“El acceso a la versión digitalizada se brinda con fines académicos y estará habilitado mientras dure las restricciones asociadas a la coyuntura nacional. Tener en cuenta las consecuencias legales que se desprenden de hacer uso indebido de estos documentos, de acuerdo al D.L. 822”

3. ¿Por qué se prefiere que el alojamiento de las colas sea en arreglos circulares en lugar de arreglos lineales?
4. ¿Puede ser circular una pila? ¿Por qué sí o por qué no?
5. Al observar el algoritmo de inserción a la cola, ¿cómo podemos identificar cuándo se mueve la cola en el sentido de las manecillas del reloj, o en sentido contrario?
6. Cuándo es conveniente usar la función *mod* en colas circulares?
7. ¿Cómo se prueba si la cola está vacía? Describa su colocación del apuntador del frente.
8. Escriba un procedimiento para agregar un elemento a una cola circular cuando el primer elemento es designado FRENTE y cuando FONDO indica el espacio inmediatamente posterior al final de la cola.
9. Escriba un algoritmo para contar el número de elementos en una cola circular.
10. Escriba un algoritmo para eliminar un elemento de la cola circular cuando FONDO apunta al final de la cola y FRENTE apunta al espacio anterior al frente de la cola.
11. Escriba un algoritmo para agregar un elemento a la cola circular cuando FONDO apunta al final de la cola y FRENTE apunta al frente.
12. Dibuje una cola con 10 elementos, indicando cuál es el frente y cuál es el fondo. Escriba un algoritmo para verificar cuando la cola está llena.
13. Escriba un algoritmo para poner dos colas de longitud variable (pero que en total puedan contener N elementos) en un arreglo de N localidades.
14. Una unidad central de procesamiento (CPU) concede dos unidades de tiempo de procesamiento como máximo a cada una de las transacciones; si la tarea se completa, entonces se envía a una salida hacia un disco o una terminal y, si no, entonces se coloca en el extremo final de la cola para atenderlo después. Dado que todas las tareas son enviadas a cola para la CPU y que existen 2 prioridades —prioridad 1, la cual da preferencia, y prioridad 2, la cual se subordina a la prioridad 1 — y dadas las siguientes transacciones:

Trans	Prioridad	Tiempo de entrada	Servicio requerido
A	1	0	1
B	2	0	2
C	2	1	6
D	1	2	4
E	1	2	3
F	2	3	2
G	1	5	1

Describa la historia de las colas aplicadas y el procesamiento de la CPU en intervalos de una unidad, hasta que todo el procesamiento se haya completado.

15. a) Hay un banco con dos cajeros. Uno con experiencia y el otro sin experiencia. El tiempo promedio de transacción para el nuevo cajero es el doble que el del cajero experimentado. Escriba un algoritmo para dirigir a los clientes al cajero apropiado, de tal forma que el tiempo promedio de espera sea igual en ambas colas.
    b) Ahora, suponga que el nuevo cajero "aprende" después de cada transacción y, su tiempo de transacción disminuye $\frac{1}{n}$ (donde n = tiempo de transacción para el cajero experimentado) después de cada transacción. Escriba un algoritmo para equilibrar los tiempos de espera en cola ante esta situación.

capítulo seis

listas ligadas

En este capítulo expondremos el uso de las listas ligadas para representar estructuras lineales de datos: listas lineales en general, pilas y colas. Después enfocaremos el uso de listas ligadas para representar estructuras no lineales de datos, como: árboles y gráficas.

REPRESENTACION DE LISTAS LIGADAS

Problemas con la representación secuencial

En el capítulo anterior representamos pilas y colas alojándolas en arreglos. Una restricción básica impuesta por este modo de representación es que la cantidad reservada de almacenamiento para la pila o la cola es fija. El espacio se reserva sin tomar en cuenta si la cola o la pila están vacías y existe la posibilidad de desborde si éstas crecen más de lo esperado. Los mismos problemas se presentan cuando se utiliza un arreglo para alojar una lista lineal general.

Sin embargo, existe una restricción todavía más severa, impuesta en el caso general. ¿Qué sucede cuando el elemento 43, digamos "IGOR", se elimina de una lista lineal de 2492 nombres? Los elementos del 1 al 42 no son afectados, pero los elementos 44, 'IVAN', hasta el 2492 'ZELDA', lógicamente deben recorrerse una posición en el arreglo; el elemento 44, ahora debe ser el elemento 43, el 45 ocupar el lugar 44, . . . , y el 2492 pasar a la localidad 2491. Ahora ¿qué sucede cuando un nuevo elemento digamos "HORACIO", se deba insertar después del elemento 41, "HOMERO"? Los elementos 42, "HORACLES", hasta el 2491 "ZELDA" se deben recorrer una posición a la derecha

en el arreglo para dejar un espacio al nuevo elemento 42. El nuevo elemento no se puede agregar simplemente al final del arreglo, porque el orden lógico de los elementos en la lista lineal es transmitido por el orden físico de los elementos en el arreglo.

Representación no-secuencial

Una solución a los problemas de movimiento de los datos que se ha encontrado al utilizar representaciones secuenciales, es la de utilizar, en su lugar, una representación no-secuencial. Una representación secuencial refleja el orden lógico de los elementos físicamente almacenados en una lista lineal; en ella, el orden físico y el orden lógico son los mismos. Con la representación no-secuencial, el orden lógico y el orden físico de los elementos *no* es necesario que sea el mismo. El orden lógico se representa de tal forma que cada elemento apunta al siguiente elemento, es decir, tales elementos se encuentran ligados. Para accesar los elementos en orden lógico se siguen dichas ligas.

Conceptos básicos

La figura 6-1 muestra una lista lineal ligada que contiene cuatro elementos. Llamaremos a cada elemento de la lista *nodo*.

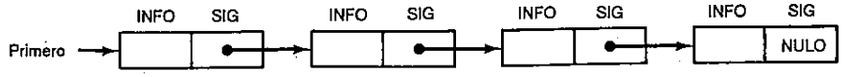

Figura 6-1 Ejemplo de lista ligada.

Hay un apuntador para el primer nodo de la lista, y cada nodo tiene una liga al siguiente nodo. El último nodo tiene un apuntador *NULO*, el cual indica que no hay ningún nodo siguiente. Cada nodo tiene dos secciones: el contenido de datos y el campo del apuntador. Cada nodo es una estructura de datos de tipo registro (véase el capítulo 3).

Los nodos ligados no tienen que estar físicamente adyacentes, tal y como puede observarse en la figura 6-2.

Una *lista vacía* es aquella que no contiene nodos. La representación de lista ligada para una lista vacía es sólo un apuntador nulo, al que llamamos "primero". La operación para crear una lista vacía es:

```
Primero := Nulo
```

Ventajas

Recordemos que otra restricción impuesta al usar arreglos, para alojar colas o pilas es que los elementos deben ser homogéneos. Esta restricción puede desaparecer al usar un esquema de representación de listas ligadas. Nodos con diferentes estructuras de datos se pueden ligar unos a otros. Los beneficios de una lista ligada aparecen cuando las operaciones de inserción y supresión se toman en consideración. Para remover el nodo

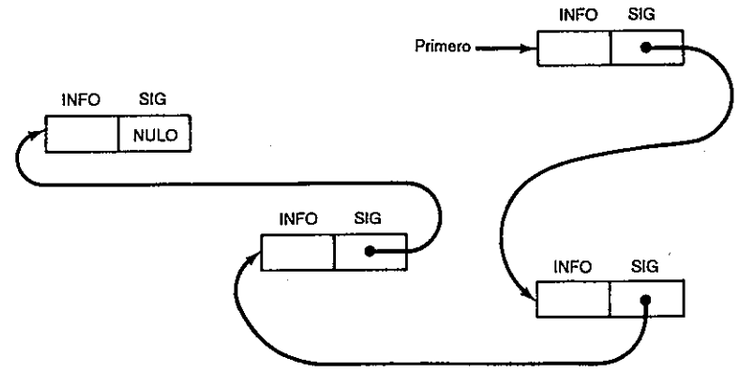

Figura 6-2 Ejemplo de lista ligada, equivalente a la figura 6-1.

43, "IGOR", de la lista 2492 nodos (Figura 6-3*a*), sólo es necesario cambiar el apuntador del nodo previo 42 para que apunte al nodo siguiente 44 (Figura 6-3*b*).

Para insertar un nuevo nodo, "HORACIO", después del nodo 41, basta con que el nodo 41 apunte al nuevo nodo y, el nuevo nodo apunte al nodo 42 (Figura 6-3*c*), sin la necesidad de recorrer los datos.

Costos

Los costos por usar una lista ligada, en lugar de una representación secuencial, son básicamente dos: Primero, el espacio requerido para una representación ligada requiere una cantidad extra para el campo del apuntador. Segundo, con frecuencia la búsqueda de un nodo en una representación ligada será más larga con respecto a la búsqueda en un arreglo que aloje a la lista. Recordemos que podemos calcular la localidad de inicio del *N*-ésimo elemento en un arreglo. Para encontrar el *N*-ésimo nodo en una lista ligada (suponiendo que no utilizamos apuntadores auxiliares), la cadena de los primeros *N*-1 nodos debe ser visitada, ya que el *N*-ésimo nodo no puede ser accesado directamente.

OPERACIONES BASICAS EN UNA LISTA LIGADA

Ahora, consideremos las operaciones básicas en listas ligadas y después veremos cómo declarar y manipular listas ligadas en programas.

Notación

Primero introduzcamos la siguiente notación: sea P una variable apuntadora, cuyo valor es una dirección, es decir, la localidad de alguna otra variable. Las operaciones válidas para las variables apuntadoras son las siguientes:

1. Examinar si un apuntador es Nulo.

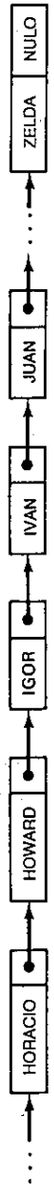

Figura 6-3a) Ejemplo de una lista ligada sencilla.

Figura 6-3b) Supresión de Igor de la lista ligada en la figura 6-3a).

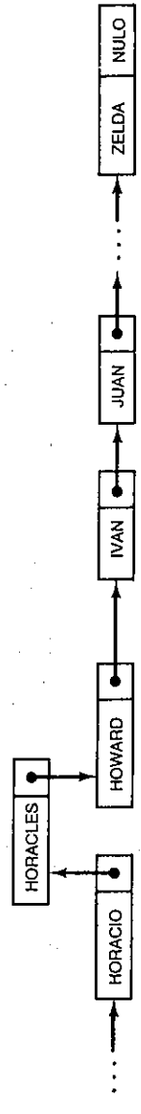

Figura 6-3c) Inserción de HORACLES en la lista ligada de la figura 6-3b).

2. Examinar la igualdad con otra variable apuntadora.
3. Inicializar con Nulo.
4. Asignar a un apuntador la dirección de un nodo.

Estas son las únicas operaciones que podemos hacer con apuntadores. Denotamos por:

Nodo(P)	el nodo apuntado por P;
Info(P)	el valor de la porción de información del nodo apuntado por P;
Sig(P)	el valor de la liga del nodo apuntado por P.

Remoción de nodos

Considere la lista ligada de la figura 6-4 *a*).

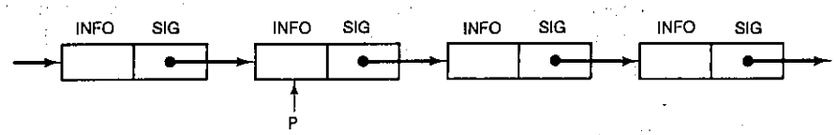

Figura 6-4a) Pasos para la supresión de un nodo de una lista ligada.

El algoritmo para remover de una lista ligada el nodo que sigue al apuntado por P, donde Q es un apuntador auxiliar usado para apuntar al nodo que debe ser removido, es el siguiente:

a) Q := Sig(P)
b) Sig(P) := Sig(Q)
c) Liberar el espacio del nodo apuntado por Q para reusarlo.

El paso a) produce la estructura mostrada en la figura 6-4b).

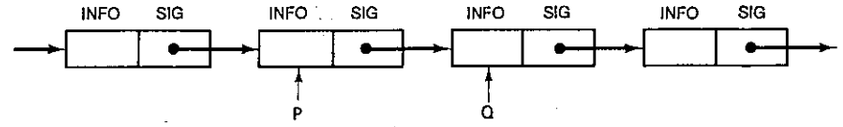

Figura 6-4b).

El paso b) produce la estructura mostrada en la figura 6-4c).

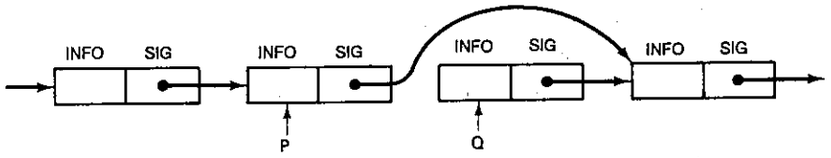

Figura 6-4c).

Y la figura 6-4d) muestra la estructura que produce el paso c). Para remover un nodo hay que referirse a dos nodos, los apuntados por P y Q; sin necesidad de accesar o cambiar ningún otro nodo

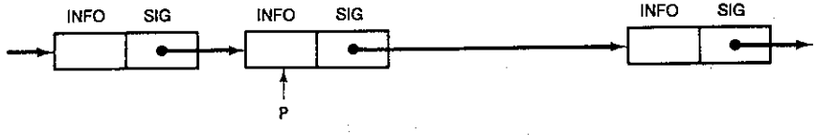

Figura 6-4d).

Esta secuencia de operaciones libera el espacio del nodo removido, excepto si P apunta al último nodo de la lista. ¿Cómo deberá modificarse el algoritmo para adecuarse a esta situación?

Inserción de un nodo

El algoritmo para insertar el contenido de una variable NOMBRE en una lista ligada, de tal forma que siga al nodo apuntado por P es el que sigue. Otra vez usamos el apuntador auxiliar Q, y otro más llamado NUEVO.

a) Asignar espacio para el nuevo nodo y hacer que NUEVO apunte a éste.
b) Info(NUEVO) := NOMBRE
c) Q := Sig(P)
d) Sig(P) := NUEVO
e) Sig(NUEVO) := Q

Después de los pasos a) y b) obtenemos las estructuras mostradas en la figura 6-5a).

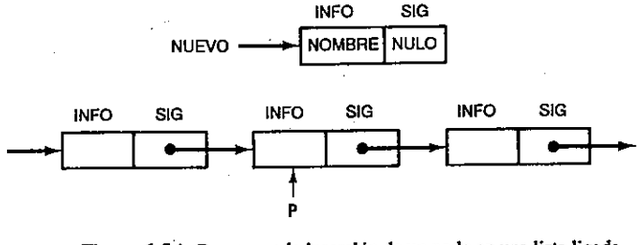

Figura 6-5a) Pasos para la inserción de un nodo en una lista ligada.

Después de los pasos c) y d) obtenemos las estructuras mostradas en la figura 6-5b).

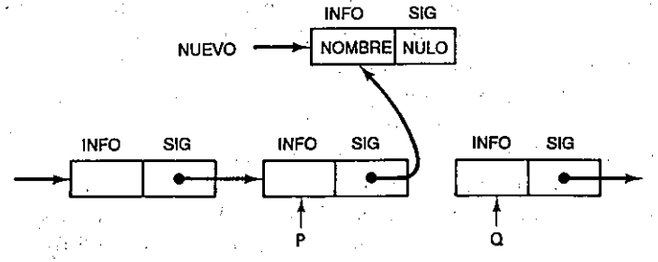

Figura 6-5b) Pasos para la inserción de un nodo en una lista ligada.

Por último, en la figura 6-5c) se muestra la estructura que produce el paso e).

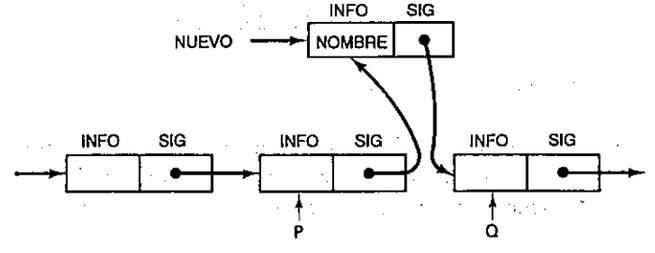

Figura 6-5c) Pasos para la inserción de un nodo en una lista ligada.

Es importante reconocer que las ligas tales como P, Q y NUEVO, así como la variable SIG apuntan todas a *nodos* y no sólo a la parte de información de los nodos. Una flecha en estos diagramas apunta a la caja completa. Insertar un nodo requiere tocar dos nodos: el nuevo nodo y el nodo apuntado por P. No hay necesidad de accesar o cambiar otros nodos.

MANEJO DEL ESPACIO DISPONIBLE

Hemos supuesto la existencia de dos funciones —una para crear espacio para el nodo y otra para regresar el espacio y reusarlo—, las cuales usamos en nuestros algoritmos de inserción y supresión. Ahora veamos estas funciones más detalladamente.

Almacenamiento compartido

El almacenamiento compartido contiene todo el espacio que no está en uso. Supongamos que este espacio es formateado en unidades con la estructura de los nodos para nuestra lista ligada. De hecho, un programa puede usar varias listas ligadas, con nodos de diferente formato, en cuyo caso tendremos varios almacenamientos compartidos con nodos vacíos. Para simplificar nuestra explicación, supongamos, por ahora, que sólo tenemos un tipo de nodo.

¿Cuál sería la estructura apropiada para el almacenamiento compartido? Tal vez no queramos utilizar una representación secuencial, debido a que los nodos serían removidos aleatoriamente de nuestra lista ligada, y desearíamos que el espacio liberado fuera reutilizable. Por ello, el almacenamiento compartido será representado, usando una estructura ligada. Sin embargo, es lógico que el espacio compartido no es otra cosa más que una lista lineal. Llamaremos DameNodo a la función que regresa la localidad del *siguiente* nodo libre en la memoria compartida.

Al inicio, cuando todas las listas de "usuario" están vacías, el almacenamiento compartido contiene todo el espacio disponible. Usemos a Disponible como el apuntador al primer nodo en la lista de espacio disponible (Figura 6-6).

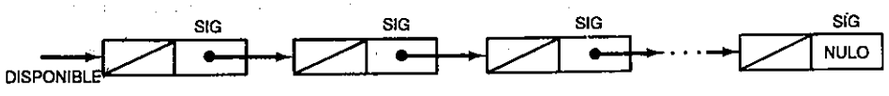

Figura 6-6 Lista de espacio disponible.

Asignación de un nuevo nodo

Cuando se necesite espacio para un nuevo nodo, éste se obtiene de la lista de espacio disponible invocando a la función DameNodo. La condición de excepción correspondiente a no tener espacio disponible se debe manipular correctamente.

```
        Si Disponible = Nulo
        entonces no hay espacio disponible
a)      sino empieza; DameNodo := Disponible;
b)              Disponible := Sig(Disponible);
c)              Sig(DameNodo) := Nulo;
        end;
```

El paso a) produce la estructura mostrada en la figura 6-7a).

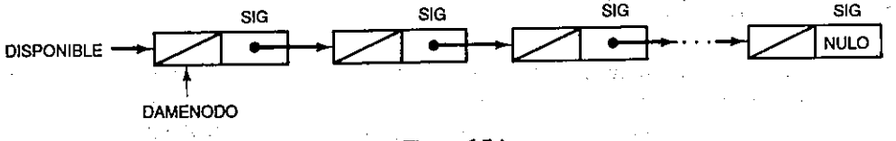

Figura 6-7a).

El paso b) produce la estructura que se muestra en la figura 6-7b).

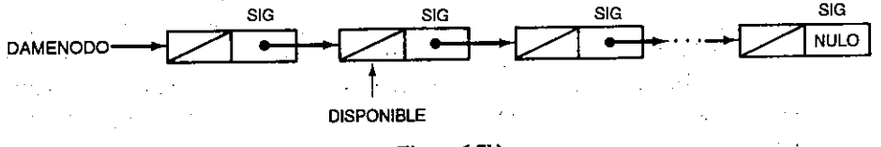

Figura 6-7b).

Y el paso c) produce la estructura de la figura 6-7c).

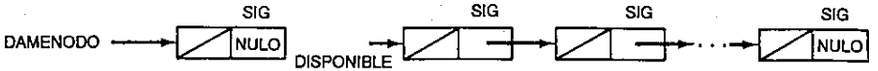

Figura 6-7c).

El asignar un nodo de la lista de espacio disponible requiere tocar solamente un nodo (el que se encuentre al frente de la lista) y cambiar el apuntador Disponible.

Liberación de un nodo

Llamemos LiberaNodo al procedimiento que regresa el nodo que ya no se necesita a la lista de espacio disponible. Esta acción hace que el espacio del nodo pueda volverse a usar. El argumento de LiberaNodo es un apuntador al nodo que ha de regresar al almacenamiento compartido. El procedimiento LiberaNodo(Q) es bastante sencillo.

a) Sig(Q):= Disponible
b) Infa(Q): = Nulo
c) Disponible: = Q.

El paso a) se muestra en la figura 6-8a.

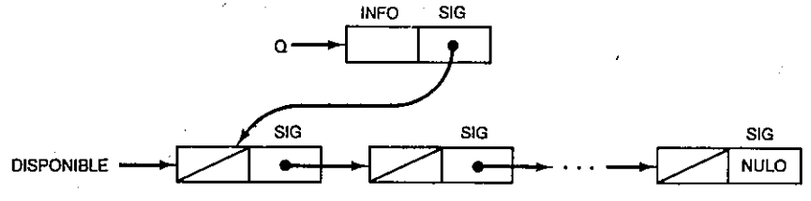

Figura 6-8a) Paso para liberar un nodo y colocarlo en la lista de disponibilidad.

Después de los pasos b) y c) obtenemos la estructura mostrada en la figura 6-8b).

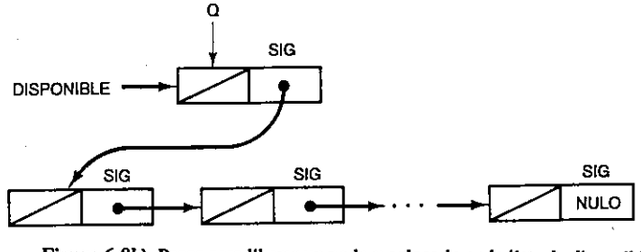

Figura 6-8b) Pasos para liberar un nodo y colocarlo en la lista de disponibilidad.

El apuntador Q, será manipulado después por la rutina de invocación de alguna forma adecuada. Q es probablemente un apuntador auxiliar. La parte Info del nuevo nodo liberado ha sido marcada con una diagonal al igual que las otras porciones Info de los nodos liberados en nuestros diagramas para indicar que han sido anuladas y se reescribirán cuando eventualmente sean removidas del almacén compartido, mediante una futura invocación del procedimiento DameNodo.

Los algoritmos DameNodo y LiberaNodo no sólo estructuran al almacenamiento compartido como una lista lineal, sino como una *pila de disponibilidad.* ¿Cómo serían los algoritmos, si el almacenamiento compartido tuviera una estructura de tipo *cola de disponibilidad?*

Después de una secuencia de operaciones de inserción y supresión en una lista de usuario, ésta y la lista de Disponibilidad podrían estar entrelazadas en forma física, aunque el manejo apropiado de ligas las conserve lógicamente desenlazadas (Figura 6-9).

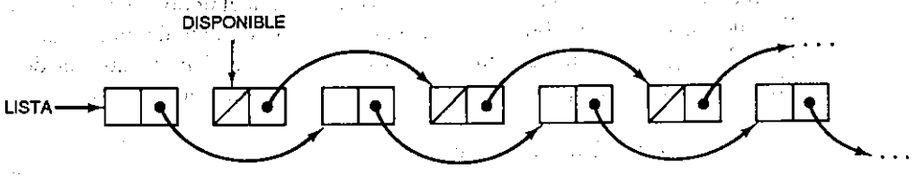

Figura 6-9 Lista de disponibilidad y lista del usuario.

En el resto de nuestra exposición sobre listas ligadas continuaremos suponiendo la existencia de los procedimientos DameNodo y LiberaNodo.

LISTAS LIGADAS EN PASCAL, USANDO VARIABLES APUNTADORAS

Definición de listas ligadas

Ahora plantearemos algunas técnicas para utilizar listas ligadas en programas. En las secciones anteriores hicimos un uso bastante extenso de apuntadores. Pascal cuenta con un tipo de datos, apuntador predefinido y con facilidades para el control de asignación dinámica de memoria. Un nodo se puede declarar en Pascal mediante:

```
type    apuntador-a-nodo = ↑ tipo-nodo;
        tipo-nombre = packed array [1 . . 10] of char;
        tipo-nodo = record
                        información : tipo-nombre;
                        sig-nodo: Apuntador-a-nodo
                    end;
```

La notación ↑ *tipo-nodo* indica que *apuntador-a nodo* es un tipo de dato que apunta a cosas que tienen un tipo *tipo-nodo.* En Pascal una variable que se declara como apuntador, está restringida a contener la dirección de un tipo particular de estructura de datos.

Después, declaremos a P como *apuntador-a-nodo* y al nodo como *tipo-nodo*

```
var p:apuntador-a-nodo;
    nodo:tipo-nodo;
```

y representamos con

```
p ↑ .información
y p ↑ .sig-nodo
```

a las partes de información y siguiente nodo, respectivamente, del nodo apuntado por p. Al apuntador p se le da un valor Nulo, mediante p:= nil.

Manejo de espacio

El programador usa las siguientes funciones para controlar el manejo del espacio en Pascal:

new(p) Reserva espacio para un tipo de variable apuntadora que es inicializada y apuntada a una dirección. Esta es la función de Pascal correspondiente a DameNodo.
dispose(p) Libera al nodo apuntado por p. Esta es la función de Pascal que corresponde a LiberaNodo.

Pascal al momento de ejecución administra el espacio libre.

Supresión de un nodo

El algoritmo para remover de una lista ligada el nodo que sigue al nodo que apunta p, se codifica en Pascal como se muestra más adelante. El contenido de la información del nodo removido se guarda en la variable *salida*, y suponemos que el nodo ha sido globalmente declarado.

```
procedure supresión(p : apuntador-a-nodo, var salida : tipo-nombre);
        var q : apuntador-a-nodo;
        begin if(p ↑ .sig-nodo = nil)
                then CONDICIÓN-SUBDESBORDE
                else begin q: = p ↑ .sig-nodo;
                        p ↑ .sig-nodo : = q ↑ sig-nodo;
                        salida: = q↑.información;
                        dispose (q)
                    end;
        end;
```

Sólo se han tocado dos nodos en la lista ligada; los que apuntan a *p* y *q*. No es necesario accesar o cambiar a ningún otro nodo.

Inserción de un nodo

El algoritmo para insertar el contenido de una variable llamada "entra" en una lista ligada, de tal forma que el nuevo nodo sea el que siga al nodo apuntado por p, se codifica en Pascal como sigue:

```
procedure inserción(p : apuntador-a-nodo,entra : tipo-nombre;
        var q : apuntador-a-nodo;
        begin new(q);
                q↑.información := entra;
                q↑. sig-nodo := p↑. sig-nodo;
                p↑. sig-nodo := q
        end;
```

Aquí, sólo se han tocado dos nodos: el nuevo (apuntado por q) y el apuntado por p.

Por seguridad, antes de invocar estos algoritmos, debe asegurar el programa que apunta a un nodo. En Pascal, p = nil, indica que no existe tal nodo.

LISTAS LIGADAS EN COBOL Y PASCAL, SIN EL USO DE VARIABLES APUNTADORAS

Utilizando un arreglo

Cuando usamos un lenguaje de programación que no tiene predefinido el tipo de dato *apuntador*, el código para declarar y manipular listas ligadas es más complejo. Sin embargo, simularemos variables apuntadoras alojando una lista ligada en un arreglo, donde el valor de la variable apuntadora se convierte en el valor de un subíndice.

La figura 6-10 muestra un arreglo que aloja una lista ligada de nombres. Cada elemento del arreglo es un registro con dos campos: Información y Sig-nodo. Usaremos el valor 0 para representar el valor nulo.

La pila de disponibilidad puede compartir el mismo espacio del arreglo (Figura 6-11). Mantener las ligas entre los nodos disponibles hace que el manejo del espacio

	INFO	SIG
1	IVAN	6
2	HOWARD	5
3		
4	HORACIO	2
5	IGOR	1
6	JUAN	0
7		

PRIMERO = 4

Figura 6-10 Arreglo que aloja una lista ligada.

	INFO	SIG
1	IVAN	6
2	HOWARD	5
3		7
4	HORACIO	2
5	IGOR	1
6	JUAN	0
7		0

PRIMERO = 4
DISPONIBLE = 3

Figura 6-11 Arreglo que aloja una lista de disponibilidad y una lista ligada del usuario.

	INFO	SIG
1	IVAN	6
2	HOWARD	1
3		7
4	HORACE	2
5		3
6	JUAN	0
7		0

PRIMERO = 4
DISPONIBLE = 5

Figura 6-12a) Lista ligada después de eliminar a Igor e insertar a Horacio.

	INFO	SIG
1	IVAN	6
2	HOWARD	1
3		7
4	HORACIO	5
5	HORACLES	2
6	JUAN	0
7		0

PRIMERO = 4
DISPONIBLE = 3

Figura 6-12b) Lista ligada después de eliminar a Igor e insertar a Horacles.

disponible sea más fácil que si cada uno tuviera un apuntador Nulo. La figura 6-12a) muestra la lista ligada después de la eliminación del elemento siguiente al que apunta P, donde P = 5 y, después de la inserción de 'HORACIO' en la secuencia alfabética apropiada (Figura 6-12b)).

Al usar arreglos, debemos establecer un límite superior para el número de nodos que contendrá la lista ligada; supongamos un máximo de 500.

Definición de una lista ligada

El arreglo puede definirse como sigue. Note que el arreglo contiene tanto una lista ligada del usuario como una lista ligada del espacio disponible. En COBOL:

```
01 ESPACIO-DEL-ARREGLO.
   02 NODO OCCURS 500 TIMES.
      03 INFORMACION      PIC X(10).
      03 SIG-NODO         PIC 9(3).
   02 PRIMER-NODO         PIC 9(3).
   02 DISPONIBLE          PIC 9(3).
```

PRIMER-NODO contiene el subíndice del primer nodo en la lista del usuario; DISPONIBLE contiene el subíndice del primer nodo en la lista del espacio disponible.

Los arreglos pueden utilizarse también para listas ligadas en Pascal, aunque no es común hacer esto ya que las técnicas para variables apuntadoras, introducidas en la sección previa, son preferibles. En Pascal, un arreglo para almacenar una lista ligada puede declararse como sigue:

```
type apuntador-a-nodo = 0. .500;
        tipo-nombre = packed array [1. .10] of char;
        tipo-nodo = record
                información: tipo-nombre;
```

```
                    sig-nodo: apuntador-a-nodo
                end;
var nodo : array [1..500] of tipo-nodo;
        primer nodo,disponible:apuntador-a-nodo;
```

En todos los casos, cuando alojamos listas ligadas en un arreglo, usamos el valor cero para representar al valor nulo.

Supresión de un nodo

El algoritmo para suprimir de la lista ligada el nodo siguiente al que apunta P requiere referirse a dos nodos: el primero con el subíndice P y el que sigue lógicamente a P con subíndice Q. Además, el índice para el espacio disponible deberá cambiarse.

En COBOL:

```
REMOVER.
        IF SIG-NODO (P) = 0
                condición-de-subdesborde
        ELSE COMPUTE Q = SIG-NODO (P)
                COMPUTE SIG-NODO(P) = SIG-NODO(Q)
                salida INFORMACION (Q)
                COMPUTE SIG-NODO (Q) = DISPONIBLE
                COMPUTE DISPONIBLE = Q.
```

En Pascal:

```
procedure remover(p:apuntador-a-nodo);
var q:apuntador-a-nodo;
begin
        if(nodo[p].sig-nodo = 0)
        then CONDICION-SUBDESBORDE
        else begin q : = nodo [p].sig-nodo;
                nodo[p].sig-nodo : = nodo[q].sig-nodo;
                writeln (nodo[q].información);
                nodo[q].sig-nodo : = disponible;
                disponible : = q
             end;
end;
```

Inserción de un nodo

El algoritmo para insertar el contenido de la variable ENTRA en la lista ligada, de tal modo que siga al que apunta P requiere tocar dos nodos: el primero, tomado de la lista de disponibilidad con subíndice Q y el otro, con subíndice P. El índice para el siguiente nodo disponible también deberá cambiarse.

En COBOL:

```
INSERCION.
        IF DISPONIBLE = 0
        THEN condición-de-desborde
        ELSE COMPUTE Q = DISPONIBLE
        COMPUTE DISPONIBLE = SIG-NODO (DISPONIBLE)
        MOVE ENTRA TO INFORMACION (Q)
        COMPUTE SIG-NODO (Q) = SIG-NODO (P)
        COMPUTE SIG-NODO (P) = Q.
```

En Pascal

```
procedure inserción(p:apuntador-a-nodo,entra:tipo-nombre);
var q:apuntador-a-nodo;
begin if(disponible = 0)
        then CONDICION-DESBORDE;
        else begin q : = disponible;
                disponible : = nodo[disponible].sig-nodo;
                nodo[q].información : = entra;
                nodo[q].sig-nodo : = nodo[p].sig-nodo;
                nodo[p].sig-nodo : = q
             end;
end;
```

OTRAS MANIPULACIONES DE LISTAS LIGADAS INDIVIDUALES

Las operaciones básicas de inserción y supresión de nodos fueron detalladas en la sección anterior, aquí desarrollaremos algoritmos para otras manipulaciones de listas ligadas y analizaremos mejoras comunes para las listas ligadas individuales. Cada algoritmo se codifica ya sea en COBOL o en Pascal.

Localización de un nodo particular

Primero, ¿cómo podemos encontrar el *i*-ésimo nodo en una lista? Si la lista estuviese alojada en un arreglo, un cálculo podría dar la dirección del nodo deseado, usando la localidad base del arreglo y el tamaño del nodo. Si la lista se representa en forma ligada, entonces necesitamos pasar a través de los nodos, contando hasta posicionarse en el *i*-ésimo. Por supuesto, es necesario verificar que por lo menos hay *I* nodos en la lista. El algoritmo se muestra en la figura 6-13. Para encontrar el *i*-ésimo nodo se requiere tocar *I* nodos, a menos que la lista contenga menos de *I* nodos. Entonces, toda la lista deberá accesarse. En COBOL la implantación de la búsqueda es como sigue, usando la declaración de ESPACIO-DEL-ARREGLO de la sección anterior.

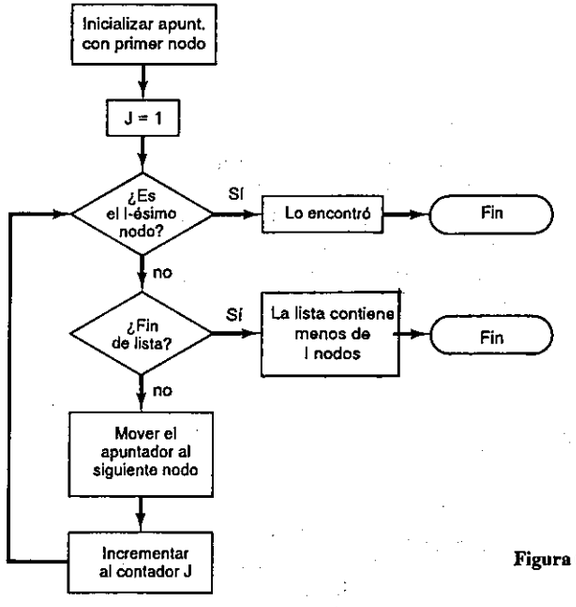

Figura 6-13 Lógica para encontrar el i-ésimo nodo en una lista ligada.

```
COMPUTE APT = PRIMER-NODO.
PERFORM BUSCA-NODO-EN-LISTA
        VARYING J FORM 1 BY 1
                UNTIL J = I OR APT = 0.
IF APT = 0
        la lista tiene menos de I nodos
ELSE salida INFORMACION (APT).
```

donde

```
BUSCA-NODO.
        COMPUTE APT = SIG-NODO (APT).
```

El algoritmo para determinar el número de nodos presentes en una lista ligada se le deja al lector.

Inserción al final de una lista

El algoritmo para insertar un nodo al principio de la lista se dio en nuestra discusión sobre la pila de disponibilidad. En esa sección, también, vimos el algoritmo para suprimir un nodo del inicio de una lista ligada. Consideremos, ahora, el algoritmo para insertar un nodo al final de una lista ligada. El algoritmo se muestra en la figura 6-14. Se tienen

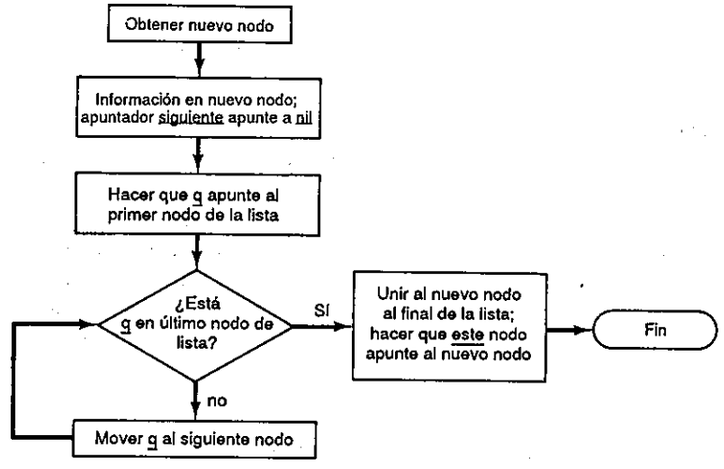

Figura 6-14 Lógica para insertar un nodo al final de una lista ligada.

que tocar todos los nodos antes de encontrar el final de la lista. Usemos las declaraciones de la sección previa.

```
procedure inserta-al-final(primer-nodo:apuntador-a-nodo,entra:tipo-nombre);
var nuevo-nodo,q:apuntador-a-nodo;
begin new(nuevo-nodo);
        nuevo-nodo ↑.información := entra;
        nuevo-nodo ↑.sig-nodo := nil;
        q: = primer-nodo;
        do while(q↑.sig-nodo<>nil)
```

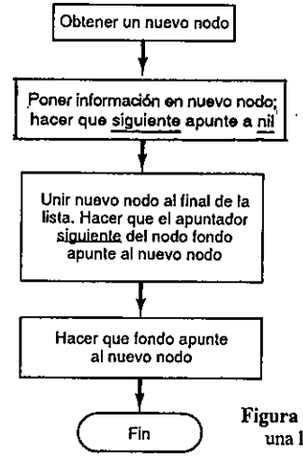

Figura 6-15 Lógica para insertar un nodo al final de una lista ligada que tiene un apuntador al final.

```
            q: = q ↑.sig-nodo;
          q↑.sig-nodo : = nuevo-nodo;
end;
```

Este código funciona excepto en el caso donde la lista inicial, apuntada por primer-nodo, esté vacía. Cambie el procedimiento de manera que trabaje correctamente en todos los casos.

Hemos visto cómo podemos utilizar listas ligadas para implantar tanto pilas como colas. Cuando una lista ligada se usa para representar una cola, deberá existir un apuntador que apunte directamente al extremo final, para evitar pasar a través de todos los nodos cuando sea necesario accesar el último. Llamemos a este apuntador Fondo. El algoritmo de inserción se muestra en la figura 6-15 y se codifica en Pascal como sigue:

```
procedure inserta-al-final(entra:tipo-nombre,var fondo:apuntador-a-nodo);
var nuevo-nodo:apuntador-a-nodo;
begin new(nuevo-nodo);
        nuevo-nodo ↑.información : = entra;
```

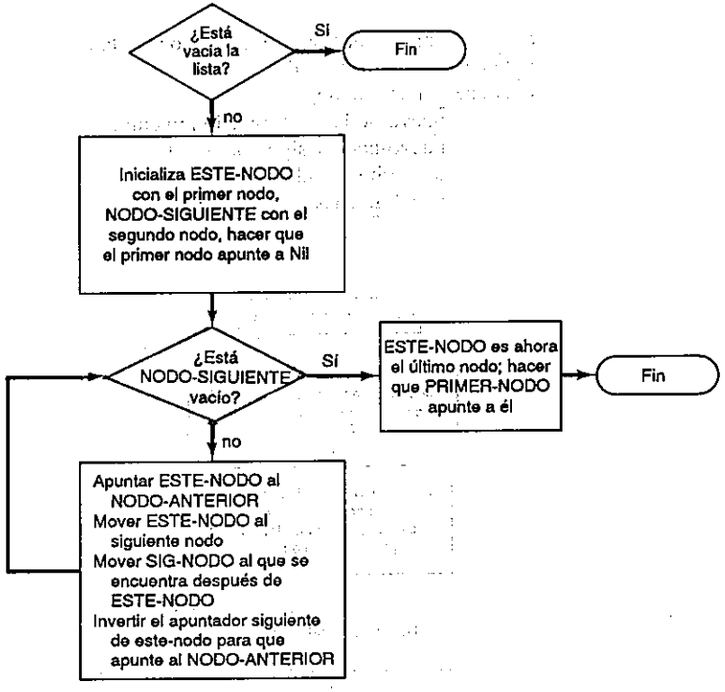

Figura 6-16 Lógica para invertir el orden de los nodos de una lista ligada.


```
            nuevo-nodo ↑.sig-nodo : = nil;
            fondo ↑.sig-nodo : = nuevo-nodo;
            fondo : = nuevo-nodo
end;
```

Sólo se tocaron dos nodos: el nuevo y el que era el último apuntado por fondo.

Inversión de una lista

Un algoritmo para invertir los elementos de una lista ligada implica pasar a través de cada nodo de la lista para cambiar todos los apuntadores, de tal manera que el último elemento se convierta en el primer elemento, y el que era el primer elemento se convierta en el último elemento. El algoritmo se muestra en la figura 6-16. En COBOL (otra vez usando la declaración de ESPACIO-DEL-ARREGLO):

```
01 APUNTADORES-AUXILIARES.
   02 ESTE-NODO              PIC 9(3).
   02 NODO-ANTERIOR          PIC 9(3).
   02 NODO-SIGUIENTE         PIC 9(3).
INVERSION-DE-LA-LISTA.
   IF PRIMER-NODO NOT = 0
   THEN COMPUTE ESTE-NODO = PRIMER-NODO
        COMPUTE NODO-SIGUIENTE = SIG-NODO (ESTE-NODO)
        COMPUTE SIG-NODO (ESTE-NODO) = 0
        PERFORM BUSCA-NODO UNTIL (NODO-SIGUIENTE = 0)
        COMPUTE PRIMER-NODO = ESTE-NODO.
BUSCA-NODO.
   COMPUTE NODO-ANTERIOR = ESTE-NODO.
   COMPUTE ESTE-NODO = NODO-SIGUIENTE.
   COMPUTE NODO-SIGUIENTE = SIG-NODO (ESTE-NODO).
   COMPUTE SIG-NODO(ESTE-NODO) = NODO-ANTERIOR.
```

Otros algoritmos útiles que puede desarrollar son los siguientes:

- Concatenar dos listas ligadas en una sola lista ligada.
- Separar una lista ligada en dos, de tal forma que el último elemento en la primera lista sea el elemento apuntado por P (Figura 6-17).

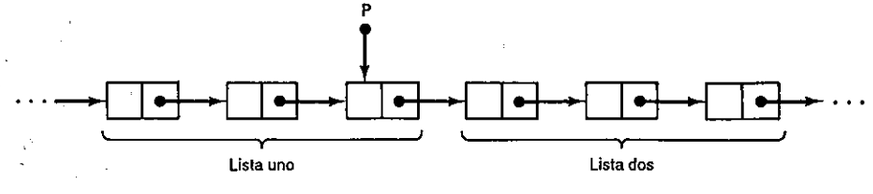

Figura 6-17 Separación de una lista ligada en dos.

- Separar una lista ligada en dos, de tal forma que la primera incluya sólo elementos donde Información < M y la segunda lista contenga elementos donde Información > = M.

LISTAS CIRCULARES LIGADAS Y NODOS PRINCIPALES

Ahora que usted ha comprendido estas operaciones fundamentales, podemos investigar algunas mejoras que permiten vencer varios problemas de las listas ligadas básicas.

Una deficiencia que ya habrá notado, es que, debido a un apuntador P en la lista ligada, no es posible alcanzar los elementos que preceden al Nodo (P). Un cambio sencillo de la estructura de datos, nos permitirá alcanzar cualquier nodo desde cualquier otro nodo. En lugar de almacenar un apuntador Nulo en el campo Sig-Nodo del último nodo de la lista, conviene hacer que este último nodo apunte hacia atrás, al inicio de la lista (Figura 6-18). Este tipo de estructura se llama *lista ligada circular.*

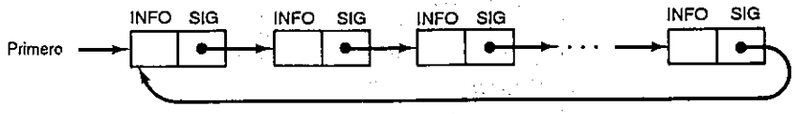

Figura 6-18 Lista ligada circular.

Usted mismo puede investigar los cambios que implica esta modificación en la estructura de la lista, para los algoritmos presentados en la sección anterior.

El problema de José

Un problema famoso que usa una lista ligada circular es el problema de José. Este problema se refiere a la supresión selectiva de elementos de una cola circular. Hay un grupo de bandidos, en el viejo oeste, que se encuentran en un predicamento desesperado. El grupo está rodeado por los alguaciles del comisario y no tienen ninguna esperanza de escapar en masa. Sólo les queda un caballo. Para seleccionar quién debe tomar el caballo y escapar con el botín, éstos (el civilizado grupo de bandidos) hacen lo siguiente. Escriben en trozos de papel el número que le corresponde a cada uno y los revuelven en un sombrero. Los bandidos se ponen en círculo. Uno de ellos es designado para ocupar la posición inicial (Figura 6-19a)).

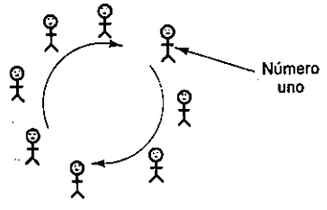

Figura 6-19a) El problema de José.

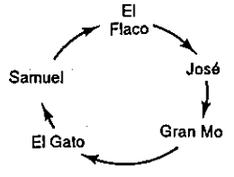

Figura 6-19b) El problema de José.

Digamos que se saca un número n del sombrero y el conteo empieza alrededor del círculo (en el sentido de las manecillas del reloj); se elimina la *n*-ésima persona. Cuando sale del círculo, éste se reduce. El conteo empieza de nuevo con el hombre que está a su izquierda. La *n*-ésima persona otra vez se elimina. El proceso continúa hasta que sólo quede un bandido, el cual coge el botín, salta al caballo y sale a campo traviesa.

Supongamos que el círculo inicial de bandidos está como se muestra en la figura 6-19b) y José es seleccionado como número 1 (porque él es el más grande). Se saca el número 4, y Samuel es el primer bandido eliminado. El conteo empieza de nuevo con el "Flaco" y, el "Gato" abandona el círculo. Otra vez el conteo empieza en el "Flaco", y el número 4 cae sobre el "Flaco". Ahora José es el primero, y el "Gran Mo" pierde con el número 4. José agarra el caballo y sale con el botín.

Debe escribir un programa que simule este procedimiento. La entrada debe ser la lista de nombres de los bandidos, ordenados de acuerdo al lugar que ocupen en el círculo, y el número *n.* Al sacar el nombre de los bandidos éstos se eliminan, sólo gana el último en quedar en la lista. Es obvia la elección de la estructura de datos para este problema: lista ligada circular.

En lo sucesivo supondremos que todas las listas ligadas son circulares, a menos que indiquemos lo contrario.

Nodos principales

Ahora consideremos el problema de buscar un nodo particular, digamos aquel que contenga el nombre "ELOISA", en una lista ligada circular. Es muy fácil caer en un ciclo infinito mientras se recorre la lista, a menos que el programador sea muy cuidadoso. Recuerde que el final de la lista no está marcado con un apuntador nulo.

Una forma común de resolver esta dificultad es utilizar un nodo principal que encabece el inicio o final de la lista (Figura 6-20).

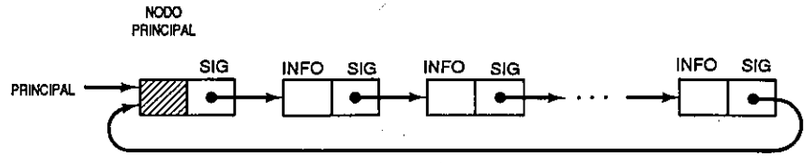

Figura 6-20 Lista ligada circular con nodo principal.

El nodo principal puede distinguirse con respecto a los otros nodos, de varias maneras:

- Puede tener un valor especial en su campo de Información, el cual es inválido como dato en los otros elementos (en este caso, hemos sombreado este campo como se muestra en la figura 6-20).
- Puede tener una bandera que lo marque como nodo principal.

Un diseñador puede poner información significativa en el nodo principal. Por ejemplo, el nodo principal puede contener el número de nodos de la lista, una descripción dirigida al usuario de la lista, una fecha de creación o cualquier otra información.

Cuando una lista ligada circular con cabeza está vacía su nodo principal está como se muestra en la figura 6-21.

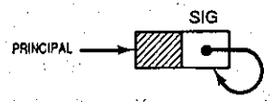

Figura 6-21 Lista ligada circular vacía con nodo principal.

Es decir, Sig(Principal) = Principal.

El algoritmo para insertar el contenido de la variable NOMBRE en la cabeza de la lista es:

a) Obtener espacio para un nuevo nodo, y hacer que NUEVO apunte a él.
b) Poner el contenido de la variable NOMBRE en la parte Información del nuevo nodo.

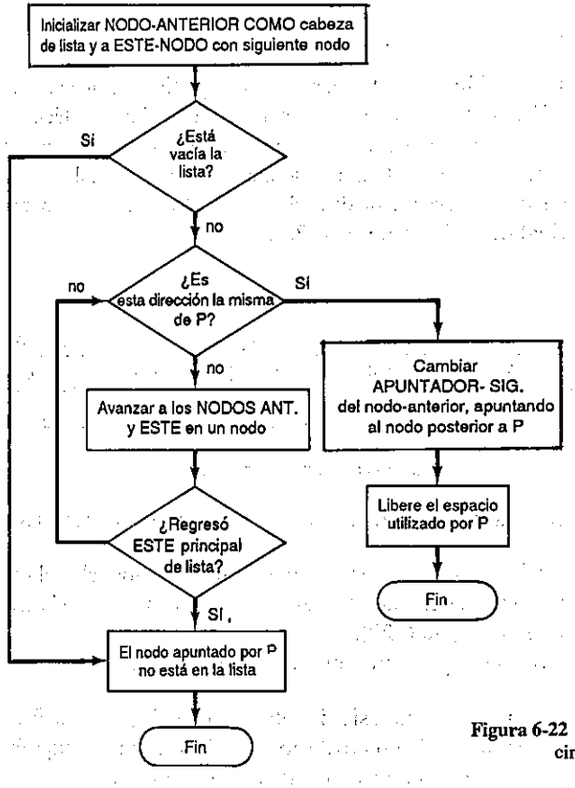

Figura 6-22 Lógica para suprimir de una lista ligada circular el nodo apuntado por P.

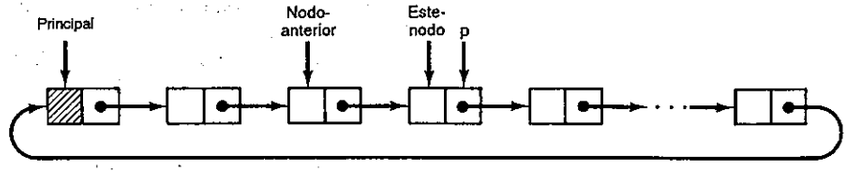

Figura 6-23 Lista con apuntadores auxiliares.

c) Hacer que el nuevo nodo apunte al nodo principal.
d) Hacer que Cabeza apunte al nuevo nodo.

¿Cómo afecta la existencia del nodo principal a los otros algoritmos presentados en este capítulo? Debe notar qué tan valioso es el dibujar diagramas para simular el manejo de apuntadores al desarrollar y analizar los algoritmos.

Supresión de un nodo particular

Como podrá haber notado en los algoritmos de supresión que se han especificado hasta aquí, el nodo a ser borrado de la lista es siempre apuntado por algún otro que le precede. ¿Cómo podemos borrar un nodo apuntado por P? Es necesario que pasemos a través de la cadena de nodos con sólo ligas al siguiente nodo, comparando las direcciones hasta posicionarse en la localidad de P, por supuesto debemos tener cuidado de no caer fuera del límite de una lista no circular. El algoritmo se muestra en la figura 6-22. Es necesario no solamente verificar si el nodo apuntado por ESTE es el nodo a ser borrado, sino también conservar el nodo anterior para que se pueda ligar al nodo posterior a ESTE si ESTE resulta ser igual a P (Figura 6-23). En promedio, la supresión del nodo apuntado por P, requiere accesar la mitad de nodos en la lista. En el peor de los casos, el cual ocurre cuando el nodo apuntado por P no está en la lista, todos los nodos de la lista deberán ser accesados. Compare este comportamiento con el de los dos algoritmos presentados anteriormente, para la supresión del nodo posterior al apuntado por P.

Una forma de codificar el algoritmo en Pascal es como sigue:

```
procedure suprime-p(principal:apuntador-a-nodo,var p:apuntador-a-nodo,
                    salida:tipo-nombre);
var anterior, este:apuntador-a-nodo;
        bandera:0..2;
begin
        anterior : = cabeza;
        éste : = principal ↑ .sig-nodo;
        bandera : = 1; {búsqueda en proceso}
        while bandera = 1
            do begin
                if(éste = principal)
                then bandera : = 2; {se ha recorrido la lista completa y no se
                                     encontró al nodo apuntado por p}
                if(éste = p)
```

```
                then bandera : = 0 {se encontró al nodo apuntado por p}
                else begin anterior : = éste;
                           éste: = éste ↑ .sig-nodo {pasa al siguiente nodo}
                     end;
                end;
if(bandera > 0)
then EL NODO APUNTADO POR P, NO ESTA EN LA LISTA
else begin {se procede a suprimir al nodo apuntado por p}
                anterior ↑ .sig-nodo: = p ↑ .sig-nodo;
                salida : = p ↑ .información;
                dispose(p)
           end;
end;
```

Puede ser un ejercicio útil desarrollar el algoritmo de arriba para una lista ligada no-circular y/o una lista ligada sin nodo cabeza.

LISTAS DOBLEMENTE LIGADAS

Conceptos básicos

Cuando es deseable poder recorrer una lista ligada hacia atrás o borrar nodos particulares, resulta ventajoso utilizar listas doblemente ligadas, en lugar de listas con una sola liga. En una *lista doblemente ligada* cada nodo no sólo tiene un apuntador al siguiente nodo, sino también un apuntador al nodo anterior (Figura 6-24).

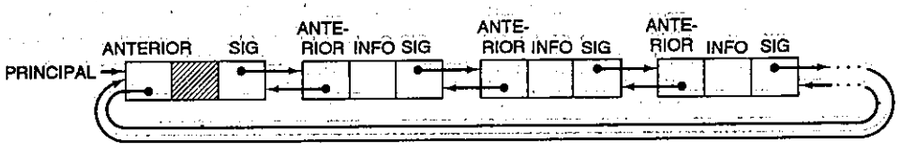

Figura 6-24 Lista doblemente ligada.

Las variaciones en una lista doblemente ligada son la omisión del nodo principal y/o no hacer que la lista ligada sea circular.

Hemos llamado al apuntador hacia el siguiente nodo como Sig(Nodo) y al apuntador hacia el nodo anterior como Anterior(Nodo). Estos apuntadores se pueden referir como apuntador *derecho* e *izquierdo*, apuntador *sucesor* y *predecesor*, apuntador *sig-liga* y *ant-liga*, etcétera.

Una propiedad fundamental de las listas doblemente ligadas es que para cualquier apuntador P en la lista,

$$Sig(Anterior(P)) = P$$

y

$$P = Anterior(Sig(P))$$

Ir hacia atrás del nodo, y después hacia adelante, lo posicionará en el nodo original. De igual forma, dando un paso adelante hacia el sucesor, y después un paso atrás hacia el predecesor, lo coloca sobre el nodo original.

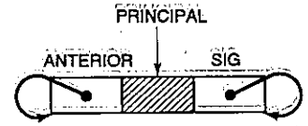

Figura 6-25 Lista doblemente ligada vacía.

En la figura 6-25 se muestra una lista doblemente ligada vacía. Tanto los apuntadores Sig y Anterior en el nodo principal apuntan al nodo principal, pues, no hay otros nodos en la lista. Esto es:

$$Anterior(Principal) = Principal$$

y

$$Sig(Principal) = Principal.$$

Definición de una lista doblemente ligada

Una estructura para una lista doblemente ligada puede declararse en Pascal, usando variables apuntadoras.

```
type apuntador-a-nodo = ↑ tipo-nodo;
     tipo-nodo = record
                   anterior:apuntador-a-nodo;
                   información:tipo-nombre;
                   sig:apuntador-a-nodo
                 end;
```

Dado que en COBOL no existen variables apuntadoras, en los programas en COBOL se usan arreglos para almacenar listas doblemente ligadas. Tal y como vimos anteriormente en nuestra discusión sobre la forma de alojar listas ligadas en arreglos, la lista de Disponibilidad de espacio libre comparte el arreglo para la lista doblemente ligada. La lista de Disponibilidad también está doblemente ligada. El programador deberá manejar los índices para la inicialización de cada lista, utilizando las variables CABEZA y DISPONIBLE. En el siguiente código, suponemos un máximo de 500 nodos.

```
01 ESPACIO DEL ARREGLO.
   02 NODO OCCURS 500 TIMES.
      03 NODO-ANTERIOR      PICTURE 9(3).
      03 INFORMACION        PICTURE X(10).
      03 NODO-SIGUIENTE     PICTURE 9(3).
   02 CABEZA                PICTURE 9(3).
   02 DISPONIBLE            PICTURE 9(3).
```

La figura 6-26 muestra un arreglo para alojar una lista doblemente ligada. Compare este ejemplo con la lista ligada mostrada en la figura 6-12.

	INFO	SIG	ANTERIOR
1	IVAN	6	2
2	HOWARD	1	4
3		7	5
4	HORACIO	2	0
5		3	0
6	JUAN	0	1
7		0	3

PRIMERO = 4
DISPONIBLE = 5

Figura 6-26 Arreglo que aloja una lista doblemente ligada.

Supresión de un nodo

Ahora podemos reconsiderar nuestros algoritmos para suprimir e insertar nodos en listas ligadas. Primero, para suprimir un nodo apuntado por P de una lista doblemente ligada, los apuntadores a ese nodo, desde su sucesor y predecesor se deben reinicializar (Figura 6-27a).

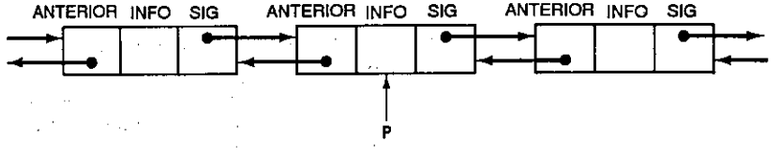

Figura 6-27a) Pasos para la supresión de un nodo de una lista doblemente ligada.

El procedimiento es como sigue:

a) Asignar un apuntador auxiliar para el nodo predecesor de P y otro para el nodo sucesor de P.
b) Cambiar el apuntador siguiente del predecesor para que apunte al sucesor de P, en lugar de a P.
c) Cambiar el apuntador Anterior del sucesor para que apunte al predecesor de P, en lugar de a P.
d) Liberar el espacio apuntado por P.

Este algoritmo requiere tocar tres nodos, el que está apuntado por P, el sucesor y el predecesor. Compare este comportamiento con el del algoritmo mostrado en la figura 6-22, el cual suprime el nodo apuntado por P de una lista ligada simple. El paso a) proporciona la estructura mostrada en la figura 6-27b),

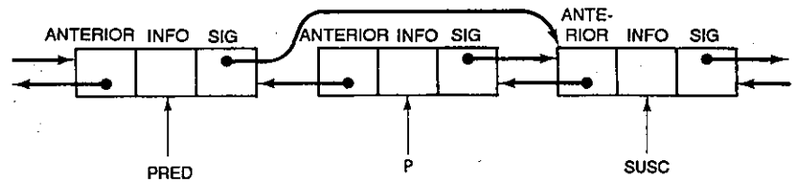

Figura 6-27b) Pasos para la supresión de un nodo de una lista doblemente ligada.

y, después del paso c) obtenemos la estructura mostrada en la figura 6-27c).

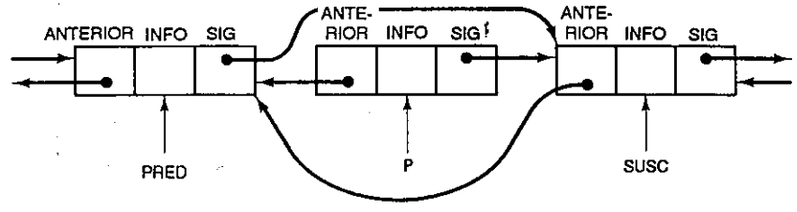

Figura 6-27c) Pasos para la supresión de un nodo de una lista doblemente ligada.

En Pascal el algoritmo puede codificarse como sigue:

```
procedure suprime-p(var p:apuntador-a-nodo, salida:tipo nombre);
var predecesor,sucesor:apuntador-a-nodo;
begin predecesor : = p ↑.anterior;
        sucesor : = p ↑ .sig;
        predecesor ↑ .sig : = sucesor;
        sucesor ↑ .anterior : = predecesor;
        salida: = p ↑ .información;
        dispose(p)
end;
```

Inserción de un nodo

Para insertar una variable ENTRADA dentro de una lista doblemente ligada, después de un nodo apuntado por P, primero establecemos el nuevo nodo en la lista, y después reinicializamos a los apuntadores apropiadamente. La clave para este algoritmo consiste en no destruir a los apuntadores que se necesitarán más tarde. El procedimiento es:

a) Obtener un nuevo nodo y llenarlo con los datos del usuario.
b) Inicializar el apuntador Siguiente del nuevo nodo para que apunte al sucesor de P y, a su apuntador Anterior para que apunte a P.
c) Hacer que el apuntador Siguiente de P apunte al nuevo nodo.
d) Hacer que el apuntador Anterior del nodo sucesor del nuevo nodo apunte al nuevo nodo, en lugar de a P.

La implantación del paso d) requiere un apuntador auxiliar para el sucesor del nuevo nodo. El algoritmo toca tres nodos, el primero es el apuntado por P, el segundo es el sucesor de P y el tercero es el nuevo nodo. En contraste, el algoritmo para insertar un nodo después del nodo apuntado por P en una lista simplemente ligada, requiere tocar sólo dos nodos; el sucesor de P no necesita ser accesado. Después del paso b), la estructura queda como en la figura 6-28a),

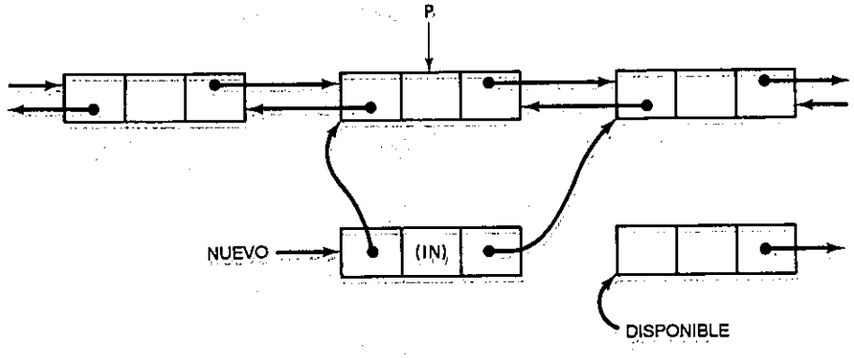

Figura 6-28a) Pasos para la inserción de un nodo en una lista doblemente ligada.

Después del paso d) obtenemos la estructura mostrada en la figura 6-28b).

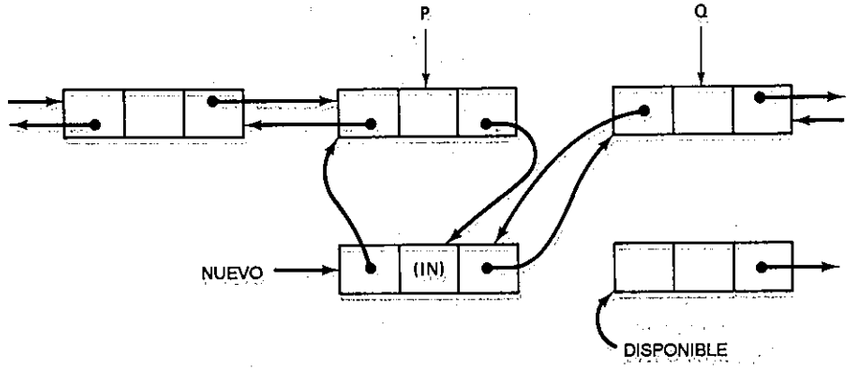

Figura 6-28b) Pasos para la inserción de un nodo en una lista doblemente ligada.

En COBOL:

```
INSERAF.
    COMPUTE NUEVO = DISPONIBLE.
    COMPUTE DISPONIBLE = NODO-SIGUIENTE (DISPONIBLE).
    MOVE IN TO INFORMACION (NUEVO).
    COMPUTE NODO-SIGUIENTE (NUEVO) = NODO-SIGUIENTE (P).
    COMPUTE NODO-ANTERIOR (NUEVO) = P.
    COMPUTE NODO-SIGUIENTE (P) = NUEVO.
    COMPUTE Q = NODO-SIGUIENTE (NUEVO).
    COMPUTE NODO ANTERIOR (Q) = NUEVO.
```

Convénzase usted mismo que este algoritmo funciona aun cuando la lista esté inicialmente vacía.

Trate de desarrollar algoritmos para listas que deberán ser tan generales como sea posible. Es decir, no deberá ser necesario tener un algoritmo para hacer algunas cosas con una lista inicialmente vacía y otro para hacer lo mismo con una lista inicialmente llena. El algoritmo mismo deberá manejar los casos 'extremos'.

Los factores básicos a considerar en el uso de listas ligadas simples y listas doblemente ligadas son el costo de mantenimiento y el almacenamiento de apuntadores contra la facilidad de manipulación de listas.

EJEMPLOS DE APLICACION DE LISTAS LIGADAS

Ahora consideramos algunas aplicaciones que hacen buen uso de la estructura de listas ligadas. El primer ejemplo utiliza listas ligadas para representar cadenas de datos en orden alfabético. El tercer ejemplo muestra cómo se puede usar la lista ligada para representar arreglos dispersos.

Polinomios

Un polinomio es una expresión algebraica de la forma:

$$a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0.$$

Cada a_i es un coeficiente de la potencia correspondiente a la variable x. Por ejemplo, en el polinomio

$$143x^4 + 201x^2 + 14x + 2$$

a_4=143, a_3=0, a_2=201, a_1=14, y a_0=2. Llamemos a este polinomio POLY 1.

Los polinomios se utilizan tanto en problemas científicos como en problemas comerciales. Los lenguajes de propósito general como Pascal, PL/I, COBOL y FORTRAN no tienen tipos de datos predefinidos o funciones para manipular directamente polinomios. En lugar de esto, es frecuente la representación de polinomios usando arreglos o listas ligadas. Una forma apropiada de estructura de nodos es (en Pascal):

```
type apuntador-a-nodo = ↑ tipo-nodo;
     tipo-nodo = record
```

```
    exp : integer;
    coef : integer;
    sig-nodo : apuntador-a-nodo
  end;
```

Donde exp es el exponente de la variable, coef es su coeficiente correspondiente y sig-nodo es la liga al siguiente nodo en la representación del polinomio. Sólo los términos con coeficiente diferente de cero necesitan ser representados.

Es conveniente ligar circularmente los nodos por orden decreciente respecto a los valores de los exponentes de los términos (Figura 6-29).

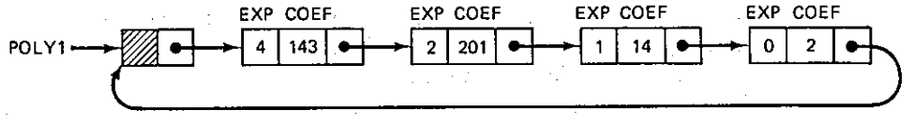

Figura 6-29 Ejemplo de representación de un polinomio con listas ligadas.

Sumar (o restar) dos polinomios requiere que se sumen (o resten) los coeficientes de los términos cuyo exponente coincide. Por ejemplo:

$$\begin{array}{rr} & 143x^4 + 201x^2 + 14x + 2 \\ + & 312x^3 - 21x^2 + 42 \\ \hline & 143x^4 + 312x^3 + 180x^2 + 14x + 44 \end{array}$$

En la figura 6-30 se muestra un algoritmo para sumar los polinomios POLY-1 y POLY-2 para formar POLYSUM. La implantación del algoritmo en Pascal le sigue. Los nodos cabeza pasan al procedimiento suma-polinomios. Poly-1 y poly-2 son apuntadores a los nodos cabeza de los polinomios POLY-1 y POLY-2, respectivamente. Polysum apunta al nodo cabeza del polinomio POLYSUM, el cual se construye por el procedimiento. Los apuntadores p1, p2 y psum son las direcciones de los términos considerados en POLY-1, POLY-2 y POLYSUM, respectivamente. Se usa el indicador de *empate* para señalar las siguientes condiciones:

exp1-mayor: Hay una potencia de x en POLY-1 que no aparece en POLY-2.
exp-igual: la potencia de x aparece tanto en POLY-1 como en POLY-2.
exp2-mayor: hay una potencia de x en POLY-2 que no aparece en POLY-1.

Las banderas fin1 y fin2 se usan para señalar que los últimos términos de POLY-1 y POLY-2, respectivamente, han sido procesados. Si POLY-1 termina antes que POLY-2, como en:

$$\begin{array}{rll} & \textbf{(POLY1)} & 14x^4 + 3x^3 \\ + & \textbf{(POLY2)} & 7x^3 + x^2 + 3 \end{array}$$

entonces los términos de POLYSUM del exponente = 2 en adelante son copiados tal cual de POLY-2, y viceversa, si POLY-2 termina antes que POLY-1.

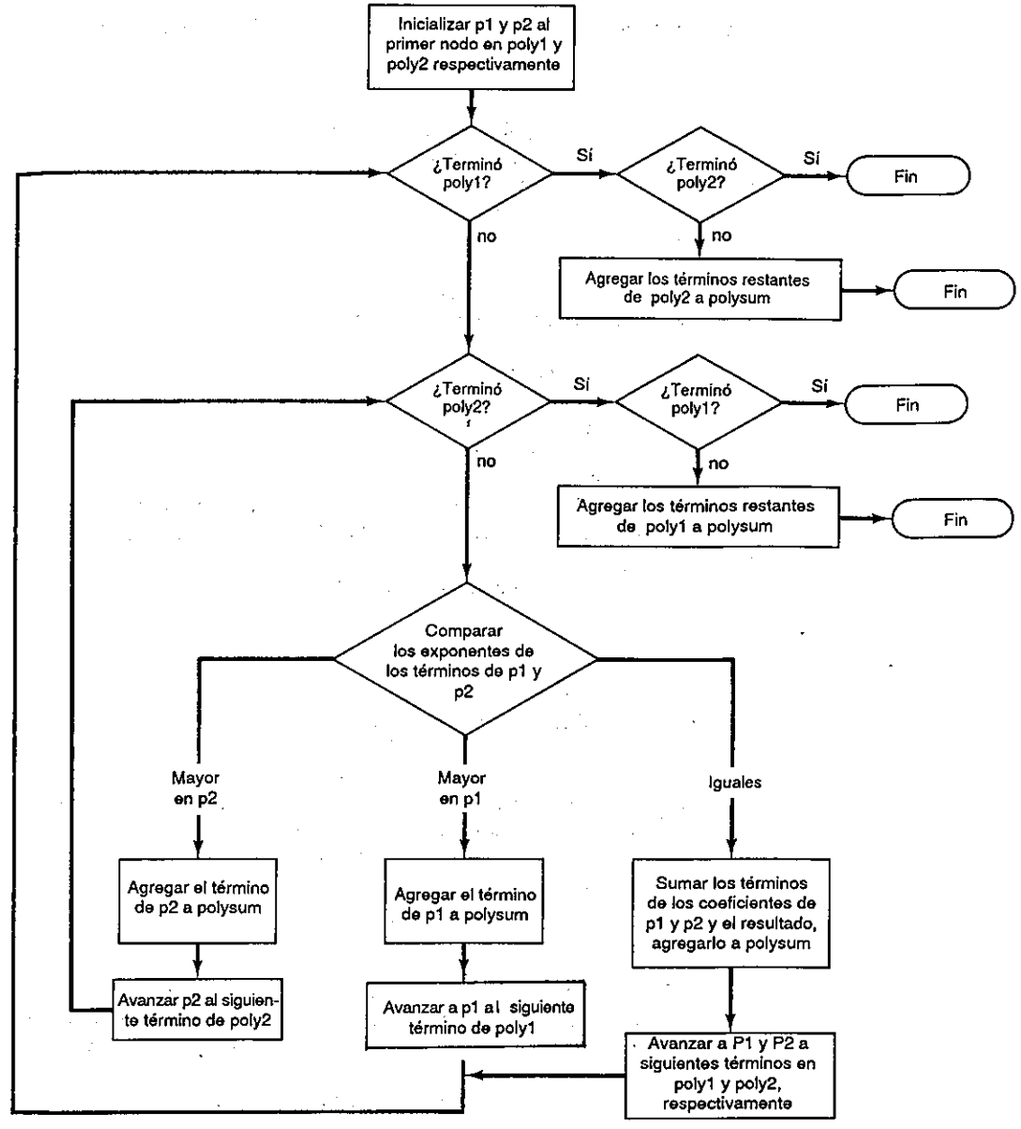

Figura 6-30 Lógica para sumar dos polinomios.

El procedimiento suma-nodos se usa para crear los nodos del polinomio resultante POLYSUM. Los parámetros pasados a éste indican cuál término se debe tomar de POLY-1 o POLY-2 y le permiten al procedimiento indicar que el último término de ese

polinomio ha sido procesado. Apuntador-a-nodo es un tipo de datos conocido globalmente.

```
procedure suma-polinomios(poly1,poly2:apuntador-a-nodo;var polysum:
                                    apuntador-a-nodo);
type empate = (exp1-mayor,exp-igual,exp2-mayor);
var p1,p2,psum:apuntador-a-nodo;
        bandera-exp:empate;
        fin1, fin2: 0..1;
procedure suma-nodos(poly:apuntador-a-nodo;var p,psum:apuntador-a-nodo,
        bandera-fin:0..1); {declaración de subprocedimiento}
var p-nuevo:apuntador-a-nodo;
begin new(p-nuevo); {añade un nuevo nodo a la suma}
        psum ↑ .sig : = p-nuevo;
        psum : = p-nuevo;
        psum ↑ .exp : = p ↑ .exp;
        psum ↑ .coef : = p ↑ .coef;
        p : = p ↑ .sig; {mueve a p al siguiente nodo en el polinomio}
        if p=poly then bandera-fin : = 1 {revisa si es el fin del polinomio}
end; {suma nodos}
begin fin1 : = 0; {bloque del programa principal}
        fin2 : = 0;
        p1 : = poly1 ↑ .sig;
        p2 : = poly2 ↑ .sig;
        psum : = polysum;
        if(p1=poly1)then fin1 : = 1; {se alcanzó final de poly1}
        if(p2=poly2)then fin2 : = 1; {se alcanzó final de poly2}
    while (fin1=0 and fin2=0) {todavía hay términos por agregar}
    do begin if (p1 ↑ .exp>p2 ↑ .exp {toma el término con mayor exponente}
        then bandera-exp : = exp1-mayor {como último término
                                        por agregar}
        else if(p1 ↑ .exp=p2 ↑ .exp)
                then bandera-exp : = expigual
                else bandera-exp : = exp2-mayor;
        case bandera-exp of
        exp1-mayor: suma-nodos(poly1,p1,psum,fin1); {agrega término poly1}
        exp-igual: begin suma-nodos(poly1,p1,psum,fin1); {agregar término poly1}
                psum ↑ .coef : = psum ↑ .coef p2 ↑ .coef; {agrega término poly2}
                p2 : = p2 ↑ .sig;
                if p2=poly2 then fin2 : = 1 {final de poly2}
            end;
        exp2-mayor : suma-nodos(poly2,p2,psum,fin2) {agrega término poly 2}
  end;
end;
{se alcanzó el final de uno de los poly}
while (fin1=0) {se alcanzó el final de poly2}
do suma-nodos(poly2,p2,psum,fin2); {agrega el resto de términos de poly1}
while (fin2=0) {se alcanzó el final de poly1}
do suma-nodos(poly2,p2,psum,fin2); {agrega el resto de términos de poly2}
```



```
    psum ↑ .sig : = polysum
end;
```

Este programa puede modificarse para manipular la resta de dos polinomios. Un ejercicio útil es escribir un programa para sumar dos polinomios que estén representados por listas ligadas, alojadas en arreglos, en lugar de usar variables apuntadoras. ¿En qué sería diferente el algoritmo, si las listas ligadas no fuesen circulares?, ¿Si no tuviesen nodos cabeza? ¿Si estuviesen doblemente ligadas? ¿Por qué se prefirió utilizar aquí una lista ligada simple como estructura de datos?

¿Cómo son los algoritmos para la multiplicación de polinomios?, y ¿para dividir polinomios?

Considere ahora el problema de manipular polinomios de dos variables; digamos x y y. Nuevamente, una lista ligada es la estructura de datos adecuada para representar estos tipos de datos abstractos. Se puede representar cada término por un nodo del siguiente tipo.

```
type apuntador-a-nodo = ↑ tipo-nodo;
        tipo-nodo = record
                        coef, x-exp, y-exp:integer;
                        sig:apuntador-a-nodo
                    end;
```

Sólo los términos con coeficientes diferentes de cero necesitan ser representados.

Los algoritmos para manipular esta representación polinomial, se pueden simplificar al hacer que los términos estén ligados en un orden específico, es decir, ordenados por el exponente x mayor y el exponente y menor. Por ejemplo:

$$14x^3 + 82x^2y - 47y^4 + 12y^2 + 6$$

sería el mostrado en la figura 6-31.

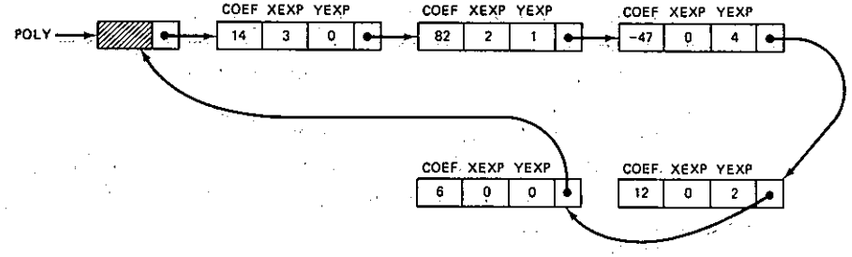

Figura 6-31 Ejemplo de un polinomio con dos variables.

Desarrolle un algoritmo para sumar dos polinomios de dos variables. ¿En qué difiere este algoritmo respecto al que se presentó para sumar polinomios de una variable?

¿Cuál sería la estructura de nodos más apropiada para los términos de un polinomio de tres variables? y de ¿n-variables?

Una lista multiligada simple

Como veremos posteriormente con mayor detalle, existen muchos casos donde es conveniente usar listas ligadas con más de dos apuntadores por nodo. Un ejemplo sencillo de estructura de datos, con tres apuntadores por nodo, es una lista doblemente ligada de datos de tipo cadena con longitud variable, donde cada nodo contiene un apuntador hacia el dato real, en lugar del valor del dato (Figura 6-32).

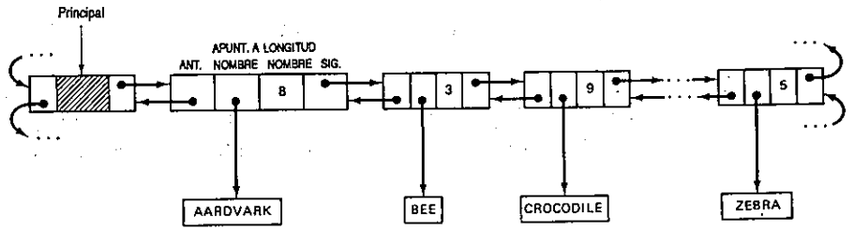

Figura 6-32 Ejemplo de una lista multiligada.

El mayor beneficio al usar esta estructura de datos, es que los nodos pueden tener una longitud fija. Cada nodo contiene valores de apuntador y un campo de tipo entero. El manejo de pedazos de memoria de longitud fija es significativamente menos complejo que el manejo de pedazos de memoria de longitud variable. Por supuesto, aquí aun tenemos que lidiar con el nombre real de los animales, el cual es de longitud variable.

Arreglos dispersos

Una aplicación común de las listas ligadas es la representación de arreglos dispersos. Ya hemos discutido la representación secuencial de arreglos dispersos. La representación ligada de arreglos dispersos tiene el potencial suficiente para reducir las necesidades de almacenamiento y reducir el número de cálculos ejecutados en algunas operaciones de arreglos. Sólo las entradas diferentes de cero en un arreglo serán representadas. La representación ligada puede eliminar sus beneficios, si el arreglo no es lo suficientemente disperso.*

Una posible representación ligada de un arreglo disperso bi-dimensional es el siguiente. Hay una lista ligada para cada renglón y columna del arreglo. Cada una de estas listas ligadas tiene un nodo principal y además es circular (con una sola liga). Cada entrada diferente de cero en el arreglo se representa por una ocurrencia de un nodo de la forma (en Pascal):

```
type apuntador-a-nodo : ↑ tipo-nodo;
     tipo-nodo = record
```

*Estas técnicas "dispersas" pueden aplicarse también cuando la mayoría de los elementos del arreglo toman un valor diferente de cero, digamos 1.


```
                renglón, columna,
                valor : integer;
                sig-columna:
                apuntador-a-nodo;
                sig-renglón: apuntador-a-nodo;
            end;
```

Cada nodo contiene una indicación del renglón y columna en el cual aparece la entrada diferente de cero, el valor de la entrada, y dos apuntadores. Un apuntador es hacia el siguiente nodo en el mismo renglón y el otro es para el siguiente nodo en la misma columna.

Por ejemplo, el siguiente arreglo de 5 por 4

1	0	0	0
0	−3	0	0
2	0	1	0
0	0	14	0
0	0	0	0

se puede representar como lo muestra la figura 6-33. Cada nodo no-cabeza participa en dos listas ligadas: una para su renglón y otra para su columna. En este ejemplo hay cuatro

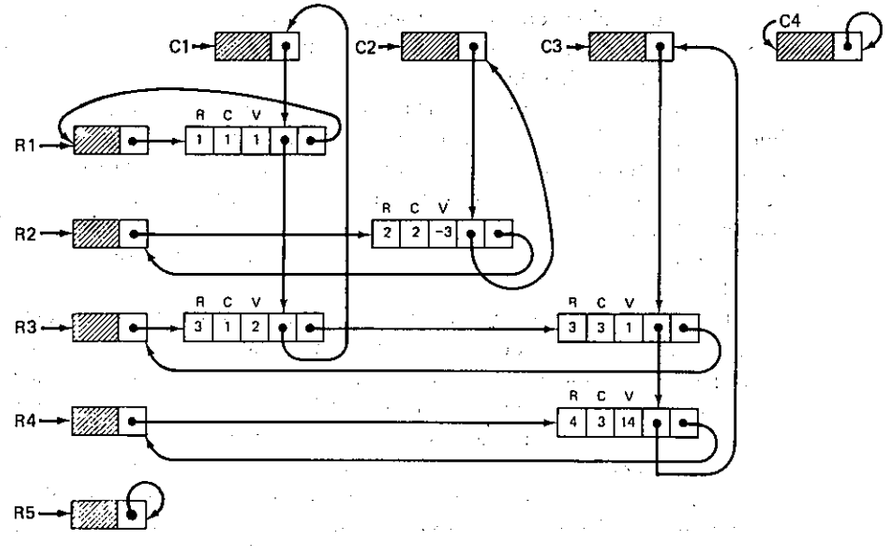

Figura 6-33 Representación ligada de un arreglo disperso

renglones diferentes de cero y tres columnas diferentes de cero; hay siete listas no nulas. Hay cinco entradas diferentes de cero en el arreglo y nueve nodos cabeza. Las quince entradas iguales a cero no ocuparon espacio de almacenamiento.

Esta técnica particular para representar un arreglo disperso es bastante satisfactoria, cuando hay actualizaciones frecuentes del arreglo. La estructura ligada hace muy fácil el trabajo de cambiar las entradas iguales a cero y las que no lo son. Para otros enfoques de representación de arreglos dispersos, el lector interesado podrá consultar U. W. Pooch and Nieder, "A survey of indexing techniques for sparse matrices", *ACM Computing Surveys*, 5(2): 109-133, junio de 1973.

Puede desarrollar los algoritmos para sumar dos arreglos dispersos (es decir, matrices) y para ejecutar multiplicación de dos matrices dispersas.

¿Cuál sería una estructura apropiada para representar un arreglo disperso tridimensional y un arreglo n-dimensional?

RESUMEN

Una lista ligada es una forma, no secuencial, de representar una estructura de datos lineal. Cada nodo de una lista ligada contiene por lo menos un apuntador al siguiente nodo. Una lista ligada puede ser circular, en cuyo caso el último nodo apunta al primer nodo de la lista. Las listas ligadas, con frecuencia se implantan con nodos cabeza. Para simplificar los algoritmos que ejecutan inserciones y supresiones de las listas ligadas se utilizan a menudo apuntadores que apuntan tanto al nodo anterior como al nodo siguiente, dando como resultado listas doblemente ligadas.

Cuando un nodo dado puede participar simultáneamente en más de una lista ligada, la estructura se dice, que es una lista multi-ligada. Nuestro estudio se enfocará más adelante a este tipo de listas.

Hemos visto cómo utilizar listas ligadas para representar, no solamente listas lineales, sino también clases especiales de listas lineales como pilas y colas. También hemos visto algunas aplicaciones representativas que usan listas ligadas. Las listas ligadas sencillas son adecuadas para la representación de polinomios; en tanto las listas doblemente ligadas se pueden usar para representar cadenas de longitud variable; y por último, las estructuras bidimensionales multi-lista pueden usarse para representar arreglos dispersos.

El beneficio básico del uso de una lista ligada es que no requiere almacenamiento secuencial. Entonces puede ser una desventaja el usar listas ligadas cuando hay un alto porcentaje de inserciones y supresiones en una estructura de datos ordenada lógicamente. Es también adecuado el uso de una lista ligada cuando es necesario poner en orden lógico elementos que no están físicamente en secuencia, por ejemplo, para implantar una pila de secciones de espacios disponibles. El costo básico al usar listas ligadas es el espacio que se requiere para los apuntadores, además de la manipulación y manejo de los apuntadores. En el resto del libro usaremos extensamente las listas ligadas, las cuales son fundamentales para las técnicas de manejo de datos.

TERMINOLOGIA

arreglo
cola
lista circularmente ligada
lista doblemente ligada
lista ligada
lista lineal
nodo
nodo principal
nulo
pila
variable apuntadora

REFERENCIAS SUGERIDAS

BERMAN, G. AND A. W. COLIJN. "A modified list technique allowing binary search," *Jour. ACM* 21(2):201–206, April 1974.

CLARK, D. W. "A fast algorithm for copying list structures," *Comm. ACM*, 21(5):351–357, May 1978.

COHEN, J. "Garbage collection of linked data structures," *ACM Computing Surveys*, 13(3):341–367, Sept. 1981.

FISHER, D. A. "Copying cyclic list structures in linear time using bounded workspace," *Comm. ACM*, 18(5):251–252, May 1975.

LINDSTROM, G. "Copying list structures using bounded workspace," *Comm. ACM*, 17(4):198–202, April 1974.

POOCH, U. W. AND A. NIEDER. "A survey of indexing techniques for sparse matrices," *ACM Computing Surveys*, 5(2):109–133, June 1973.

SCHNEIDERMAN, B. AND P. SCHEUERMANN. "Structured data structures," *Comm. ACM*, 17(10):566–574, Oct. 1974.

EJERCICIOS DE REPASO

1. Llene los datos del apuntador siguiente, de manera que el siguiente arreglo que representa una lista ligada quede ordenado alfabéticamente, según el contenido del campo Info. ¿Cómo se vería si utiliza una pila de espacio disponible?

	INFO	SIG
1	PAPAYA	
2	AGUACATE	
3		
4	CHABACANO	
5	GUAYABA	

Figura P6-1

2. Una lista de elementos aparece en un arreglo como sigue:

	DATA	SIG.
1	ITEM G	
2	ITEM D	
3	ITEM J	
4	ITEM B	
5	ITEM E	
6	ITEM F	
7		
8		
9		

Figura P6-2

Muestre el contenido de los campos siguientes para ligar los elementos en orden alfabético.

3. Inserte los elementos C, A y Z en el arreglo anterior, de tal manera que la lista ligada resultante quede en orden alfabético.
4. Escriba un algoritmo para encontrar el J-ésimo nodo en una lista ligada sencilla.
5. Escriba un algoritmo para regresar una lista circular ligada T al banco de nodos disponibles.
6. ¿Cuál sería el resultado de la función SIG(ANTERIOR(P)) en una lista doblemente ligada?
7. Sea X un apuntador a algún nodo arbitrario en una lista ligada. Escriba un algoritmo que guarde '1234' en el campo INFORMACION de un nodo adicional e inserte este nodo adicional después del nodo apuntado por X.
8. Escriba un algoritmo para insertar un nodo S, en una lista doblemente ligada, inmediatamente antes del nodo X.
9. Escriba un algoritmo para invertir una lista ligada no circular, y otro para invertir una lista doblemente ligada.
10. Escriba los algoritmos correspondientes a la asignación de espacio para un nodo, y para la liberación de espacio de un nodo, los cuales son almacenados en una estructura de tipo cola.
11. ¿Cuántas ligas deberán cambiarse para borrar un ítem de una lista doblemente ligada?
12. Escriba los algoritmos para agregar y suprimir elementos de una pila representada por una lista ligada.
13. Escriba los algoritmos para agregar y suprimir elementos de una cola representada por una lista ligada.
14. Escriba una rutina para concatenar dos listas ligadas A y B.
15. ¿Cómo puede ser representada la siguiente matriz dispersa mediante listas ligadas?

0	0	5	2
0	10	4	0

0	0	0	2
0	0	0	0

16. Escriba un algoritmo para borrar el nodo que contiene "GATO" de la siguiente lista ligada:

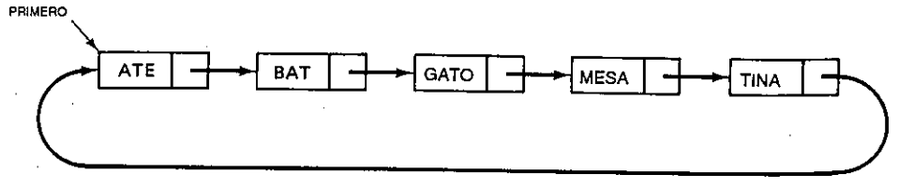

Figura P6-16

17. Escriba un algoritmo para separar la lista anterior entre los nodos "BAT" y "GATO" y designe un nodo-principal, de la lista de Disponibilidad, al comienzo de la nueva lista.
18. ¿Por qué es recomendable una lista ligada circular para el problema de José? Escriba el algoritmo para resolver el problema de José.
19. Ilustre el uso de las listas ligadas para representar los polinomios: $5x^3 + 7x^2 + 9$, y $3x^2 + 4x + 7$.
20. Muestre cómo se pueden utilizar las listas ligadas para representar los siguientes polinomios:

$$7x^4y - 3x^2y^2 + 5y + 12y^2 - 2$$
$$9x^3y^2 + 2x^2y^2 - 11xy + 3y^3$$

21. Escriba un algoritmo para contar el número de nodos en una lista ligada.
22. Considere el uso de un arreglo para almacenar una lista ligada de información de un inventario de libros. El arreglo se define en COBOL, como sigue:

```
01 ARREGLO-LIBROS.
  02 NODO-LIBRO OCCURS 15 TIMES.
     03 AUTOR                  PIC X(20).
     03 SIG-AUTOR              PIC 99.
     03 TITULO                 PIC X(30).
     03 SIG-TITULO             PIC 99.
     03 NUM-ALMACEN            PIC 9(5).
     03 SIG-NUM-ALMACEN        PIC 99.
     03 ANTERIOR-NUM-ALMA      PIC 99.
     03 SIG-DISPONIBLE         PIC 99.

  02 CABEZAS.
     03 PRIMER AUTOR           PIC 99.
     03 PRIMER TITULO          PIC 99.
     03 PRIMER-NUM-ALMACEN     PIC 99.
     03 ULTIMO-NUM-ALMACEN     PIC 99.
     03 DISPONIBLE             PIC 99.
```

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