Las entradas están ligadas en orden alfabético por el apellido del autor, alfabéticamente por la primera palabra del título y, en orden ascendente por el número de almacén. Muestre el contenido de la estructura ARREGLO-LIBROS, después de completar cada una de las transacciones siguientes:

Operación	No. almacén	Título	Autor
INSERCION	53526	CREATIVE PHOTOGRAPHY	F. STOP FITZGERALD
INSERCION	98374	NEWSPAPER ORIGAMI	LINUS TYPE
INSERCION	14683	COOL HAND LUKE	HADDA DUWITT
INSERCION	23764	THE LONGEST YARD	LOWEN MAUER
INSERCION	49261	LA MARKE DE LA FRANCAISE	ASCENT AGU
INSERCION	19822	BEER BASTED HOT DOGS	DR. FRANK ANNSTEIN
INSERCION	76482	FOUR-WAY ROMANCE	QUAD LAMOORE
INSERCION	17760	COMPUTER SIMULATION	ARTIE ABACUS
INSERCION	38641	BETSY WORE BLUE JEANS	DENN M. STRETCHER
INSERCION	73920	FINGER LICKIN' GOOD	C. SANDERS
SUPRESION	14683		
SUPRESION	76482		

23. ¿Qué ventajas se tienen al tener un nodo principal en una lista? ¿Qué desventajas?

24. ¿Qué ventajas tiene una lista ligada circular sobre una lista no-circular? y ¿qué desventajas?

25. ¿Qué ventajas tiene una lista circular doblemente ligada sobre una lista circular ligada sencilla? y ¿qué desventajas?

26. Considere un arreglo con 500 nodos para alojar una lista doblemente ligada. Suponga que los nodos disponibles están ligados en una lista ligada sencilla.

a) Escriba un algoritmo para suprimir un nodo de la lista doblemente ligada.

b) Escriba un algoritmo para insertar un nodo en la lista doblemente ligada.

c) ¿Cuál es el número promedio de nodos tocados en su algoritmo de supresión?

d) ¿Cuál es el número promedio de nodos tocados en su algoritmo de inserción?

e) Compare sus respuestas de a) hasta d) con la situación donde los nodos disponibles *están* también en una lista doblemente ligada.

capítulo siete

grafos

Las estructuras de datos que hemos analizado en este libro, han sido estructuras lineales, donde para cada elemento hay un *siguiente* elemento. Esta linealidad es típica de las cadenas, de los elementos a lo largo de una sola dimensión en un arreglo, de los campos en un registro, de las entradas en una pila, de las entradas en una cola, y de los nodos en una lista ligada simple. En este capítulo comenzaremos con el estudio de estructuras de datos *no lineales*. En estas estructuras cada elemento puede tener varios elementos "siguientes", lo cual introduce el concepto de estructuras de ramificación. Estas estructuras de datos de ramificación son llamadas *grafos* y *árboles*.

En este capítulo analizaremos los grafos, su representación y sus operaciones. En el capítulo 8 iniciaremos la discusión sobre el concepto de árboles, que son un caso especial de grafos.

DEFINICIONES

Intuitivamente, un grafo es un conjunto de puntos y un conjunto de líneas, con cada línea se une un punto a otro. Los puntos se llaman los *nodos* del grafo, y las líneas se llaman aristas. Denotemos al conjunto de nodos de un grafo dada G, por V_G y, al conjunto de aristas, por E_G. Por ejemplo, en el grafo G de la figura 7-1, $V_G = \{a,b,c,d\}$ y $E_G = \{1,2,3,4,5,6,7,8\}$. El número de elementos de V_G es llamado el *orden* del grafo G. Un *grafo nulo* es un grafo con orden cero.

Una arista está determinada por los nodos que conecta. La arista 4, por ejemplo, conecta los nodos c y d y se dice que es de la *forma* (c,d). Un grafo está completamente

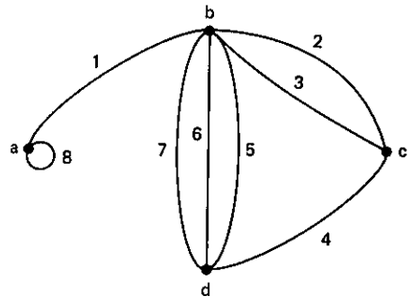

Figura 7-1 Ejemplo de un grafo.

definido por sus conjuntos de nodos y aristas. La posición real de estos elementos en la página no tiene importancia. La gráfica de la figura 7-1 es equivalente al grafo de la figura 7-2.

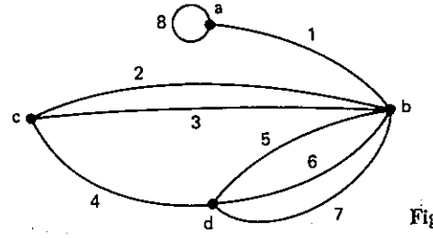

Figura 7-2 Grafo equivalente a la figura 7-1.

Nótese que puede haber varias aristas conectando a dos nodos, por ejemplo, los arcos 5, 6 y 7 todos son de la forma (b,d). Algunos pares de nodos pueden estar desconectados; por ejemplo, no hay aristas de la forma (a,c) o (a,d). Algunas aristas pueden conectar un nodo a sí mismo, por ejemplo, la arista 8 es de la forma (a,a), a estas aristas se les llama *bucles*.

Un grafo G se llama *grafo simple* si las siguientes condiciones son válidas.

1. No tiene ciclos, esto es, no existe una arista en E_G de la forma (v,v), donde V está en V_G.
2. No hay más de una arista uniendo un par de nodos, esto es, no existe más de una arista en E_G de la forma (v_1,v_2), para cualquier par de elementos v_1 y v_2 en V_G.

La figura 7-3 muestra un grafo simple derivado del ejemplo mostrado en la figura 7-1.

Un grafo que no es simple algunas veces es llamado *multigrafo*. Encontrará, que a las aristas también se les conoce como *arcos* y a los nodos como *vértices*.

Un *grafo conexo* es una gráfica que no se puede dividir en dos gráficas, sin eliminar por lo menos una de las aristas. El grafo de la figura 7-4 es un grafo *no* conexo.

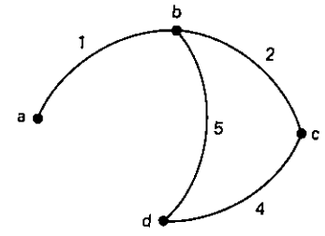

Figura 7-3 Ejemplo de grafo simple derivado de la figura 7-1.

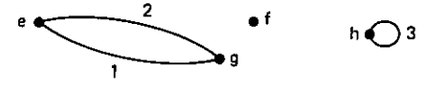

Figura 7-4 Ejemplo de un grafo desconectado.

Trayectorias

Una *trayectoria* en un grafo es una secuencia de una o más aristas que conecta a dos nodos. Denotamos por $P(v_i,v_j)$ a una trayectoria que conecta a los nodos v_i y v_j. Para que $P(v_i,v_j)$ exista, debe haber en E_G una secuencia de arcos de la siguiente forma:

$$P(v_i,v_j) = (v_i,x_1)\ (x_1,x_2)\ \ldots\ (x_{n-1},x_n)\ (x_n,v_j)$$

La *longitud* de la trayectoria es el número de aristas que la componen. En el grafo simple de la figura 7-3 se tienen las siguientes trayectorias entre los nodos b y d:

P(b,d) = (b,c) (c,d)	de longitud = 2
P(b,d) = (b,c) (c,b) (b,c) (c,d)	de longitud = 4
P(b,d) = (b,d)	de longitud = 1
P(b,d) = (b,c) (c,b) (b,d)	de longitud = 3

En general, sólo estamos interesados en trayectorias en las cuales un nodo dado es "visitado" no más de una vez. Esto restringe nuestro interés a la primera y tercera trayectorias anteriores. La segunda trayectoria visita dos veces a los nodos b y c; la cuarta trayectoria visita al nodo b dos veces. Siempre estaremos interesados en *no* recorrer la misma arista dos veces en una trayectoria. Esto evitará que nuestros algoritmos se tarden mucho por regresar sobre sus pasos.

Ciclos

Un *ciclo* es una trayectoria sobre la cual se cumple con las siguientes dos condiciones:

1. Ninguna arista puede aparecer más de una vez en una secuencia de aristas.
2. El nodo inicial de la trayectoria es el mismo que el nodo terminal, es decir, P(v,v).

En otras palabras, un ciclo regresa a su punto de inicio. El grafo de la figura 7-2 tiene varios ciclos, por ejemplo:

P(a,a) = (a,a)
P(b,b) = (b,c) (c,b)
P(b,b) = (b,c) (c,d) (d,b)
P(d,d) = (d,b) (b,c) (c,d)
P(d,d) = (d,b) (b,d)

Un grafo sin ciclos se dice que es *acíclica*. Las figuras 7-5a) y b) representan grafos acíclicos.

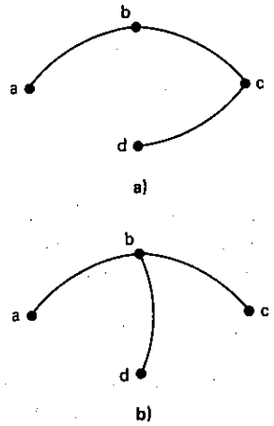

Figura 7-5 Ejemplo de grafos acíclicos.

Grafos dirigidos

Otro caso especial de la estructura de grafos general es el *grafo dirigido*, en el cual se les asigna dirección a las aristas de la gráfica. Se proporciona un ejemplo en la figura 7-6.

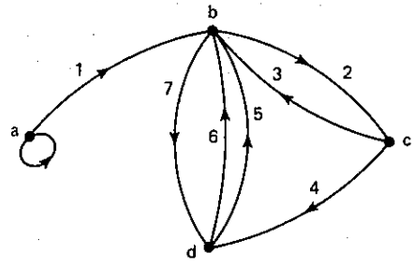

Figura 7-6 Ejemplo de grafo dirigido.

Cada arista del grafo dirigido incluye una flecha para indicar la dirección.

El *grado interno* de un nodo en un grafo es el número de aristas que terminan en ese nodo; el *grado externo* de un nodo, es el número de aristas que salen de este nodo. El *grado* de un nodo es la suma de sus grados internos y externos. Por ejemplo, en la figura 7-6 los valores son:

grado-interno(a) = 1	grado-externo(a) = 2	grado(a) = 3
grado-interno(b) = 4	grado-externo(b) = 2	grado(b) = 6
grado-interno(c) = 1	grado-externo(c) = 2	grado(c) = 3
grado-interno(d) = 2	grado-externo(d) = 2	grado(d) = 4

A lo largo de este capítulo nos referimos a los nodos de los grafos por sus etiquetas. En realidad, un nodo puede contener cualquier clase de información. Tenderemos a ignorar el contenido de estos nodos, sin embargo, no olvide que en última instancia el propósito de gráficas y árboles es el de representar la estructura lógica de esta información.

GRAFOS EN PROGRAMAS

Tal y como hemos estudiado otras estructuras de datos en capítulos anteriores, el tipo de datos llamado "grafo" no está predefinido en los lenguajes de programación comunes. Por esto, las características del grafo se simulan, alojándolas en otra estructura de datos. Existen tres formas principales para representar grafos: representación matricial, representación de lista y representación de multi-lista. Después de considerar cada una de éstas entraremos en la discusión de algunas de las operaciones más comunes sobre grafos: recorrido de gráficas y análisis de trayectorias.

REPRESENTACION DE LA MATRIZ DE ADYACENCIAS

Considere un grafo G con un conjunto de nodos V_G y un conjunto de arcos E_G. Suponga que la gráfica es de orden N, para $N >= 1$. Una forma de representar esta gráfica es utilizando una *matriz de adyacencias*, la cual es un arreglo A de N-por-N, donde

$$A(i,j) = \begin{cases} 1 \text{ si y sólo si la arista } v_i,v_j \text{ está en } E_G \\ 0 \text{ en otro caso.} \end{cases}$$

Si hay una arista que conecta los nodos i y j, entonces $A(i,j,) = 1$. La matriz de adyacencia del grafo no dirigida, mostrada en la figura 7-7 es:

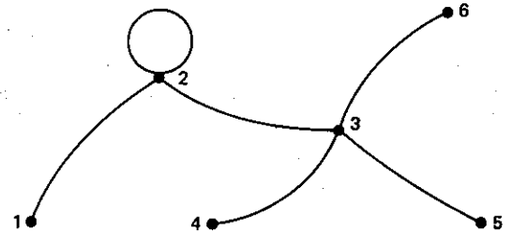

Figura 7-7 Ejemplo de grafo no-dirigido.

i \ j	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	1	1	0	0	0
3	0	1	0	1	1	1
4	0	0	1	0	0	0
5	0	0	1	0	0	0
6	0	0	1	0	0	0

Grafos dirigidos

Una arista de un *grafo dirigido* tiene su fuente en un nodo y su final en otro nodo. Por convención, la arista (v_i,v_j) denota la dirección del nodo v_i al nodo V_j.

La matriz de adyacencia para el grafo dirigido de la figura 7-8 es:

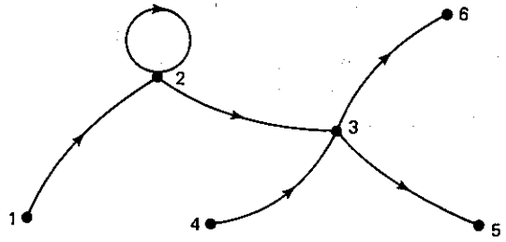

Figura 7-8 Ejemplo de grafo dirigido.

i \ j	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	0	0	1	1
4	0	0	1	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Matrices dispersas

Un grafo de orden *N* tiene una matriz de adyacencia de *N*-por-*N*. En muchos casos estas matrices de adyacencia son matrices dispersas y se pueden almacenar como arreglos dispersos. En estos casos, la representación matricial en general no es tan recomendable como la forma de listas ligadas, que introduciremos más tarde en este capítulo.

La matriz de adyacencia de un grafo no dirigido es simétrica. Por lo tanto, aun cuando la matriz no sea dispersa, los requerimientos de almacenamiento pueden reducirse

casi a la mitad al almacenar sólo el triángulo superior (o inferior) de la matriz. Regrese a los capítulos 2 y 6 para refrescar su comprensión del uso de arreglos triangulares dispersos.

Definición de grafos en COBOL y Pascal

La siguiente definición del arreglo puede ser usada para declarar un grafo llamado GRAFICA, de orden 24, en COBOL. La gráfica requiere de una matriz de adyacencias de 24-por-24.

```
01 GRAFO
   02 RENGLON OCCURS 24 TIMES.
      03 COLUMNA OCCURS 24 TIMES.
         04 ARCO PIC 9.
```

El ARCO (*I*, *J*) tiene entonces el valor de 1 si existe una arista entre los nodos *I* y *J*, y 0 en otro caso.

Otra forma para interpretar una matriz de adyacencia es como un tipo de datos booleano. Por ejemplo, en Pascal:

```
type grafo : array [1 . . 24, 1 . . 24] of boolean;
```

Aquí el grafo [*i*,*j*] tiene el valor *true* (verdadero) si existe una arista entre los nodos *i* y *j*, y *false* (falso) en otro caso.

Cálculo de aristas

Por lo general es conveniente hacer aritmética en matrices de adyacencia. Por ejemplo, el grado del nodo *i* en un grafo no dirigido es:

$$\sum_{j=1}^{N} A(i,j)$$

El grado interno de un nodo *i* en un grafo dirigido es la suma de las columnas de la matriz:

$$\sum_{k=1}^{N} A(k,i)$$

Y el grado externo del nodo *i* es la suma de los renglones de la matriz:

$$\sum_{k=1}^{N} A(i,k)$$

Discutiremos otros resultados comunes de la manipulación de datos adyacentes más adelante en el capítulo. Si la matriz de adyacencia se define de tipo booleano, entonces estas manipulaciones comunes no son realmente tan simples. En lugar de sumas, se requiere de una serie de operaciones booleanas (*and* y *or*).

Aristas ponderadas

Existen muchas aplicaciones de grafos en las cuales la matriz de adyacencia es la representación adecuada. Considere la gráfica que se muestra en la figura 7-9, la cual se puede usar en un sistema de información de una compañía de transporte. Este sería un ejemplo de un grafo de aristas ponderadas. Los nodos representan a las ciudades y las aristas representan las trayectorias de tráfico entre las ciudades. Cada arista se etiqueta con la distancia entre un par de ciudades interconectadas. En lugar de usar una matriz de bits para representar este sistema de transporte, podemos usar una matriz de aristas ponderadas.

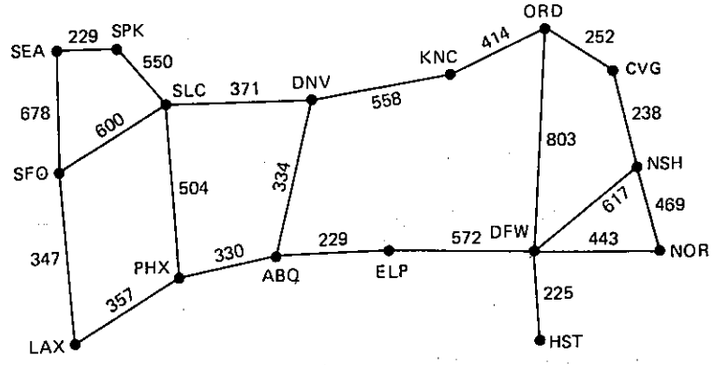

Figura 7-9 Ejemplo de grafo que muestra las distancias entre ciudades.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		0	0	334	229	0	0	0	0	0	0	330	0	0	0	0
2			0	0	0	0	0	0	238	0	252	0	0	0	0	0
3				0	572	225	0	0	617	443	803	0	0	0	0	0
4					0	0	558	0	0	0	0	0	0	0	371	0
5						0	0	0	0	0	0	0	0	0	0	0
6							0	0	0	0	0	0	0	0	0	0
7								0	0	0	414	0	0	0	0	0
8									0	0	0	357	0	347	0	0
9										469	0	0	0	0	0	0
10											0	0	0	0	0	0
11												0	0	0	0	0
12													0	0	504	0
13														678	0	229
14															600	0
15																550
16																

Figura 7-10 Representación matricial de aristas ponderadas de la figura 7-9.

donde el nodo 1 = ABQ, 2 = CVG, 3 = DFW, 4 = DNV, 5 = ELP, 6 = HST, 7 = KNC, 8 = LAX, 9 = NSH, 10 = NOR, 11 = ORD, 12 = PHX, 13 = SEA, 14 = SFO, 15 = SLC, 16 = SPK

Las aristas ponderadas se usan con frecuencia, en aplicaciones de transporte, donde los pesos representan normalmente las distancias, como en las figuras anteriores. En aplicaciones de flujos, los pesos representan las capacidades. Por ejemplo, los nodos en un grafo pueden representar la capacidad en litros por minuto, de una línea de ductos de transporte de líquidos entre diversas localidades, o la capacidad en bits/seg de comunicación en una red de computadoras.

En otras aplicaciones, el peso de las aristas representa el tiempo. Por ejemplo, se pueden utilizar las gráficas para representar redes de actividades, donde cada arista representa una tarea o actividad. El peso de la arista representa la cantidad de tiempo requerido para completar la actividad. Cada nodo representa un evento: la terminación del conjunto de actividades representadas por las aristas que llegan a cada nodo. Ninguna actividad en un arco de salida puede comenzarse hasta que todas las actividades en los arcos de entrada a un nodo se hayan completado. (Variaciones en el grafo de actividades básica permiten que una terminación del evento sea señalada por un subconjunto de aristas de actividades incidentes.) Consideraremos grafos de actividades con más detalle en una discusión posterior sobre análisis de trayectorias o rectas críticas.

REPRESENTACIONES LIGADAS

La representación de un grafo, usando la técnica de matriz de adyacencias, requiere del almacenamiento de información de las aristas, para cada posible par de nodos. Para una gráfica de N nodos, con pocas aristas en relación a las N^2 posibles conexiones, una representación ligada resulta ser más adecuada.

Un programador nuevo que trabaja sobre un proyecto en el centro de cómputo de la ciudad, descubrió con mucho esfuerzo los beneficios obtenibles al seleccionar la representación adecuada de una estructura de datos para su aplicación. Era el responsable de un programa en COBOL que iba a ser parte de un sistema de transporte. El programa iba a usarse para vigilar los volúmenes de tráfico en las calles principales de la ciudad. Los nodos del grafo representarían las intersecciones y las aristas representarían las calles. Los pesos de las aristas representarían los volúmenes de tráfico. Cada arista para una calle de un solo sentido tenía un solo valor de peso; para una calle de doble sentido, tenía dos valores de peso. Los datos recolectados correspondían a 500 intersecciones. El programador utilizó una matriz de adyacencias para representar el grafo de tráfico. Pocas de estas intersecciones representaban la confluencia de más de cuatro calles por lo tanto, gran parte de la matriz de adyacencias contenía ceros y los requerimientos de almacenamiento del programa le impedían caber en la memoria principal de la computadora de la ciudad. El programador no se percató de que el problema era la presentación de la matriz y gastó muchas horas compactando el código de la *DIVISION DE PROCEDIMIENTOS*. De esta forma la fecha de terminación del programa fue rebasada varias veces sin que el código pudiera probarse. Es obvio, que despidieron al programador novato, y en su lugar un programador experimentado encontró la horrible codificación en su escritorio, identificó el gran volumen de almacenamiento requerido por la matriz, convirtió el programa para usar una representación ligada y finalmente, completó el proyecto.

En comparación con la técnica de matriz de adyacencias, la cual almacena información de cada posible arista la representación ligada solamente almacena la información de las aristas que existen. A medida que las aristas se suman o se borran de la gráfica ligada, se debe modificar la representación en consecuencia. Existen dos tipos fundamentales de estructuras ligadas para representar grafos: una se llama representación de directorio de nodos y la otra representación de multi-lista.

Representación de directorio de nodos

La *representación de directorio de nodos* incluye dos partes: un directorio y un conjunto de listas ligadas. Hay una entrada en el directorio para cada nodo del grafo. La entrada del directorio para el nodo *i* apunta a una lista ligada, que representa los nodos que están conectados al nodo *i*. Cada registro de la lista ligada tiene dos campos: el primero es una identificación del nodo, y el otro es una liga al siguiente elemento de la lista. El directorio representa nodos y, la lista ligada representa aristas.

Una representación del directorio de nodos del grafo no dirigido de la figura 7-7 se da en la figura 7-11.

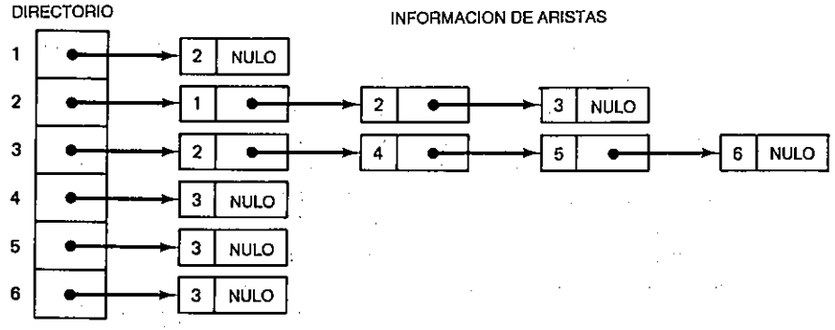

Figura 7-11 Representación de directorio de nodos de la figura. 7-7.

Un grafo no dirigido de orden *N* con *E* aristas, requiere de N entradas en el directorio y 2**E* entradas de listas ligadas, excepto cuando existen bucles en cuyo caso se reduce el número de entradas de listas ligadas en uno (¿por qué?).

Una representación del directorio de nodos del grafo dirigido de la figura 7-8 se da en la figura 7-12.

Un grafo dirigido de orden *N* con *E* aristas, requiere de *N* entradas en el directorio y *E* entradas de listas ligadas.

La lista ligada encabezada por la *i*-ésima entrada del directorio corresponde al *i*-ésimo renglón de la matriz de adyacencia. Las entradas del directorio se ordenan secuencialmente de acuerdo al identificador del nodo. Aunque también hemos ordenado las entradas de la lista ligada según el nodo identificador, podrían haber aparecido en cualquier orden, puesto que cada entrada contiene la identidad del nodo.

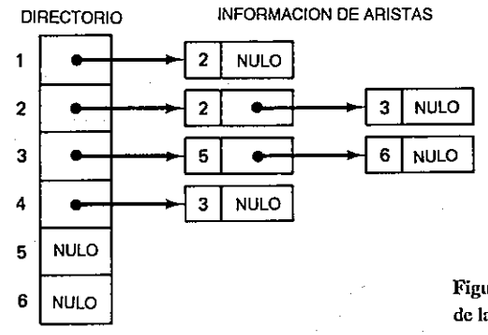

Figura 7-12 Representación de directorio de nodos de la figura 7-8.

Una declaración en COBOL de la representación del directorio de nodos, para un grafo no dirigido, de orden 24 con un máximo de 100 aristas es:

```
01 GRAFO.
   02 NODO OCCURS 24 TIMES
                PICTURE 9(3).
   02 ARCOS.
      03 ENTRADAS-DE-LISTA OCCURS 200 TIMES.
         04 NODO-ID PICTURE 99.
         04 SIG-ENTRADA PICTURE 9(3).
```

Este método requiere de *N* entradas en el directorio y 2*máx(*E*) entradas en un arreglo, donde *N* es el orden de la gráfica y máx (*E*) es el número máximo de aristas. Máx(*E*) podría ser tan grande como *N*(*N* – 1) originando que el total de entradas en el directorio y el arreglo sea de N^2, lo cual es lo mismo que si se hubiera utilizado una matriz de adyacencias.

El uso de un lenguaje, como Pascal, el cual maneja asignación dinámica de memoria puede reducir los requerimientos de espacio. Un grafo no dirigido de orden 24 y un máximo de 100 aristas puede declararse en Pascal de la siguiente forma:

```
type id-nodo = 0 . . 24;
     apt-arco = ↑ info-arco;
     info-arco = record nodo: nodo-identificador;
                        sig-nodo: apt-arco end;
     directorio = array [1 . . 24] of apt-arco;
```

Nótese que el número máximo de arcos es inmaterial en la declaración; los registros info-arco se asignan a medida que se insertan nodos en la gráfica. Este método requiere de *N* entradas en el directorio y 2**E* registros de info-arco, donde *N* es el orden del grafo y *E* es el número de arcos. La implantación es igual al diseño de lista-ligada, presentado anteriormente en esta sección. La implantación del arreglo en COBOL se aproxima al diseño de lista-ligada.

Aristas ponderadas

Representar un grafo con aristas ponderadas, requiere reservar espacio en la estructura de datos para el almacenamiento dé los pesos. La figura 7-13 muestra un grafo dirigido con aristas ponderadas. Este ejemplo en particular es el de un grafo de actividades: cada nodo representa un evento y cada arista representa una tarea que, al completarse ayuda a disparar el siguiente evento, que inicia otras tareas. Cada peso de arista es su tiempo requerido. Esta clase de grafo es comúnmente usada en sistemas de administración de proyectos.

Una representación de directorio de nodos para este grafo se muestra en la figura 7-14. El registro para cada entrada de arista contiene el identificador del nodo destino, el peso de la arista y un apuntador al siguiente nodo, que tenga el mismo nodo fuente.

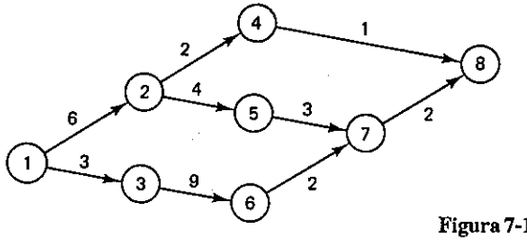

Figura 7-13 Ejemplo de un grafo de actividades.

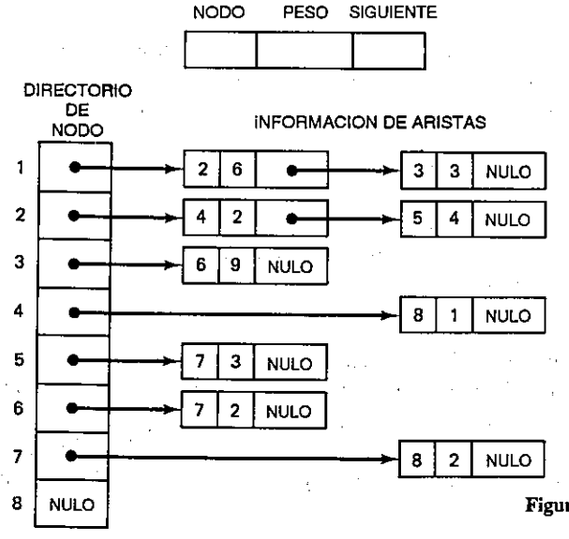

Figura 7-14 Representación de directorio de nodos de la figura 7-13.

Cálculo de aristas

Para determinar el grado de un nodo, en un grafo no dirigido, se requiere contar el número de entradas en su lista ligada. El grado externo de un nodo, en un grafo dirigido, también puede ser determinado contando el número de entradas en su lista ligada.

La determinación del grado-interno del nodo *i* es más compleja. Cada una de las listas ligadas se debe accesar para detectar cuando aparece un nodo *i* como nodo destino. Para facilitar la localización de nodos anteriores y la determinación de los grados internos puede utilizarse un directorio auxiliar de aristas que llegan a los nodos.

Algunos tipos de análisis de grafos se facilitan más al usar matrices de adyacencia, en lugar de listas ligadas. Sin embargo, si el espacio de almacenamiento fuese el único criterio de selección, entonces, la representación ligada requerirá en general de menos espacio, a menos que el grafo esté altamente conectado. Si un grafo tiene una estructura altamente volátil, entonces puede requerir menos trabajo el agregar o borrar aristas en una representación de matriz de adyacencia, que en la de un directorio de nodos. Cambiar una entrada a una matriz es, generalmente más rápido que agregar (o borrar) otras entradas en una lista ligada.

Representación de multi-lista

En una *representación de multi-lista* de estructuras de grafos, hay que considerar de nuevo dos partes: un directorio de información de nodos y un conjunto de listas ligadas de información de aristas. Hay una entrada en el directorio de nodos para cada nodo de el grafo. La entrada del directorio para el nodo *i* apunta a una lista ligada de adyacencia para el nodo *i*. Cada registro de la lista ligada aparece en dos listas de adyacencia: una para cada nodo en cada extremo de la arista representada. Usando la estructura de datos de la figura 7-15, para cada entrada de la arista (v_i,v_j),

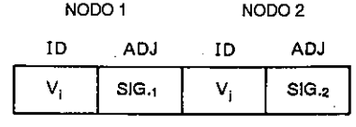

Figura 7-15 Estructura de datos para la arista (v_i, v_j).

una representación de multi-lista para el ejemplo del grafo mostrado en la figura 7-7, se presenta en la figura 7-16. Esta no es la única representación de multi-lista para este grafo, el cual está basado en el conjunto de aristas {(1,2),(2,2),(2,3),(4,3),(3,5),(3,6)}. Una representación alternativa, basada en el conjunto de aristas {(2,1),(2,2),(2,3), (3,4),(5,3),(3,6)} se muestra en la figura 7-17. Si el grafo fuera dirigido, no habría tal flexibilidad en la selección de identificadores para las aristas.

Un peso de arista puede almacenarse con la demás información registrada para la arista, tal y como se hizo en los ejemplos anteriores de representaciones de grafos.

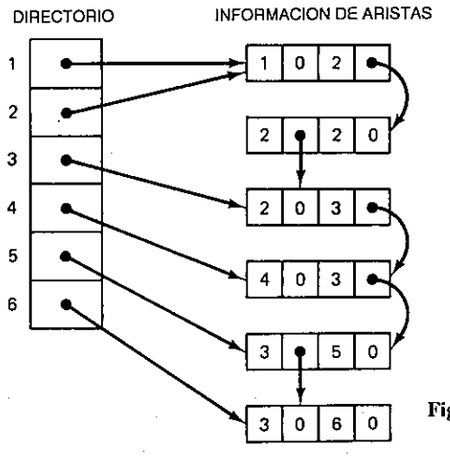

Figura 7-16 Representación multi-lista de la figura 7-7.

Una declaración en Pascal de una representación multi-lista para un grafo ponderado de orden 24, es la que sigue:

```
type id-nodo = 0 . . 24;
     tipo-nodo = ↑ info-arco;
     apt-arco = ↑ info-arco;
     info-arco = record nodo-1 : id-nodo;
                        nodo-2 : id-nodo;
                        lista-adj-1 : apt-arco;
                        lista-adj-2 : apt-arco;
                        peso : integer
                 end;
grafo = array [1 . . 24] of tipo-nodo;
```

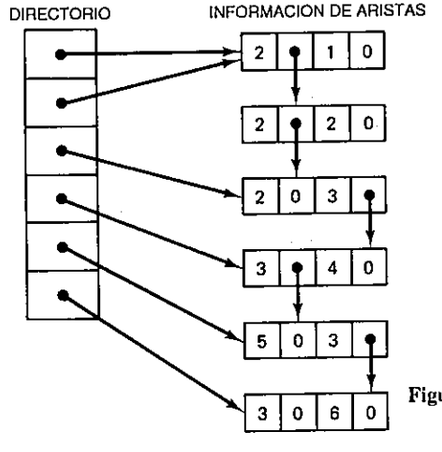

Figura 7-17 Otra representación de multi-lista de la figura 7-7.

RECORRIDO DE GRAFOS

En muchas aplicaciones es necesario visitar todos los nodos de un grafo por ejemplo. Al necesitar imprimir todos los eventos de un grafo de actividades (por ejemplo, figura 7-13), o para determinar qué ciudades están influidas en un mapa de distancias (Figura 7-9), o determinar total entre ciudades. En el mapa de distancias.

Las dos técnicas básicas de recorrido de grafos que representamos aquí son: recorrido en amplitud y recorrido profundidad. Con frecuencia es necesario tomar precauciones para visitar cada nodo y cada arista sólo una vez. El visitar un nodo más de una vez, ocasionaría listarlo más de una vez y pasar por una misma arista puede hacer que se repitan nuestros pasos a través del grafo: pueden existir varias trayectorias entre cualquier par de nodos. Los algoritmos de recorrido de grafos por lo general *marcan* cada nodo a medida que es visitado para no volver a visitarlo. Al mismo tiempo, un algoritmo de recorrido de gráficas puede marcar cada arista cuando ésta haya sido recorrida. Una arista marcada previamente no podrá quedar incluida en otra trayectoria. Los "bits de Marca" se pueden almacenar con la demás información sobre nodos o aristas.

Recorrido en amplitud

En un *recorrido en amplitud* de grafos, un nodo se selecciona como posición inicial, éste se visita y se marca, después, todos los nodos no visitados, adyacentes a ese nodo, visitan y se marcan en algún orden secuencial. Por último, los nodos no visitados, inmediatamente adyacentes a esos nodos, se visitan y marcan, y así sucesivamente, hasta que todos los nodos del grafo hayan sido visitados en el recorrido.

El recorrido en amplitud del grafo (Figura 7-13), resulta en el orden siguiente: 1, 2, 3, 4, 5, 6, 7, 8. La secuencia 1, 3, 2, 6, 5, 4, 7, 8 también es un orden de visita válida para el recorrido en amplitud.

El algoritmo de recorrido utiliza una cola para almacenar los nodos, de cada "nivel" del grafo que se visita. Estos nodos almacenados son tratados uno por uno, luego sus nodos adyacentes son visitados, y así sucesivamente hasta que todos los nodos hayan sido visitados. Esta condición de terminación se alcanza cuando la cola queda vacía. Un diagrama de flujo de algoritmo se da en la figura 7-18.

El algoritmo puede implantarse en Pascal como sigue. Suponga que la representación del directorio de nodos, ya explicada anteriormente, es utilizada, con bits de marca agregados a la información del nodo. La estructura de datos tipo cola y los procedimientos para insertar y suprimir se definieron en el capítulo 5.

```
type id-nodo = 0 . . orden-grafo;
     apt-arco = ↑ info-arco;
     tipo-nodo = record marca : 0 . . 1;
                        lista-ady : apt-arco
                 end;
     info-arco = record nodo : id-nodo;
                        peso : integer;
                        sig : apt-arco
```

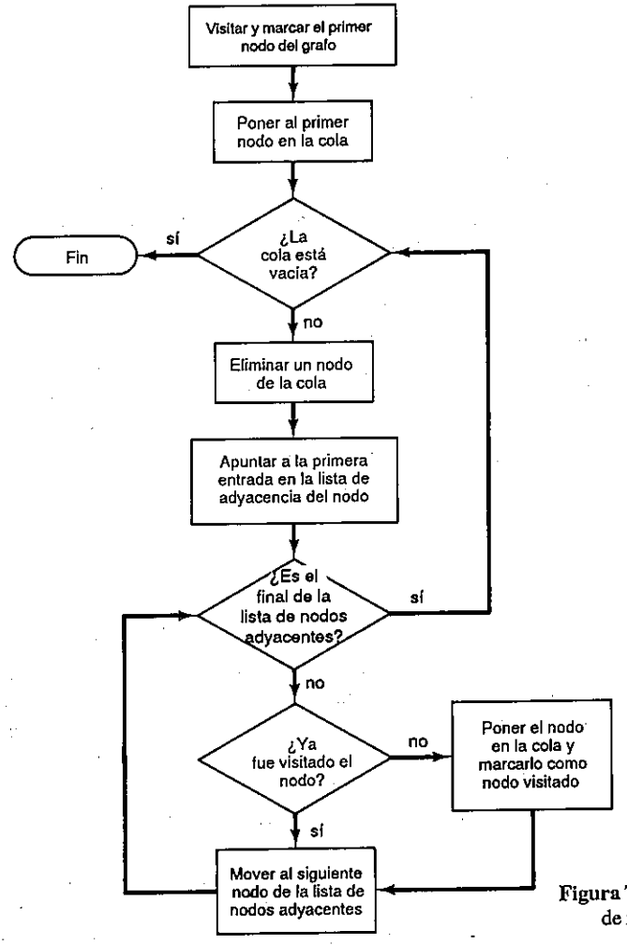

Figura 7-18 Diagrama de flujo de algoritmo de recorrido de grafos en amplitud.

```
                end;
        tipo-grafo = array [1 . . orden-grafo] of tipo-nodo;
var gráfica : tipo-grafo;
        primer-nodo : id-nodo;
```

El procedimiento iterativo puede programarse como sigue:

```
procedure amplitud (primer-nodo:id-nodo);
var : estructura-cola;
        nodo-guardado : id-nodo;
        apt-ady : apt-arco;
begin {aquí visita al primer nodo}
        grafo[primer-nodo].marca := 1;
        insertar(primer-nodo);
```



```
while g.frente <> 0
do begin suprimir (nodo-guardado);
        apt-ady := grafo[nodo-guardado].lista-ady;
        {visita el nodo adyacente al nodo-guardado}
        while apt-ady <> nil
        do begin nodo-guardado := apt-ady ↑ .nodo;
                if (grafo[nodo-guardado].marca = 0)
                then begin insertar(nodo-guardado);
                        {aquí se visita el nodo-guardado}
                        grafo[nodo-guardado].marca := 1
                    end;
                apt-ady := apt-ady ↑ .sig
            end;
    end;
end;
```

Recorrido en profundidad

Mientras que el recorrido en amplitud procede nivel por nivel, el recorrido en profundidad sigue primero una trayectoria desde el nodo inicial hasta un nodo terminal, después otra trayectoria desde el mismo punto inicial hasta alcanzar otro nodo final, y así sucesivamente hasta que todos los nodos hayan sido visitados.

El recorrido en profundidad de la gráfica mostrada en la figura 7-13, resulta en una visita de los nodos en el orden siguiente: 1, 2, 4, 8, 5, 7, 3, 6. Una trayectoria se sigue hasta que todos los nodos no visitados hayan sido alcanzados; después, el algoritmo regresa al último nodo que fue visitado y que tiene un nodo adyacente no visitado. Una secuencia igualmente válida, como resultado del recorrido en profundidad es el siguiente: 1, 3, 6, 7, 8, 2, 5, 4.

El recorrido en amplitud se describió fácilmente usando un procedimiento iterativo; el recorrido en profundidad conduce hacia una definición recursiva. El diagrama de flujo para un recorrido en profundidad recursivo se da en la figura 7-19. El código en Pascal para implantar el recorrido de gráficas en profundidad es como sigue:

```
procedure profundidad (var este-nodo:id-nodo);
var nodo-guardado : id-nodo;
    apt-ady : apt-arco;
begin {este-nodo es visitado aquí}
        grafo[este-nodo].marca := 1;
        apt-ady := grafo [este-nodo].lista-ady;
        while apt-ady <> nil
          do begin nodo-guardado := apt-ady ↑ .nodo;
                if (grafo[nodo-guardado].marca = 0)
                then profundidad(nodo-guardado);
                apt-ady := apt-ady ↑ .sig
             end;
end;
```

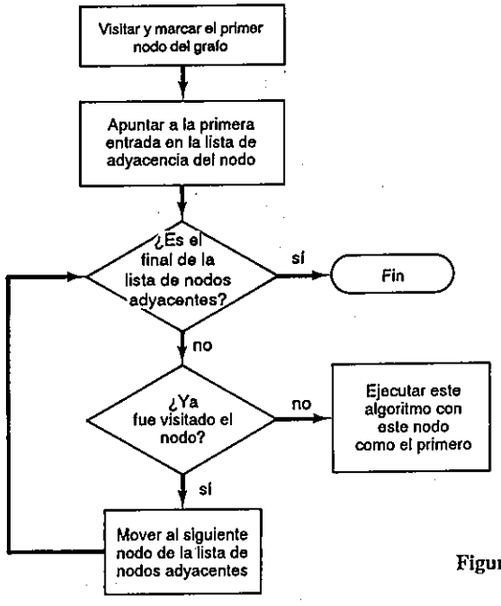

Figura 7-19 Diagrama de flujo para el algoritmo recursivo de recorrido en profundidad.

Comparaciones

Los ejercicios al final de este capítulo, sugieren que escriba los algoritmos de recorrido en amplitud y profundidad para grafos que estén representado por estructuras multi-lista y para grafos que estén representados por matrices de adyacencia. Ahora, compararemos el tiempo requerido para estos recorridos usando representaciones ligadas y no ligadas.

En el recorrido en amplitud, cada nodo de la gráfica entra en la cola sólo una vez. Así el ciclo externo *while* se ejecuta N veces, donde N es el orden del grafo. Si el grafo está representado por una estructura ligada, entonces solamente aquellos nodos que están adyacentes al nodo que está al frente de la cola se examinan. Después, el ciclo interno *while* se ejecuta en total E veces, donde E es el número de aristas en el grafo. El recorrido en amplitud, de una estructura ligada, es por lo tanto de orden $O(N*E)$.

Sin embargo, si el grafo está representado por una matriz de adyacencia, entonces el ciclo interno *while* se ejecuta una vez por cada nodo en el grafo, por lo tanto el renglón completo de la matriz de adyacencia deberá examinarse. Por lo que el recorrido en amplitud del grafo será de orden $O(N^2)$.

Se puede aplicar el mismo tipo de lógica para mostrar que el recorrido en profundidad de un grafo, representado por una estructura ligada, es también de $O(N*E)$ y el recorrido en profundidad de un grafo representado por una matriz de adyacencia es también de $O(N^2)$.

Por tanto, las representaciones ligadas economizan espacio y producen recorridos más cortos, hasta que el grafo se encuentre más densamente conectado. Mientras que

representaciones ligadas son más eficientes. Recordemos que el valor máximo para E es $N^2 - N$, lo cual ocurre cuando hay una arista entre cada par de nodos.

¿Cómo podemos determinar cuándo un recorrido en profundidad o amplitud es apropiado? Esta elección está típicamente determinada por la lógica de la aplicación.

ALCANCE Y TRAYECTORIAS MAS CORTAS

El análisis de grafos con frecuencia involucra recorrer trayectorias a través del grafo. Existen muchos problemas interesantes en el análisis de trayectorias de grafos, de los cuales consideraremos sólo dos en esta sección: alcance y trayectoria más cortas. El uso de matrices de adyacencia facilitará la discusión inicial.

En algunos problemas el peso de las aristas es importante. Por ejemplo, para determinar la trayectoria más corta entre un par de ciudades del grafo mostrado en la figura 7-9, se utilizan los pesos de las aristas, los cuales son las distancias. Por otra parte, para determinar si están o no, conectados dos nodos no se necesita considerar el peso de la arista. Inicialmente, consideremos que el peso de las aristas es, para todos, de uno.

Alcance

La figura 7-20 representa la matriz de adyacencia para el grafo de la figura 7-9. Una entrada diferente de cero para un par de ciudades indica que hay una ruta entre ellas sin

i \ j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
3	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0
4	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
5	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
9	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
14	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0
15	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

Figura 7-20 Matriz de adyacencia para la figura 7-9.

ciudades intermedias. Al multiplicar la matriz de adyacencia por sí misma nos mostrará para que par de ciudades existe una ruta con una ciudad intermedia. Llamemos a la matriz de adyacencia A. La multiplicación de matrices convencional se usa para encontrar A^2. Es decir, el elemento i,j-ésima de A^2 es el resultado de multiplicar el renglón i por la columna j de A:

$$A_{ij}^2 = \sum_{k=1}^{n} A_{ik}A_{kj}$$

$A_{ij}^2 = 1$, si hay una trayectoria de longitud 2 desde el nodo i al nodo j; $A_{ij}^2 = 0$, en cualquier otro caso.

En general, la m-ésima potencia de la matriz de adyacencia binaria es:

$$A_{ij}^m = \begin{cases} 1 \text{ si hay una ruta de longitud } m \text{ desde el nodo } i \text{ al } j \\ 0 \text{ en otro caso.} \end{cases}$$

Para el ejemplo de distancias entre ciudades, A^m mostrará qué pares de ciudades tienen rutas que requieren paradas en las $m - 1$ ciudades intermedias. El problema de determinar cuando un par de nodos están, o no, conectados, se resuelve entonces encontrando:

$$\sum_{t=1}^{N} A^t$$

cuyo resultado es una matriz llamada *cerradura transitiva* de A, con frecuencia denotada por A^+. En realidad no se necesita realizar la suma N veces, donde N es el orden de la gráfica, sino sólo tantas veces como la longitud de la trayectoria más larga. La cerradura transitiva de A es también denominada como *matriz de alcance* de A.

Trayectorias más cortas

Además de determinar si el nodo j se puede alcanzar desde el nodo i, es de común interés encontrar la trayectoria más corta desde el nodo i al nodo j. El desarrollo del algoritmo se deja como ejercicio y sólo haremos un par de observaciones acerca de la trayectoria más corta aquí. Estos puntos de vista le ayudarán a desarrollarlo. Refiriéndonos, otra vez, a la figura 7-9, la tabla de la página 159 proporciona las trayectorias más cortas desde DNV a cada ciudad, listadas en el orden en el cual un algoritmo de la trayectoria más corta, las detectaría.

Hay varias rutas posibles entre muchos pares de ciudades. Note que:

1. Las trayectorias más cortas se detectan en orden no decreciente de sus distancias; es decir, el siguiente nodo destino es aquél con la distancia mínima desde el nodo fuente de todos aquellos nodos que aún no son seleccionados.
2. La trayectoria más corta al siguiente nodo destino pasa a través de los nodos que ya fueron seleccionados.

Si los pesos de aristas tuviesen que representar costos en lugar de distancias, entonces el mismo algoritmo podría usarse para detectar las trayectorias más económicas. Si estos

Trayectoria	Distancia
DNV-ABQ	334
DNV-SLC	371
DNV-KNC	558
DNV-ABQ-ELP	334 + 229 = 563
DNV-ABQ-PHX	334 + 330 = 664
DNV-SLC-SPK	371 + 550 = 921
DLV-SLC-SFO	371 + 600 = 971
DNV-KNC-ORD	558 + 414 = 972
DNV-ABQ-PHX-LAX	334 + 330 + 357 = 1021
DNV-ABQ-ELP-DFW	334 + 229 + 572 = 1135
DNV-SLC-SPK-SEA	371 + 550 + 229 = 1150
DNV-KNC-ORD-CVG	558 + 414 + 252 = 1224
DNV-ABQ-ELP-DFW-HST	334 + 229 + 572 + 225 = 1360
DNV-KNC-ORD-CVG-NSH	558 + 414 + 252 + 238 = 1462
DNV-ABQ-ELP-DFW-NOR	334 + 229 + 572 + 443 = 1578

pesos representasen velocidades (por ejemplo, kilobits/seg), entonces el algoritmo podría modificarse fácilmente para detectar las trayectorias más rápidas.

RUTAS CRITICAS

El grafo de la figura 7-13 representa una gráfica de actividades. Casi todos los proyectos se pueden representar por estos grafos. Se han desarrollado técnicas bastante buenas para ayudar a la evaluación y análisis de gráficas de actividades, por ejemplo, el Método de la Ruta Crítica (MRC), la Técnica de Evaluación Desempeño y Revisión (TEDR), y la Técnica de Asignación de Recursos y Planeación de Multi-proyectos (TARPM). Aquí, sólo introduciremos los aspectos básicos del análisis de grafos de actividades.

La estructura de grafo de un proyecto puede mostrar que varias tareas pueden ejecutarse en paralelo. Por ejemplo, las tareas desde los eventos 2 y 3 pueden proceder al mismo tiempo (Figura 7-13). El interés del manejo de proyectos, es el de determinar cuáles eventos son críticos para terminar el proyecto oportunamente. Un evento es *crítico* si, el retraso en su terminación, retrasa la terminación de todo el proyecto. El tiempo más corto de terminación posible para el proyecto es la trayectoria más larga a través del grafo. A esta trayectoria más larga se le llama *ruta crítica*; y los eventos críticos se encuentran a lo largo de esta ruta. A la longitud de esta ruta se le llama *tiempo de la ruta crítica* (TRC). Un sistema de información podría incluir programas para encontrar rutas críticas y TRC, así como diseñar y modificar los grafos de actividades.

Una forma de detectar la ruta crítica y determinar los tiempos de corrimiento permisibles para los eventos que no están sobre la ruta crítica, consiste en encontrar los tiempo de inicio más tempranos y los más tardíos de cada evento. El *tiempo de inicio más temprano* del evento i (EST_i), es el tiempo de arranque más próximo posible para dicho evento. Para el grafo de la figura 7-13.

Evento	EST
1	0
2	6
3	3
4	8
5	10
6	12
7	14
8	16

El EST_i se calcula a partir de la trayectoria más larga desde el nodo de inicio hasta el nodo i. Nótese que las tareas desde los nodos 5 y 6 se deberán completar antes de que arranque el evento 7. Más formalmente,

$$EST_i = \text{máx}_{j \text{ pertenece a } P(i)} \{EST_j + T(j,i)\}$$

donde $P(i)$ es el conjunto de predecesores inmediatos del nodo i
$T(j,i)$ es el peso de la arista desde el nodo j al nodo i.

El *tiempo de inicio más tardío* del evento i (LST_i), es el tiempo más tardío posible en el cual el evento i puede ser arrancado para que el grafo se complete en tiempo crítico. El conjunto de LST_i se calcula hacia atrás a partir del último evento. Para el grafo de la figura 7-13.

Evento	LST
1	0
2	13
3	3
4	15
5	11
6	12
7	14
8	16

Formalmente, $LST_i = \text{mín}_{j \text{ pertenece a } S(i)} \{LST_j - T(i,j)\}$

donde $S(i)$ es el conjunto de sucesores inmediatos del nodo i.

La diferencia entre el LST_i y el EST_i es el corrimiento permisible del evento i.

Evento	LST-EST
1	0
2	7
3	0
4	7
5	1
6	0
7	0
8	0

Por lo tanto, los eventos críticos están representados por los nodos 1, 3, 6, 7, y 8, los cuales se encuentran sobre la trayectoria más larga del grafo. Estos son los eventos que necesitan controlarse para completar a tiempo el proyecto. Nótese que, si un evento no crítico tuviese un corrimiento mayor al que le es permitido, entonces debería cambiarse la ruta crítica del grafo.

ARBOLES DE EXPANSION

Un *árbol de expansión* es un árbol que contiene todos los nodos de un grafo y ningún otro más. Muchas aplicaciones exigen la identificación de un árbol de expansión para una gráfica conexa. Las figuras 7-21 y 7-22 muestran dos de los muchos árboles de expansión correspondientes al grafo de la figura 7-9. Cada árbol de expansión muestra una manera de planear las rutas de transporte de tal forma que cada ciudad pueda ser atendida. ¿Podría usted diseñar otras?

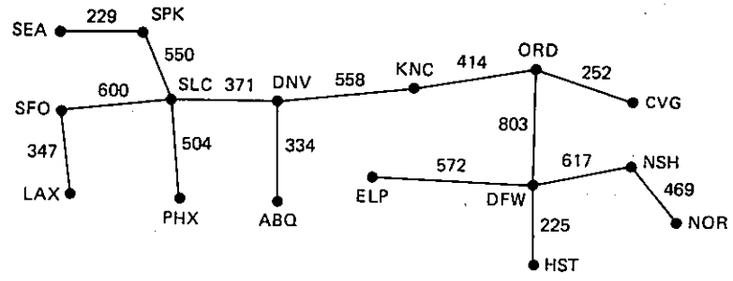

Figura 7-21 Arbol de expansión de la figura 7-9.

Es de interés primordial identificar un *árbol de expansión mínima* para un grafo. El costo de un árbol de expansión es la suma de los pesos de sus aristas. El costo del árbol de expansión de la figura 7-21 es de 6845 millas; el costo del árbol de expansión de la figura 7-22 es de 6567 millas. Ninguno de estos costos parece corresponder al árbol de expansión con el menor costo posible.

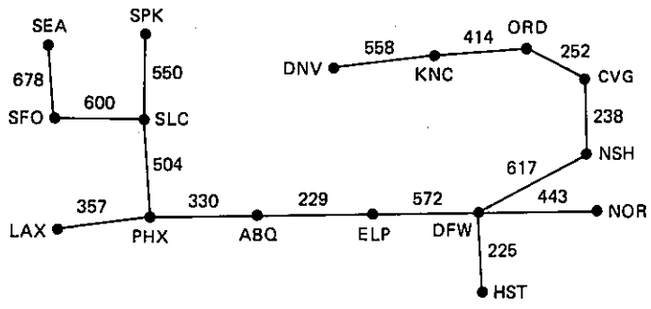

Figura 7-22 Otro árbol de expansión de la figura 7-9.

Algoritmo de Kruskal

Un método para encontrar un árbol de expansión mínima es un algoritmo desarrollado por Kruskal. Este algoritmo considera la inclusión de las aristas del grafo en orden, por costos crecientes; una arista se incluye si no forma un ciclo. Un ciclo significaría que existen dos rutas entre un par de nodos. Al aplicar el algoritmo de Kruskal al grafo de la figura 7-9 se procede de la siguiente manera:

1. Menor costo: DFW-HST (225)
2. Menor costo: ABQ-ELP (229)
3. Menor costo: SEA-SPK (229)
4. Menor costo: CVG-NSH (238)
5. Menor costo: ORD-CVG (252)

El desarrollo del árbol de expansión mínima hasta el momento se muestra en la figura 7-23a).

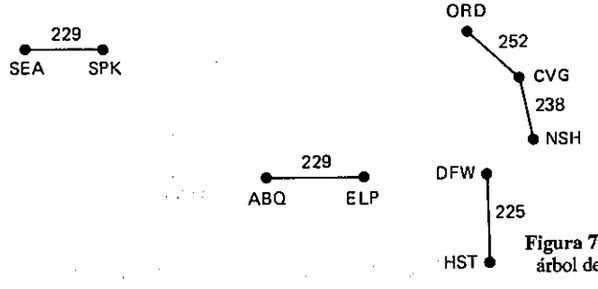

Figura 7-23a) Pasos iniciales para encontrar un árbol de expansión mínima para la figura 7-9.

6. Menor costo: PHX-ABQ (330)
7. Menor costo: DNV-ABQ (334)
8. Menor costo: SFO-LAX (347)

9. Menor costo: LAX-PHX (357)
10. Menor costo: KNC-ORD (414)

El árbol de expansión mínima ahora, se muestra en la figura 7-23b).

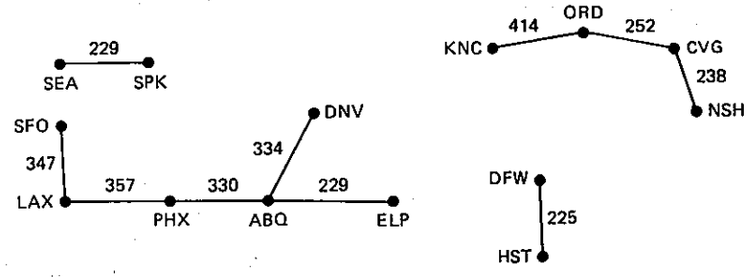

Figura 7-23b) Pasos iniciales para encontrar un árbol de expansión mínima para la figura 7-9.

11. Menor costo: DFW-NOR (443)
12. Menor costo: NSH-NOR (469)
13. Menor costo: PHX-SLC (504)
14. Menor costo: SLC-SPK (550)
15. Menor costo: DNV-KNC (558)

¡Hemos tenido mucha suerte de no encontrar ciclos! El árbol de expansión mínima se muestra en la figura 7-23c).

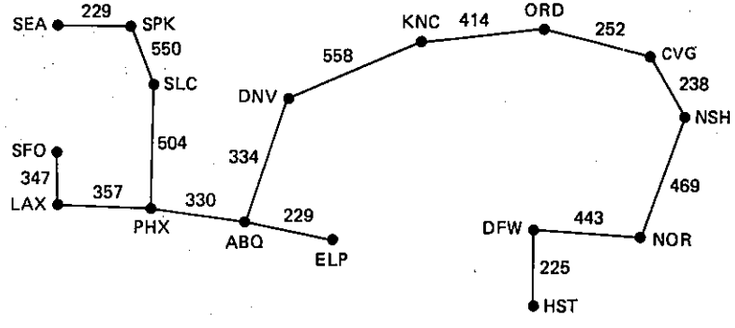

Figura 7-23c) Arbol de expansión mínima para la figura 7-9.

En efecto hemos incluido todos los nodos y el árbol de expansión mínima está completo, con un costo de 5479.

Supongamos por el momento, que todos los nodos aún *no* hubieran sido incluidos. Considerar que la siguiente arista de menor costo (ELP-DFW) detectaría un ciclo. Esta arista sería rechazada y, la arista con el siguiente costo mínimo sería considerado, y así sucesivamente hasta que todos los nodos fuesen incluidos.

Los árboles de expansión mínima representan, por lo tanto, la manera de menor costo para conectar los nodos de un grafo. El costo, por supuesto, puede ser cuantificado en términos de distancia, tiempo, dinero u otros términos cuantitativos, dependiendo de las unidades de las aristas. En la lista de Referencias Sugeridas, al final de este capítulo, hay información que contiene varios algoritmos para encontrar árboles de expansión mínima.

RESUMEN

En este capítulo hemos analizado los grafos con cierto grado de detalle. Primero introdujimos los conceptos básicos de las estructuras de datos de grafos. Se presentaron tres técnicas fundamentales para la representación de grafos: la representación de matriz de adyacencia y dos formas de representación ligada—directorio de nodos y multi-listas. El uso de matrices de adyacencia tiende a hacer que los algoritmos de análisis de grafos sean al menos del orden $O(N^2)$, mientras que el uso de representaciones ligadas hace que los algoritmos de análisis de grafos sean de orden $O(N*E)$. Se proporcionaron algunos ejemplos de declaración de gráficas en COBOL y Pascal.

Se introdujeron dos técnicas básicas de recorrido de grafos: en amplitud y en profundidad. Estos algoritmos cuidan de visitar cada nodo sólo una vez, marcando el nodo cada vez que es visitado y cada arista cuando es recorrida. El recorrido en amplitud por lo general se programa como un procedimiento iterativo, mientras que el recorrido en profundidad se programa generalmente como un procedimiento recursivo.

Se analizaron varios problemas de análisis de trayectorias. Las potencias de las matrices de adyacencia de grafos se pueden usar, para detectar trayectorias de una longitud dada entre pares de nodos, y para determinar si un nodo es alcanzable desde otro nodo. Para ilustrar un algoritmo que permite detectar las trayectorias más cortas entre un nodo y los demás nodos, se utilizó un ejemplo. El análisis de la ruta crítica se introdujo, junto con los conceptos de tiempos de inicio más temprano y más tardío. Por último se introdujeron los árboles de expansión y una técnica (el algoritmo de Kruskal) para identificar un árbol de expansión mínima de un grafo.

TERMINOLOGIA

árbol de expansión
árbol de expansión mínima
arco
arista
bucle
cerradura transitiva
ciclo
grado (de un nodo)
grado externo
grado interno
grafo
grafo acíclico
grafo conexo
grafo de actividades
grafo dirigido
grafo no dirigido
grafo nulo
grafo simple
longitud de trayectoria
matriz de adyacencia
matriz de alcance
multigráfica
nodo
orden (de un grafo)
recorrido de amplitud
recorrido en profundidad
ruta crítica
tiempo de inicio más tardío
tiempo de inicio más temprano
trayectoria
vértice

REFERENCIAS SUGERIDAS

CHANDY, K. M. and J. MISRA. "Distributed computation on graphs: shortest path algorithms," *Comm. ACM*, 25(11):833–837, Nov. 1982.

CHIN, F. Y., J. LAM, and I-N. CHEN. "Efficient parallel algorithms for some graph problems," *Comm. ACM*, 25(9):659–665, Sept. 1982.

EBERT, J. "A versatile data structure for edge-oriented graph algorithms," *Comm. ACM*, 30(6):513–519, June 1987.

FORD, L. and D. FULKERSON. *Flows in Networks*. Princeton: Princeton UP, 1962.

LAWLER, E. L. "Comment on computing the k shortest paths in a graph," *Comm. ACM*, 20(8):603–604, Aug. 1977.

LITKE, J. D. "An improved solution to the traveling salesman problem with thousands of nodes," *Comm. ACM*, 27(12):1227–1236, Dec. 1984.

LOUI, R. P. "Optimal paths in graphs with stochastic or multidimensional weights," *Comm. ACM*, 26(9):670–676, Sept. 1983.

MINIEKA, E. "On computing sets of shortest paths in a graph," *Comm. ACM*, 17(6):351–353, June 1974.

QUINN, M. J. and N. DEO. "Parallel graph algorithms," *ACM Computing Surveys*, 16(3):319–348, Sept. 1984.

TARJAN, R. E. "Applications of path compression on balanced trees," *Jour. ACM*, 26(4):690–715, Oct. 1979.

WHITNEY, V. K. M. "Minimal spanning tree," *Comm. ACM*, 15(4): 273–274, April 1972.

EJERCICIOS DE REPASO

1. Supongamos que un grafo dirigido de Orden *N* se representa por una matriz de adyacencia, donde *grafo* $(i,j) = 1$, si un arco conecta los nodos *i* y *j*, y cero en los demás casos. ¿Cuáles son las expresiones para determinar lo siguiente: a) el grado interno del nodo *i*, b) el grado externo del nodo *i*?
2. Dibuje un grafo dirigido. ¿Cuál es la matriz de adyacencia? ¿Cuál es la matriz de alcance? Represente el grafo mediante una estructura de directorio de nodos. Represente el grafo mediante una estructura multi-lista. Recorra el grafo utilizando el método de amplitud y después el método de profundidad. ¿Cuál es el árbol de expansión mínima del grafo?
3. ¿Puede haber un ciclo en un grafo simple? si así es, dé un ejemplo y, si no, diga ¿por qué?

4. Dibuje varios grafos. Para cada grafo responda a las siguientes cuestiones. ¿Es el grafo: a) acíclico, b) conexo, c) dirigido, d) simple?, e) ¿cuál es el orden del grafo?, ¿cuál es: f) el grado interno, g) el grado externo, h) el grado de cada nodo?
5. ¿Qué estructura de datos es típicamente apropiada para almacenar una matriz de adyacencia?
6. Una representación ligada de un grafo mantiene un directorio de nodos y listas ligadas de las aristas correspondientes. Desarrolle una representación ligada alternativa que mantenga un directorio de aristas y sus correspondientes listas ligadas de nodos. Analice los requerimientos de almacenamiento para su representación y compárelos con los requerimientos de la representación ligada dada en el capítulo.
7. Escriba programas para recorrer: a) en profundidad y b) en amplitud un grafo representado por una estructura multi lista.
8. Escriba programas para recorrer a) en profundidad y b) en amplitud una gráfica representada por una matriz de adyacencia.
9. Escriba un programa para encontrar la ruta crítica de un grafo ponderado, representado por a) una matriz de adyacencia, b) una estructura de directorio de nodos, c) una estructura multi-lista.
10. Desarrolle un grafo de actividades para sus propias actividades de la siguiente semana o siguiente mes. Encuentre la ruta crítica a través de la gráfica.
11. Escriba un programa para generar el conjunto de trayectorias más cortas, desde el nodo *i* a todos los demás nodos, en un grafo de orden *n*. Suponga que el grafo está representado por a) una matriz de adyacencia, b) una estructura de directorio de nodos, y c) una estructura multi-lista.

capítulo ocho

árboles generales y binarios

Una clase importante de grafos son aquellos que son estructurados como *árboles*. Un árbol es un grafo conexo, simple y acíclico. Un árbol no contiene ni ciclos ni bucles; existe una sola arista entre cualquier par de nodos. Las gráficas de la figura 7-5 son árboles; los grafos de las figuras precedentes en el capítulo 7 no son árboles.

ARBOLES GENERALES

Nuestra atención se restringirá a la clase de árboles conocida como *árboles enraizados*. Un árbol se dice que está enraizado si tiene un nodo (llamado *raíz*), el cual se distingue de los demás nodos. La raíz del árbol T es denotada por raíz (T).

De manera más formal, un árbol T es un conjunto finito de cero o más nodos (v_1, v_2, . . . , v_n) de tal manera que:

1. Hay un nodo especialmente designado (digamos v_1) llamado Raíz(T).
2. Los nodos restantes (v_2, . . . , v_n) son particionados en $m \geq 0$ conjuntos *disjuntos*, llamados T_1, T_2, . . . , T_m tales que, cada T_i es por sí mismo un árbol.

Los conjuntos T_1, . . . , T_m son llamados *subárboles* de la Raíz(T). Note la naturaleza recursiva de esta definición; pues hemos definido árboles en términos de árboles. Un árbol sin nodos es un árbol nulo.