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.

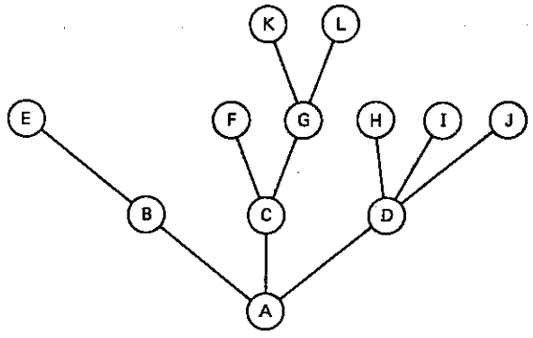

Figura 8-1. Ejemplo de un árbol.

En la figura 8-1 se muestra un árbol con nodos etiquetados con una letra dentro de un círculo. Esta es la notación que usaremos normalmente al dibujar árboles. Aquí la Raíz(T) = A. Los tres subárboles de la raíz A, están enraizados en B, C, y D respectivamente. B es la raíz de un árbol con un subárbol, cuya raíz es E, y tal subárbol ya no tiene subárboles. El árbol con raíz en C tiene dos subárboles, con raíz en F y G respectivamente.

Algunas veces aplicamos características direccionales a las aristas del árbol. Una arista va *desde* un nodo raíz *hasta* una raíz de un subárbol, tal como se muestra en la figura 8-2. Aquí hemos dibujado el árbol de la manera típica de los textos de manejo de datos, con la raíz en la parte superior y sus arcos (o *ramificaciones*) creciendo hacia abajo.

Hemos dicho que un árbol tiene un nodo, especialmente designado, llamado raíz. La raíz no está simplemente seleccionada de manera aleatoria; más bien es un nodo que se distingue por la propiedad de:

Grado-interno (v) = 0 para v = Raíz(T).

La raíz no tiene ramificaciones de entrada. Debido a que un árbol es un grafo conexo, no puede haber más de un nodo con esta propiedad. Considere los subárboles de A, del árbol de la figura 8-2, tal como se muestra en la figura 8-3. Nótese que las aristas que

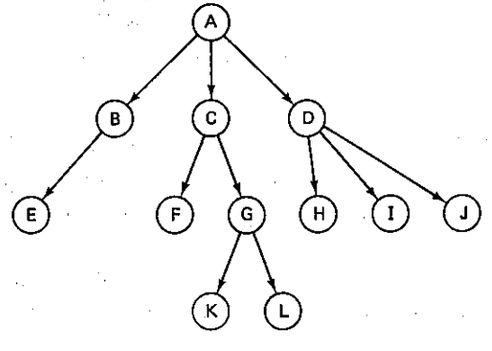

Figura 8-2 Arbol con aristas dirigidas.

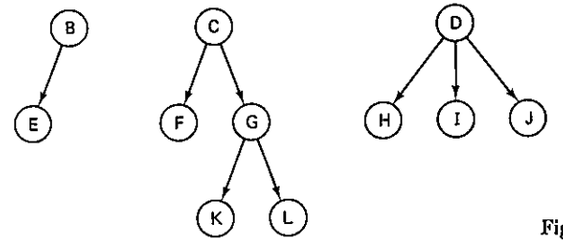

Figura 8-3 Tres subárboles de la figura 8-2

conectan A a B, C y D, no aparecen en estos subárboles, porque A no es uno de sus nodos.

La terminología usada al hablar sobre árboles generales es similar a la que se usa con árboles genealógicos y árboles vivientes. Si existe una arista (A, B), entonces A se dice que es el *padre* de B; y B se dice que es el *hijo* de A. Un nodo hijo, por lo tanto, tiene sólo un nodo padre. Dos nodos con el mismo nodo padre, se dice que son nodos *hermanos*, y así sucesivamente con cada nodo del árbol. Los nodos con grado externo igual a cero se dice que son *hojas* del árbol. Los nodos hojas del árbol de la figura 8-2 son E, F, K, L, H, I y J.

Se dice que los nodos de un árbol están en *niveles*, donde el nivel de los nodos se determina por la longitud de la trayectoria desde la raíz hasta dicho nodo. Por ejemplo, en el árbol de la figura 8-2:

nivel 0	A
nivel 1	B, C, D
nivel 2	E, F, G, H, I, J
nivel 3	K, L.

La *altura* de un árbol es igual a uno más el número del nivel más alto sobre el cual hay nodos. El *peso* del árbol es el número de nodos hoja. La altura y peso del árbol de la figura 8-2 son 4 y 7 respectivamente.

Una colección de árboles enraizados se llama *bosque*. La figura 8-3 muestra un bosque de tres árboles.

Formas de representación

Hemos usado una notación gráfica para representar árboles. Otras notaciones gráficas para estructuras de árboles incluyen 1) conjuntos anidados (Figura 8-4) y 2) paréntesis anidados:

(A(B(E))(C(F)(G(K)(L))) (D(H)(I)(J)))

en donde los paréntesis encierran a la raíz y a los subárboles de cada árbol, y 3) la indentación como se muestra en la figura 8-5.

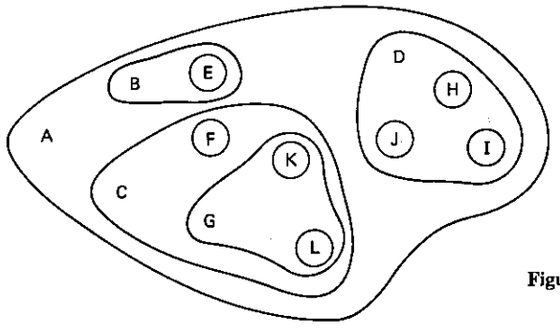

Figura 8-4 Representación de conjuntos anidados para el árbol de la figura 8-2.

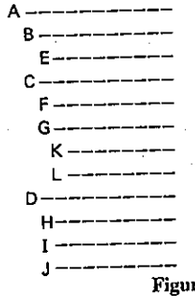

Figura 8-5 Representación sangrada del árbol de la figura 8-2.

ARBOLES BINARIOS

La clase más importante de árboles generales son los árboles binarios. Un *árbol binario* es un conjunto finito de nodos, el cual puede ser vacío o, puede contener un par de árboles binarios disjuntos, que son llamados subárbol *izquierdo* y subárbol *derecho*. Note que, además del requerimiento de que el grado externo máximo de cualquier nodo de un árbol binario sea 2, existe la restricción adicional de que a los subárboles se les impone la identificación como derechos o izquierdos. Los árboles binarios de la figura 8-6a) y b) *no* son los mismos; son dos árboles diferentes, uno con un subárbol izquierdo y otro con un subárbol derecho. El árbol de la figura 8-6c) no es un árbol binario, porque su subárbol no es izquierdo ni derecho.

Se dice que dos árboles binarios son *similares* si tienen la misma estructura, tal y como

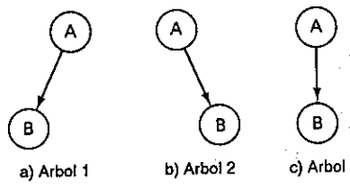

Figura 8-6 Ejemplos de árboles.

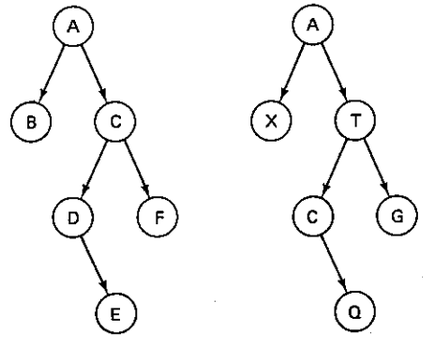

Figura 8-7 Arboles similares.

se ejemplifica en la figura 8-7. Dos árboles binarios son *equivalentes* si son similares y contienen la misma información. Los dos árboles de la figura 8-7 no son equivalentes.

Consideremos ahora algunas propiedades de los árboles binarios que los distinguen de los árboles generales. Considere un árbol binario con K niveles, como el que representa en la figura 8-8. Note que ahora empezamos a omitir las flechas sobre las aristas del árbol, dado que no existe duda de donde está la raíz. Se dice que un árbol binario está *completo* si contiene el número máximo de nodos posibles para su altura. ¿Cuántos nodos contiene un árbol binario completo con K niveles? El número de aristas que llegan a cualquier nivel (excepto al nivel 0) es dos veces el número de nodos en el nivel anterior. Por lo tanto, el número máximo de nodos en el I-ésimo nivel es 2^I. Así, un árbol binario con K niveles contiene

$$\sum_{I=0}^{K-1} 2^I$$

nodos, lo cual es igual a $2^K - 1$. Un árbol binario completo con tres niveles contiene siete nodos; un árbol binario con diez niveles contiene 1023 nodos.

Un árbol binario con K niveles, se dice que es *casi completo*, si los niveles desde el 0 hasta el $K - 2$ están llenos, y el nivel $K - 1$ se está llenando de izquierda a derecha, tal como se muestra en la figura 8-9, para un árbol de 5 niveles.

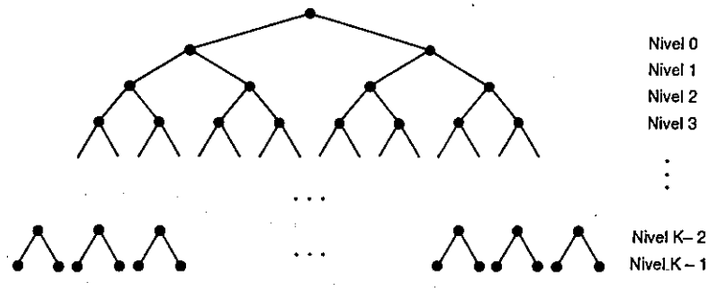

Figura 8-8 Un árbol binario completo con K niveles.

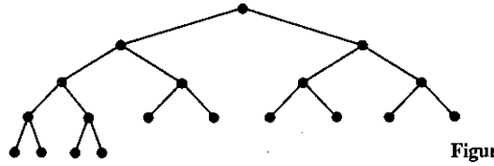

Figura 8-9 Un árbol binario casi completo con 5 niveles.

La longitud de trayectoria máxima (denotada por $PL_{máx}$) para un árbol binario, se alcanza al hacer que un árbol se llene hasta donde sea posible. Un árbol casi completo tiene la mínima altura para su conjunto de nodos. La altura mínima (denotada por $H_{mín}$) para un árbol binario con N nodos es:

$$H_{mín} = \lceil \log_2(N + 1) \rceil$$

donde $[x]$ es una función que produce el entero más pequeño $\geq x$. Cuando $N = 1$ entonces $H_{mín} = 1$. Cuando $N = 3$, el nivel 1 está lleno y $H_{mín} = 2$, cuando $N = 7$, el nivel 2 está lleno y $H_{mín} = 3$. (Recordemos que si $\log_2 X = y$, entonces, $2^y = X$.)

La altura del peor caso se logra cuando el árbol binario se alarga por un solo lado, tal como se muestra en la figura 8-10.

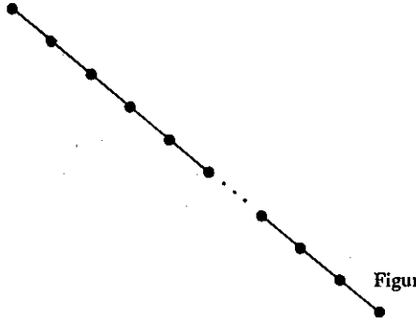

Figura 8-10 Arbol unialargado con altura máxima.

Ahora, la altura máxima (denotada por $H_{máx}$) para un árbol binario con N nodos es

$$H_{máx} = N.$$

Después veremos que uno de los usos más importantes de árboles, es el de estructurar una colección de datos, de tal forma que facilite la búsqueda de un elemento en particular. En general, es aconsejable arreglar los datos en el árbol de tal forma que las longitudes de trayectoria de búsqueda sean mínimas. Por esto, los árboles casi completos se usan frecuentemente.

REPRESENTACION DE ARBOLES BINARIOS

Los árboles binarios se representan frecuentemente por listas ligadas. Cada nodo puede tener tres campos elementales: un área de información y los apuntadores a los subárboles izquierdo y derecho. Por ejemplo, el árbol binario de la figura 8-11 se puede representar por listas ligadas como se muestra en la figura 8-12.

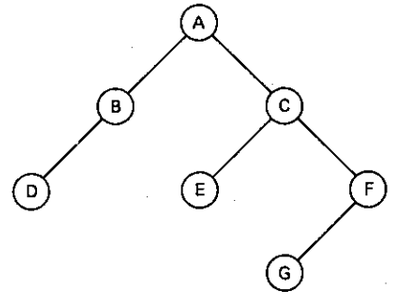

Figura 8-11 Ejemplo de árbol binario.

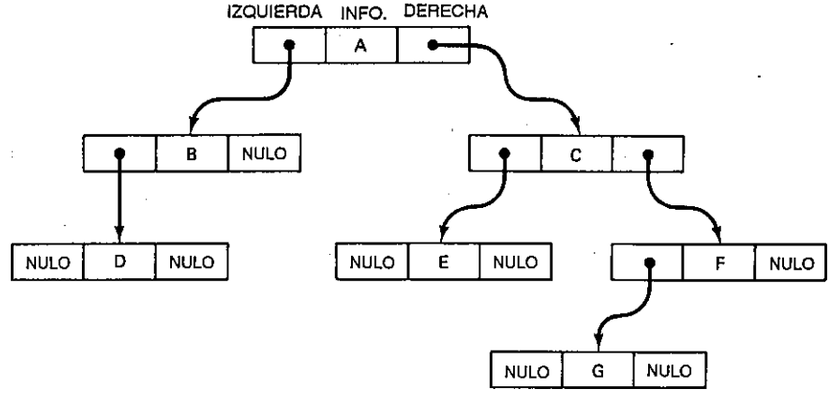

Figura 8-12 Representación de lista ligada del árbol de la figura 8-11.

La colocación topológica de los nodos de la lista ligada en esta página, por supuesto no es significativa; la ubicación real de los nodos en el almacenamiento puede estar diseminada a través de la memoria.

Definición de árboles en Pascal

En Pascal, una estructura de datos de un árbol binario se puede definir usando apuntadores, de la misma manera como lo hicimos para listas ligadas, en el capítulo seis. Supongamos que los campos de información contienen valores enteros.

```
type apuntador-a-nodo = ↑ tipo-nodo;
     tipo-nodo = record
                    izquierdo : apuntador-a-nodo;
                    información : integer;
                    derecho : apuntador-a-nodo
                 end;
```

Cada vez que se crea un nodo, tiene esta estructura. El programa debe inicializar los valores de los campos apuntadores (izquierdo y derecho) correctamente, para establecer las interconexiones entre los nodos que forman el árbol. El espacio puede ser asignado dinámicamente a los nodos a medida que son agregados al árbol y puede ser liberado dinámicamente, a medida que se remueven los nodos.

Definición de árboles en COBOL

En COBOL, un árbol binario con un máximo de 500 nodos se puede declarar de la siguiente manera:

```
01  ARBOL-BINARIO.
    02  NODO OCCURS 500 TIMES.
        03  INFORMACION    PICTURE 9(3).
        03  IZQUIERDA      PICTURE 9(5).
        03  DERECHA        PICTURE 9(3).
```

Debido a que COBOL no maneja variables tipo apuntador, el árbol binario se aloja en un arreglo. Para manejar el espacio en el arreglo se requiere usar una lista de DISPONIBILIDAD de nodos vacíos, tal y como se vio en el capítulo 6. Cabe aclarar que la asignación del espacio debe ser lo suficientemente grande para almacenar el tamaño máximo del árbol puesto que COBOL no permite asignaciones dinámicas.

ARBOLES BINARIOS COMO REPRESENTACION DE ARBOLES GENERALES

Es considerablemente más fácil representar árboles binarios en programas que representar árboles generales. Con un árbol general, es impredecible cuantificar el número de nodos que surgirán a partir de otro nodo en un momento dado. Por esto, el espacio ha de manejarse de tal forma que a cada nodo se le permita un número variable de apuntadores a subárboles ó que, a cada nodo le sea asignado espacio para un número fijo de apuntadores a subárboles, ya sea que se necesiten todos o ninguno. Los árboles binarios resultan más atractivos de usar, puesto que cada nodo tiene un máximo número predecible de apuntadores a subárboles: 2.

Afortunadamente, hay una técnica directa para convertir un árbol general a un árbol binario. El algoritmo consiste en dos pasos muy sencillos:

1. Insertar aristas conectando a nodos hermanos y eliminar todas las aristas que conectan a los nodos padre con sus nodos hijo, excepto el nodo más a la izquierda.

2. Girar el diagrama resultante 45 grados para distinguir entre los subárboles izquierdo y derecho.

Por ejemplo, considere el árbol general de la figura 8-2, el cual se reproduce en la figura 8-13a), es obvio que no es un árbol binario. En esta figura puede ver que hay nodos con más de dos subárboles, y que los subárboles no tienen identificado cuales son

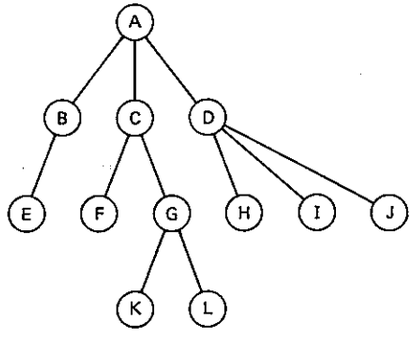

Figura 8-13a) Ejemplo de árbol general.

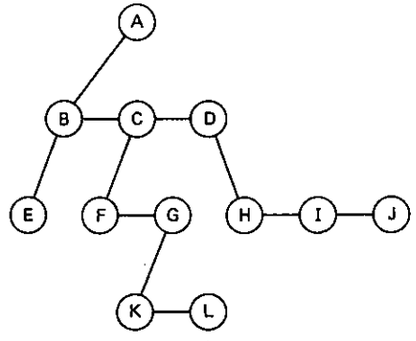

Figura 8-13b) Transformación inicial del árbol general de la figura 8-13a).

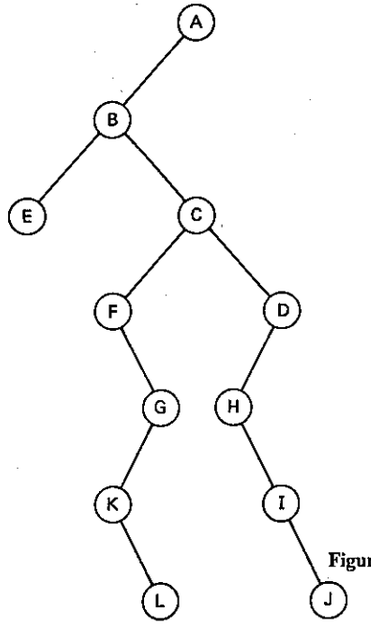

Figura 8-13c) Transformación final de árbol general figura 8-13a) a árbol binario.

los nodos derechos e izquierdos. El paso 1 de la transformación produce el resultado mostrado en la figura 8-13b). Note que *no* hay una arista que conecte a los nodos E y F, y no son nodos hermanos en el árbol general, porque tienen diferentes padres (B y C). El paso 2 produce el árbol binario mostrado en la figura 8-13c).

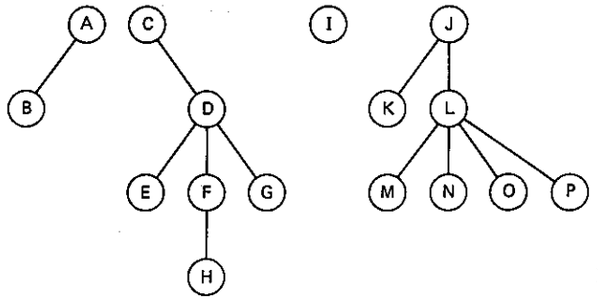

Figura 8-14a) Ejemplo de un bosque.

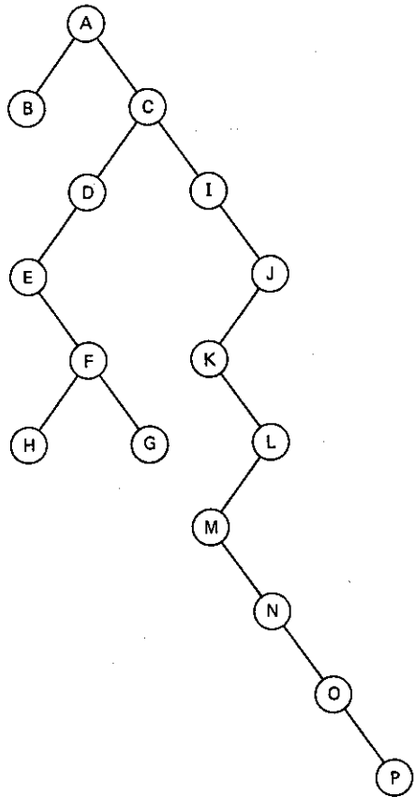

Figura 8-14 b) Representación de árbol binario del bosque de la figura 8-14a).

En el árbol binario resultante, los apuntadores izquierdos son siempre de un nodo padre a su primer nodo hijo (más izquierdo) del árbol general original. Los apuntadores derechos salen siempre de un nodo a uno de sus nodos hermanos en el árbol original.

¿Puede reconstruirse un árbol general a partir del árbol binario resultante? ¿Qué sucede cuando se aplica esta transformación a árboles generales que ya son árboles binarios?

Este algoritmo de transformación también se puede aplicar para convertir un bosque de árboles generales a un árbol binario individual, considerando que las raíces de los árboles son hermanos. Por ejemplo, el bosque de la figura 8-14a) se puede representar por el árbol binario de la figura 8-14b).

Los árboles binarios generados por este tipo de conversión tienen alturas bastantes grandes para el número de nodos que contienen. Sus trayectorias de búsqueda son relativamente largas, pero el manejo de memoria no es complicado.

EJEMPLOS DE ARBOLES

Veremos que los árboles se usan con frecuencia para estructurar datos y facilitar la búsqueda de nodos en particular. Los árboles también son útiles para representar colecciones de datos que tienen estructuras lógicas ramificadas. Por ejemplo, los árboles binarios 1 y 2 de la figura 8-15 representan instrucciones aritméticas. Cada nodo sin hojas es un operador y sus subárboles izquierdos y derechos son sus operandos. Los compiladores frecuentemente construyen árboles binarios en el proceso de reconocimiento léxico-gráfico, reconocimiento sintáctico, e interpretación semántica al generar el código para la evaluación de expresiones aritméticas. El árbol 3 de la figura 8-15 representa una colección de elementos de datos, que están posicionados en el árbol de tal manera que si K es la etiqueta de un nodo, todas las etiquetas de los nodos en su correspondiente subárbol izquierdo son menores (o iguales) a K y todas las etiquetas de los nodos en su subárbol derecho son mayores que K. Esta clase de árbol se usa con frecuencia para estructurar colecciones de nombres, llaves, o etiquetas. La búsqueda de

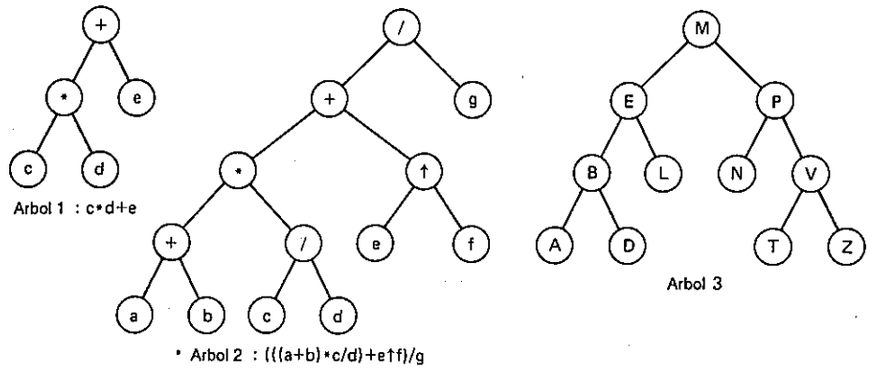

Figura 8-15 Ejemplo de árboles.

un nodo en el árbol puede ser considerablemente más rápida que la búsqueda en la misma colección en forma secuencial. Consideremos esta clase de árboles más en detalle.

ARBOLES DE BUSQUEDA BINARIOS

Un *árbol de búsqueda binario* sobre la colección de n registros con llaves $K_1, K_2, \ldots, K_n$ es un árbol binario, donde cada uno de sus nodos R_i contiene una de las llaves K_i, para

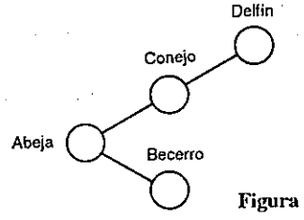

Figura 8-16 Ejemplo de árbol binario de búsqueda.

$i = 1, 2, \ldots, n$. Las llaves son los identificadores de los nodos. La búsqueda de un nodo particular se hará, buscando su valor de llave. Los nodos del árbol binario están arreglados de tal modo que, para cada nodo R_i, se aplican las siguientes propiedades:

1. Todas las llaves de los nodos en el subárbol izquierdo de R_i preceden a la llave etiquetada por R_i.

   Si R_j pertenece a izquierdo(R_i)

   entonces $K_j < K_i$.

2. La llave etiquetada por R_i precede a las llaves de todos los nodos en el subárbol derecho de R_i.

   Si R_j pertenece a derecha(R_i)

   entonces $K_i < K_j$.

En general requerimos que las llaves en la colección sean distintas aunque esta restricción algunas veces es eliminada. La precedencia de llaves es determinada por su ordenamiento lineal, de acuerdo a una secuencia de ordenamiento.

Por ejemplo, el árbol binario de la figura 8-16 se sujeta a las propiedades ya mencionadas y se califica como un árbol de búsqueda binario sobre las llaves ABEJA, BECERRO, CONEJO Y DELFIN.

Es importante notar que hay muchas maneras de ordenar las llaves para que la búsqueda binaria sea válida. Por ejemplo, la figura 8-17 muestra cuatro árboles de búsqueda binarios con las mismas llaves. ¿Usted puede construir otras?

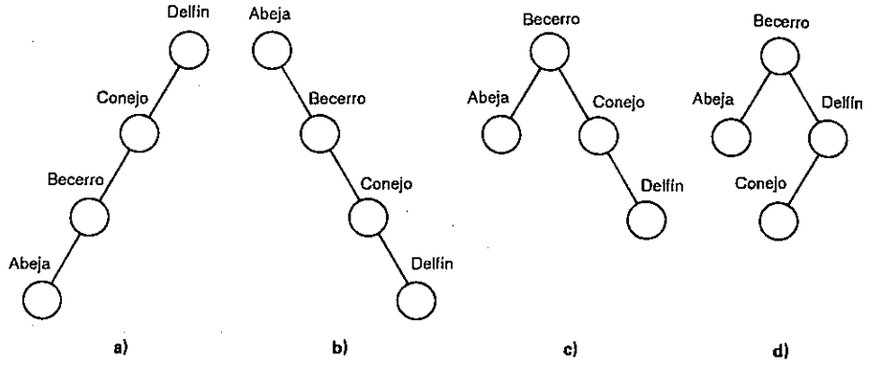

Figura 8-17 Más árboles binarios de búsqueda.

Cuando construya árboles de búsqueda binarios, podrá notar que una vez que ha comenzado el árbol y ha decidido qué llave debe insertar primero, no hay duda para saber donde colocar la siguiente. Las propiedades que definen un árbol de búsqueda binario dictan las relaciones entre los nodos y determinan donde debe residir el siguiente nodo. La estructura de un árbol de búsqueda binario particular está determinada por el orden en el cual los nodos son colocados en el árbol. A menos que permitamos que el árbol sea reestructurado durante el proceso de inserción, un nuevo nodo es siempre insertado como nodo hoja.

La estructura de nodos para un árbol de búsqueda binario puede ser declarada en Pascal como:

```
type apuntador-a-nodo = ↑ tipo-nodo;
     nodo-llave = packed array [1..10] of char;
     tipo-nodo = record
                   izq:apuntador-a-nodo;
                   llave:nodo-llave;
                   información:integer;
                   der:apuntador-a-nodo
                 end;
```

para nodos con llaves de diez caracteres e información de tipo entero.

BUSQUEDAS SECUENCIALES

Un proceso para desplazarse a través de un árbol en forma tal que, cada nodo sea visitado una y sólo una vez se llama *recorrido de árbol.* Cuando un árbol se recorre, su colección total de nodos se revisa. Hay varios métodos bien conocidos de recorrido de árboles binarios. Cada uno impone una ordenación secuencial y lineal de los nodos de un árbol. Se dice que un nodo es *visitado* cuando se encuentra en el recorrido y se puede efectuar cualquier proceso en ese momento sobre su contenido.

Los algoritmos

Hay tres clases de actividades básicas en cada uno de los algoritmos de recorrido de árboles binarios:

Visitar la raíz.
Recorrer el subárbol izquierdo.
Recorrer el subárbol derecho.

Los métodos difieren en el orden en que estos tipos de actividades son ejecutados. Los algoritmos que aquí presentamos son todos recursivos; cada uno está definido en sus propios términos.

RECORRIDO EN PRE-ORDEN

1. Visitar la raíz.
2. Recorrer el subárbol izquierdo en pre-orden.
3. Recorrer el subárbol derecho en pre-orden.

RECORRIDO EN EN-ORDEN

1. Recorrer el subárbol izquierdo en en-orden.
2. Visitar la raíz.
3. Recorrer el sub-árbol derecho en en-orden.

RECORRIDO EN POST-ORDEN

1. Recorrer el subárbol izquierdo en post-orden.
2. Recorrer el subárbol derecho en post-orden.,
3. Visitar la raíz.

Los nodos de los árboles de la figura 8-15 son visitados en las siguientes secuencias, cuando los árboles se recorren de acuerdo a los métodos ya descritos.

Arbol 1:	Pre-orden:	+*cde
	En-orden:	c*d+e
	Post-orden:	cd*e+
Arbol 2:	Pre-orden:	/+*+ab/cd ↑ efg
	En-orden:	a+b*c/d+e ↑ f/g
	Post-orden:	ab+cd/*ef ↑ +g/
Arbol 3:	Pre-orden:	MEBADLPNVTZ
	En-orden:	ABDELMNPTVZ
	Post-orden:	ADBLENTZVPM

El resultado de los recorridos de los árboles 1 y 2 pueden recordarle el ejemplo de aplicación de pilas, en el capítulo 4, para la notación de expresiones aritméticas. El recorrido en Post-orden resulta en una representación de cadena en forma postfija; un operador es precedido por sus dos operandos. El recorrido en pre-orden resulta en una representación de cadena en forma prefija; un operador precede a sus operandos. El recorrido en en-orden resulta en una representación de cadena en forma infija; un operador aparece entre sus operandos. Note que las representaciones en prefijo e infijo pueden perder la precedencia apropiada del operador si las expresiones aritméticas originales requieren el uso de paréntesis para invalidar las reglas naturales de precedencia. En la notación postfija, se conserva la precedencia del operador y se indica solamente por el orden del operador. Para este tipo de ejemplo, el recorrido en post-orden es el más adecuado.

En contraste, el recorrido en-orden del árbol 3 parece que da mejores resultados, pues los nodos se recorren en orden alfabético. El árbol 3 es un árbol de búsqueda binario donde el recorrido En-Orden es usado para búsquedas secuenciales.

El recorrido en pre-orden es el más útil en otro tipo de aplicaciones. Una de las más importantes de éstas es el recorrido de árboles manejado mediante el manejo Jerárquico de bases de datos en sistemas como el Information Management System (IMS) de IBM. El recorrido en pre-orden es equivalente al orden de *secuencia jerárquica* del IMS.

Por lo tanto, las tres técnicas de recorrido son importantes y tienen aplicación práctica. La técnica más adecuada para una situación particular está determinada por la manera en la cual la información se haya estructurado en el árbol.

Recorrido en-orden

Un procedimiento en Pascal para recorrer un árbol de búsqueda binario en en-orden, es el siguiente. Note que este procedimiento, como las definiciones de recorridos, es recursivo.

```
procedure en-orden(var raíz:apuntador-a-nodo);
begin if raíz < > nil
        then begin en-orden(raíz ↑.izquierdo);
                   writeln(raíz ↑ .información);
                   en-orden(raíz ↑ .derecho)
             end;
end;
```

Las estructuras de los nodos y apuntadores fueron definidas con anterioridad. Es relativamente fácil escribir los procedimientos recursivos correspondientes para implantar los recorridos en pre-orden y post-orden. Los recorridos en post-orden y pre-orden probablemente no serían usados en árboles de búsqueda binarios, pero se podrían usar para otro tipo de árboles binarios.

Recorrido en-orden no recursivo

En algunas implantaciones, rutinas no recursivas que explícitamente ejecutan su propio apilamiento y desapilamiento son más eficientes que las rutinas recursivas. Un algoritmo

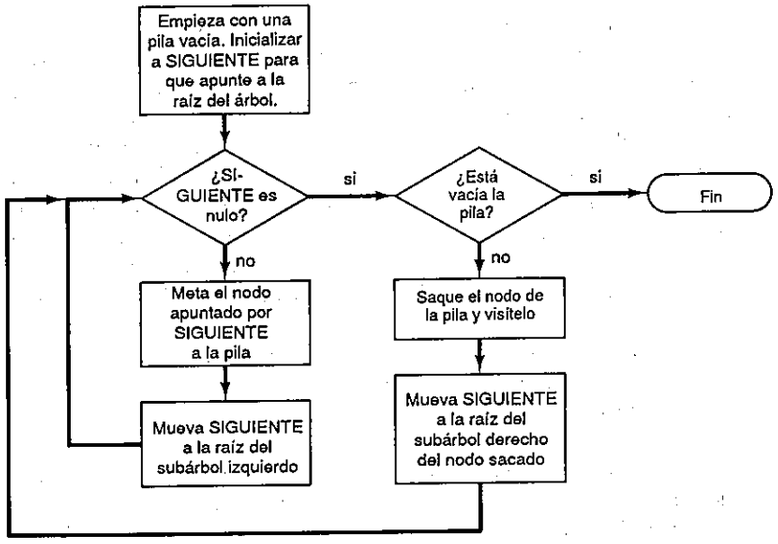

Figura 8-18 Algoritmo no recursivo para recorrido en en-orden.

no recursivo para recorrer un árbol binario en En-orden se muestra en la figura 8-18. Cada nodo del árbol se mete en una pila de tal forma que se saca en en-orden. El algoritmo primero busca el nodo izquierdo más hacia abajo, después su raíz y después el nodo derecho (el cual puede tener subárboles), después sube un nivel (que es un nivel más abajo en la pila) para visitar esa raíz y conservar su nodo derecho, y así sucesivamente hasta que todos los nodos hayan sido visitados.

Recorrido en-orden en Pascal

En seguida se muestra un procedimiento en Pascal para implantar este algoritmo. El procedimiento utiliza el tipo de variable apuntador predefinido para implantar el árbol binario y un arreglo para implantar la pila que se necesita para registrar la localidad de los nodos en el árbol durante el recorrido. El apuntador para la raíz del árbol es raíz.

```
procedure en-orden(raíz : apuntador-a-nodo);
type estructura-pila = record pila:array [1 . . 500] of apuntador-a-nodo;
                              apuntador-tope:0 . . 500
                       end;
var siguiente:apuntador-a-nodo;
    bandera-recorrido:boolean;
    p:estructura-pila;
begin p.apuntador-tope : = 0;
      siguiente : = raíz; {comienza en la raíz}
      bandera-recorrido : = true;
```



```
      while (bandera-recorrido)
      do begin {construye la pila de apuntadores a los nodos}
            {el ciclo se detiene cuando encuentre un apuntador nulo izquierdo}
         while (siguiente < > nil)
         do begin p.apuntador-tope : = p.apuntador-tope + 1;
                  p.pila[p.apuntador-tope] : = siguiente;
                  siguiente : = siguiente ↑ .izquierdo
            end;
         {visita al nodo del tope de la pila}
         if (p.apuntador-tope > 0)
         then begin siguiente : = p.pila [p.apuntador-tope];
                    p.apuntador-tope : = p.apuntador-tope - 1;
                    writeln (siguiente ↑ .información );
                    {recorre el subárbol derecho}
                    siguiente : = siguiente ↑ .derecho
              end;
         else bandera-recorrido : = false; {la pila está vacía}
      end;
end;
```

Recorrido post-orden en COBOL

Un procedimiento no recursivo en COBOL para recorrer un árbol binario en post-orden sigue. Este procedimiento aloja al árbol binario y a la pila en arreglos. Nosotros suponemos que el árbol binario tiene un máximo de 500 nodos y que el valor 0 se usa para simular un apuntador nulo.

El árbol y la pila se declaran en la *DIVISION DE DATOS.*

```
01  ARBOL BINARIO.
    02  NODO                OCCURS 500 TIMES.
        03  INFORMACION     PICTURE 9(3).
        03  IZQUIERDO       PICTURE 9(5).
        03  DERECHO         PICTURE 9(3).
01  APUNTADORES-AUXILIARES.
    02  RAIZ                PICTURE 9(3).
    02  SIGUIENTE           PICTURE 9(3).
01  ESTRUCTURA PILA.
    02  PILA                OCCURS 500 TIMES.
                             PICTURE 9(3).
    02  APUNTADOR-TOPE      PICTURE 9(3) VALUE 0.
```

En la *DIVISION DE PROCEDIMIENTOS.*

```
POST-ORDEN.
    MOVE RAIZ TO SIGUIENTE.
    PERFORM CICLO.
    PERFORM CICLO UNTIL APUNTADOR-TOPE = 0.
```

donde

```
CICLO.
    PERFORM SALVA-NODO UNTIL SIGUIENTE = 0.
    MOVE PILA (APUNTADOR-TOPE) TO SIGUIENTE.
    COMPUTE APUNTADOR-TOPE = APUNTADOR-TOPE - 1.
    DISPLAY INFORMACION (SIGUIENTE).
    MOVE DERECHO (SIGUIENTE) TO SIGUIENTE.
SALVA-NODO.
    COMPUTE APUNTADOR-TOPE = APUNTADOR-TOPE + 1.
    MOVE SIGUIENTE TO PILA(APUNTADOR-TOPE).
    MOVE IZQUIERDO (SIGUIENTE) TO SIGUIENTE.
```

ARBOLES BINARIOS ENHILADOS

Los procedimientos no recursivos, para recorrer árboles binarios, dados en la sección precedente no son triviales. De hecho, el uso de pilas para registrar cuáles nodos han sido visitados (en especial para recorridos en post-orden) es realmente embrollado. Para facilitar el recorrido de árboles binarios, algunas veces *enhilamos* al árbol con apuntadores que explícitamente muestran el orden de recorrido. De manera intuitiva el enhilado liga a los nodos del árbol en la secuencia del método de recorrido.

Hay dos tipos de enhilado: *el enhilado derecho*, liga a un nodo con su nodo sucesor según el orden de recorrido; *el enhilado izquierdo*, liga a un nodo con su nodo predecesor según el orden de recorrido. A partir de estas definiciones, se debe establecer que un árbol sólo se puede enhilar de acuerdo a un método de recorrido; en pre-orden *o* en-orden *o* en post-orden. (Por supuesto un árbol se puede multi-enhilar.) Un árbol de búsqueda binario es normalmente enlazado en pre-orden.

El enhilado es presentado en las figuras con líneas discontinuas. La figura 8-19a) muestra los tres árboles de la figura 8-15 enhilados en pre-orden; en la figura 8-19b) se muestra el enlazado en en-orden; y la figura 8-19c) muestra el enhilado en post-orden. Solamente se ilustraron los enhilados derechos.

Representación de nodos

Una alternativa para la representación de árboles binarios enhilados consiste en tener simplemente cada nodo estructurado como sigue:

```
type apuntador a-nodo = ↑ tipo-nodo;
     tipo-nodo = record
                   izquierdo:apuntador-a-nodo;
                   información:integer;
                   derecho:apuntador-a-nodo;
                   lazo-der:apuntador-a-nodo;
                   lazo izq:apuntador-a-nodo
                 end;
```

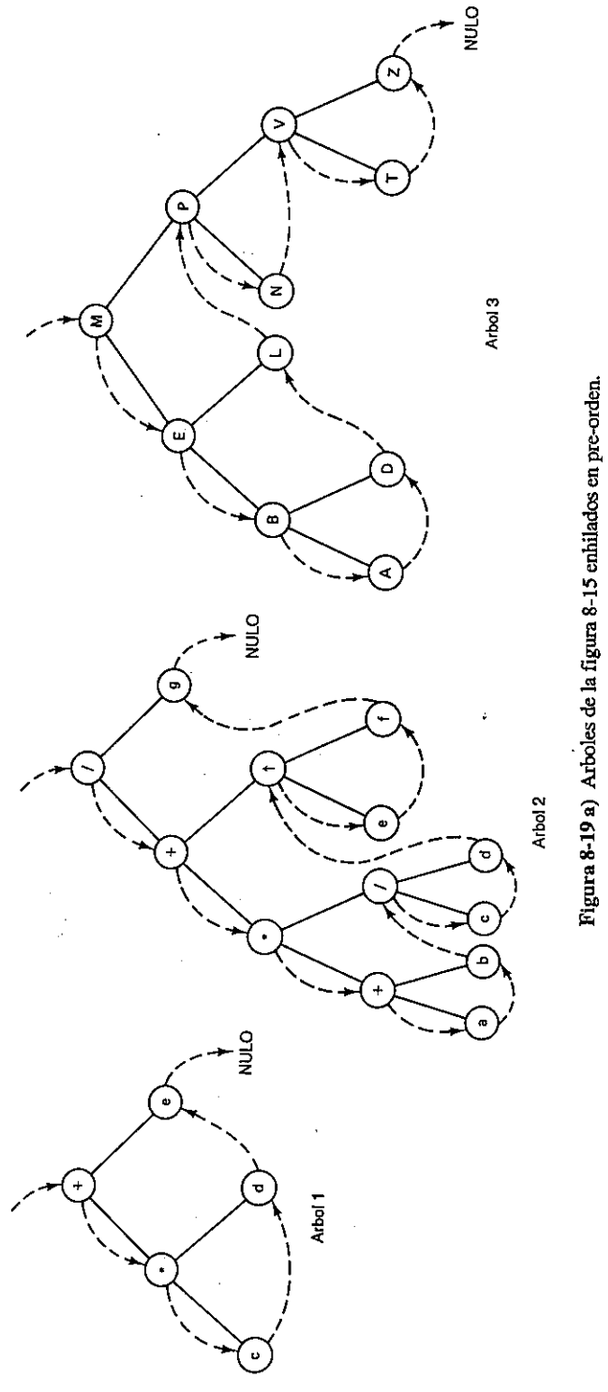

Figura 8-19 a) Arboles de la figura 8-15 enhilados en pre-orden.

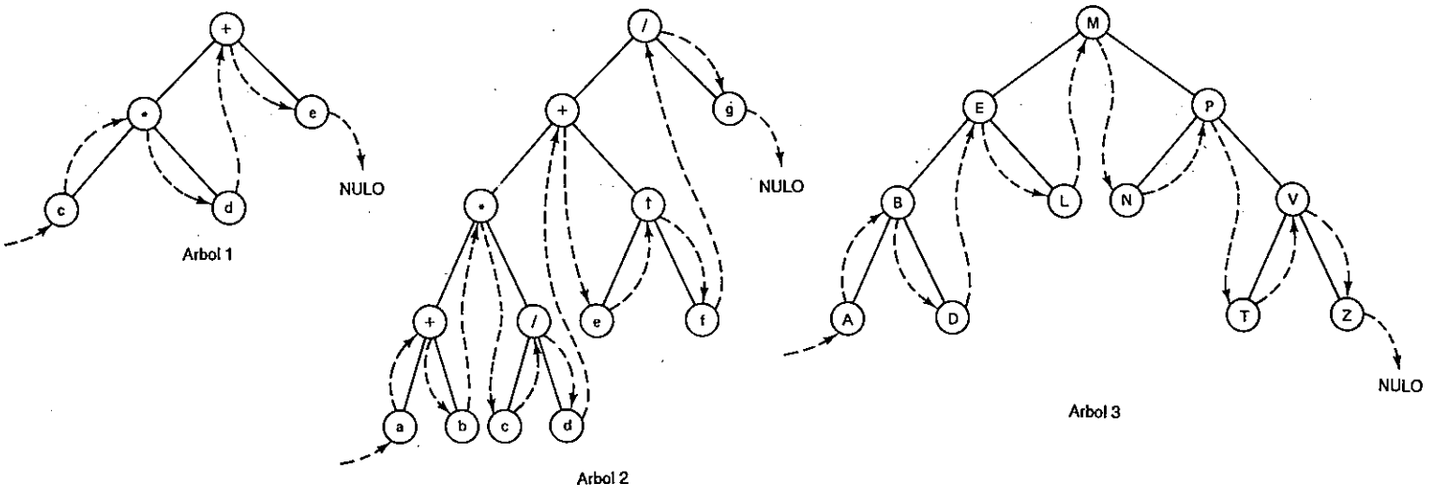

Figura 8-19 b) Arboles de la figura 8-15 enhilados en-orden.

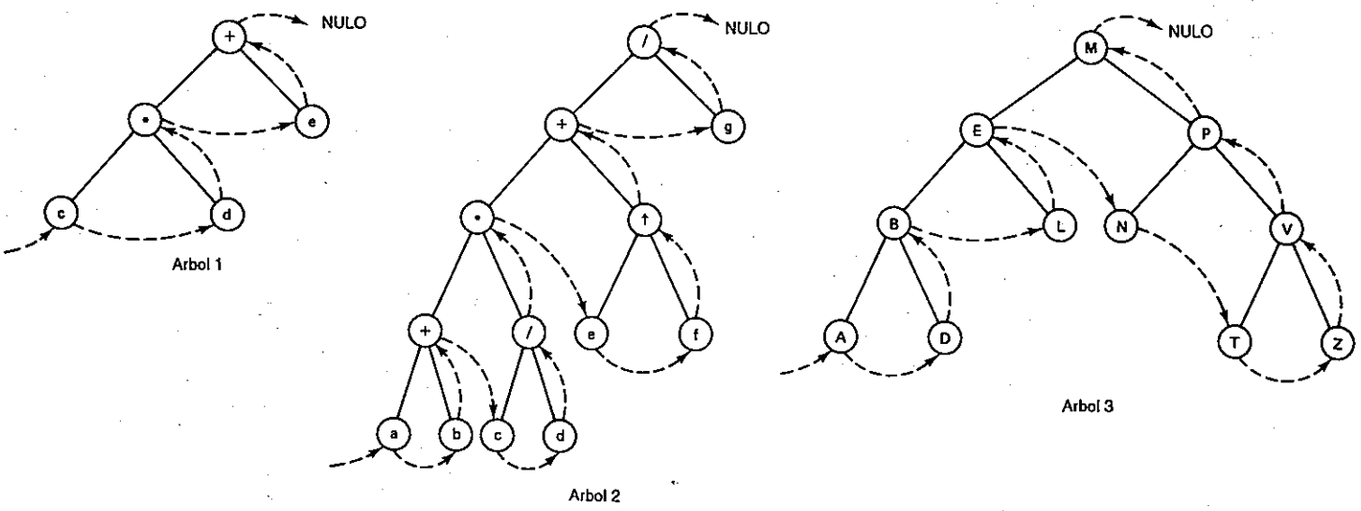

Figura 8-19 c) Arboles de la figura 8-15 enhilados en post-orden.

Cada nodo tiene sus apuntadores usuales derecho e izquierdo, sus campos de información, y sus apuntadores de enhilamiento derecho e izquierdo. Todos los apuntadores son a otros nodos con esta misma estructura. Si el árbol fuera un árbol de búsqueda binario, cada nodo tendría también, un campo llave.

Recorrido en en-orden

A continuación se presenta un procedimiento no recursivo para recorrer en en-orden un árbol binario, el cual está enhilado en en-orden. El apuntador primero-que-entra apunta al primer nodo del árbol en en-orden. Los lazos derechos se siguen desde cada nodo al siguiente.

```
procedure en-orden(primero-que-entra:apuntador-a-nodo);
var p:apuntador-a-nodo;
begin p : = primero-que-entra;
        while p < > nil
        do begin writeln (p ↑ .información);
                 p : = p ↑ .lazo-der
           end;
end;
```

¡Esto es lo más simple que puede ser un algoritmo de recorrido!

Una representación alternativa

Una segunda alternativa para representar un árbol binario enhilado requiere de menos "consumo" de almacenamiento para los apuntadores, pero puede hacer que el manejo de apuntadores y algoritmos de recorrido sea un poco más complejo. Primero, ¿cuántos apuntadores nulos hay en un árbol binario no enlazado con N nodos? Semejante árbol tiene un total de $2N$ apuntadores (nulos y no nulos); N apuntadores para los subárboles izquierdos y N apuntadores para los subárboles derechos. De éstos, sólo $N - 1$ son no-nulos, puesto que cada nodo excepto la raíz tiene sólo un apuntador a él. Por tanto siempre habrá $N + 1$ apuntadores nulos. Si examina un árbol que se ha enhilado en pre-orden, encontrará enhilados de nodos que no son hoja, paralelos a los apuntadores izquierdos reales. En realidad no hay necesidad de duplicar estos apuntadores. Los nodos hoja, que necesitan el lazo, tienen apuntadores nulos. Entonces podemos representar un árbol binario enlazado en pre-orden, teniendo cada nodo estructurado como sigue:

```
type apuntador-a-nodo = ↑ tipo-nodo;
     tipo-nodo = record
                   izquierdo:nodo-puntero;
                   bandera-enhilar:0 . . 1;
                   información:integer;
                   derecho:apuntador-a-nodo;
                 end;
```


Para evitar confusiones entre los apuntadores a subárboles y los enhilados, la bandera de enhilar puede ser 0 cuando el apuntador izquierdo represente un apuntador a un subárbol izquierdo y, cuando represente un lazo derecho pueda ser 1. Alternativamente, cuando un árbol binario sea alojado en un arreglo, los apuntadores izquierdos se pueden representar por valores positivos para el subíndice y, los enhilados derechos se pueden repre sentar por valores negativos para el subíndice, o viceversa. Los árboles de la figura 8-15, enhilados con un sólo apuntador, se muestran en la figura 8-20.

El procedimiento de recorrido en pre-orden es como sigue. Los apuntadores izquierdos son seguidos desde cada nodo al siguiente, algunas veces estos apuntadores son lazos derechos, y algunas veces son verdaderos apuntadores izquierdos.

```
procedure preorden-enhilado(raíz:apuntador-a-nodo);
var p:apuntador-a-nodo;
begin p : = raíz;
        while p < > nil
        do begin writeln(p ↑ .información);
                 p : = p ↑ .izquierdo
           end;
end;
```

Note la similitud de este procedimiento con el que se dio para el recorrido en en-orden de árboles enhilados.

BUSQUEDAS DIRECTAS

Las propiedades de un árbol de búsqueda binario implican la existencia de un procedimiento para determinar si un nodo con una llave dada, reside en el árbol y para encontrar tal nodo cuando exista. Para encontrar el nodo con la llave K en el árbol de búsqueda binario, con raíz en R_i, se deben seguir los siguientes pasos:

1. Si el árbol está vacío, la búsqueda termina sin éxito.
2. Si $k = K_i$, la búsqueda termina satisfactoriamente; el nodo encontrado es R_i.
3. Si $k < K_i$, se debe buscar en el subárbol izquierdo de R_i, es decir, R_i : = izquierdo (R_i).
4. Si $k > K_i$, se debe buscar en el subárbol derecho de R_i, es decir, R_i : = derecho(R_i).

Este algoritmo se programa, en Pascal como un procedimiento recursivo, donde inicialmente r apunta a la raíz del árbol.

```
procedure búsqueda(k : nodo-llave,var r,nodo-encontrado:apuntador-a-nodo);
begin if(r=nil)
        then nodo-encontrado : = nil
        else if (k=r ↑ .llave)
                then nodo-encontrado : = r
                else if (k < r ↑ .llave)
```

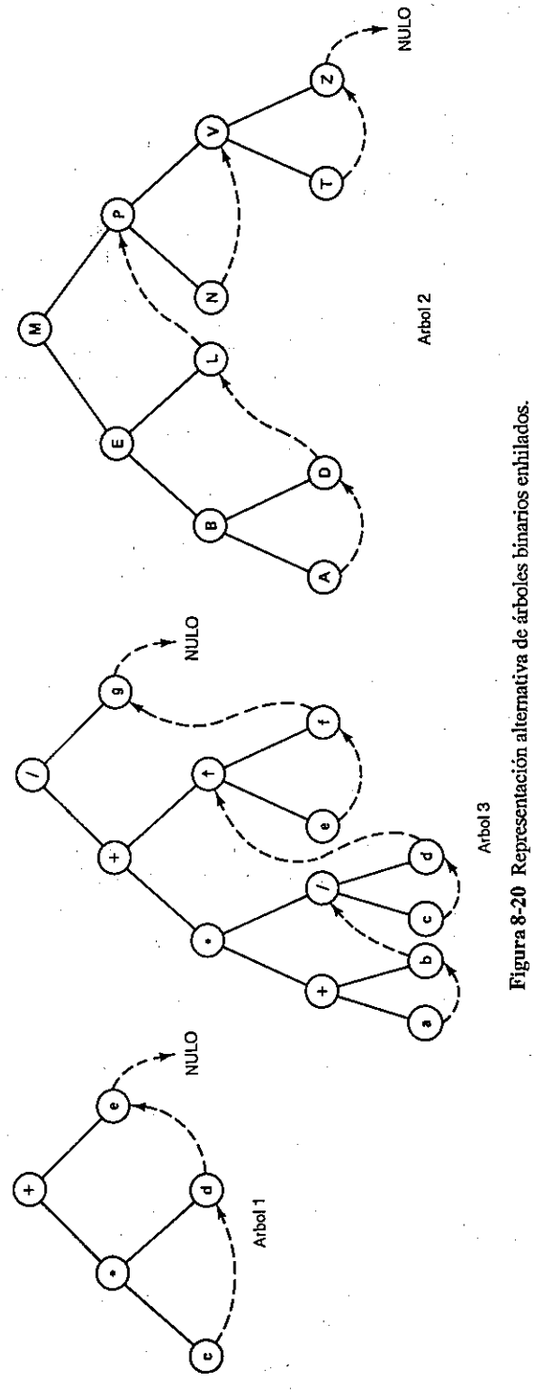

Figura 8-20 Representación alternativa de árboles binarios enhilados.


```
                then búsqueda (k,r ↑ .izquierdo,nodo-encontrado)
                else {k > r ↑ .llave}
                        búsqueda(k,r ↑ .derecho,nodo-encontrado)
    end;
```

La búsqueda termina cuando nodo-encontrado apunta al nodo solicitado, si es que existe, o con un valor nulo si la llave *k* no está en el árbol.

Encontrar un nodo particular en un árbol que no es un árbol de búsqueda binario es usualmente realizado mediante una búsqueda secuencial más que mediante una búsqueda directa, a menos que el árbol esté especialmente estructurado y los nodos tengan campos llave. Una búsqueda secuencial en un árbol con *N* nodos requerirá un promedio de *N*/2 nodos visitados para encontrar un nodo particular. En el peor de los casos, todos los *N* nodos necesitarán ser recorridos para determinar que el nodo buscado no existe en el árbol.

El esfuerzo requerido para encontrar un registro particular en un árbol de búsqueda binario depende de la posición del registro en el árbol. Hemos visto que hay muchos posibles árboles de búsqueda binarios para organizar una colección dada de registros. Entre más lejos de la raíz del árbol se encuentre el registro buscado, más esfuerzo se hará para encontrarlo. El esfuerzo de búsqueda se mide por el número de comparaciones hechas, antes de terminar la búsqueda, con éxito o sin él.

Por ejemplo, encontrar el registro con el nombre DELFIN en los diferentes árboles de búsqueda binarios de la figura 8-17, requiere el siguiente número de comparaciones:

árbol a) 1 comparación.
b) 4 comparaciones.
c) 3 comparaciones.
d) 2 comparaciones.

Si fuera necesario facilitar la búsqueda de DELFIN, el árbol a) sería probablemente mucho mejor que el árbol b). En general, ¿cómo podemos determinar cuando un árbol de búsqueda binario es mejor que otro? ¿Qué es un buen árbol de búsqueda binario? ¿Cómo se pueden construir buenos árboles de búsqueda binarios?

Un árbol de búsqueda binario no puede evaluarse bien sobre la única base de la trayectoria de búsqueda que proporciona para un solo registro. Más bien se deberían evaluar las trayectorias de búsqueda a través de todo el árbol, por lo tanto, la longitud esperada (es decir, el peso promedio) de las trayectorias de búsqueda en el árbol es útil.

Sea s_i la probabilidad de que la llave K_i sea encontrada, donde K_i ($i = 1, \ldots, n$) es una de las llaves del árbol, dado que la búsqueda sea satisfactoria. Se tiene que, la suma de probabilidades es igual a:

$$\sum_{i=1}^{n} s_i = 1.$$

Considerando primero sólo las búsquedas exitosas, la longitud de búsqueda esperada para un árbol binario es:

$$\sum_{i=1}^{n} s_i c_i$$

donde c_i es el número de comparaciones requeridas para alcanzar la llave K_i. $c_i - 1$ es el número del nivel sobre el cual el nodo R_i se encuentra. Por ejemplo, si las probabilidades de acceso para las llaves estructuradas en los árboles de búsqueda binarios de la figura 8-17 fueran:

ABEJA	.2
BECERRO	.4
CONEJO	.3
DELFIN	.1

entonces las longitudes de búsqueda esperada para estos valores serían:

	Abeja		Becerro		Conejo		Delfín		
a)	.2*4	+	.4*3	+	.3*2	+	.1*1	=	2.7
b)	.2*1	+	.4*2	+	.3*3	+	.1*4	=	2.3
c)	.2*2	+	.4*1	+	.3*2	+	.1*3	=	1.7
d)	.2*2	+	.4*1	+	.3*3	+	.1*2	=	1.9

Note que, aquí, para reducir la longitud de búsqueda esperada, el árbol se estructura de tal forma que los nombres accesados con mayor frecuencia, quedan colocados tan cerca como sea posible de la raíz. Para lograr esto, las llaves se deben insertar dentro del árbol en el orden apropiado. Obviamente, ¡habrá problemas con este método, si las probabilidades de acceso son desconocidas! Ante esta situación el diseñador, algunas veces podrá suponer las frecuencias de acceso, y después reestructurar el árbol, habiendo obtenido los valores de uso reales.

Note también que la longitud de la ruta esperada para un árbol de búsqueda binario cambiará a medida que los nodos sean insertados al árbol; lo que puede ser una buena estructura de árbol hoy, mañana podrá ser una estructura relativamente pobre, debido a las inserciones de nodos y a cambios en los patrones de acceso a los contenidos del árbol.

Antes de discutir como acomodar estos cambios, también debemos considerar los efectos de las búsquedas sin éxito. Sea u_i la frecuencia con la cual una búsqueda sin éxito termina en el nodo R_i. Note que $u_i = 0$ para cualquier nodo R_i que tenga sus subárboles derecho e izquierdo no-nulos, porque una búsqueda nunca podrá terminar sin éxito en semejante nodo. Luego, la longitud de búsqueda esperada es:

$$p_s \sum_{i=1}^{n} s_i c_i + p_u \sum_{i=1}^{n} u_i c_i$$

donde p_s es la probabilidad de una búsqueda exitosa y p_u es la probabilidad de una búsqueda sin éxito, y

$$p_s + p_u = 1.$$

En algunas aplicaciones, la p_u puede ser mucho mayor que p_s. Por ejemplo, considere el uso de un árbol de búsqueda binario para almacenar números de tarjetas de crédito robadas en un sistema interactivo de manejo de crédito. Antes de hacer una venta, un vendedor teclea el número de la tarjeta de crédito en una terminal y el sistema busca este número en el árbol. En la mayoría de los casos, la búsqueda termina sin éxito. La longitud de búsqueda esperada se puede minimizar haciendo que estas búsquedas terminen con la menor cantidad de comparaciones posibles. Esto implica que el árbol debería ser tan espeso como sea posible. La longitud de búsqueda se minimiza aquí cuando la longitud de trayectoria máxima en el árbol se minimiza. Note que, aquí también, hay el problema de manipular inserciones impredecibles en el árbol, de tal forma que las ejecuciones de búsqueda no se deterioren.

INSERCION DE NODOS

Las operaciones de inserción de nodos en árboles binarios y la supresión de los mismos son realmente sencillas. Por ejemplo, considere la inserción de un nodo etiquetado como GNU, el cual se convertirá en la raíz del subárbol derecho de algún nodo X de un árbol binario. Existen dos casos: Que GNU se vuelva una hoja o que no se vuelva, dependiendo de si el subárbol derecho de X esté inicialmente vacío. Los resultados posibles al insertar GNU son mostrados en la figura 8-21a).

Inserción desenhilada

Un procedimiento en Pascal, para hacer estas inserciones en un árbol desenhilado, es como sigue:

```
procedure inserción(x : apuntador-a-nodo;var gnu:apuntador-a-nodo);
begin new(gnu);
        gnu ↑ .izquierdo : = nil;
        gnu ↑ .derecho : = x ↑ .derecho;
        x ↑ .derecho : = gnu
end;
```

El aspecto más difícil de la inserción de nodos en árboles es la determinación de donde deben ir.

Inserción enhilada

Si el árbol está enhilado, entonces el procedimiento de inserción es un poco más complejo. La figura 8-21b) muestra al árbol de la figura 8-21a), antes y después de la inserción de GNU, cuando el árbol está enhilado en En-orden.

El procedimiento de inserción es entonces:

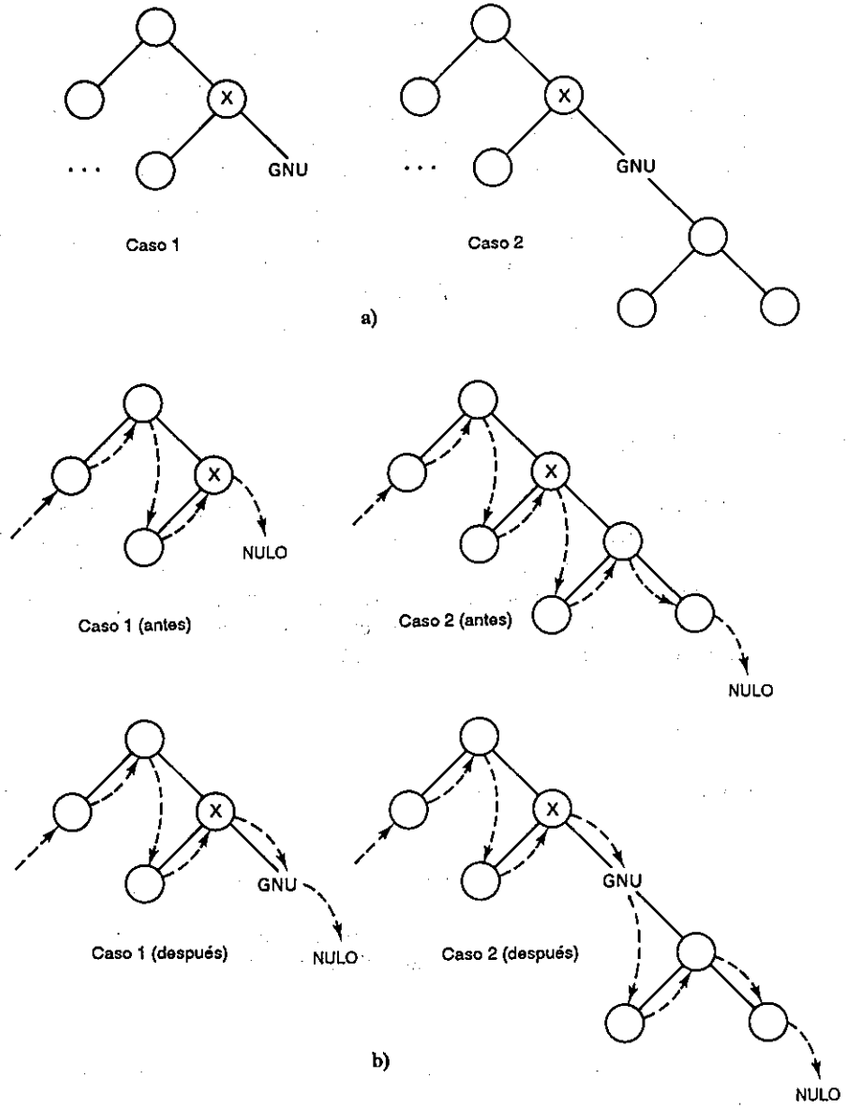

Figura 8-21 Inserción de un nodo.


```
procedure inserción(x:apuntador-a-nodo;var gnu:apuntador-a-nodo);
begin new(gnu);
        gnu ↑ .izquierdo : = nil;
        gnu ↑ .derecho : = x ↑ .derecho;
        gnu ↑ .lazo-izquierdo : = x;
        gnu ↑ .lazo-derecho : = x ↑ .lazo-derecho;
        x ↑ .derecho : = gnu;
        x ↑ .lazo-derecho : = gnu
end;
```

Usted puede desarrollar el algoritmo de inserción para árboles binarios enhilados en pre-orden y post-orden. Para el caso de pre-orden, determine qué diferencias (si existe alguna) son necesarias cuando una bandera es utilizada para indicar que izquierdo puede ser un apuntador de subárbol izquierdo, o bien un lazo derecho.

INSERCION DE NODOS EN UN ARBOL DE BUSQUEDA BINARIO

Un nodo nuevo en un árbol de búsqueda binario es siempre colocado como hoja; la estructura del árbol está dictada totalmente por las propiedades de un árbol de búsqueda binario y por el orden de inserción de los nodos. El algoritmo de inserción es muy similar al algoritmo de búsqueda directa. Para insertar un nuevo nodo con llave k en el árbol de búsqueda binario con raíz en R, *se deben seguir los siguientes pasos:*

1. Si el árbol está vacío, el nodo con llave k será la raíz.
2. Si $k = K_i$ la inserción no podrá hacerse, pues ya existe la llave.
3. Si $k < K_i$, el subárbol izquierdo de R_i es recorrido hasta encontrar la posición adecuada para insertar el nuevo nodo.
4. Si $k > K_i$, el subárbol derecho de R_i es recorrido hasta encontrar la posición adecuada para insertar el nuevo nodo.

Este algoritmo es programado muy fácilmente usando la misma estructura lógica establecida en el procedimiento escrito anteriormente para buscar en un árbol de búsqueda binario. Sea nuevo un apuntador al nodo, que ha de ser insertado en el árbol de búsqueda binario, en *r*. En Pascal:

```
procedure inserción(nuevo: apuntador-a-nodo;var r:apuntador-a-nodo);
begin if (r=nil)
        then r : = nuevo
        else if (nuevo ↑ .llave = r ↑ .llave)
            then ENTRADA-DUPLICADA
            else if (nuevo ↑ .llave < r ↑ .llave)
                then inserción(nuevo,r ↑ .izquierdo)
                else {nuevo ↑ .llave > r ↑ .llave}
                    inserción(nuevo,r ↑ .derecho)
end;
```

La condición que causa que el procedimiento de búsqueda termine sin éxito, es la condición que causa que el procedimiento de inserción termine con éxito, y la condición que causa que la búsqueda termine con éxito es la misma condición para que la inserción termine sin éxito.

Note que, la inserción ordenada de llaves en un árbol de búsqueda binario, produce un árbol largo sin ramificaciones (véase el árbol b) de la figura 8-17).

SUPRESION DE NODOS

Eliminación enlazada

La supresión de nodos de un árbol binario es similar a la inserción, en cuanto está involucrado el "cambio" de los apuntadores a subárboles y de los lazos. Su experiencia con listas ligadas puede ilustrarle que es un poco más fácil suprimir nodos, cuando las ligas de los subárboles son bidireccionales. Para instrumentar subárboles con ligas bidireccionales se requiere únicamente de un campo de apuntador adicional por nodo: al padre del nodo. En Pascal:

```
type apuntador-a-nodo = ↑ tipo-nodo;
     tipo-nodo = record
                   izquierdo:apuntador-a-nodo;
                   información:integer;
                   derecho:apuntador-a-nodo;
                   lazo-izquierdo:apuntador-a-nodo;
                   lazo-derecho:apuntador-a-nodo;
                   padre:apuntador-a-nodo
                 end;
```

La eliminación de un nodo (llamado *X*) de un árbol binario es más compleja, si este nodo no es una hoja, dado que se debe tomar una decisión con respecto a la disposición de los hijos de los nodos removidos. Podría ser el caso de que estos nodos hijos también tengan que ser removidos del árbol. La dificultad para instrumentar esta política reside en el cambio de los enhilados. Otra alternativa consiste en mover la raíz del subárbol izquierdo de *X*, o del subárbol derecho a la posición de *X*. Esto es fácil (excepto para los enhilados) cuando sólo uno de estos nodos es no-nulo. Si ambos son no-nulos se debe escoger uno de ellos. Si el que fue seleccionado tiene ambos subárboles derecho e izquierdo no-nulos, se producirán dificultades posteriores. (¡Inténtelo!) Una alternativa razonable en algunas situaciones es la de evitar estos problemas, al no permitir la remoción de un nodo, a menos que sea una hoja.

SUPRESION DE NODOS DE UN ARBOL DE BUSQUEDA BINARIO

La operación de supresión de nodos de un árbol de búsqueda binario puede ser compleja. Si el nodo a borrar es una hoja, entonces el proceso es simple pues sólo se arranca la

hoja. En cambio, si el nodo a borrar no es una hoja, entonces el proceso deberá hacer algo para conservar los subárboles del nodo.

Considere primero la supresión del nodo PGMR del árbol de búsqueda binario de la figura 8-22. Remover PGMR no implica borrar también a PILOT, más bien PILOT se puede cambiar al lugar de PGMR convirtiéndose en subárbol izquierdo de SECY. De igual forma si SECY se borrara, entonces PGMR podría moverse al lugar vacante; el árbol con raíz en PGMR se convertiría en el subárbol izquierdo de SWEEPER. En forma similar, al suprimir un nodo con un subárbol nulo, esto implica mover la raíz del subárbol con nodos no-nulos a la posición vacante.

Considere ahora la supresión del nodo con llave CLERK, la cual tiene dos subárboles, ¿Cuál de las raíces deberá ocupar el lugar vacante? Cualquiera de ellas, ya que no se violan las reglas del árbol de búsqueda binario.

La supresión del nodo etiquetado como MNGR es más difícil; Si FIREMAN toma la posición de MNGR, ¿cómo serán manipulados los subárboles con raíces en CLERK y HELPER? Usted puede desarrollar un algoritmo que manipule la supresión de un nodo, de cualquier posición, en un árbol de búsqueda binario, pero primero, asegúrese de haber comprendido bien la complejidad del problema.

Otro método de supresión es evitar (al menos temporalmente) la complejidad del problema simplemente marcando como inactivos los nodos. Una posterior búsqueda o inserción necesita utilizar las llaves de los nodos marcados para encontrar su trayectoria a través del árbol. Si la llave buscada por el procedimiento de búsqueda o inserción corresponde a la llave de un nodo marcado, entonces el procedimiento debe tomar acciones diferentes a las que hará normalmente. Modifique usted las rutinas de búsqueda e inserción para acomodar nodos marcados.

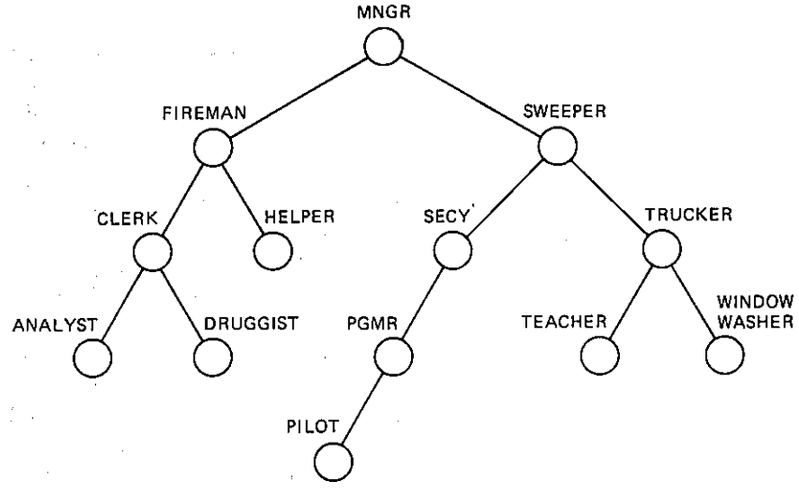

Figura 8-22 Un árbol de búsqueda binario.

Para mayor información sobre el borrado de nodos véase el artículo de J. Cohen, "Garbage collection of linked data structures", en *AMC Computing Surveys*, 13(3): 341-367, septiembre 1981.

Las colecciones de datos con llaves para los cuales se requiere tanto de un acceso (directo) rápido a registros particulares, como un acceso secuencial a través de la colección de registros, se estructuran normalmente como árboles de búsqueda binarios.

BALANCEO DE ARBOLES DE BUSQUEDA BINARIOS

Hemos visto que el desempeño de un árbol de búsqueda binario se determina por la estructura del árbol y las probabilidades de acceso a los nodos. La estructura del árbol está afectada por la secuencia de modificaciones al árbol; tanto las inserciones como las supresiones cambian el esquema del árbol.

Realmente, lograr la estructura *óptima* de un árbol de búsqueda binario puede requerir de más trabajo de lo que vale la pena; de hecho, esto puede ser imposible, por carecer del suficiente conocimiento sobre las probabilidades de acceso. Sin embargo, las investigaciones han mostrado que es competitivo el comportamiento de los árboles de búsqueda binarios que están construidos con algoritmos heurísticos razonables con respecto al de los árboles óptimos, proporcionando mucho más bajos costos de diseño. Los dos algoritmos heurísticos más obvios son los que hemos introducido:

1. Poner los nodos con las llaves más frecuentemente accesadas, cerca de la raíz del árbol.
2. *Balancear* el árbol: esto es, para cada nodo, los subárboles derecho e izquierdo deberán tener un número de nodos, tan similar como sea posible, minimizando por lo tanto la longitud máxima de ruta.

Si las probabilidades de acceso fuesen conocidas y no hubiese inserciones al árbol, el árbol de búsqueda binario que produce un comportamiento óptimo podría construirse con algo de trabajo y daría mejores resultados que un árbol construido heurísticamente. Sin embargo, la heurística es realmente buena y requiere de muy poco esfuerzo de aplicación. Se ha demostrado que, en general, los árboles de búsqueda binarios construidos con el segundo punto de vista heurístico (árboles balanceados) dan una longitud de búsqueda casi óptima; los árboles balanceados se desempeñan mejor en general que los árboles construidos con el primer punto de vista heurístico. Este resultado es bastante reconfortante dado que las probabilidades de acceso requeridas por el primer algoritmo frecuentemente no están disponibles.

Para más detalles de análisis de varios algoritmos heurísticos para construir árboles de búsqueda binarios, puede consultar las referencias sugeridas al final del capítulo.

Hacer que un árbol de búsqueda binario quede balanceado, cuando existen inserciones y supresiones de nodos, es una tarea difícil. La dificultad se debe a que hay que conservar las relaciones apropiadas entre las llaves de los nodos y las llaves en sus subárboles. Por ejemplo, considere el árbol de búsqueda binario de la figura 8-23.

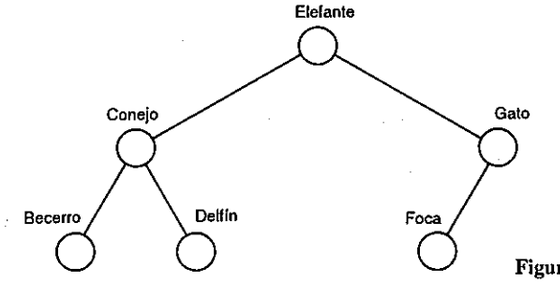

Figura 8-23 Un árbol de búsqueda binario.

Insertar el nodo "ZORRO" puede ser fácil; éste podría ser el subárbol derecho del nodo "GATO", y el árbol podría conservarse balanceado. Sin embargo, intente insertar la palabra "ABEJA", la posición de "ABEJA" será la de subárbol izquierdo de "BECERRO", lo cual hace que el árbol se alargue y salga de balance. El árbol cuando está balanceado, puede ser como se muestra en la figura 8-24, donde *todos* los nodos del árbol original debieron ser movidos. Un problema similar puede ocurrir cuando hay supresión de nodos.

Para reducir el total de trabajo requerido para mantener un árbol de búsqueda binario mientras proporcione un buen comportamiento de búsqueda, podemos relajar las restricciones y permitir que los árboles no queden tan balanceados. Sin embargo, la desviación permitida debe ser tan pequeña que la longitud promedio esperada de búsqueda sea apenas un poco mayor que la de los árboles completamente balanceados. En las siguientes dos secciones introducimos dos clases de árboles de búsqueda binarios que permiten una modificación relativamente fácil.

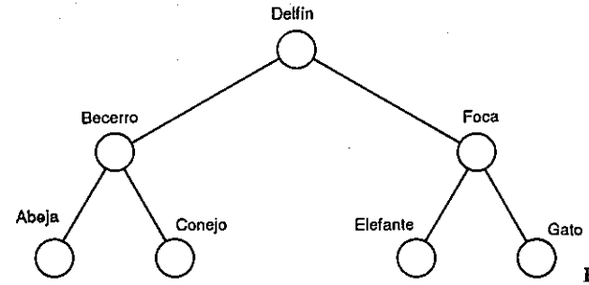

Figura 8-24 Arbol binario de búsqueda balanceado.

ARBOLES BALANCEADOS POR SU ALTURA (AVL)

Un tipo de árboles "casi completamente" balanceados es conocido como *árbol balanceado por su altura*, o *árbol AVL*, debido a sus descubridores rusos: G. M. Adelson-Velskii y E. M. Landis (1962). Un árbol está balanceado por su altura si sus nodos quedan arreglados de tal forma que, para cada nodo R_i, se cumple con la siguiente propiedad:

> La altura del subárbol izquierdo de R_i y la altura del subárbol derecho de R_i difieren en un máximo de 1.

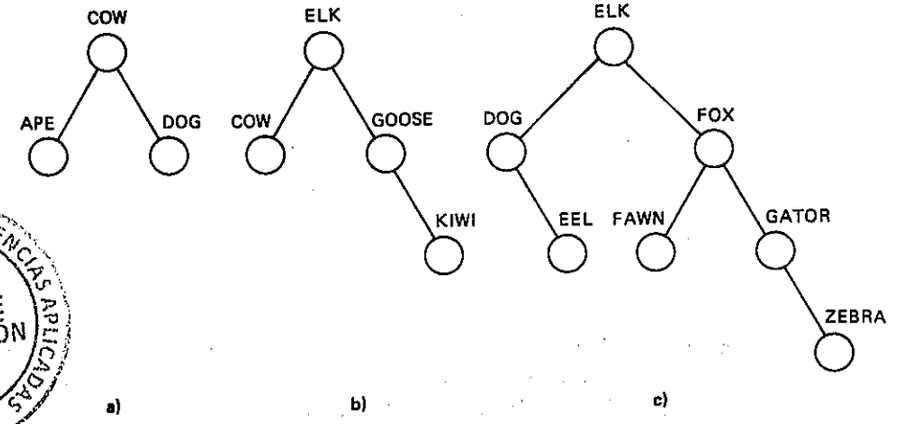

Figura 8-25 Arbol balanceado por su altura.

Recordemos que la altura de un árbol es de uno más el número del nivel más alto en el cual tiene nodos.

Los árboles de búsqueda binarios mostrados en la figura 8-25, están todos balanceados por su altura. Note que aunque está balanceado por su altura, el árbol c) no está completamente balanceado. Los árboles de búsqueda binarios de la figura 8-26 *no* están balanceados por su altura; la restricción de altura se viola en el nodo APE del árbol a), en el nodo COW del árbol b) y en el nodo FOX del árbol c).

Los árboles AVL proporcionan una buena y eficiente búsqueda directa. Aunque tiendan a verse algo más esparcidos que los que están completamente balanceados, las búsquedas que requieren son moderadamente mayores. La peor longitud de búsqueda posible para un árbol completamente balanceado, con n-nodos es $\log_2 (n + 1)$, mientras que la peor longitud de búsqueda para un árbol AVL de n-nodos es $1.44 \log_2 (n + 1)$. El

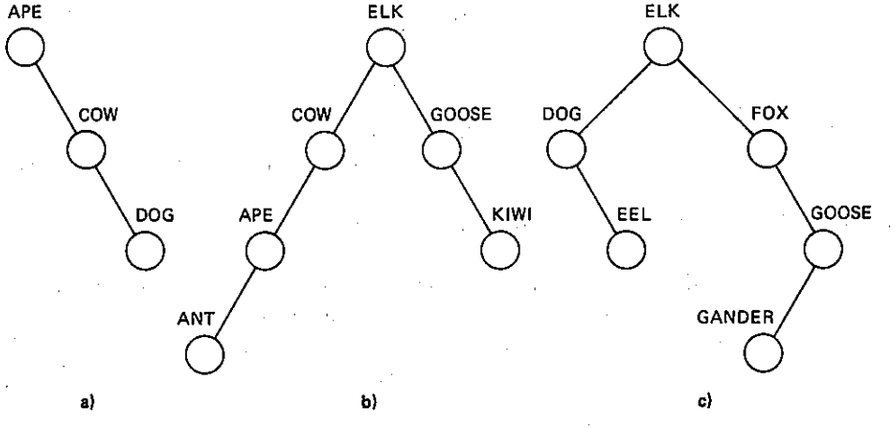

Figura 8-26 Arboles binarios que no están balanceados por su altura.

promedio de longitud de búsqueda esperado para un árbol completamente balanceado de n-nodos es $\log_2 (n + 1) - 2$, y el promedio de longitud para un árbol AVL de n nodos es de $\log (n + 1) + \alpha$, donde α es una constante, que depende de las características de la distribución. (Consulte Knuth [1973, pp.453-454] para más detalles de este análisis). Por tanto los árboles AVL son casi tan buenos como los árboles completamente balanceados. La restricción de la altura impide que un árbol AVL se vuelva demasiado distante de un árbol completamente balanceado.

Si una inserción o supresión provoca que un árbol AVL pueda desbalancearse, entonces el árbol deberá ser reestructurado para regresar a un estado balanceado. Este proceso de reestructuración, es conocido comúnmente como *rotación* y no se analiza aquí. El lector interesado podría comenzar la investigación sobre las técnicas de balanceo de árboles, leyendo cuidadosamente el libro de Knuth (1973, pp. 453-463).

ARBOLES BALANCEADOS POR UN LIMITE (BB)

Otra clase de árbol balanceado "casi completamente" es conocido como *árbol balanceado por un límite*. Estos árboles son conocidos también como *árboles BB* o *árboles balanceados por peso*. La idea básica de un árbol balanceado por un límite es la de imponer una restricción con respecto a qué tan lejos puede desviarse un árbol de estar completamente balanceado. Este límite restringe la diferencia entre el número de nodos en el subárbol izquierdo y en el subárbol derecho. Aunque basados en una restricción de tamaño en lugar de la altura de los subárboles, los árboles balanceados por un límite son similares en sus propiedades a los árboles balanceados por su altura. Ambos intentan negociar un compromiso entre longitudes de búsqueda cortas y los requerimientos de balanceo.

Un árbol balanceado por un límite lleva un parámetro β, el cual controla este compromiso. En un árbol balanceado por un límite de balance β, donde $0 < \beta \leq \frac{1}{2}$, para cada nodo del árbol, la fracción de nodos en el subárbol izquierdo estará entre β y $1 - \beta$. Más formalmente, si el TAMAÑO (N) es el número de nodos en el árbol con raíz en el nodo N, entonces la restricción es:

$$\beta \leq \frac{\text{TAMAÑO(IZQ(N))} + 1}{\text{TAMAÑO(N)} + 1} \leq 1 - \beta$$

Si $\beta = \frac{1}{2}$, entonces el subárbol izquierdo y el derecho tendrán el mismo número de nodos, es decir, el árbol estará completamente balanceado. Si $\beta = \frac{1}{3}$, entonces uno de los subárboles podrá tener hasta tres veces tantos nodos como tenga el otro. A medida que el valor de β se acerca a $\frac{1}{2}$, la restricción de limitación se hace más cerrada. Un diseñador de árboles selecciona el valor apropiado de β para cada aplicación.

El promedio esperado de longitud de búsqueda para un árbol balanceado por un límite es $0(\log_2 n)$, donde n es el número de nodos del árbol. Si la inserción o supresión causa que un árbol de esta clase se vuelva desbalanceado, con relación a su límite β,

entonces los mismos tipos de rotación, que se aplicaron a la clase de árboles balanceados por su altura, se aplican para hacer que el árbol vuelva a un estado balanceado.

El trabajo requerido para conservar un árbol AVL o BB en conformancia con su razón de altura o de tamaño es considerablemente menor que el trabajo requerido para conservar un árbol *completamente* balanceado. Si se necesita una transformación para conservar un árbol completamente balanceado, después de una inserción o supresión, esta transformación requiere mover $0(n)$ nodos, donde n es el número de nodos en el árbol. A diferencia de que si se necesita una transformación para mantener la conformancia en la razón de altura o de tamaño de un árbol AVL o BB , después de agregar o retirar nodos entonces se necesitarán $0(\log_2 n)$ movimientos de nodos. La reducción de trabajo se logra localizando el movimiento de los nodos a lo largo de una sola trayectoria, desde la raíz hasta una hoja en el árbol AVL o BB; mientras que el movimiento en un árbol completamente balanceado afecta al árbol entero.

RESUMEN

Este capítulo introdujo la clase especial de estructura de *grafos* llamada *árbol general*, la cuál es un grafo simple acíclico y conexo. También se introdujo la terminología pertinente para árboles; después, un caso especial de árbol general, enfocó nuestra atención: el *árbol binario*. En un árbol binario cada nodo tiene a lo más dos subárboles, los cuales se pueden identificar como subárbol derecho e izquierdo. Debido a que es más fácil manejar árboles binarios que árboles generales, es útil representar a un árbol general (o un bosque de árboles generales) mediante una forma de árbol binario. También presentamos un algoritmo para esta conversión.

Una clase importante de árboles binarios son aquellos que son *casi completos*. La estructuración de una colección de nodos como un árbol binario casi completo minimiza la longitud de ruta máxima del árbol, minimizando por lo tanto la ruta de mayor búsqueda, a través del árbol para un nodo en particular.

Algunas veces, en lugar de buscar un nodo en particular, es necesario recorrer un árbol binario, esto es, visitar todos los nodos del árbol. Se presentaron varios algoritmos para recorrer árboles binarios: pre-orden, en-orden y post-orden. Estos algoritmos difieren en el orden en el cual la raíz, el subárbol izquierdo y el subárbol derecho son recorridos. Cada orden es adecuado para alguna aplicación en especial. Para facilitar el recorrido, un árbol se debe enhilar de acuerdo con alguno de los algoritmos de recorrido. Entonces cada nodo apunta a su sucesor en la ruta de recorrido. Los enhilados son comúnmente bidireccionales, lo cual hace que la inserción y supresión de nodos sea más fácil. El recorrido de un árbol general se logra convirtiendo primero el árbol a su forma binaria.

El *árbol de búsqueda binario* se aplica para organizar grandes conjuntos de registros de datos con llaves de identificación, donde se requieren tanto el acceso directo a un registro particular como del acceso secuencial al conjunto completo de registros. Se presentaron algoritmos para ejecutar las operaciones básicas en árboles de búsqueda binarios: búsqueda directa, búsqueda secuencial, inserción de nodos y supresión de nodos.

La comprensión de la naturaleza de los árboles de búsqueda binarios hace obvio que algunas estructuras obtengan una mejor eficiencia de búsqueda que otras. Se analizaron dos criterios heurísticos para construir árboles de búsqueda binarios; uno de ellos —cuidar el balance del árbol— produce longitudes de búsqueda casi óptimas sin la necesidad de trabajar en la construcción de un árbol óptimo.

Se discutió la dificultad de lograr que un árbol de búsqueda binario se mantenga completamente balanceado cuando hay inserciones o supresiones de nodos y se introdujeron dos clases de árboles de búsqueda binarios "casi balanceados" que producen una eficiencia de búsqueda buena, aunque requiere de menos trabajo de mantenimiento que para los árboles completamente balanceados. Los *árboles AVL* (también conocidos como *árboles balanceados por su altura*) restringen la diferencia de altura entre el subárbol izquierdo y derecho de cada nodo a no pasar de 1. Los *árboles balanceados por un límite* (también conocidos como *árboles BB* o *árboles balanceados por peso*) restringen la diferencia entre el tamaño (es decir, el número de nodos) del subárbol izquierdo y derecho de cada nodo. El diseñador de árboles, aquí, especifica el parámetro utilizado para establecer la restricción.

La inserción y supresión de nodos en un árbol de búsqueda binario balanceado puede desbalancear el árbol, pues la restricción de altura o tamaño puede ser violada. Para regresar el árbol a una condición de balance, una rotación puede ser aplicada al árbol.

La longitud de búsqueda promedio esperada para un árbol de búsqueda binario balanceado es de $0(\log_2 n)$.

TERMINOLOGIA

altura del árbol
árbol
árbol AVL
árbol balanceado por peso
árbol balanceado por su altura
árbol binario
árbol binario balanceado por un límite
árbol binario casi completo
árbol binario completo
árbol de búsqueda binario
árbol enhilado
árbol enraizado
árboles equivalentes
árboles similares
bosque
enhilado derecho
enhilado izquierdo
longitud de trayectoria
nivel
nodo
nodo hoja
peso del árbol
recorrido de árboles
recorrido en-orden
recorrido en post-orden
recorrido en pre-orden
rotación

REFERENCIAS SUGERIDAS

ALLEN, B. and I. MUNRO. "Self-organizing binary search trees," *Jour. ACM*, 25(4): 526–535, Oct. 1978.

BAER, J.-L. and B. SCHWAB. "A comparison of tree-balancing algorithms," *Comm. ACM*, 20(5): 322–330, May 1977.

BALLARD, D. H. "Strip trees: a hierarchical representation for curves," *Comm. ACM*, 24(5): 310–321, May 1981.

BECKLEY, D. A., M. W. EVANS, and V. K. RAMAN. "Multikey retrieval from K-d trees and quad-trees," *Proc. ACM-SIGMOD 1985 Internat. Conf. on Management of Data, May 28–31, 1985, Austin TX*, pp. 291–301.

BRUNO, J., and E. G. COFFMAN. "Nearly optimal binary search trees," *Proc. IFIP Congress 1971*, North-Holland, pp. 99–103.

BURGE, W. H. "An analysis of binary search trees formed from sequences of nondistinct keys," *Jour. ACM*, 23(3): 451–454, July 1976.

CHANG, H. and S. S. IYENGAR. "Efficient algorithms to globally balance a binary search tree," *Comm. ACM*, 27(7): 695–702, July 1984.

DEJONGE, W., A. S. TANENBAUM, and R. P. VAN DE RIET. "Two access methods using compact binary trees," *IEEE Trans. on Software Engineering*, SE-13(7): 799–810, July 1987.

DRISCOLL, J. R. and Y. E. LIEN. "A selective traversal algorithm for binary search trees," *Comm. ACM*, 21(6): 445–447, June 1978.

DYER, C. R., A. ROZENFELD, and H. SAMET. "Region representation: boundary codes from quadtrees," *Comm. ACM*, 23(3): 171–179, March 1980.

EPPINGER, J. L. "An empirical study of insertion and deletion in binary search trees," *Comm. ACM*, 26(9): 663–669, Sept. 1983.

FOSTER, C. C. "Information storage and retrieval using AVL trees," *Proc. ACM 20th Natl. Conf.*, pp. 192–205, 1965.

GARGANTINI, I. "An effective way to represent quadtrees," *Comm. ACM*, 25(12); 905–910, Dec. 1982.

GONNET, G. H. "Balancing binary trees by internal path reduction," *Comm. ACM*, 26(12): 1074–1081, Dec. 1983.

HIBBARD, T. "Some combinatorial properties of certain trees with applications to searching and sorting," *Jour. ACM*, 9(1): 13–28, January 1962.

HIRSCHBERG, D. S. "An insertion technique for one-sided height-balanced trees," *Comm. ACM*, 19(8): 471–473, Aug. 1976.

HUANG, S-H. S. "Height-balanced trees of order (b,g,d)," *ACM Trans. on Database Syst.*, 10(2) 261–284, June 1985.

KARLTON, P. L., S. H. FULLER, R. E. SCROGGS, and E. B. KAEHLER. "Performance of height-balanced trees," *Comm. ACM*, 19(1): 23–28, Jan. 1976.

KESSELS, J. L. W. "On-the-fly optimization of data structures," *Comm. ACM*, 26(11): 895–901, Nov. 1983.

KNOTT, G. D. "A numbering system for binary trees," *Comm. ACM*, 20(2): 113–115, Feb. 1977.

KNUTH, D. E. *The Art of Computer Programming: Vol. 3, Sorting and Searching*. Reading, MA. Addison-Wesley Publishing Co . 1973. pp. 422–471.

KOSARAJU, S. R. "Insertions and deletions in one-sided height-balanced trees," *Comm. ACM*, 21(3): 226–227, March 1978.

KUNG, H. T. and P. L. LEHMAN. "Concurrent manipulation of binary search trees," *ACM Trans. on Database Syst.*, 5(3): 354–382, Sept. 1980.

LEE, K. P. "A linear algorithm for copying binary trees using bounded workspace," *Comm. ACM*, 23(3): 159–162, March 1980.

LUCCIO, F. and L. PAGLI. "Comment on generalized AVL trees," *Comm. ACM*, 23(7): 394–395, July 1980.

LUCCIO, F. and L. PAGLI. "Power trees," *Comm. ACM*, 21(11): 941–947, Nov. 1978.

LUCE, R. D. and H. RAIFFA. *Games and Decisions*. New York: John Wiley & Sons, Inc., 1958.

MANBER, U. "Concurrent maintenance of binary search trees," *IEEE Trans. on Software Engineering*, SE-10(6): 777–784, Nov. 1984.

MOITRA, A. and S. S. IYENGAR. "Derivation of a parallel algorithm for balancing binary trees," *IEEE Trans. on Software Engineering*, SE-12(3): 442–449, March 1986.

NEWBORN, M. *Computer Chess*. New York: Academic Press, 1975.

NIEVERGELT, J. "Binary search trees and file organization," *ACM Computing Surveys*, 6(3): 195–207, Sept. 1974.

NIEVERGELT, J. and C. K. WONG. "Upper bounds for the total path length of binary trees," *Jour. ACM*, 20(1): 1–6, Jan. 1973.

OTTMANN, T., H. W. SIX, and D. WOOD. "Right brother trees," *Comm. ACM*, 21(9): 769–776, Sept. 1978.

PFALTZ, J. L. "Representing graphs by Knuth trees," *Jour. ACM*, 22(3): 361–366, July 1975.

PROSKUROWSKI, A. "On the generation of binary trees," *Jour. ACM*, 27(1): 1–2, Jan. 1980.

RAIHA, K-J. and S. H. ZWEBEN. "An optimal insertion algorithm for one-sided height-balanced binary search trees," *Comm. ACM*, 22(9): 508–512, Sept. 1979.

ROTEM, D. and Y. L. VAROL. "Generation of binary trees from ballot sequences," *Jour. ACM*, 25(3): 369–404, July 1978.

SAMET, H. "Data structures for quadtree approximation and compression," *Comm. ACM*, 28(9): 973–993, Sept. 1985.

SAMET, H. "The quadtree and related hierarchical data structures," *ACM Computing Surveys*, 16(2): 187–260, June 1984.

SAMET, H. "A quadtree medial axis transform," *Comm. ACM*, 26(9): 680–693, Sept. 1983.

SAMET, H. "Deletion in two-dimensional quad trees," *Comm. ACM*, 23(12): 703–710, Dec. 1980.

SAMET, H. "Region representation: quadtrees from boundary codes," *Comm. ACM*, 23(3): 163–170, March 1980.

SARNAK, N. and R. E. TARJAN. "Planar point location using persistent search trees," *Comm. ACM*, 29(7): 669–679, July 1986.

SCOTT, D. S. and S. S. IYENGAR. "TIF—a translation invariant data structure for storing images," *Comm. ACM*, 29(5): 418–429, May 1986.

SEVERANCE, D. G. "Identifier search mechanisms: a survey and generalized model," *ACM Computing Surveys*, 6(3): 175–194, Sept. 1974.

SOLOMON, M. N. and R. A. FINKEL. "A note on enumerating binary trees," *Jour. ACM*, 27(1): 3–5, Jan. 1980.

STANDISH, T. A. "Trees," Chapter 3 in *Data Structure Techniques*. Reading, MA: Addison-Wesley Publishing Co., 1980, pp. 44–129.

STASKO, J. T. and J. S. VITTER. "Pairing heaps: experiments and analysis," *Comm. ACM*, 30(3): 234–249, March 1987.

STOUT, Q. F. and B. L. WARREN. "Tree rebalancing in optimal time and space." *Comm. ACM*, 29(9): 902–908, Sept. 1986.

TOMPA, F. W., J. GECSEI, and G. V. BOCHMANN. "Data structuring facilities for interactive videotex systems," *Computer*, 14(8): pp. 72–81, Aug. 1981.

YAU, M-M. and S. N. SRIHARI. "A hierarchical data structure for multidimensional digital images," *Comm. ACM*, 26(7): 504–515, July 1983.

ZWEBEN, S. H. and M. A. MCDONALD. "An optimal method for deletions in one-sided height-balanced trees," *Comm. ACM*, 21(6): 441–445, June 1978.

EJERCICIOS DE REPASO

1. Dibuje dos árboles binarios que sean similares.
2. Dibuje dos árboles binarios que sean equivalentes.
3. ¿Puede haber dos árboles binarios equivalentes pero no similares? Dé un ejemplo si su respuesta es afirmativa, en caso contrario explique ¿por qué no?
4. ¿Un árbol puede ser cíclico?, en caso afirmativo dé un ejemplo, en caso contrario explique ¿por qué no?
5. ¿Cuál es el número de nodos en el i-ésimo nivel de un árbol binario completo, de altura K, donde $I < K$?
6. ¿Cuál es el número de nodos en el i-ésimo nivel de un árbol binario casi completo, de altura K, donde $I < K$?
7. Dibuje un árbol binario completo con 10 nodos. Dibuje un árbol binario casi completo con diez nodos. ¿Cuál es la mínima longitud de trayectoria máxima en un árbol binario con 10 nodos? ¿Cuál es la longitud máxima de la trayectoria más larga en un árbol binario de 10 nodos.

8. ¿Por qué es más fácil representar árboles binarios en programas que representar árboles generales?
9. Considere el siguiente árbol binario:

   consigne los nodos del árbol en a) pre-orden, b) post-orden, c) en-orden. Después dibuje el árbol como árbol enhilado representando los enhilados con líneas punteadas y enlácelo en d) pre-orden, e) post-orden, f) en-orden.

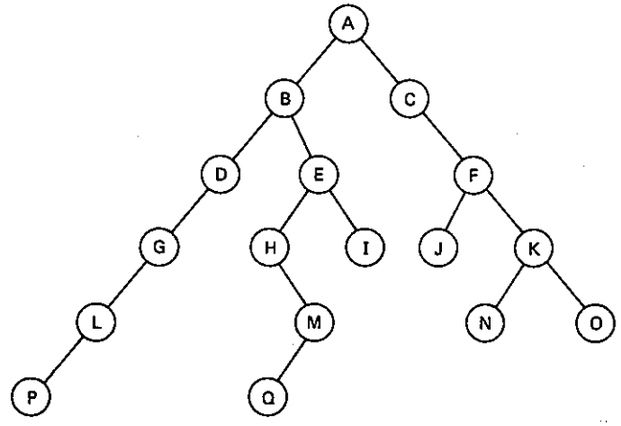

Figura P8–9

10. ¿La siguiente proposición es cierta o falsa? "Las hojas de un árbol binario aparecen en la misma posición relativa en pre-orden, post-orden y en-orden".
11. Dibuje el árbol binario que corresponda al bosque de la figura P8-11.

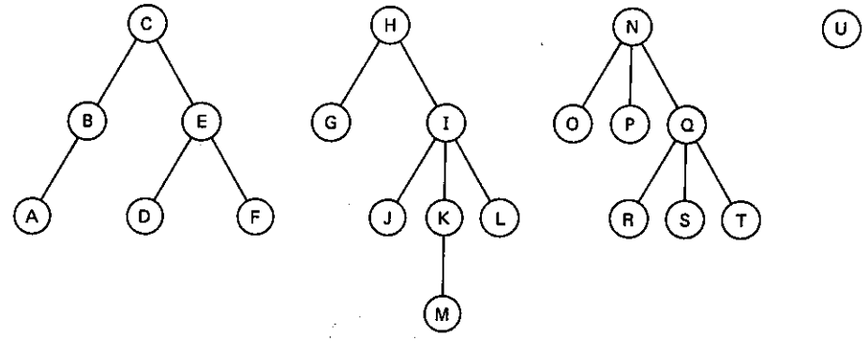

Figura P8–11

12. Transforme el siguiente árbol general en árbol binario.

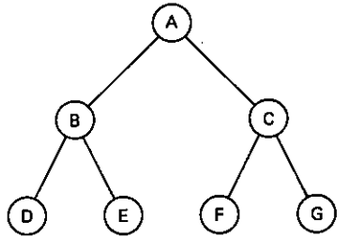

Figura P8–12

13. El siguiente árbol binario es el resultado de la transformación de un bosque a una forma binaria. ¿Cuál era el bosque?

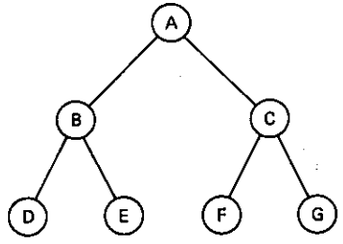

Figura P8–13

14. Represente a la expresión $((A + B)*C)/(D - E)$ como árbol binario, y recorra al árbol en pre-orden, en post-orden, en en-orden. ¿Por qué cree que el recorrido en post-orden fue elegido para la evaluación de la expresión, es decir, qué puede usted apreciar en el recorrido en post-orden que hace que el cálculo sea más fácil?

15. Muestre la representación de árbol binario para la cadena ICDFEHABC producida a) usando pre-orden, b) usando post-orden, y c) usando en-orden. Utilice la estructura de árbol, mostrada en la figura P8-15.

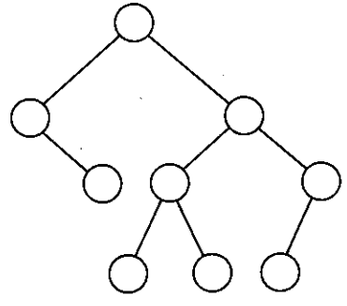

Figura P8–15

16. Dada una colección de llaves $K_1, K_2, \ldots, K_n$. ¿En que secuencia deben ser insertadas estas llaves para formar un árbol de búsqueda binario balanceado?

17. ¿Todos los compiladores de que usted dispone utilizan la misma secuencia de comparación? si no, ¿cuál es la diferencia?

18. Muestre como se enhilarían los nodos del siguiente árbol, si se enlazan en en-orden.

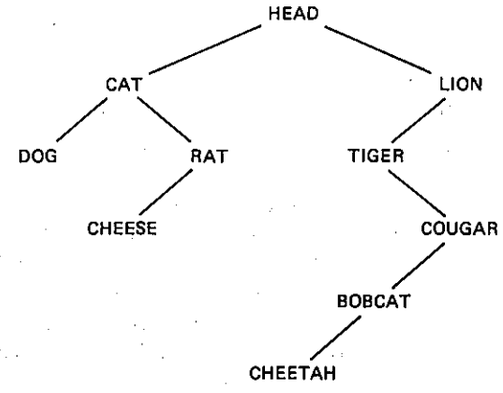

Figure P8–18

19. ¿Qué tipo de enhilado debe tener un árbol binario para facilitar la visita de los nodos en la secuencia de sus nombres?

20. Construya dos árboles de búsqueda binarios adicionales para el conjunto de llaves estructuradas en la figura 8-17, utilizando las siguientes probabilidades de acceso.

ABEJA	.2
BECERRO	.4
CONEJO	.3
DELFIN	.1

y calcule la longitud de búsqueda esperada para sus árboles.

21. Dé un ejemplo donde el recorrido en post-orden sea la forma más adecuada de visitar el árbol. Haga lo mismo para los recorridos en en-orden y en pre-orden.

22. Describa la estructura de nodos para un árbol binario que esté enhilado en en-orden y otro en post-orden.

23. ¿Un árbol binario real puede ser enlazado por más de un tipo de recorrido? ¿Por qué si o por qué no?

24. Considere un árbol binario T con n nodos, $n \geq 0$. Si T no está vacío el nivel máximo que cualquier nodo de T puede alcanzar es n. Si T es no vacío, entonces al menos un nodo alcanza un nivel máximo para ese árbol. ¿Cuál es el valor *mínimo* que este nivel máximo puede tener? Por ejemplo, para $n = 1$, la respuesta es 1, mientras que para $n = 2$, la respuesta es 2, y para $n = 3$, también es 2. Exprese su respuesta como una fórmula, en términos de n, $\log_2$ (logaritmo base 2), y la función de redondeo $\lceil\ \rceil$. Pruebe que su resultado es verdadero.

25. Se identificaron tres órdenes básicas para recorrer un árbol binario, otra alternativa podría ser:

a) visitar la raíz,
b) recorrer el subárbol derecho,
c) recorrer el subárbol izquierdo,

utilizando la misma regla recursivamente en todos los subárboles no-vacíos. ¿Este nuevo orden guarda alguna relación simple con los otros tres ya descritos? ¿Que clase de enhilamientos deberían definirse para hacer que el recorrido en este orden sea más fácil?

26. Describa qué sucede cuando una transformación de árbol general a binario se aplica a un árbol binario.

27. Escriba un procedimiento para reconstruir un árbol general a partir del árbol binario que resulte de aplicar el algoritmo de transformación, presentado en este capítulo.

28. Escriba un programa que tome árboles generales como entrada y cuya salida sean los correspondientes árboles binarios.

29. Escriba procedimientos iterativos para recorrer en en-orden un árbol binario que está enhilado en en-orden; otro para ejecutar un recorrido en pre-orden en un árbol binario enhilado en pre-orden; otro más para recorrer un árbol binario en post-orden y que está enhilado en post-orden. Escriba programas para implantar estos procedimientos.

30. Escriba procedimientos recursivos para implantar los recorridos en en-orden, pre-orden y post-orden para árboles binarios no enhilados. Escriba el correspondiente programa.

31. Escriba los correspondientes procedimientos no recursivos (e.i. iterativos) para recorrer árboles binarios no enhilados en post-orden, en-orden y pre-orden. Escriba el respectivo programa.

32. ¿Que ventajas y desventajas deberán tomarse en cuenta para decidir si el recorrido del árbol debiera implantarse mediante rutinas recursivas o iterativas?

33. Escriba procedimientos para insertar nuevos nodos en un árbol binario que está enhilado en en-orden, en pre-orden, en post-orden.

34. Escriba los procedimientos para borrar nodos de un árbol binario y de un árbol binario enhilado.

35. ¿Cuál es la diferencia entre un árbol balanceado y un árbol balanceado por su altura?

36. Escriba procedimientos no recursivos (i.e. iterativos) para buscar e insertar nodos en un árbol de búsqueda binario.

37. Modifique los procedimientos para buscar e insertar nodos en árboles de búsqueda binarios de tal forma que reconozcan nodos que han sido marcados como nodos borrados, pero aún no han sido removidos del árbol.

38. Escriba un procedimiento para borrar un nodo dado de un árbol de búsqueda binario. Asegúrese de considerar las situaciones en que un nodo tiene 0, 1 o 2 subárboles no-nulos.

39. Lea acerca de las técnicas para conservar un árbol en balance. Escriba un programa para implantar una de las técnicas.

capítulo nueve

búsqueda y ordenamiento

En este capítulo tomaremos un breve descanso con respecto al aprendizaje de nuevas estructuras de datos y en su lugar estudiaremos las técnicas para encontrar valores de datos particulares en las estructuras y para ordenar dichos valores. La búsqueda y el ordenamiento son procesos estrechamente relacionados, como ya veremos.

Considere una colección de registros, cada uno de los cuales tiene una *llave* que puede usarse para identificarlo. Una llave puede estar compuesta por uno o más campos. El valor de la llave puede ser el único identificador del registro, aunque también se pueden permitir valores duplicados. Si se permiten valores de llave duplicados para los registros, es aconsejable diferenciarlos de acuerdo al orden en que vayan apareciendo. El concepto de llave se utilizará a través de los capítulos subsecuentes de este libro. La *búsqueda* es el proceso de localizar un registro con valor de llave particular. El *ordenamiento* es el proceso de arreglar registros de tal forma que queden ordenados por su valor llave.

BUSQUEDA SECUENCIAL

Un algoritmo de búsqueda es una técnica para encontrar un registro que tenga algún valor de llave en especial. Llamaremos al valor de la llave, digamos k, el *argumento* de la búsqueda. La búsqueda termina exitosamente cuando se localiza el registro que contenga la llave k, o termina sin éxito, cuando se determina que no aparece ningún registro con la llave k.