gg1:
  IF ntri = 0 THEN GOSUB tecla ELSE GOSUB ilumina3
  COLOR 2
  LOCATE 29, 9
  PRINT "Turno de:";
  IF k$ = "o" OR k$ = "O" THEN GOTO otro4d
  IF k$ = "s" OR k$ = "S" THEN RETURN
  GOTO gg1
END IF
IF njg1 < 32000 AND njg2 < 32000 THEN GOTO otro4d
RETURN

REM ************************************************************************
REM ************************************************************************

pontablero2:          ' pinta toda la pantalla
CLS
FOR i = 0 TO 5
 LINE (i, i)-(639 - i, 479 - i), 6, B
NEXT i
LINE (316, 0)-(321, 479), 6, BF          ' Vertical central
```

```
LINE (316, 322)-(639, 328), 6, BF        ' Horizontal inferior de hexágono
LINE (0, 425)-(639, 430), 6, BF          ' Penúltima horizontal
LINE (480, 322)-(482, 425), 6, BF        ' Vertical entre jugadores
LINE (0, 40)-(320, 42), 6, BF            ' Bajo titulo

COLOR ncolor(1)
LOCATE 23, 45
PRINT "1er Jugador";
LOCATE 25, 45
PRINT nombrej$(jugador1); ":";
COLOR ncolor(2)
LOCATE 23, 65
PRINT "2o Jugador";
LOCATE 25, 65
PRINT nombrej$(jugador2); ":";

GOSUB ponpuntaje
FOR i = 1 TO nl
lineac(i) = 0
NEXT i
xcc = 39.5: ycc = 65: rc = 25: GOSUB pintaconfiguracion2
COLOR 8
LOCATE 3, 68: PRINT "1";
LOCATE 3, 52: PRINT "2";
LOCATE 11, 45: PRINT "3";
LOCATE 19, 52: PRINT "4";
LOCATE 19, 69: PRINT "5";
LOCATE 11, 75: PRINT "6";

njugador = 0                    ' Para que pintalinea dibuje gris
FOR nlajugar = 1 TO nl
v1t = lir(nlajugar)
v2t = lic(nlajugar)
GOSUB pintalinea
NEXT nlajugar


FOR i = 1 TO nv
 FOR j = 1 TO nv
  nestado(i, j) = 0
 NEXT j
NEXT i

LOCATE 2, 8
COLOR 2
PRINT "El juego de Trikis Trakis"
LOCATE 29, 53
PRINT "Velocidad: ";
LOCATE 29, 9
PRINT "Turno de:";
LOCATE 29, 64
COLOR 14
PRINT nombrev$(ivelocidad);
GOSUB portada1
RETURN
```

```
REM ************************************************************************
REM ************************************************************************

ponpuntaje:
COLOR 14
LOCATE 25, 53: PRINT "       ";
LOCATE 25, 53
PRINT njg1;
LOCATE 25, 73: PRINT "       ";
LOCATE 25, 73
PRINT njg2;
RETURN

REM ************************************************************************
REM ************************************************************************

pontablero3:    ' Pinta únicamente lo que corresponde a un juego nuevo
GOSUB ponpuntaje

FOR i = 1 TO nl
lineac(i) = 0
NEXT i
xcc = 39.5: ycc = 65: rc = 25: GOSUB pintaconfiguracion2

njugador = 0                ' Para que pintalinea dibuje gris
FOR nlajugar = 1 TO nl
v1t = lir(nlajugar)
v2t = lic(nlajugar)
GOSUB pintalinea
NEXT nlajugar

FOR i = 1 TO nv
 FOR j = 1 TO nv
  nestado(i, j) = 0
 NEXT j
NEXT i
RETURN

REM ************************************************************************
REM ************************************************************************

limpiabuffer:
FOR ilb = 1 TO 15
k$ = INKEY$
NEXT ilb
RETURN

REM ************************************************************************
REM ************************************************************************

espera2:                ' Espera un décimo de segundo y regresa
t1 = TIMER + .1         ' Se usa para resaltar los triángulos
ee2:
tt = TIMER
IF tt < t1 AND (t1 - tt) < t12 THEN GOTO ee2
RETURN
```

```
REM ************************************************************************
REM ************************************************************************

portada1:                          ' pantalla principal
LINE (7, 44)-(314, 424), 0, BF
COLOR 3
LOCATE 5, 3
PRINT "Elija una de las siguientes opciones:";
COLOR 10
LOCATE 8, 9: PRINT "F1";
LOCATE 10, 9: PRINT "F2";
LOCATE 12, 9: PRINT "F3";
LOCATE 14, 9: PRINT "F4";
LOCATE 16, 9: PRINT "F5";
COLOR 3
LOCATE 8, 12: PRINT "- Ayuda";
LOCATE 10, 12: PRINT "- Elegir jugadores";
LOCATE 12, 12: PRINT "- Elegir velocidad";
LOCATE 14, 12: PRINT "- Jugar";
LOCATE 16, 12: PRINT "- Salir del programa";
COLOR 10
LOCATE 8, 14: PRINT "A";
LOCATE 10, 14: PRINT "E";
LOCATE 12, 21: PRINT "V";
LOCATE 14, 14: PRINT "J";
LOCATE 16, 14: PRINT "S";
RETURN

REM ************************************************************************
REM ************************************************************************

portada2:                          ' Juega máquina contra máquina
LINE (7, 44)-(314, 424), 0, BF
COLOR 3
LOCATE 15, 6
PRINT "Utilice <Esc> para interrumpir";
LOCATE 17, 3
PRINT "(Se interrumpe al terminar el juego)";

COLOR 10
LOCATE 15, 14
PRINT "<Esc>";
RETURN

REM ************************************************************************
REM ************************************************************************

portada3:                          ' Cuando termina un juego de humano
LINE (7, 44)-(314, 424), 0, BF
COLOR 3
LOCATE 13, 17
PRINT "Otro juego";
LOCATE 17, 17
PRINT "Salir";
COLOR 10
LOCATE 13, 17
PRINT "O";
LOCATE 17, 17
```

```
PRINT "S";
COLOR 2
LOCATE 29, 9
PRINT "    Gan$:";
COLOR ncolor(njc)
LOCATE 29, 19
IF njc = 1 THEN PRINT nombrej$(jugador1);  ELSE PRINT nombrej$(jugador2);
RETURN

REM ************************************************************************
REM ************************************************************************

tecla:
k$ = INKEY$
IF k$ = "" THEN GOTO tecla
kt = ASC(RIGHT$(k$, 1))
kl = LEN(k$)
RETURN

REM ************************************************************************
REM ************************************************************************

trompetilla:

SOUND 440, 5
RETURN

REM ************************************************************************
REM ************************************************************************

fanfarrias:
IF jugador1 <> 1 AND jugador2 <> 1 THEN SOUND 1320, .2: RETURN
IF nbsalida = 2 THEN
  SOUND 440, 3
  SOUND 0, 1              ' Se rindió
  SOUND 220, 4
  SOUND 0, .3
  nbsalida = 0
  RETURN
END IF
IF (jugador1 = 1 AND nj = 2) OR (jugador2 = 1 AND nj = 1) THEN
  SOUND 880, 3
  SOUND 40, .2          ' ganó humano
  SOUND 1320, 3
  SOUND 40, .2
  SOUND 1760, 5
  RETURN
END IF
IF (jugador1 = 1 AND nj = 1) OR (jugador2 = 1 AND nj = 2) THEN
  SOUND 220, 3
  SOUND 0, 4              'perdió humano
  SOUND 220, 3
  SOUND 0, 4
  SOUND 220, 3
  SOUND 0, .3
  SOUND 220, 5
  SOUND 0, .2
 RETURN
```

```
END IF
RETURN

REM ************************************************************************
REM ************************************************************************
escalas:                              ' para tener control sobre las
escalas.
sx = (npxf - npxi) / (vxf - vxi)
bx = npxi - sx * vxi

sy = (npyf - npyi) / (vyi - vyf)
by = npyf - sy * vyi
RETURN

REM ************************************************************************
REM ************************************************************************

parpadea:    ' prende un cuadrito parpadeante en (npxc1, npyc1)-(npxc2,
npyc2)
             ' y termina con un teclazo.
npxc1 = ncolc * 8 - 8: npxc2 = npxc1 + 7
npyc1 = nrenc * 16 - 16: npyc2 = npyc1 + 15
comienzaparpadeo:
GOSUB pintacuadro
t1 = TIMER + tprendido
prende1: k$ = INKEY$
IF k$ <> "" THEN GOTO teclazo
tt = TIMER
IF tt < t1 AND (t1 - tt) < t12 THEN GOTO prende1
GOSUB quitacuadro
t1 = TIMER + tapagado
apaga1: k$ = INKEY$
IF k$ <> "" THEN GOTO teclazo
tt = TIMER
IF tt < t1 AND (t1 - tt) < t12 THEN GOTO apaga1
GOTO comienzaparpadeo

teclazo: GOSUB quitacuadro
LOCATE nrenc, ncolc
COLOR 14
PRINT k$
kt = ASC(RIGHT$(k$, 1))
RETURN
REM ************************************************************************
REM ************************************************************************

quientira2:      ' Regresa nj=número de jugador que debe tirar
                 ' y nlt = número de líneas tiradas.
nl1 = 0          ' Para que cualquier jugador pueda jugar en cualquier
momento
nl2 = 0          ' viendo el estado del tablero.
FOR i = 2 TO nv
 FOR j = 1 TO i - 1
  IF nestado(i, j) = 1 THEN nl1 = nl1 + 1    ' Cuenta el número de líneas
de
  IF nestado(i, j) = 2 THEN nl2 = nl2 + 1    ' cada color.
 NEXT j
NEXT i
```

```
IF nl1 = nl2 THEN nj = 1 ELSE nj = 2          ' Decide quien juega.
nlt = nl1 + nl2                          ' Calcula el total de lineas
jugadas.
RETURN

REM ************************************************************************
REM ************************************************************************
pintacuadro:
LINE (npxc1, npyc1)-(npxc2, npyc2), ncc, BF
RETURN

REM ************************************************************************
REM ************************************************************************

quitacuadro:
LINE (npxc1, npyc1)-(npxc2, npyc2), 0, BF
RETURN

REM ************************************************************************
REM ************************************************************************

definecolores:  ' Define los colores para el parpadeo de líneas y
triángulos.
FOR i = 1 TO 6
zr = 39 + i * 4
zg = i * 4
zb = i * 4
zrgb = zb
zrgb = zrgb * 256 + zg
zrgb = zrgb * 256 + zr
zcolor(i, 1) = zrgb
NEXT i
FOR i = 1 TO 6
zr = i * 5
zg = 33 + i * 5
zb = 33 + i * 5
zrgb = zb
zrgb = zrgb * 256 + zg
zrgb = zrgb * 256 + zr
zcolor(i, 2) = zrgb
NEXT i
RETURN

REM ************************************************************************
REM ************************************************************************

linea2edot:
FOR i = 1 TO nl
nestadot(lir(i), lic(i)) = linea(i)
NEXT i
RETURN
REM ************************************************************************
REM ************************************************************************

edo2edot:
FOR ii = 1 TO nv
 FOR jj = 1 TO nv
  nestadot(ii, jj) = nestado(ii, jj)
```

```
 NEXT jj
NEXT ii
RETURN

REM ************************************************************************
REM ************************************************************************

buscatriangulo:          ' busca triángulos del jugador nj en nestadot()
                         ' regresa el número de triángulos en ntri
                         ' y los vértices de los triángulos en ntriangulos().
FOR i = 2 TO nv
 FOR j = 1 TO i - 1
 nestadot(j, i) = nestadot(i, j)  ' Llena triángulo superior derecho de
la
 NEXT j                           ' matriz nestadot() porque así lo
requiere
NEXT i                            ' el algoritmo.
ntri = 0                          ' Inicialmente hay cero triángulos.
FOR i = 1 TO nv
 FOR j = i + 1 TO nv
 IF nestadot(i, j) = nj THEN      ' Localizó una línea del color nj de i a j.
  FOR k = j + 1 TO nv
   IF nestadot(i, k) = nj THEN    ' Localizó otra línea del color nj de i
a k.
    IF nestadot(j, k) = nj THEN  ' Localizó la línea de j a k que cierra el
     ntri = ntri + 1          ' triángulo e incrementa la cuenta de triángulos.
     ntriangulos(ntri, 1) = i     ' Guarda los vértices del triángulo.
     ntriangulos(ntri, 2) = j
     ntriangulos(ntri, 3) = k
    END IF
   END IF
  NEXT k
 END IF
 NEXT j
NEXT i
RETURN

REM ************************************************************************
REM ************************************************************************

Trikis:                ' Rutina que juega fijándose únicamente en el
jugador
                       ' que gana; no utiliza probabilidades. Redefine
                       ' calificación de probabilidades para este fin.
tt1 = TIMER + ttespera(ivelocidad)   ' Para controlar velocidad.
GOSUB quientira2                     ' Para saber:
njugador = nj                        ' el número de jugador que tira y
nivels = nlt + 2             ' el nivel (siguiente) de la jugada en el
árbol.
iconfig1 = iniconfigs(nivels)          ' Índice inicial del nivel
siguiente.
iconfig2 = iniconfigs(nivels + 1) - 1 ' Índice final del nivel siguiente.
GOSUB edo2edot                         ' Copia el Estado a un arreglo
temporal.
GOSUB edot2lineam                      ' Copia Estado a Línea (madre).
GOSUB generahijas2    ' Genera las hijas y también llena ljugada()
FOR inh = 1 TO ndeh   ' hace lista de qnc's de las hijas en qnch()
 FOR jnh = 1 TO nl
```

```
  linea(jnh) = lineah(inh, jnh)  ' Copia la hija número inh a linea().
 NEXT jnh
 GOSUB linea2edot               ' Pone información de la hija en edot().
 GOSUB varitamagica             ' Calcula el número característico de la
hija.
 qnch(inh) = qnc                ' Guarda el número característico de la
hija.
NEXT inh

FOR inh = 1 TO ndeh  ' Hace lista de calificaciones de las hijas.
 qnc = qnch(inh)     ' Número característico de la hija en turno.
 iconfigt = iconfig1 ' Índice donde debe comenzar a buscar el nc.
ss1:
 IF qnc <> qconfigs(iconfigt) THEN iconfigt = iconfigt + 1: GOTO ss1 ' Busca
                                                                  ' Encontró.
  IF ncalificacion(iconfigt) = 32000 THEN   ' Redefine calificaciones
     lcalificacionh(inh) = 1                ' fijándose únicamente en el
  ELSE                                      ' jugador que gana.
     lcalificacionh(inh) = 2
  END IF
NEXT inh

nhg = 0              ' número de hijas jugables (tiradas buenas)
FOR inh = 1 TO ndeh
IF lcalificacionh(inh) = nj THEN     ' Si la calificación es de jugada
buena
 nhg = nhg + 1                       ' aumenta el número de hijas jugables
y
 indicem(nhg) = inh                  ' la guarda en la lista.
END IF
NEXT inh
IF nhg > 0 THEN GOTO finm         ' Si hay jugadas buenas se va a elegir una.

FOR inh = 1 TO ndeh               ' Cuando no hay jugadas buenas, pone en la
 indicem(inh) = inh               ' lista a todas las jugadas; cualquiera es
NEXT inh                          ' buena para perder.
nhg = ndeh              ' El número de jugadas a jugar es el número de hijas.
finm: iijugada = INT(nhg * RND) + 1 ' Se escoge al azar un número de 1 a nhg.
ijugada = indicem(iijugada)         ' Se escoge la jugada con ese número.
nlajugar = ljugada(ijugada)    ' Se toma el número de la línea que corresponde
                               ' a esa jugada.
ee3:
tt = TIMER
IF tt < tt1 AND (tt1 - tt) < t12 THEN GOTO ee3  ' Para controlar la
velocidad.

v1t = lir(nlajugar)              ' Vértice 1 de la línea a jugar
v2t = lic(nlajugar)              ' Vértice 2 de la línea a jugar.
nestado(v1t, v2t) = njugador     ' Registra la jugada en nestado().
nestado(v2t, v1t) = njugador
GOSUB pintalineas                ' Traza la línea en la pantalla.
RETURN

REM ************************************************************************
REM ************************************************************************

Trakis:                           ' Juega intentando aumentar su
probabilidad
```

```
                                ' de ganar cuando es el primer jugador.
                                ' Como segundo jugador elige al azar una
                                ' de las jugadas ganadoras.
tt1 = TIMER + ttespera(ivelocidad)   ' Para control de velocidad.
GOSUB quientira2                     ' Para saber quien tira.
IF nj = 2 THEN GOSUB trikis: RETURN ' Cuando trakis es el jugador 2 juega
                                     ' como trikis.
njugador = nj                        ' Ver comentarios en trikis.
nivels = nlt + 2             ' el nivel (siguiente) de la jugada en el
 rbol.
iconfig1 = iniconfigs(nivels)          ' Índice inicial del nivel
siguiente.
iconfig2 = iniconfigs(nivels + 1) - 1  ' Índice final del nivel
siguiente.
GOSUB edo2edot                       ' Copia el Estado a un arreglo
temporal.
GOSUB edot2lineam                    ' Copia Estado a Línea (madre).
GOSUB generahijas2         ' Genera las hijas y también llena ljugada()
FOR inh = 1 TO ndeh        ' hace lista de qnc's de las hijas en qnch()
 FOR jnh = 1 TO nl
  linea(jnh) = lineah(inh, jnh)    ' Copia la hija número inh a linea().
 NEXT jnh
 GOSUB linea2edot            ' Pone información de la hija en edot().
 GOSUB varitamagica          ' Calcula el número característico de la
hija.
 qnch(inh) = qnc             ' Guarda el número característico de la hija.
NEXT inh

ncalifmax = 0                ' calificación máxima de entrada es cero.
FOR inh = 1 TO ndeh          ' recorre todas las hijas.
 qnc = qnch(inh)             ' toma el número característico de la hija.
 iconfigt = iconfig1         ' Índice donde debe comenzar a buscar el nc.
ss2:
 IF qnc <> qconfigs(iconfigt) THEN iconfigt = iconfigt + 1: GOTO ss2 ' busca
                                                                ' encontró
 ncalift = ncalificacion(iconfigt)           ' Lee  calificación
 lcalificacionh(inh) = ncalift               ' La guarda
 IF ncalift > ncalifmax THEN ncalifmax = ncalift  ' La registra si es máxima.
NEXT inh

nhg = 0              ' número de hijas jugables (con calificación máxima)
FOR inh = 1 TO ndeh
IF lcalificacionh(inh) = ncalifmax THEN  ' Si la hija tiene calificación
máx.
 nhg = nhg + 1                           ' aumenta la cuenta y
 indicem(nhg) = inh                      ' la registra.
END IF
NEXT inh

iijugada = INT(nhg * RND) + 1  ' Se escoge al azar un número de 1 a nhg.
ijugada = indicem(iijugada)    ' Se escoge la jugada con ese número.
nlajugar = ljugada(ijugada)    ' Se toma el número de la línea que corresponde
                               ' a esa jugada.
ee4:
tt = TIMER
IF tt < tt1 AND (tt1 - tt) < t12 THEN GOTO ee4  ' Para controlar la
velocidad.
```

```
v1t = lir(nlajugar)              ' Vértice 1 de la línea a jugar
v2t = lic(nlajugar)              ' Vértice 2 de la línea a jugar.
nestado(v1t, v2t) = njugador     ' Registra la jugada en nestado().
nestado(v2t, v1t) = njugador
GOSUB pintalineas                ' Traza la línea en la pantalla.
RETURN

REM ************************************************************************
REM ************************************************************************

Trukis:        ' si hay tiradas en las que no se hace triángulo
               ' escoge una de esas al azar (sin fijarse en la
               ' calificación). Si en todas se hace triángulo tira
cualquiera

tt1 = TIMER + ttespera(ivelocidad)
GOSUB quientira2
njugador = nj
GOSUB edo2edot
GOSUB edot2lineam
GOSUB generahijas2    ' también llena ljugada()

ijugada = INT(ndeh * RND) + 1     ' escoge una hija al azar
nlajugar = ljugada(ijugada)       ' número de línea a jugar

ee5:
tt = TIMER
IF tt < tt1 AND (tt1 - tt) < t12 THEN GOTO ee5

v1t = lir(nlajugar)
v2t = lic(nlajugar)
nestado(v1t, v2t) = njugador
nestado(v2t, v1t) = njugador
GOSUB pintalineas
RETURN


REM ************************************************************
REM *
REM *    Rutina para generar la gráfica dirigida y calificarla.
REM *
REM ************************************************************

arbolcalificado:
FOR i = 1 TO nconfigsmax
  ncalificacion(i) = -1    ' Inicialmente todas tienen -1. Al construir la
                           ' gráfica se asigna una calificación a las que
tienen
                           ' triángulo y queda el -1 en aquellas que la
                           ' rutina que califica debe calificar.
NEXT i
GOSUB hazarbol
GOSUB califica
RETURN


califica:
```

```
iconfig1 = iniconfigs(nivelmax)          ' Índice de la primera
configuración
                                         ' del último nivel.
iconfig2 = iniconfigs(nivelmax + 1) - 1  ' Índice de la última
configuración
                                         ' del último nivel.
FOR iconfigt = iconfig1 TO iconfig2      ' Recorre todas las
configuraciones
                                         ' del último nivel (16)
   qnc = qconfigs(iconfigt)
   GOSUB nc2edot                         ' Genera nestadot() para la config.
   FOR nj = 1 TO 2
     IF nj = 1 THEN njb = 2 ELSE njb = 1
     GOSUB buscatriangulo                ' Busca triángulos en la config.
     IF nj = 1 THEN ncalt = 0 ELSE ncalt = 32000
     IF ntri > 0 THEN ncalificacion(iconfigt) = ncalt  ' Califica de
acuerdo
                                                       ' al color del
tri.
   NEXT nj
NEXT iconfigt            ' Lo anterior es útil para menos de 6 vértices.


FOR nivel = nivelmax - 1 TO 1 STEP -1   ' Recorre del penúltimo nivel al 0
 iconfig1 = iniconfigs(nivel)           ' Índices inicial y final del
nivel
 iconfig2 = iniconfigs(nivel + 1) - 1
 iconfig1s = iniconfigs(nivel + 1)      ' Primero del siguiente nivel
 nj = ((nivel + 1) MOD 2) + 1           ' jugador que tira
 IF nj = 1 THEN njb = 2 ELSE njb = 1    ' Número del otro jugador
 FOR iconfigt = iconfig1 TO iconfig2    ' Recorre todas las
configuraciones
                                        ' del nivel en turno.
  IF ncalificacion(iconfigt) = -1 THEN ' Califica sólo a las sin
triángulo.
   qnc = qconfigs(iconfigt)
   GOSUB nc2edot
   GOSUB edot2lineam
   GOSUB generahijas
   GOSUB sacatodoslosnumeros    ' Regresa qnc's de hijas sin triángulo o
                                ' de todas si todas tienen triángulo.
   IF nj = 1 THEN               ' cuando juega 1 tomamos el máximo
     ncalmax = 0
     FOR iht = 1 TO nncd
      qnch = qncdh(iht)
      ict = iconfig1s
compara: IF qnch <> qconfigs(ict) THEN ict = ict + 1: GOTO compara
      ncht = ncalificacion(ict)
      IF ncht > ncalmax THEN ncalmax = ncht
     NEXT iht
     ncalificacion(iconfigt) = ncalmax
   END IF
   IF nj = 2 THEN                ' cuando juega 2 tomamos el promedio
     qncalprom = 0#
     FOR iht = 1 TO nncd
      qnch = qncdh(iht)
      ict = iconfig1s
compara2: IF qnch <> qconfigs(ict) THEN ict = ict + 1: GOTO compara2
```

```
      ncht = ncalificacion(ict)
      qncalprom = qncalprom + ncht
     NEXT iht
     ncalificacion(iconfigt) = INT(qncalprom / nncd)
   END IF
  END IF
 NEXT iconfigt
NEXT nivel
RETURN

REM ************************************************************************
REM
REM                Rutina para construir la gráfica dirigida
REM
REM ************************************************************************


hazarbol:
nntt = 0              ' Número de configuraciones con triángulo
iniconfigs(1) = 1
iniconfigs(2) = 2
qconfigs(1) = 0#
iconfigu = 1          ' Última configuración registrada
nivel = 1
nj = 1: njb = 2       ' nj= jugador en turno; njb= el jugador que no tira
nivelnuevo:
iconfig1 = iniconfigs(nivel)          ' Primer índice de "nivel" (ya
terminado)
iconfig2 = iniconfigs(nivel + 1) - 1 ' Último    "         "          "
iconfigt = iconfig1                   ' Índice de la configuración en
turno
iconfigi = iconfig2 + 1               ' Índice de config. inicial del
                                      ' siguiente nivel
confignueva:
qnc = qconfigs(iconfigt)         ' Número característico de config en
turno
GOSUB nc2edot
IF nj = 1 THEN nj = 2 ELSE nj = 1       ' Porque hay que buscar
triángulos
                                        ' del jugador que tiró
GOSUB buscatriangulo
IF nj = 1 THEN nj = 2 ELSE nj = 1       ' Para regresar al valor anterior
IF nj = 1 THEN ncalt = 32000 ELSE ncalt = 0  ' Por si hay triángulo.
IF ntri > 0 THEN ncalificacion(iconfigt) = ncalt: nntt = nntt + 1: SOUND
800, .5: GOTO siguienteconfig
GOSUB edot2lineam
GOSUB generahijas
GOSUB sacanumerosdiferentes
GOSUB agregancsnuevos
siguienteconfig:
IF iconfigt < iconfig2 THEN iconfigt = iconfigt + 1: GOTO confignueva
iniconfigs(nivel + 2) = iconfigu + 1
PRINT (nivel + 2); iniconfigs(nivel + 2)
IF nivel = nl THEN RETURN
nivel = nivel + 1
njt = nj: nj = njb: njb = njt
GOTO nivelnuevo
```

```
REM ************************************************************************
REM
REM        Rutina para agregar nuevos nc a la gráfica dirigida
REM
REM ************************************************************************

agregancsnuevos:
FOR i = 1 TO nncd
 qnc = qncdh(i)
 nbdif = 0
 FOR j = iconfigi TO iconfigu
  IF qnc = qconfigs(j) THEN nbdif = 1  'compara lo ncs con los de la
gráfica
 NEXT j                                'si no existe lo agrega.
 IF nbdif = 0 THEN iconfigu = iconfigu + 1: qconfigs(iconfigu) = qnc
NEXT i
RETURN

REM ************************************************************************
REM
REM      Rutina para generar los ncs de los estados generados y descartar
REM      aquellos que son equivalentes.
REM
REM ************************************************************************


sacanumerosdiferentes:
nncd = 0
FOR nht = 1 TO ndeh
 FOR ii = 1 TO nl
  linea(ii) = lineah(nht, ii)
 NEXT ii
 GOSUB linea2edot
 GOSUB varitamagica
 nbdif = 0
 FOR ii = 1 TO nncd
  IF qnc = qncdh(ii) THEN nbdif = 1  'compara los ncs nuevos, descarta
 NEXT ii                             'aquellos que son equivalentes.
 IF nbdif = 0 THEN nncd = nncd + 1: qncdh(nncd) = qnc
NEXT nht
RETURN

REM ************************************************************************
REM ************************************************************************

sacatodoslosnumeros:    ' regresa en qncdh() los nncd números
característicos
                        ' de las hijas sin triángulo (cuando las hay).
                        ' Si todas tienen triángulo, regresa todas.
nncd = 0
FOR nht = 1 TO ndeh
 FOR ii = 1 TO nl
  linea(ii) = lineah(nht, ii)
 NEXT ii
 GOSUB linea2edot
 GOSUB buscatriangulo
 IF ntri = 0 THEN
    GOSUB varitamagica
```

```
    nncd = nncd + 1
    qncdh(nncd) = qnc
 END IF
NEXT nht
IF nncd > 0 THEN RETURN          ' Regresa cuando hubo hijas sin triángulo
FOR nht = 1 TO ndeh              ' Todas tienen triángulo
 FOR ii = 1 TO nl
  linea(ii) = lineah(nht, ii)
 NEXT ii
 GOSUB linea2edot
 GOSUB varitamagica
 nncd = nncd + 1
 qncdh(nncd) = qnc
NEXT nht
RETURN

REM ************************************************************************
REM
REM                Rutina para generar los nodos descendientes.
REM
REM ************************************************************************

generahijas:                     ' Genera todas las hijas.
ndeh = 0                         ' Comienza sin hijas
FOR i1 = 1 TO nl                 ' Recorre todas las celdas de la madre
IF lineam(i1) = 0 THEN           ' Busca los ceros (lugares para tirar)
   ndeh = ndeh + 1               ' Cuando encuentra donde tirar genera una
hija
   FOR i2 = 1 TO nl              ' Copia madre
   lineah(ndeh, i2) = lineam(i2)      '     "
   NEXT i2                       '     "
   lineah(ndeh, i1) = nj               ' Anota el número del jugador en la
hija
END IF
NEXT i1
RETURN

REM ************************************************************************
REM
REM            Rutina para generar nodos descendientes sin triángulo
REM
REM ************************************************************************


generahijas2:                    ' Genera todas las hijas sin triángulo.
ndeh = 0                         ' Comienza sin hijas
FOR i1 = 1 TO nl                 ' Recorre todas las celdas de la madre
IF lineam(i1) = 0 THEN           ' Busca los ceros (lugares para tirar)
   GOSUB lineam2linea
   linea(i1) = nj
   GOSUB linea2edot
   GOSUB buscatriangulo
   IF ntri = 0 THEN
      ndeh = ndeh + 1              ' Cuando encuentra donde tirar genera
una hija
      FOR i2 = 1 TO nl             ' Copia madre
      lineah(ndeh, i2) = lineam(i2)      '     "
      NEXT i2                       '     "
```

```
      lineah(ndeh, i1) = nj        ' Anota el número del jugador en la
hija
      ljugada(ndeh) = i1
    END IF
END IF
NEXT i1
IF ndeh > 0 THEN RETURN          ' Regresa si hay hijas sin triángulo
                                 ' Sino registra hijas con triángulo
FOR i1 = 1 TO nl                 ' Recorre todas las celdas de la madre
IF lineam(i1) = 0 THEN           ' Busca los ceros (lugares para tirar)
   ndeh = ndeh + 1               ' Cuando encuentra donde tirar genera una
hija
   FOR i2 = 1 TO nl              ' Copia madre
   lineah(ndeh, i2) = lineam(i2)      '     "
   NEXT i2                            '     "
   lineah(ndeh, i1) = nj         ' Anota el número del jugador en la hija
   ljugada(ndeh) = i1
END IF
NEXT i1
RETURN

REM ************************************************************************
REM ************************************************************************

lineam2linea:
FOR i3 = 1 TO nl
 linea(i3) = lineam(i3)
NEXT i3
RETURN

REM ************************************************************************
REM ************************************************************************

deflocalizacion2:
nlt = 1
FOR i = 2 TO nv
 FOR j = 1 TO i - 1
  lic(nlt) = j
  lir(nlt) = i
  localizacion(i, j) = nlt
  nlt = nlt + 1
 NEXT j
NEXT i
ncolor(0) = 8
ncolor(1) = 12
ncolor(2) = 11
RETURN

REM ************************************************************************
REM ************************************************************************

edot2lineam:
FOR nr = 2 TO nv
 FOR nc = 1 TO nr - 1
  nlt = localizacion(nr, nc)
  lineam(nlt) = nestadot(nr, nc)
 NEXT nc
NEXT nr
```

```
RETURN

REM ************************************************************************
REM ************************************************************************

nc2edot:
FOR i = 1 TO nv
FOR j = 1 TO nv
nestadot(i, j) = 0
NEXT j
NEXT i
FOR i = 1 TO nl
 qnr = qnc AND 2 ^ (i - 1)
 IF qnr <> 0 THEN nestadot(lir(i), lic(i)) = 1
 qnr = qnc AND 2 ^ (i - 1) * qnci
 IF qnr <> 0 THEN nestadot(lir(i), lic(i)) = 2
NEXT i
RETURN

REM ********************************************************************
REM
REM  Rutina para asignar el número característico a cada estado.
REM
REM ********************************************************************

varitamagica:  ' recibe nestadot() y regresa el número característico qnc

FOR i = 1 TO nv: npermutacion(i) = i: NEXT i      ' Permutación inicial
GOSUB generaestadop
GOSUB asignancp: qncmin = qncp
otra:
GOSUB siguientepermutacion
IF indicp = 1 THEN
 GOSUB generaestadop
 GOSUB asignancp
 IF qncmin > qncp THEN qncmin = qncp
 GOTO otra
END IF
qnc = qncmin
RETURN

REM ************************************************************************
REM ************************************************************************

asignancp:     ' Calcula el número correspondiente a estadop
qncp = 0#
FOR i = 1 TO nl
 nrenglon = lir(i)
 ncolumna = lic(i)
 IF nestadop(nrenglon, ncolumna) = 1 THEN qncp = qncp + 2 ^ (i - 1)
 IF nestadop(nrenglon, ncolumna) = 2 THEN qncp = qncp + qnci * 2 ^ (i -
1)
NEXT i
RETURN

REM ********************************************************************
REM ********************************************************************
```

```
generaestadop:     ' genera estadop con el estado permutado
FOR i = 1 TO nl
 nrenglon = lir(i)
 ncolumna = lic(i)
 ntemp = nestadot(nrenglon, ncolumna)
 nrenglonp = npermutacion(nrenglon)
 ncolumnap = npermutacion(ncolumna)
 IF nrenglonp < ncolumnap THEN
   nntemp = nrenglonp
   nrenglonp = ncolumnap
   ncolumnap = nntemp
 END IF
 nestadop(nrenglonp, ncolumnap) = ntemp
NEXT i
RETURN

REM*************************************************************************
REM *************************************************************************
REM *     Rutina que recibe una permutación y regresa la siguiente en orden
REM *     creciente y avisa cuando ya no hay
REM *
REM *************************************************************************

siguientepermutacion:
indicp = 1
im = nv - 1
ciclop: nm = npermutacion(im)     'Número que vamos a comparar
inb = 0: nb = nv + 1       'Número buscado
FOR i = im + 1 TO nv
 IF npermutacion(i) > nm AND npermutacion(i) < nb THEN  'Compara números
    inb = I                                            'encontró uno menor
    nb = npermutacion(i)
 END IF
NEXT i
IF inb = 0 THEN                                        'Si no hay número menor
 IF im = 1 THEN indicp = 0: RETURN
 im = im - 1                                           'Toma el dígito siguiente
 GOTO ciclop
END IF
FOR i = im TO nv: npt(i) = npermutacion(i): NEXT i
npermutacion(im) = nb: npt(inb) = nv + 1         'Toma el dígito siguiente
FOR i = im + 1 TO nv                             'Colocamos los dígitos
 ns = nv + 1                                     'restantes en orden decreciente.
 FOR j = im TO nv
  IF npt(j) < ns THEN ns = npt(j): ins = j
 NEXT j
 npermutacion(i) = ns: npt(ins) = nv + 1
NEXT i
RETURN
```

Bibliografía

1. Chartrand G., Introductory Graph Theory, Dover, (1985).
2. Morton D., Teoría de Juegos, Alianza Universidad, (1997).
3. Owen G., Game Theory, W.B. Saunders Company, (1968).
4. Stuart R., Peter N., Inteligencia Artificial: Un enfoque moderno, Prentice Hall, (1995).