Capítulo 6

ÁRBOLES

6.1 INTRODUCCIÓN

Hasta el momento sólo se han estudiado estructuras de datos lineales, tanto estáticas como dinámicas: a cada elemento siempre le sucede o le precede como máximo otro elemento. Al estudiar la estructura de datos árboles se introduce el concepto de ramificación entre componentes o nodos. Es decir, a un elemento le pueden preceder o suceder varios elementos.

Los **árboles** son las estructuras de datos **no lineales** y **dinámicas** de datos más importantes del área de computación. Dinámicas, puesto que las mismas pueden cambiar tanto de forma como de tamaño durante la ejecución del programa. No lineales, puesto que cada elemento del árbol puede tener más de un sucesor. Los **árboles balanceados** o **AVL** son la estructura de datos más eficiente para trabajar con la memoria principal —interna— del procesador, mientras que los **árboles *B*** y, especialmente la versión ***B*⁺**, representan la estructura de datos más eficiente para trabajar en memoria secundaria o externa.

En la tabla 6.1 se presentan las principales estructuras de datos clasificadas de acuerdo con su capacidad para cambiar en forma y tamaño, durante la ejecución del programa.

Es de observar que las pilas y colas no fueron consideradas en esta clasificación, puesto que dependen de la estructura que se utilice para implementarlas. Si se usan arreglos, se tratarán como estructuras estáticas. Si se implementan con listas, serán estructuras dinámicas. En ambos casos son lineales.

En la tabla 6.2 se presentan las principales estructuras de datos clasificadas según la distribución de sus elementos.

TABLA 6.1
Estructuras de datos estáticas y dinámicas

Estructuras estáticas	Estructuras dinámicas
Arreglos	Listas
Registros	Árboles
	Gráficas

TABLA 6.2
Estructuras de datos lineales y no lineales

Estructuras lineales	Estructuras no lineales
Arreglos	Árboles
Registros	Gráficas
Pilas	
Colas	
Listas	

6.2 ÁRBOLES EN GENERAL

Un **árbol** se puede definir como una estructura jerárquica aplicada sobre una colección de elementos u objetos llamados nodos, uno de los cuales es conocido como raíz. Además se crea una relación o parentesco entre los nodos dando lugar a términos como padre, hijo, hermano, antecesor, sucesor, ancestro, etcétera.

Formalmente se define un árbol de tipo *T* como una estructura homogénea resultado de la concatenación de un elemento de tipo *T* con un número finito de árboles disjuntos, llamados subárboles. Una forma particular de árbol es el árbol vacío. Los árboles son estructuras recursivas, ya que cada subárbol es a su vez un árbol.

Los árboles se pueden aplicar para la solución de una gran cantidad de problemas. Por ejemplo, se pueden utilizar para representar fórmulas matemáticas, para registrar la historia de un campeonato de tenis, para construir un árbol genealógico, para el análisis de circuitos eléctricos y para enumerar los capítulos y secciones de un libro.

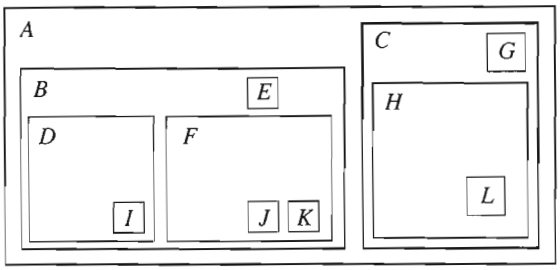
Un árbol se puede representar de diferentes formas y todas ellas se consideran equivalentes. En la figura 6.1 se presentan cinco notaciones diferentes correspondientes al mismo árbol. En la figura 6.1*a* se utiliza un diagrama de Venn; en la figura 6.1*b* la indentación de paréntesis; en la figura 6.1*c* la notación decimal de Dewey; en la figura 6.1*d* la notación indentada, y, por último, en la figura 6.1*e* un grafo. Esta última representación es la que comúnmente se utiliza, y ha originado el término árbol por su parecido abstracto con el vegetal —raíz, ramas, hojas—, a pesar de que la raíz se dibuja arriba, aunque en el vegetal se encuentre abajo.

En el grafo se distinguen **nodos** —círculos— y **arcos** —líneas con flechas—. Los primeros se usan para almacenar la información y los últimos para establecer la relación entre los nodos. En el ejemplo de la figura 6.1*e* los nodos guardan letras y los arcos permiten ir de ciertos nodos a otros.

6.2.1 Características y propiedades de los árboles

La estructura tipo árbol tiene ciertas características y propiedades que la distinguen. A continuación se presentan las más importantes:

a) Todo árbol que no es vacío tiene un único nodo raíz.



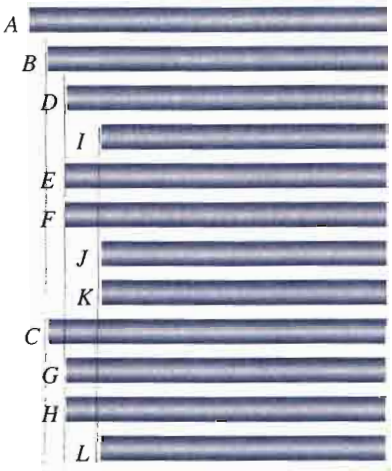
a)

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

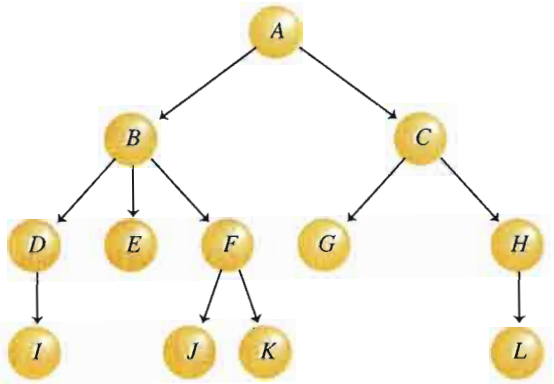
b)

1.A, 1.1.B, 1.1.1.D, 1.1.1.1.I, 1.1.2.E, 1.1.3.F, 1.1.3.1.J, 1.1.3.2.K, 1.2.C, 1.2.1.G, 1.2.2.H, 1.2.2.1.L

c)



d)



e)

FIGURA 6.1

Diferentes formas de representar una estructura de árbol. *a)* Diagramas de Venn. *b)* Notación de paréntesis. *c)* Notación decimal de Dewey. *d)* Notación indentada. *e)* Grafo.

b) Un nodo *X* es descendiente directo de un nodo *Y*, si el nodo *X* es apuntado por el nodo *Y*. En este caso es común utilizar la expresión *X* **es hijo de** *Y*.

c) Un nodo *X* es antecesor directo de un nodo *Y*, si el nodo *X* apunta al nodo *Y*. En este caso es común utilizar la expresión *X* **es padre de** *Y*.

d) Se dice que todos los nodos que son descendientes directos —hijos— de un mismo nodo —padre— son **hermanos**.

e) Todo nodo que no tiene ramificaciones —hijos—, se conoce con el nombre de **terminal** u **hoja**.

f) Todo nodo que no es raíz ni terminal u hoja se conoce con el nombre de **interior**.

g) **Grado** es el número de descendientes directos de un determinado nodo.

h) **Grado del árbol** es el máximo grado de todos los nodos del árbol.

i) **Nivel** es el número de arcos que deben ser recorridos para llegar a un determinado nodo. Por definición la raíz tiene nivel 1.

j) **Altura** del árbol es el máximo número de niveles de todos los nodos del árbol.

A continuación se presenta un ejemplo para clarificar estos conceptos.

Ejemplo 6.1

Dado el árbol general de la figura 6.2, se puede afirmar lo siguiente:

1. *A* es la raíz del árbol.

2. *B* es hijo de *A*.
   C es hijo de *A*.
   D es hijo de *B*.
   E es hijo de *B*.
   L es hijo de *H*.

3. *A* es padre de *B*.
   B es padre de *D*.
   D es padre de *I*.
   C es padre de *G*.
   H es padre de *L*.

4. *B* y *C* son hermanos.
   D, *E* y *F* son hermanos.
   G y *H* son hermanos.
   J y *K* son hermanos.

5. *I*, *E*, *J*, *K*, *G* y *L* son nodos terminales u hojas.

6. *B*, *D*, *F*, *C* y *H* son nodos interiores.

7. El grado del nodo *A* es 2.
   El grado del nodo *B* es 3.
   El grado del nodo *C* es 2.
   El grado del nodo *D* es 1.
   El grado del nodo *E* es 0.
   El grado del árbol es 3.

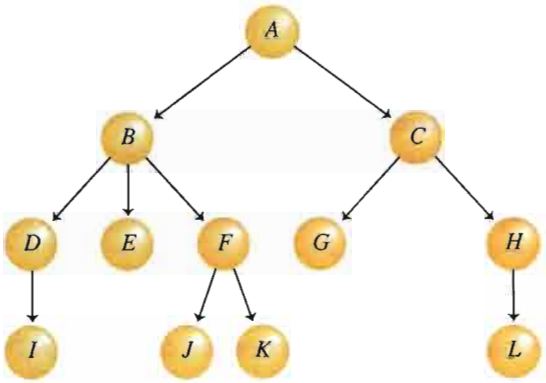
8. El nivel del nodo *A* es 1.
   El nivel del nodo *B* es 2.
   El nivel del nodo *D* es 3.
   El nivel del nodo *C* es 2.
   El nivel del nodo *L* es 4.

9. La altura del árbol es 4.

6.2.2 Longitud de camino interno y externo

Se define la longitud de camino del nodo *X* como el número de arcos que se deben correr para llegar desde la raíz hasta el nodo *X*. Por definición la raíz tiene longitud

FIGURA 6.2
Árbol general.



camino 1, sus descendientes directos longitud de camino 2 y así sucesivamente. Considere el árbol de la figura 6.2. El nodo *B* tiene longitud de camino 2, el nodo *I* longitud de camino 4 y el nodo *H* longitud de camino 3.

Longitud de camino interno

La **longitud de camino interno** (LCI) del árbol es la suma de las longitudes de camino de todos los nodos del árbol. Esta medida es importante porque permite conocer los caminos que tiene el árbol. Se calcula por medio de la siguiente fórmula:

$$LCI = \sum_{i=1}^{h} n_i * i \qquad \textbf{Fórmula 6.1}$$

donde i representa el nivel del árbol, h su altura y n_i el número de nodos en el nivel i.

La LCI del árbol de la figura 6.2 se calcula de esta forma:

$$LCI = 1 * 1 + 2 * 2 + 5 * 3 + 4 * 4 = 36$$

Ahora bien, la media de la longitud de camino interno (LCIM) se calcula dividiendo la LCI entre el número de nodos del árbol (n). La media es importante porque permite conocer, en promedio, el número de decisiones que se deben tomar para llegar a un determinado nodo partiendo desde la raíz. Se expresa:

$$LCIM = LCI/n \qquad \textbf{Fórmula 6.2}$$

La LCIM del árbol de la figura 6.2 se calcula como se muestra a continuación:

$$LCIM = 36/3 = 3$$

Longitud de camino externo

Para definir la longitud de camino externo de un árbol es necesario primero explicar conceptos de **árbol extendido** y **nodo especial**.

Un árbol extendido es aquel en el que el número de hijos de cada nodo es igual grado del árbol. Si alguno de los nodos del árbol no cumple con esta condición, entonces deben incorporarse al mismo tantos nodos especiales como se requiera para llegar a cumplirla.

Los nodos especiales tienen como objetivo remplazar las ramas vacías o nulas pueden tener descendientes y normalmente se representan con la forma de un cuadrado En la figura 6.3 se presenta el árbol extendido de la figura 6.2. El número de nodos especiales de este árbol es 25.

Se puede definir ahora la **longitud de camino externo** (LCE) de un árbol como suma de las longitudes de camino de todos los nodos especiales del árbol. Se calcula por medio de la siguiente fórmula:

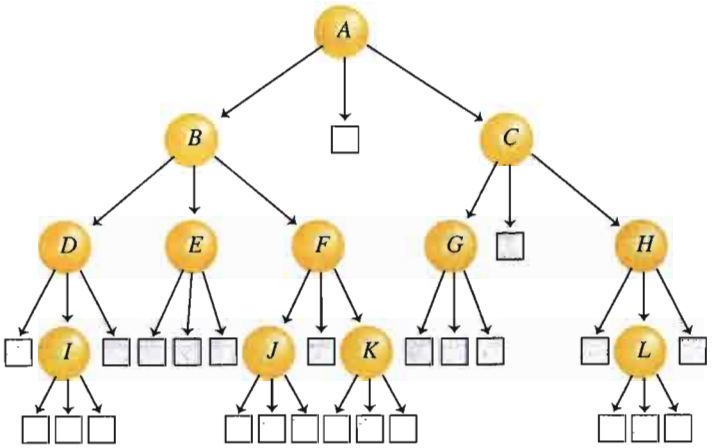
$$LCE = \sum_{i=2}^{h+1} n_{ei} * i \qquad \textbf{Fórmula}$$

donde i representa el nivel del árbol, h su altura y ne_i el número de nodos especiales en el nivel i. Observe que i comienza desde 2, puesto que la raíz se encuentra en el nivel y no puede ser un nodo especial.

La LCE del árbol de la figura 6.3 se calcula de esta manera:

$$LCE = 1 * 2 + 1 * 3 + 11 * 4 + 12 * 5 = 109$$

FIGURA 6.3
Árbol extendido.



Ahora bien, la media de la longitud de camino externo (LCEM) se calcula dividiendo la LCE entre el número de nodos especiales del árbol (*ne*). Observe el lector la siguiente fórmula:

$$LCEM = LCE/ne \qquad \textbf{Fórmula 6.4}$$

e indica el número de arcos que se deben recorrer en promedio para llegar, partiendo desde la raíz, a un nodo especial cualquiera del árbol.

La LCEM del árbol de la figura 6.3 se calcula de la siguiente manera:

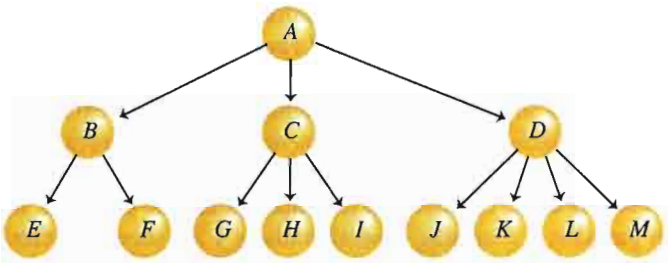
$$LCEM = 109/25 = 4.36$$

El siguiente ejemplo clarificará los conceptos de longitud de camino interno y externo.

Ejemplo 6.2

Dado el árbol general de la figura 6.4 y el árbol extendido de la figura 6.5 se calcula la longitud de camino interno:

FIGURA 6.4
...general.

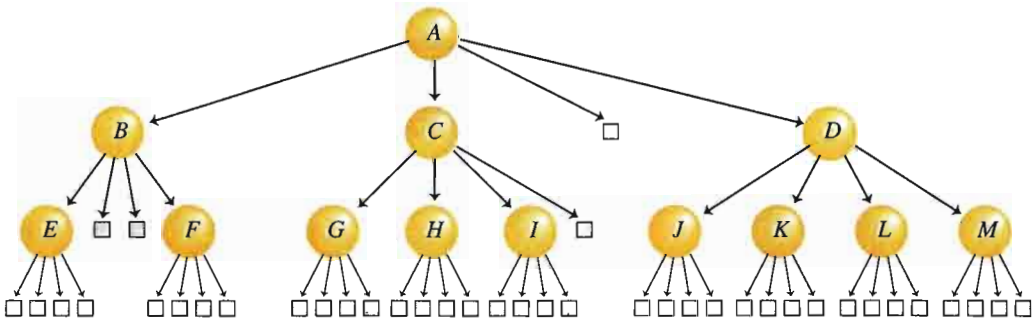


$$LCI = 1 * 1 + 3 * 2 + 9 * 3 = 34$$

La media de la longitud de camino interno:

$$LCIM = 34/13 = 2.61$$

FIGURA 6.5
...extendido.



La longitud de camino externo:

$$LCE = 1 * 2 + 3 * 3 + 36 * 4 = 155$$

La media de la longitud de camino externo:

$$LCEM = 155/40 = 3.87$$

6.3 ÁRBOLES BINARIOS

Un árbol ordenado es aquel en el cual la distribución de las ramas sigue un cierto
Los árboles ordenados de grado 2 son de especial interés en el área de la comp
porque permiten representar la información relacionada con la solución de muchos
blemas. Estos árboles son conocidos con el nombre de **árboles binarios**.

En un árbol binario cada nodo puede tener como máximo dos subárboles y és
distinguen entre sí como el subárbol izquierdo y el subárbol derecho, según su ubic
con respecto al nodo raíz.

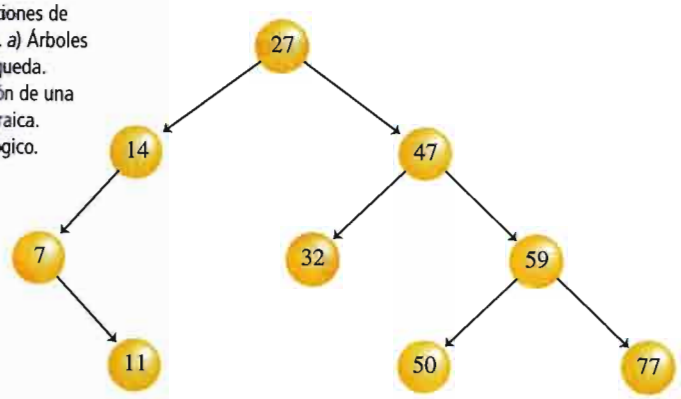
**Formalmente se define un árbol binario tipo *T* como una estructura ho
nea, resultado de la concatenación de un elemento de tipo *T*, llamado raíz, co
árboles binarios disjuntos, llamados subárbol izquierdo y subárbol derech
árbol binario especial es el árbol vacío.**

Los árboles binarios tienen múltiples aplicaciones. Se les puede utilizar pa
presentar la solución de un problema para el cual existen dos posibles alter
(árbol de decisiones), para representar un árbol genealógico (construido en f
ascendente y donde se muestran los ancestros de un individuo dado), para repre
la historia de un campeonato de tenis (construido en forma ascendente y donde
un ganador, 2 finalistas, 4 semifinalistas y así sucesivamente) y para represent
presiones algebraicas construidas con operadores binarios. Esto sólo por citar al
de sus múltiples usos.

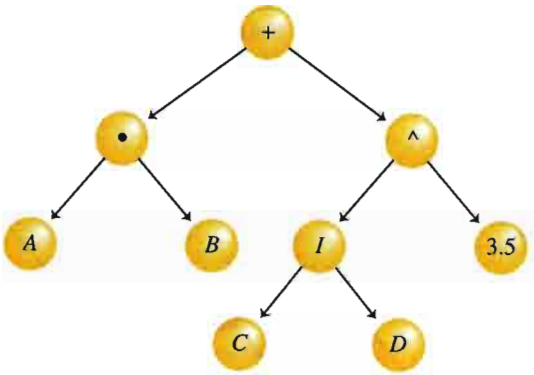
En la figura 6.6 se muestran tres diagramas correspondientes a una estruct
árbol binario. En la figura 6.6*a* hay un árbol binario de búsqueda (esta variante se
sentará con detalle más adelante), en la figura 6.6*b* el árbol binario que repres
expresión $(A * B) + (C/D) \wedge 3.5$ y en la figura 6.6*c* un árbol genealógico.

Los árboles ordenados de grado mayor a 2 representan también estructuras
portantes. Se conocen con el nombre de árboles multicaminos y serán estudiados
adelante en este mismo capítulo.

...aciones de ...narios. *a)* Árboles ... de búsqueda. ...representación de una ... algebraica. ... genealógico.



a)



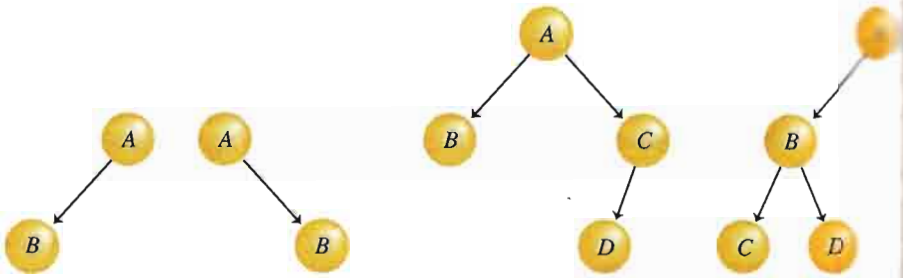
b)

c)

6.3.1 Árboles binarios distintos, similares y equivalentes

Dos árboles binarios son **distintos** cuando sus estructuras —la distribución de nodos y arcos— son diferentes. En la figura 6.7 se presentan dos ejemplos de árboles binarios distintos.

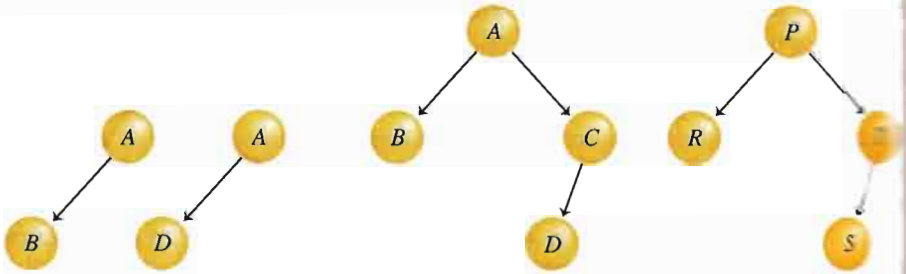
FIGURA 6.7
Árboles binarios distintos.



a) *b)*

Dos árboles binarios son **similares** cuando sus estructuras son idénticas, pero la información que contienen sus nodos difiere entre sí. En la figura 6.8 se presentan dos ejemplos de árboles binarios similares.

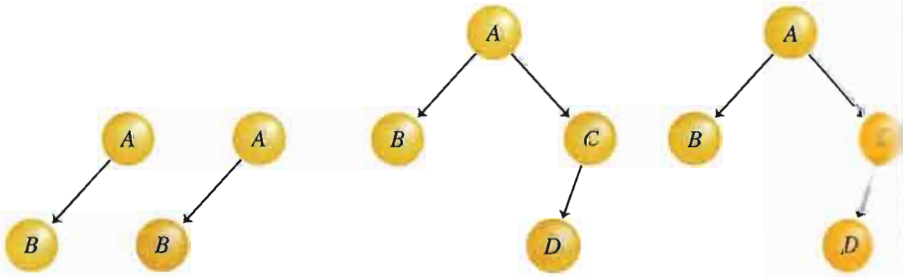
FIGURA 6.8
Árboles binarios similares.



a) *b)*

Por último, los árboles binarios **equivalentes** se definen como aquellos que son similares y además los nodos contienen la misma información. En la figura 6.9 se muestran dos ejemplos de árboles binarios equivalentes.

FIGURA 6.9
Árboles binarios equivalentes.



a) *b)*

Con el fin de clarificar los conceptos anteriores, se presenta el siguiente ejemplo.

Ejemplo 6.3

Dados los árboles binarios de la figura 6.10, se puede afirmar lo siguiente:

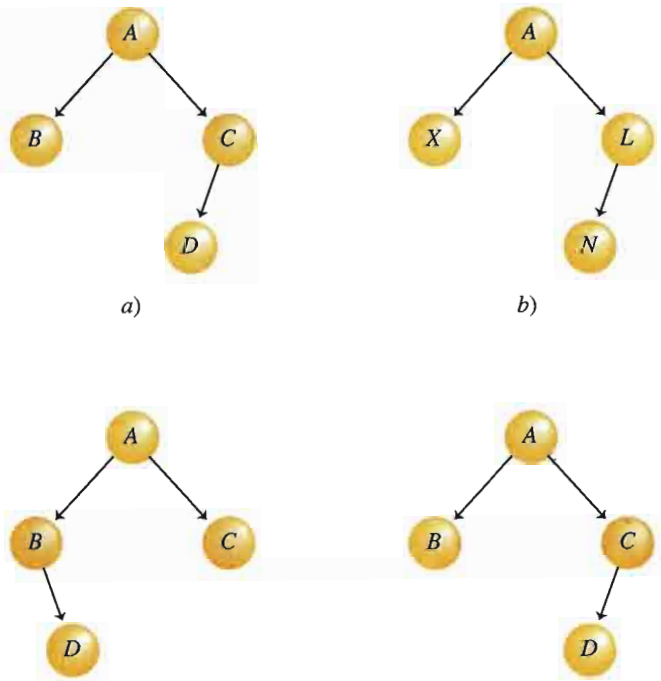
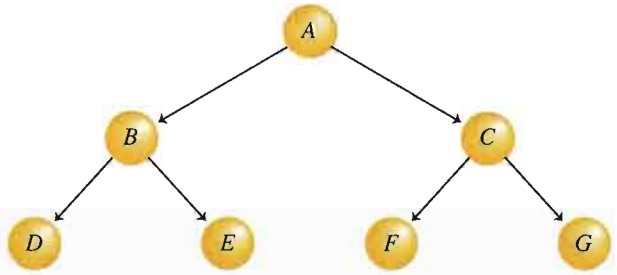


FIGURA 6.10
Árboles binarios distintos, similares y equivalentes.

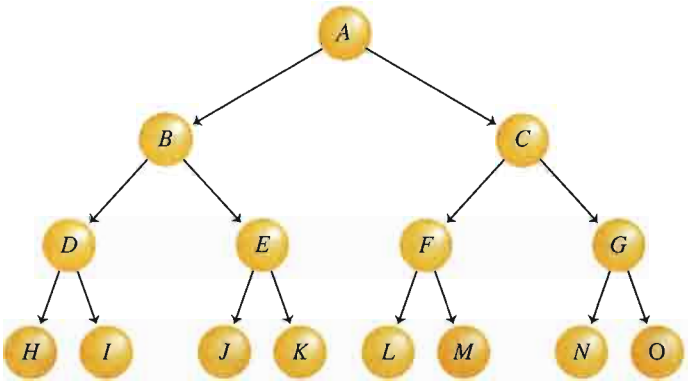
- El árbol de la figura 6.10*c* es distinto de los árboles de la figura 6.10*a*, 6.10*b* y 6.10*d*.
- Los árboles de la figura 6.10*a*, 6.10*b* y 6.10*d* son similares.
- Los árboles de la figura 6.10*a* y 6.10*d* son equivalentes.

6.3.2 Árboles binarios completos

Se define un **árbol binario completo (ABC)** como un árbol en el que todos sus nodos, excepto los del último nivel, tienen dos hijos: el subárbol izquierdo y el subárbol derecho. En la figura 6.11 se presentan dos ejemplos de árboles binarios completos.



a)



b)

FIGURA 6.11
Árboles binarios completos. *a)* De altura 3. *b)* De altura 4.

El número de nodos de un árbol binario completo de altura h, se puede cal aplicando la siguiente fórmula:

NÚMERO DE NODOS(ABC) = $2^h - 1$ **Fórmula**

Así, por ejemplo, un árbol binario completo de altura 5 tendrá 31 nodos, un altura 9 tendrá 511 nodos y un árbol de altura 17 tendrá 131 071 nodos.

Cabe aclarar que existen algunos autores que definen un árbol binario comple otra forma y otros que utilizan el término *lleno* para referirse a lo que en este libr denomina *completo.*

6.3.3 Representación de árboles generales como binarios

Los árboles binarios, por las razones ya mencionadas, se aplican en la solución co tacional de muchos problemas. Además, su uso se ve favorecido por su dinamism no linealidad entre sus elementos y por su sencilla programación. Por lo tanto, re muy útil poder convertir árboles generales, con 0 a n hijos, en árboles binarios.

En esta sección se darán los pasos necesarios para lograrlo. Considere el árbol neral de la figura 6.12*a*. Las operaciones que se deben aplicar para lograr la conver del árbol general al árbol binario correspondiente son las siguientes:

1. Enlazar los hijos de cada nodo en forma horizontal —los hermanos—.
2. Relacionar en forma vertical el nodo padre con el hijo que se encuentra más a la izquierda. Además, se debe eliminar el vínculo de ese padre con el resto de sus hijos.
3. Rotar el diagrama resultante, aproximadamente 45 grados hacia la izquierda, y así se obtendrá el árbol binario correspondiente.

En la figura 6.12*b* se visualiza el árbol luego de aplicar los dos primeros pasos. En la figura 6.12*c* se observa el árbol binario, obtenido luego de aplicar el tercer paso.

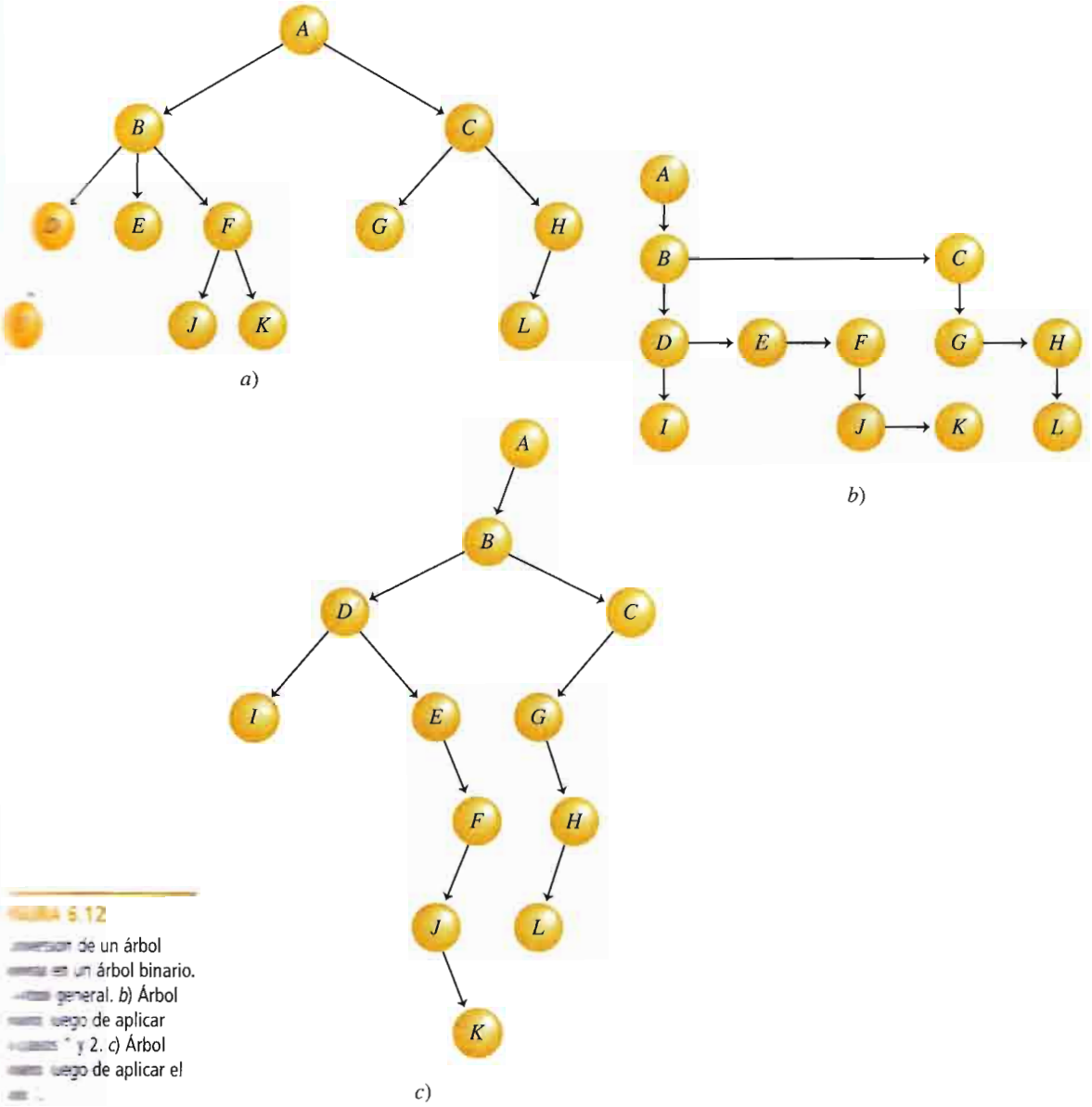
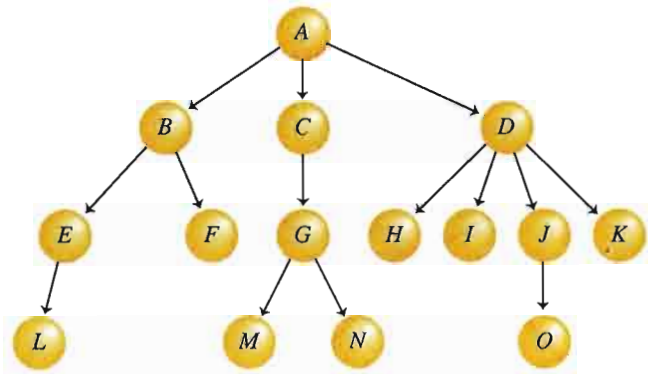


FIGURA 6.12
...de un árbol ... en un árbol binario. ... general. *b)* Árbol ... luego de aplicar ... y 2. *c)* Árbol ... luego de aplicar el ...

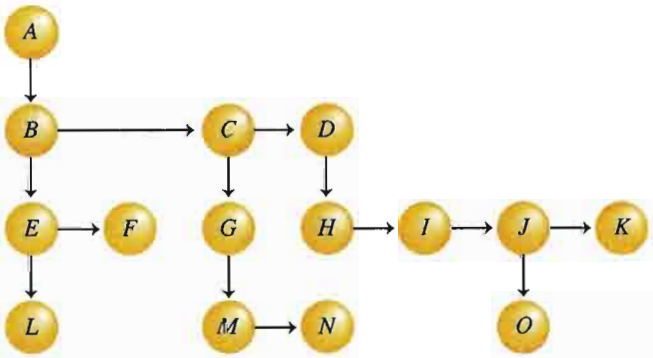
Ejemplo 6.4

Dado como dato el árbol general de la figura 6.13*a*, se debe convertir a un árbol bi
En la figura 6.13*b* se observa una gráfica del árbol luego de aplicar los dos prime
sos. En la figura 6.13*c* se observa el árbol binario que se obtiene luego de que se
el tercer paso.

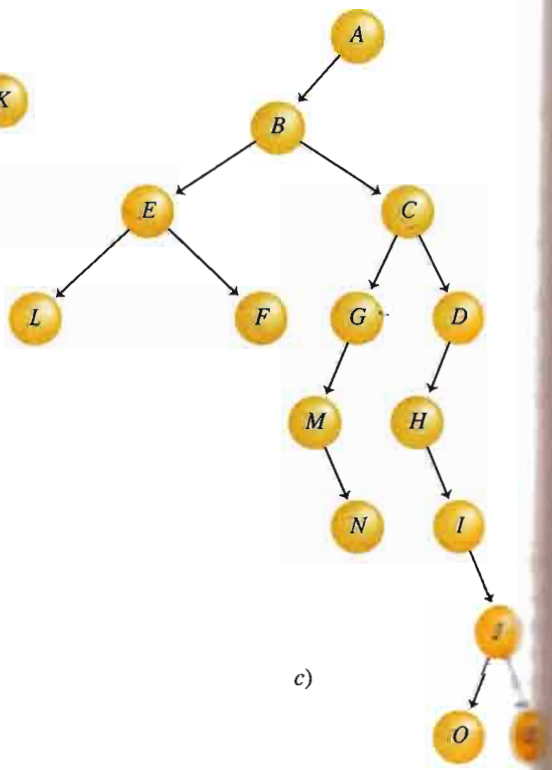
FIGURA 6.13
Conversión de un árbol general en un árbol binario. *a*) Árbol general. *b*) Árbol binario luego de aplicar los pasos 1 y 2. *c*) Árbol binario luego de aplicar el paso 3.



a)



b)



c)

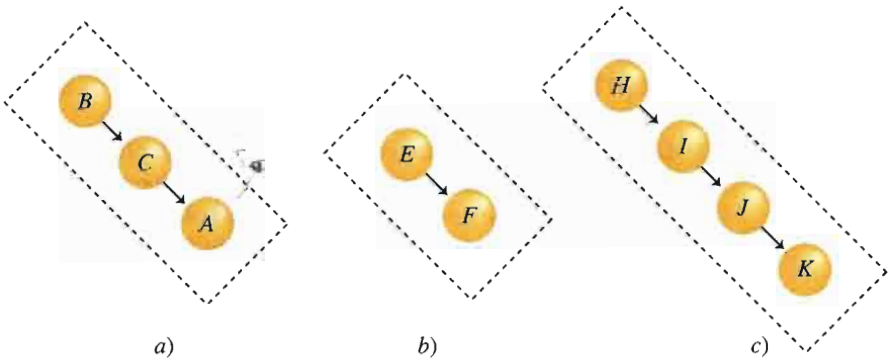
Observe que para todo nodo de un árbol binario, generado a partir de un árbol general, se debe cumplir lo siguiente:

1. Si la rama derecha de cada nodo, excepto el nodo raíz, es distinta de vacío se encuentra un nodo que era hermano de éste en el árbol general. De la figura 6.13*c* podemos deducir que *C* era hermano de *B*. Aplicando el mismo criterio deducimos también que *D* era hermano de *C* y, por lo tanto, por transitividad, hermano de *B*. Otras deducciones que se pueden realizar observando la figura 6.13*c* son las siguientes:

   - *E* y *F* eran hermanos.
   - *M* y *N* eran hermanos.
   - *H*, *I*, *J* y *K* eran hermanos.

   Es de notar que los hermanos se encuentran en la gráfica en una línea oblicua continua, orientada 45 grados hacia la derecha. En la figura 6.14 se presentan tres diagramas diferentes donde se pueden observar algunos ejemplos.

URA 6.14
ermanos.



a) *b)* *c)*

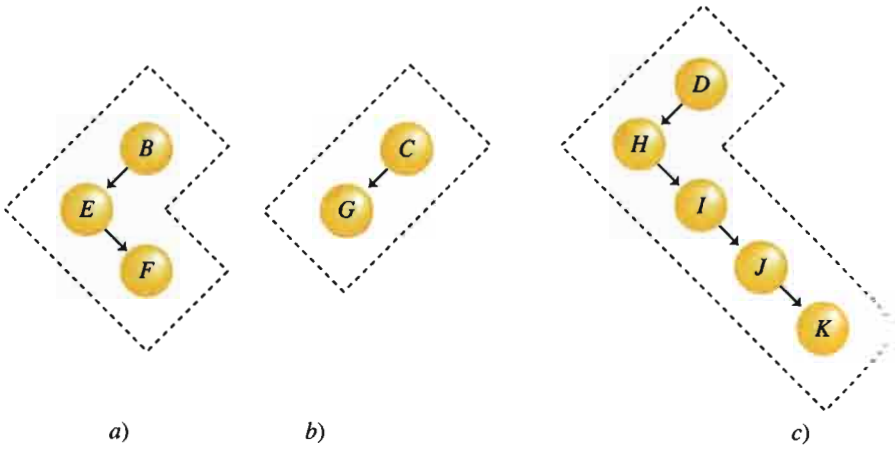
2. En la rama izquierda de cada nodo —si ésta es distinta de vacío— se encuentra un nodo que era hijo de éste en el árbol general. De la figura 6.13*c* podemos deducir que *E* era hijo de *B* y como por (1) *F* era hermano de *E*, podemos afirmar que *F* era también hijo de *B*. Otras deducciones que se pueden realizar observando la figura 6.13*c* son las siguientes:

   - *B*, *C* y *D* eran hijos de *A*.
   - *M* y *N* eran hijos de *G*.
   - *G* era hijo de *C*.

Es de notar que los hijos de un nodo se encuentran en la gráfica, primero en una línea oblicua continua orientada 45 grados hacia la izquierda y luego en una línea continua oblicua orientada 45 grados hacia la derecha. En la figura 6.15 hay tres diagramas diferentes donde se observan algunos ejemplos.

En la figura 6.15*b* no existe línea oblicua orientada hacia la derecha porque *G* es el único hijo del nodo *C*.

FIGURA 6.15
Nodos hijos.

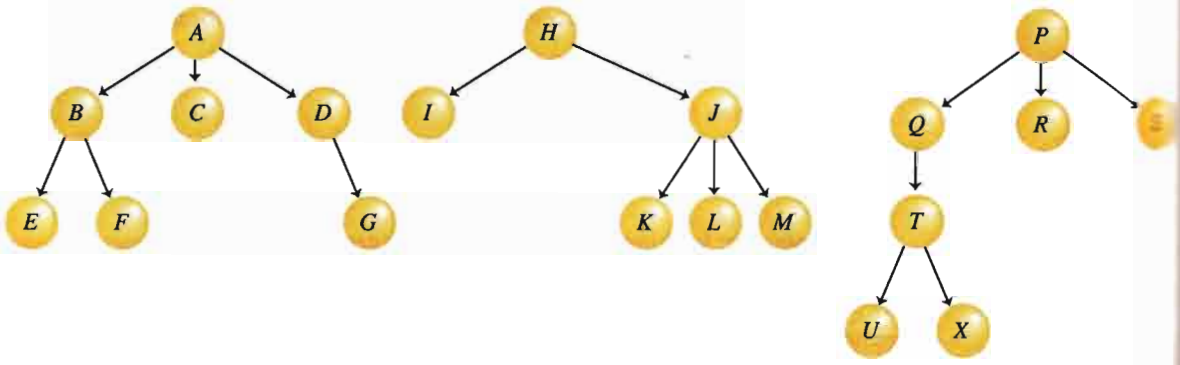


6.3.4 Representación de un bosque como árbol binario

Un **bosque** representa un conjunto normalmente ordenado de uno o más árboles generales. Es posible utilizar el algoritmo de conversión analizado en la sección anterior, con algunas modificaciones, para generar un árbol binario a partir de un bosque.

Considere por ejemplo el bosque, formado por tres árboles generales, de la figura 6.16. Los pasos que se deben aplicar para lograr la conversión del bosque a un árbol binario son los siguientes:

FIGURA 6.16
Bosque de árboles generales.

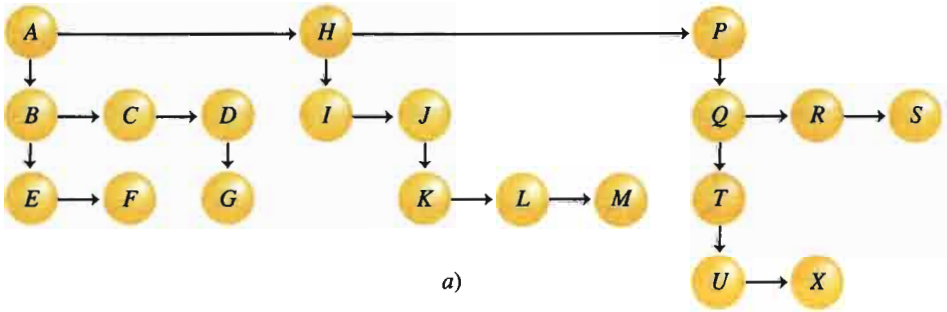


1. Enlazar en forma horizontal las raíces de los distintos árboles generales.
2. Relacionar los hijos de cada nodo —los hermanos— en forma horizontal.
3. Enlazar en forma vertical el nodo padre con el hijo que se encuentra más a la izquierda. Además, se debe eliminar el vínculo del padre con el resto de sus hijos.
4. Rotar el diagrama resultante aproximadamente 45 grados hacia la izquierda y así se obtendrá el árbol binario correspondiente.

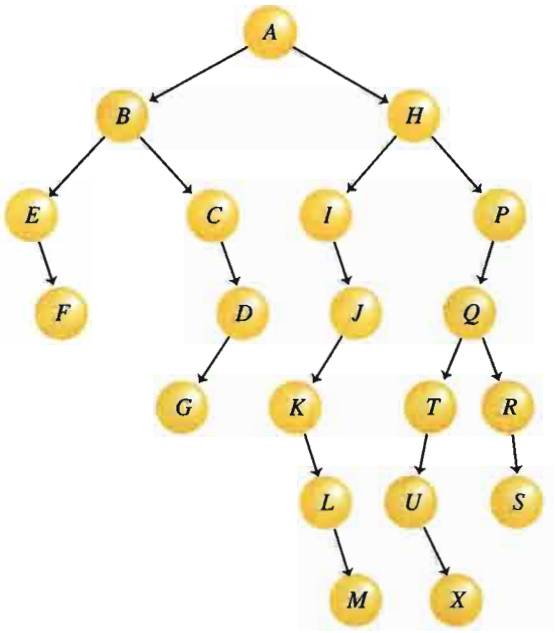
En la figura 6.17*a* se muestra el árbol luego de aplicar los tres primeros pasos. En la figura 6.17*b* se observa el árbol binario obtenido luego de que se realiza el cuarto paso.

FIGURA 6.17

Conversión de un bosque en árbol binario. *a*) Árbol binario luego de aplicar los pasos 1, 2 y 3. *b*) Árbol binario luego de aplicar el paso 4.



a)



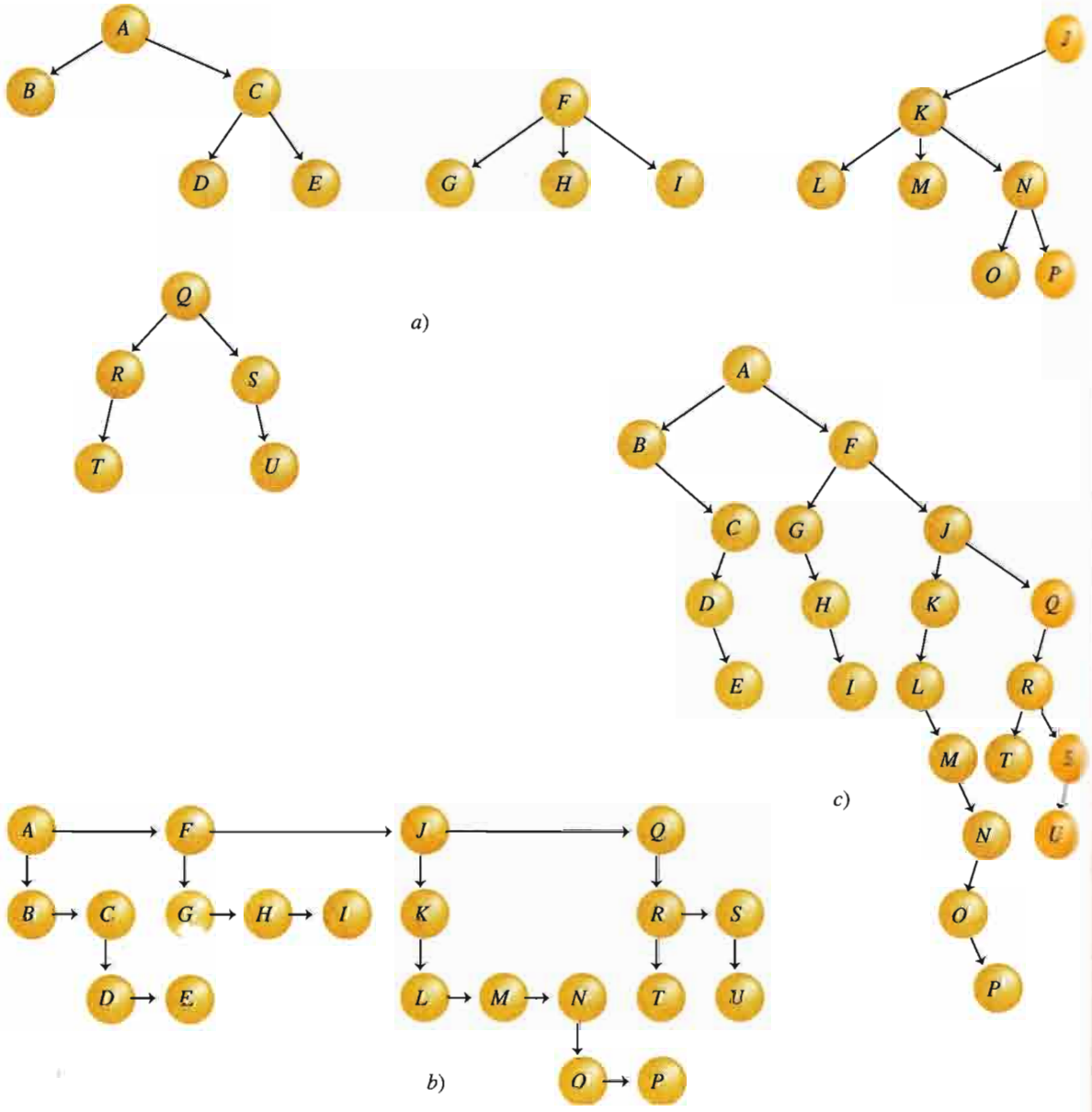
b)

Ejemplo 6.5

Dado como dato el bosque de la figura 6.18*a*, se desea convertirlo a un árbol binario. En la figura 6.18*b* se observa una gráfica del árbol luego de aplicar los tres primeros pasos. En la figura 6.18*c* se puede apreciar el árbol binario que se obtiene luego de que se aplica el cuarto paso.

FIGURA 6.18

Conversión de un bosque en árbol binario. *a*) Bosque. *b*) Árbol luego de aplicar el primero, segundo y tercer pasos. *c*) Árbol binario luego de aplicar el cuarto paso.



Es de notar que para todo nodo de un árbol binario, que se obtiene a partir de un bosque, se cumplen los dos incisos señalados en la conversión de un árbol general en un árbol binario.

6.3.5 Representación de árboles binarios en memoria

Las dos maneras más comunes de representar un árbol binario en memoria son:

1. Por medio de datos tipo puntero, también conocidos como variables dinámicas.
2. Por medio de arreglos.

En este libro se explicará y utilizará la primera forma, puesto que representa la más natural para tratar una estructura de datos de este tipo.

Al final del capítulo se presentará una breve introducción a los árboles, desde el punto de vista del paradigma orientado a objetos. Sin embargo, lo que se estudiará a continuación sigue siendo válido para las clases. Como en las otras estructuras de datos presentadas en este libro, los conceptos explicados son los fundamentos requeridos para el uso de las mismas, independientemente del paradigma y del lenguaje utilizado para su implementación.

Los nodos del árbol binario se representan como registros. Cada uno de ellos contiene como mínimo tres campos. En un campo se almacenará la información del nodo. Los dos restantes se utilizarán para apuntar los subárboles izquierdo y derecho, respectivamente, del nodo en cuestión.

Dado el nodo *T*:

En él se distinguen tres campos:

- **IZQ:** es el campo donde se almacena la dirección del subárbol izquierdo del nodo *T*.
- **INFO:** representa el campo donde se almacena la información del nodo. Normalmente en este campo y en el transcurso de este libro se almacenará un valor simple: número o carácter. Sin embargo, en la práctica es común almacenar en este campo cualquier tipo de dato.
- **DER:** es el campo donde se almacena la dirección del subárbol derecho del nodo *T*.

La definición de un árbol binario en lenguaje algorítmico es como sigue:

```
ENLACE = ^NODO
NODO   = REGISTRO
           IZQ: tipo ENLACE
           INFO: tipo de dato
           DER: tipo ENLACE
{Fin de la definición}
```

Nota: Es importante observar que se utiliza el símbolo ^ para representar el concepto de dato tipo puntero.

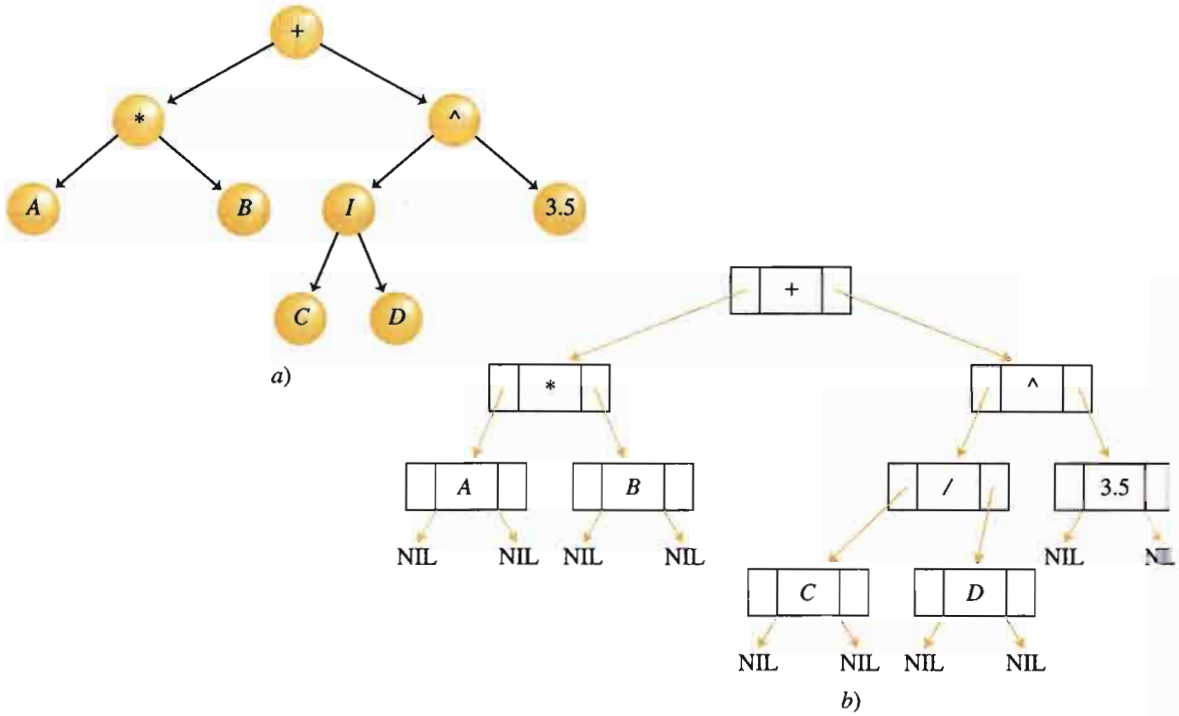
Ejemplo 6.6

Considere el árbol binario de la figura 6.19*a*, que representa la expresión algebraica (*A* * *B*) + (*C* / *D*) ^ 3.5. Su representación en memoria es como la que se muestra en la figura 6.19*b*.

Note el lector que en la figura 6.19*b* se utiliza el término NIL para hacer referencia al árbol vacío.

FIGURA 6.19

Representación de un árbol binario en memoria. *a*) Árbol binario. *b*) Su representación en memoria.



Como todas las estructuras de datos, los árboles tienen asociadas ciertas operaciones. A continuación se presentan los algoritmos de algunas de estas operaciones y más adelante, cuando se estudien otros tipos de árboles, se explicarán otras.

6.3.6 Operaciones en árboles binarios

Una de las operaciones básicas de un árbol binario es la creación del mismo en memoria. Un algoritmo muy simple para formar un árbol, por medio de la creación dinámica de nodos y la asignación a éstos de información, es el que se muestra a continuación:

Algoritmo 6.1 Crea_árbol

Crea_árbol (APNODO)

{El algoritmo crea un árbol binario en memoria. APNODO es una variable de tipo ENLACE —puntero a un nodo—. La primera vez APNODO se crea en el programa principal}
{INFO, IZQ y DER son campos del registro NODO. INFO es de tipo carácter. IZQ y DER son de tipo puntero. Las variables RESP y OTRO son de tipo carácter y de tipo ENLACE, respectivamente}

```
1. Leer APNODO^.INFO  {Lee la información y se guarda en el nodo}
2. Escribir "¿Existe nodo por izquierda: 1(Sí) – 0(No)?"
3. Leer RESP
4. Si (RESP = "Sí")
     entonces
        Crear(OTRO) {Se crea un nuevo nodo}
        Hacer APNODO^.IZQ ← OTRO
        Regresar a Crea_árbol con APNODO^.IZQ {Llamada recursiva}
     si no
        Hacer APNODO^.IZQ ← NIL
5. {Fin del condicional del paso 4}
6. Escribir "¿Existe nodo por derecha: 1(Sí) – 0(No)?"
7. Leer RESP
8. Si (RESP = "Sí")
     entonces
        Crear(OTRO) {Se crea un nuevo nodo}
        Hacer APNODO^.DER ← OTRO
        Regresar a Crea_árbol con APNODO^.DER  {Llamada recursiva}
     si no
        Hacer APNODO^.DER ← NIL
9. {Fin del condicional del paso 8}
```

Una vez que se crea el árbol binario, se pueden realizar otras operaciones sobre sus elementos: recorrer todos los nodos, insertar un nuevo nodo, eliminar alguno de los existentes o buscar un valor determinado.

Una de las operaciones más importantes que se realiza en un árbol binario es el recorrido de los mismos. Recorrer significa visitar los nodos del árbol en forma ordenada, de tal manera que todos los nodos del mismo sean visitados una sola vez. Existen tres formas diferentes de efectuar el recorrido y todas ellas de naturaleza recursiva; éstas son:

a) Recorrido en **preorden**
   - Visitar la raíz
   - Recorrer el subárbol izquierdo
   - Recorrer el subárbol derecho

b) Recorrido en **inorden**
   - Recorrer el subárbol izquierdo

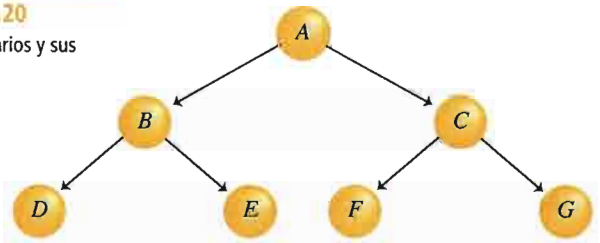
- Visitar la raíz
- Recorrer el subárbol derecho

c) Recorrido en **posorden**
- Recorrer el subárbol izquierdo
- Recorrer el subárbol derecho
- Visitar la raíz

En la figura 6.20 se muestran tres árboles binarios con el resultado que se obtiene al efectuar los diferentes tipos de recorrido. En este ejemplo, la visita del nodo implica la impresión de su contenido.

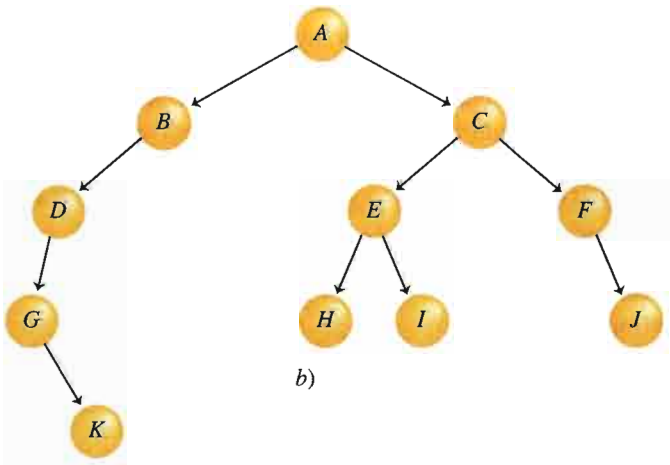
Note que en un árbol binario que representa una expresión algebraica, por ejemplo, el árbol de la figura 6.20*c*, la impresión de la información de sus nodos, usando el recorrido preorden, produce la notación polaca prefija. En el caso del recorrido inorden se obtiene la notación convencional y, por último, el recorrido posorden produce la notación polaca posfija. Aunque, cabe aclarar, sin los paréntesis respectivos que indican la precedencia de los distintos operadores.

FIGURA 6.20
Árboles binarios y sus recorridos.



PREORDEN: *A B D E C F G*
INORDEN: *D B E A F C G*
POSORDEN: *D E B F G C A*

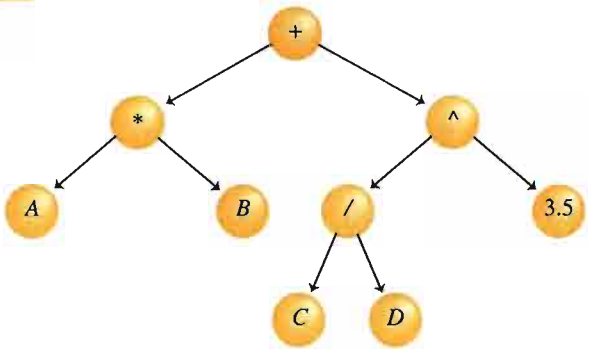
a)



PREORDEN: *A B D G K C E H I F J*
INORDEN: *G K D B A H E I C F J*
POSORDEN: *K G D B H I E J F C A*

b)

FIGURA 6.20
(continuación)



c)

PREORDEN: + * *A B* ^ / *C D* 3.5

INORDEN: *A* * *B* + / *D* ^ 3.5

POSORDEN: *A B* * *C D* / 3.5 ^+

Se analizan a continuación los algoritmos que efectúan los diferentes tipos de recorridos en un árbol binario.

Algoritmo 6.2 Preorden

Preorden (APNODO)

{Este algoritmo realiza el recorrido preorden de un árbol binario. APNODO es un dato de tipo ENLACE —puntero a un nodo—}
{INFO, IZQ y DER son campos del registro nodo. INFO es una variable de tipo carácter. IZQ y DER son variables de tipo puntero}
1. *Si* (APNODO ≠ NIL) *entonces*
   Visitar el APNODO {Escribir NODO^.INFO}
   Regresar a Preorden con APNODO^.IZQ
   {Llamada recursiva a Preorden con la rama izquierda del nodo en cuestión}
   Regresar a Preorden con APNODO^.DER
   {Llamada recursiva a Preorden con la rama derecha del nodo en cuestión}
2. {Fin del condicional del paso 1}

Nota: Cabe destacar que el término *visitar* se puede reemplazar por cualquier otra instrucción válida, por ejemplo escribir, sumar o comparar la información del nodo. Note que esta aclaración se aplica también para los otros tipos de recorridos.

Ejemplo 6.7

En la siguiente tabla se presentan los pasos necesarios para obtener el recorrido preorden del árbol binario de la figura 6.20 *a*), utilizando el algoritmo 6.2.

En la columna *Pila: rama pendiente de visitar*, la llamada (*N*) indica el orden en el cual las ramas pendientes de visitar se introdujeron en la pila. En la columna *Nodo actual* la llamada (*N*) señala la rama que se extrajo de la pila. Observe el lector que el orden en que los nodos se visitaron es:

A - B - D - E - C - E - G

TABLA 6.3
Recorrido preorden

Paso	Nodo actual	Nodo visitado	Rama a visitar	Pila: rama pendiente de visitar
1	*A*	*A*	*A*^.IZQ → *B*	*A*^.DER → *C* (1)
2	*B*	*B*	*B*^.IZQ → *D*	*B*^.DER → *E* (2)
3	*D*	*D*	*D*^.IZQ → NIL	*D*^.DER → NIL (3)
4	NIL			
5	(3) NIL			
6	(2) *E*	*E*	*E*^.IZQ → NIL	*E*^.DER → NIL (4)
7	NIL			
8	(4) NIL			
9	(1) *C*	*C*	*C*^.IZQ → *F*	*C*^.DER → *G* (5)
10	*F*	*F*	*F*^.IZQ → NIL	*F*^.DER → NIL (6)
11	NIL			
12	(6) NIL			
13	(5) *G*	*G*	*G*^.IZQ → NIL	*G*^.DER → NIL (7)
14	NIL			
15	(7) NIL			

Algoritmo 6.3 Inorden

Inorden (APNODO)

{Este algoritmo realiza el recorrido inorden de un árbol binario. APNODO es un registro de tipo ENLACE —puntero a un nodo—}
{INFO, IZQ y DER son campos del registro nodo. INFO es una variable de tipo carácter. IZQ y DER son variables de tipo puntero}

1. *Si* (APNODO ≠ NIL) *entonces*
   Regresar a Inorden con APNODO^.IZQ
   {Llamada recursiva a Inorden con la rama izquierda del nodo en cuestión}
   Visitar el APNODO {Escribir APNODO^.INFO}
   Regresar a Inorden con APNODO^.DER
   {Llamada recursiva a Inorden con la rama derecha del nodo en cuestión}
2. {Fin del condicional del paso 1}

Ejemplo 6.8

En la tabla 6.4 se muestra la generación del recorrido inorden del árbol de la figura 6.20*a*, usando el algoritmo 6.3.

TABLA 6.4
Recorrido inorden

Paso	Nodo actual	Rama a visitar	Nodo visitado	Pila: rama pendiente de visitar	
1	*A*	*A*^.IZQ → *B*		*A*^.DER → *C* *A*	(1) (2)
2	*B*	*B*^.IZQ → *D*		*B*^.DER → *E* *B*	(3) (4)
3	*D*	*D*^.IZQ → NIL		*D*^.DER → NIL *D*	(5) (6)
4	NIL		*D* (6)		
5	(5) NIL		*B* (4)		
6	(3) *E*	*E*^.IZQ → NIL		*E*^.IZQ → NIL *E*	(7) (8)
7	NIL		*E* (8)		
8	(7) NIL		*A* (2)		
9	(1) *C*	*C*^.IZQ → *F*		*C*^.DER → *G* *C*	(9) (10)
10	*F*	*F*^.IZQ → NIL		*F*^.DER → NIL *F*	(11) (12)
11	NIL		*F* (12)		
12	(11) NIL		*C* (10)		
13	(9) *G*	*G*^.IZQ → NIL		*G*^.DER → NIL *G*	(13) (14)
14	NIL		*G* (14)		
15	(13) NIL				

En la columna *Pila: rama pendiente de visitar*, la llamada (*N*) indica el orden en el cual las ramas pendientes de visitar fueron introducidas a la pila. En las columnas *Nodo actual* y *Nodo visitado*, las llamadas (*N*) indican las instrucciones que se extrajeron de la pila. Note que esta observación también es válida para la tabla 6.5.

El orden en que se visitaron los nodos es:

D - *B* - *E* - *A* - *F* - *C* - *G*

Algoritmo 6.4 Posorden

Posorden (APNODO)

{Este algoritmo realiza el recorrido posorden de un árbol binario. APNODO es un dato de tipo ENLACE —puntero a un nodo—}
{INFO, IZQ y DER son campos del registro nodo. INFO es una variable de tipo carácter. IZQ y DER son variables de tipo puntero}

1. *Si* (APNODO ≠ NIL) *entonces*
   Regresar a Posorden con APNODO^.IZQ
   {Llamada recursiva a Posorden con la rama izquierda del nodo en cuestión}
   Regresar a Posorden con APNODO^.DER
   {Llamada recursiva a Posorden con la rama derecha del nodo en cuestión}
   Visitar el APNODO {Escribir APNODO^.INFO}
2. {Fin del condicional del paso 1}

Ejemplo 6.9

En la tabla 6.5 se presentan los pasos necesarios que se siguieron para efectuar el recorrido posorden del árbol de la figura 6.20*a*, aplicando el algoritmo anterior.

TABLA 6.5
Recorrido posorden

Paso	Nodo actual	Rama a visitar	Nodo visitado	Pila: rama pendiente de visitar
1	*A*	*A*^.IZQ → *B*		*A* (1)
				A^.DER → *C* (2)
2	*B*	*B*^.IZQ → *D*		*B* (3)
				B^.DER → *E* (4)
3	*D*	*D*^.IZQ → NIL		*D* (5)
				D^.DER → NIL (6)
4	NIL			
5	(6) NIL		*D* (5)	
6	(4) *E*	*E*^.IZQ → NIL		*E* (7)
				E^.DER → NIL (8)
7	NIL			
8	(8) NIL		*E* (7)	
9			*B* (3)	
10	(2) *C*	*C*^.IZQ → *F*		*C* (9)
				C^.DER → *G* (10)
11	*F*	*F*^.IZQ → NIL		*F* (11)
				F^.DER → NIL (12)
12	NIL			
13	(12) NIL		*F* (11)	
14	(10) *G*	*G*^.IZQ → NIL		*G* (13)
				G^.DER → NIL (14)
15	NIL			
16	(14) NIL		*G* (13)	
17			*C* (9)	
18			*A* (1)	

6.3.7 Árboles binarios de búsqueda

En esta sección se presenta un tipo especial de árboles binarios. Su principal característica es que la información se almacena en los nodos cuidando de mantener cierto orden.

Formalmente se define un **árbol binario de búsqueda** de la siguiente manera: **Para todo nodo *T* del árbol se debe cumplir que todos los valores almacenados en el subárbol izquierdo de *T* sean menores o iguales a la información guardada en el nodo *T*. De forma similar, todos los valores almacenados en el subárbol derecho de *T* deben ser mayores o iguales a la información guardada en el nodo *T*.** Los valores a los que se hace referencia en la definición refieren al contenido del campo de información del nodo. Por ejemplo, si en el árbol se almacena información de los empleados de una empresa, los valores utilizados para generar el árbol de manera ordenada corresponderán al número de cada empleado —campo clave—.

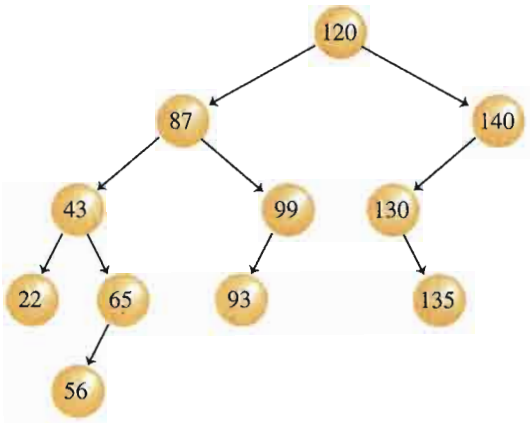
El árbol binario de búsqueda es una estructura de datos obre la cual se pueden realizar eficientemente las operaciones de búsqueda, inserción y eliminación. Comparando esta estructura con otras, se pueden observar ciertas ventajas. Por ejemplo, en un arreglo es posible localizar datos eficientemente si éstos se encuentran ordenados, pero las operaciones de inserción y eliminación resultan costosas, porque involucran movimiento de los elementos dentro del arreglo. En las listas, por otra parte, dichas operaciones se pueden llevar a cabo con facilidad, pero la operación de búsqueda, en este caso, es una operación que demanda recursos, pudiendo incluso requerir recorrer todos los elementos de ella para llegar a uno en particular.

Ejemplo 6.10

La figura 6.21 contiene un árbol binario de búsqueda.

Observe el lector que si en dicho árbol se sustituyen los valores 140 por 160, 99 por 105 y 43 por 55, el árbol continúa siendo un árbol binario de búsqueda.

Pero por otra parte, si en dicho árbol se reemplaza el valor 87 por 125, entonces el árbol deja de ser un árbol binario de búsqueda, puesto que viola el principio que dice: "Todos los nodos del subárbol izquierdo del nodo *T* deben almacenar valores menores o iguales al nodo´´ —en este caso 125 no es menor a 120—.



RA 6.21
inario de búsqueda.

También es posible observar que si se efectúa un recorrido inorden sobre un árbol de búsqueda se obtendrá una clasificación de los nodos en forma ascendente. El recorrido inorden del árbol de la figura 6.21 produce el siguiente resultado:

22 - 43 - 56 - 65 - 87 - 93 - 99 - 120 - 130 - 135 - 140

Búsqueda

La operación de **búsqueda en un árbol binario de búsqueda** es mucho más eficiente que en un árbol binario general, ya que al comparar el valor buscado con la información del nodo visitado, si no es igual, se deberá continuar sólo por alguno de los dos subárboles. Por ejemplo, si se compara el valor buscado 90 con el valor del nodo visitado 81, la búsqueda sólo debe continuar por el camino de la derecha. El camino de la izquierda se desecha, porque contiene nodos cuyos valores serán menores o iguales a 85.

Algoritmo 6.5 Búsqueda_ABB

Búsqueda_ABB (APNODO, INFOR)

{Este algoritmo localiza el nodo del árbol binario de búsqueda que contiene la información INFOR, que estamos buscando. APNODO es un parámetro de tipo ENLACE —la primera vez apunta a la raíz del árbol—. Se asume que el árbol no es vacío}

```
1. Si (INFOR < APNODO^.INFO)
     entonces
   1.1 Si (APNODO^.IZQ = NIL)
       entonces
           Escribir "La información no se encuentra en el árbol"
       si no
           Regresar a Búsqueda_ABB con APNODO^.IZQ e INFOR
           {Llamada recursiva}
   1.2 {Fin del condicional del paso 1.1}
     si no
   1.3 Si (INFOR > APNODO^.INFO)
         entonces
     1.3.1 Si (APNODO^.DER = NIL)
               entonces
                   Escribir "La información no se encuentra en el árbol"
               si no
                   Regresar a Búsqueda_ABB con APNODO^.DER e INFOR
                   {Llamada recursiva}
     1.3.2 {Fin del condicional del paso 1.3.1}
         si no
             Escribir "La información está en el árbol"
   1.4 {Fin del condicional del paso 1.3}
2. {Fin del condicional del paso 1}
```

Analice el algoritmo de búsqueda con el siguiente ejemplo.

Ejemplo 6.11

Supongamos que se desea localizar las claves 93 y 123 en el árbol binario de búsqueda de la figura 6.21. En las tablas 6.6 y 6.7 se presentan los pasos (P), número de comparaciones (C) y preguntas y acciones necesarias para localizar las claves 93 y 123, respectivamente.

TABLA 6.6
Localización de la clave 93 (INFOR ← 93)

P	C	Preguntas y acciones
1	1	¿Es 93 < 120?
		Sí. ¿Es el subárbol izquierdo de 120 (87) = NIL?
	2	No. Entonces se regresa a Búsqueda con el subárbol izquierdo de 120 (87) e INFOR
	3	¿Es 93 < 87?
2	4	No. ¿Es 93 > 87?
		Sí. ¿Es el subárbol derecho de 87 (99) = NIL?
	5	No. Entonces se regresa a Búsqueda con el subárbol derecho de 87 (99) e INFOR
	6	¿Es 93 < 99?
3		Sí. ¿Es el subárbol izquierdo de 99 (93) = NIL?
	7	No. Entonces se regresa a Búsqueda con el subárbol izquierdo de 99 (93) e INFOR
	8	¿Es 93 < 93?
4	9	No. ¿Es 93 > 93?
		No. Entonces ÉXITO

TABLA 6.7
Localización de la clave 123 (INFOR ← 123)

P	C	Preguntas y acciones
	1	¿Es 123 < 120?
1	2	No. ¿Es 123 > 120?
		Sí. ¿Es el subárbol derecho de 120 (140) = NIL?
	3	No. Entonces se regresa a Búsqueda con el subárbol derecho de 120 (140) e INFOR
	4	¿Es 123 < 140?
2		Sí. ¿Es el subárbol izquierdo de 140 (130) = NIL?
	5	No. Entonces se regresa a Búsqueda con el subárbol izquierdo de 140 (130) e INFOR
	6	¿Es 123 < 130?
3	7	Sí. ¿Es el subárbol izquierdo de 130 (NIL) = NIL?
		Sí. Entonces FRACASO

Otra forma de escribir el algoritmo de búsqueda presentado anteriormente es la que se muestra a continuación. En esta variante se contempla el caso de que el árbol esté vacío.

Algoritmo 6.6 Búsqueda_v1_ABB

```
Búsqueda_v1_ABB (APNODO, INFOR)

{El algoritmo localiza el nodo del árbol binario de búsqueda que contiene cierta información
—INFOR—. Parámetro de tipo entero. APNODO es una variable de tipo ENLACE. La
primera vez, apunta a la raíz del árbol}

1. Si (APNODO ≠ NIL)
   entonces
   1.1 Si (INFOR < APNODO^.INFO)
       entonces
           Regresar a Búsqueda_v1_ABB con APNODO^.IZQ e INFOR
           {Llamada recursiva}
       si no
       1.1.1 Si (INFOR > NODO^.INFO)
           entonces
               Regresar a Búsqueda_v1_ABB con APNODO^.DER e INFOR
               {Llamada recursiva}
           si no
               Escribir "La información se encuentra en el árbol"
       1.1.2 {Fin del condicional del paso 1.1.1}
   1.2 {Fin del condicional del paso 1.1}
   si no
       Escribir "La información no se encuentra en el árbol"
2. {Fin del condicional del paso 1}
```

Inserción en un árbol binario de búsqueda

La **inserción en un árbol binario de búsqueda** es una operación que se puede realizar eficientemente en este tipo de estructura de datos. La estructura crece conforme se insertan elementos al árbol. Los pasos que se deben realizar para agregar un nuevo nodo a un árbol binario de búsqueda son los siguientes:

1. Comparar la clave a insertar con la raíz del árbol. Si es mayor, se sigue con el subárbol derecho. Si es menor, se continúa con el subárbol izquierdo.
2. Repetir sucesivamente el paso 1 hasta que se cumpla alguna de las siguientes condiciones:
   2.1 El subárbol derecho, o el subárbol izquierdo, es igual a vacío, en cuyo caso se procederá a insertar el elemento en el lugar que le corresponde.

2.2 La clave que se quiere insertar está en el nodo analizado, por lo tanto no se lleva a cabo la inserción. Este caso es válido sólo cuando la aplicación exige que no se repitan elementos.

Ejemplo 6.12

Supongamos que se quiere insertar las siguientes claves en un árbol binario de búsqueda que se encuentre vacío:

120 - 87 - 43 - 65 - 140 - 99 - 130 - 22 - 56

Los resultados parciales que ilustran cómo funciona el procedimiento se presentan en la figura 6.22.

...A 6.22

...en un árbol binario de búsqueda.

...as líneas en color indican el elemento que acaba de insertarse.

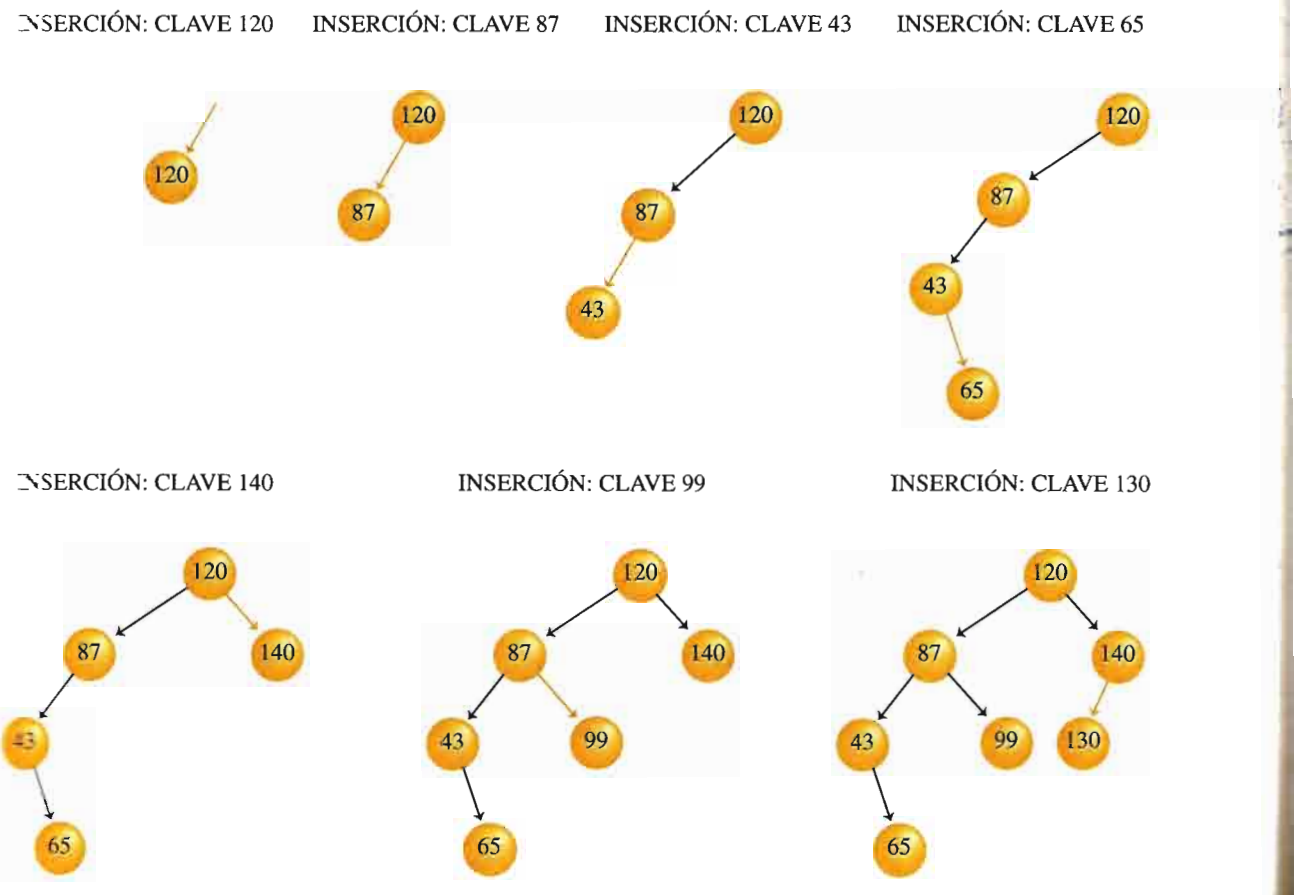
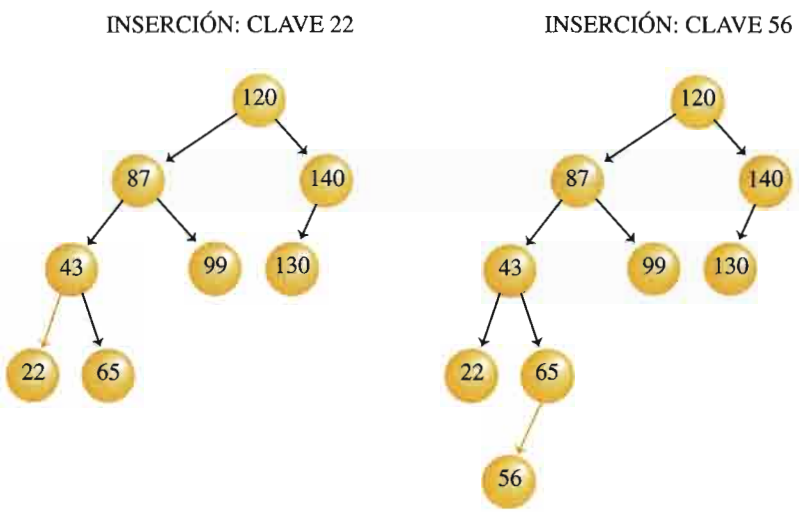


FIGURA 6.22
(*continuación*)



El algoritmo de inserción es el siguiente.

Algoritmo 6.7 Inserción_ABB

Inserción_ABB (APNODO, INFOR)

{El algoritmo realiza la inserción de un nodo en un árbol binario de búsqueda. APNODO es una variable de tipo puntero y la primera vez debe ser distinta de vacío. INFOR es un parámetro de tipo entero que contiene la información que se quiere insertar en un nuevo nodo}
{Se utiliza además, como auxiliar, la variable OTRO de tipo puntero}

```
1. Si (INFOR < APNODO^.INFO)
     entonces
   1.1 Si (APNODO^.IZQ = NIL)
         entonces
            Crear(OTRO) {Se crea un nuevo nodo}
            Hacer OTRO^.IZQ ← NIL, OTRO^.DER ← NIL, OTRO^.INFO ← INFOR
                  y APNODO^.IZQ ← OTRO
         si no
            Regresar a Inserción_ABB con APNODO^.IZQ e INFOR
            {Llamada recursiva}
   1.2 {Fin del condicional del paso 1.1}
     si no
   1.3 Si (INFOR > APNODO^.INFO)
         entonces
      1.3.1 Si (APNODO^.DER = NIL)
              entonces
                 Crear(OTRO) {Se crea un nuevo nodo}
                 Hacer OTRO^.IZQ ← NIL, OTRO^.DER ← NIL,
                       OTRO^.INFO ← INFOR y NODO^.DER ← OTRO
```

```
        si no
            Regresar a Inserción_ABB con NODO^.DER e INFOR
            {Llamada recursiva}
    1.3.2 {Fin del condicional del paso 1.3.1}
      si no
          Escribir "El nodo ya se encuentra en el árbol"
  1.4 {Fin del condicional del paso 1.3}
2. {Fin del condicional del paso 1}
```

Ejemplo 6.13

Supongamos que se desea insertar las claves 93 y 135 en el árbol binario de búsqueda de la figura 6.22. En las tablas 6.8 y 6.9 se presentan los pasos (P), número de comparaciones (C) y preguntas y acciones necesarias para insertar estas claves.

TABLA 6.8
de la clave 93
← 93)

P	C	Preguntas y acciones
	1	¿Es 93 < 120?
1		Sí. ¿Es el subárbol izquierdo de 120 (87) = NIL?
	2	No. Entonces regresar a Inserción con el subárbol izquierdo de 120 (87) e Infor
	3	¿Es 93 < 87?
2	4	No. ¿Es 93 > 87?
		Sí. ¿Es el subárbol derecho de 87 (99) = NIL?
	5	No. Entonces regresar a Inserción con el subárbol derecho de 87 (99) e Infor
	6	¿Es 93 < 99?
3		Sí. ¿Es el subárbol izquierdo de 99 (NIL) = NIL?
	7	Sí. Entonces crear otro nodo, realizar los enlaces y cargar la información

TABLA 6.9
de la clave 135
← 135)

P	C	Preguntas y acciones
	1	¿Es 135 < 120?
1	2	No. ¿Es 135 > 120?
		Sí. ¿Es el subárbol derecho de 120 (140) = NIL?
	3	Sí. Entonces regresar a Inserción con el subárbol derecho de 120 (140) e Infor
	4	No. ¿Es 135 > 140?
2		Sí. ¿Es el subárbol izquierdo de 140 (130) = NIL?
	5	No. Entonces regresar a Inserción con el subárbol izquierdo de 140 (130) e Infor
	6	¿Es 135 < 130?
3	7	No. ¿Es 135 > 130?
		Sí. ¿Es el subárbol derecho de 130 (NIL) = NIL?
	8	No. Entonces crear otro nodo, realizar los enlaces y cargar la información

En la figura 6.23 se presenta el árbol binario de búsqueda luego de realizada esta operación.

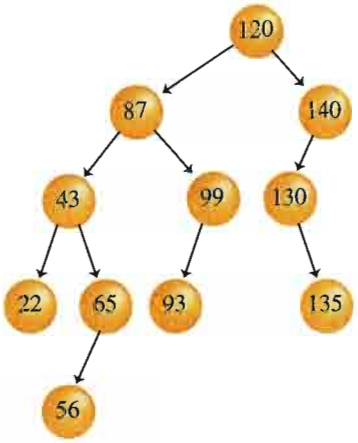


FIGURA 6.23
Árbol binario de búsqueda. Inserción de las claves 93 y 135.

Otra forma de expresar el algoritmo de inserción es la siguiente.

Algoritmo 6.8 Inserción_v1_ABB

Inserción_v1_ABB (APNODO, INFOR)

{El algoritmo realiza la inserción de un elemento en un árbol binario de búsqueda. APNODO es una variable de tipo ENLACE, la primera vez apunta a la raíz del árbol. INFOR es un parámetro de tipo entero que contiene la información del elemento que se quiere insertar. El algoritmo considera el caso de un árbol vacío}

```
1. Si (APNODO ≠ NIL)
     entonces
   1.1 Si (INFOR < APNODO^.INFO)
         entonces
           Regresar a Inserción_v1_ABB con APNODO^.IZQ e INFOR
           {Llamada recursiva}
         si no
       1.1.1 Si (INFOR > APNODO^.INFO)
             entonces
               Regresar a Inserción_v1_ABB con APNODO^.DER e INFOR
               {Llamada recursiva}
             si no
               Escribir "La información ya se encuentra en el árbol"
       1.1.2 {Fin del condicional del paso 1.1.1}
   1.2 {Fin del condicional del paso 1.1}
     si no
       Crear (OTRO) {Se crea un nuevo nodo}
       Hacer OTRO^.IZQ ← NIL, OTRO^.DER ← NIL, OTRO^.INFO ← INFOR
             y APNODO ← OTRO
2. {Fin del condicional del paso 1}
```

Eliminación en un árbol binario de búsqueda

La operación de **eliminación en un árbol binario de búsqueda** es un poco más complicada que la de inserción. Ésta consiste en eliminar un nodo sin violar los principios que definen un árbol binario de búsqueda. Se deben distinguir los siguientes casos:

1. Si el elemento a eliminar es terminal u hoja, simplemente se suprime redefiniendo el puntero de su predecesor.
2. Si el elemento a eliminar tiene un solo descendiente, entonces tiene que sustituirse por ese descendiente.
3. Si el elemento a eliminar tiene los dos descendientes, entonces se tiene que sustituir por el nodo que se encuentra más a la izquierda en el subárbol derecho o por el nodo que se encuentra más a la derecha en el subárbol izquierdo.

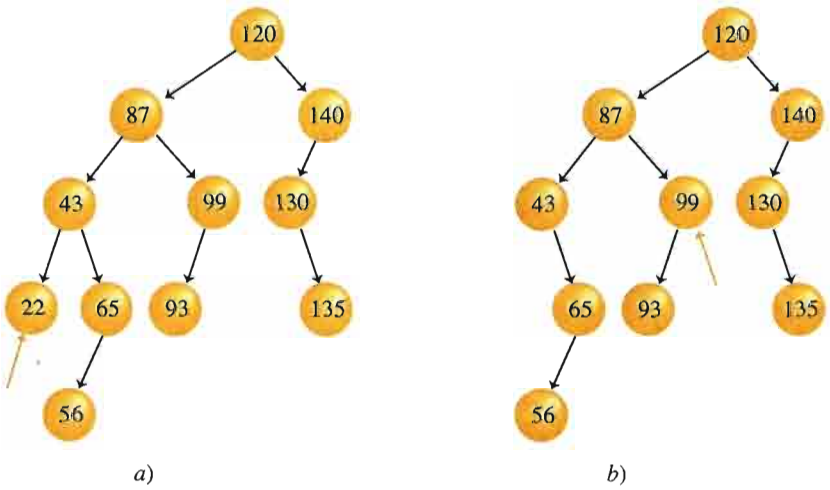
Cabe destacar que antes de eliminar un nodo, se debe localizar éste en el árbol. Para esto se utiliza el algoritmo de búsqueda presentado anteriormente.

Ejemplo 6.14

Supongamos que se desea eliminar las siguientes claves del árbol binario de búsqueda de la figura 6.23:

22 - 99 - 87 - 120 - 140 - 135 - 56

Los resultados parciales que ilustran cómo funciona el procedimiento se presentan en la figura 6.24.



a) *b)*

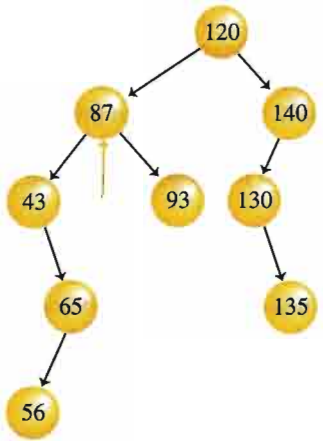
FIGURA 6.24
...ación en un árbol ... de búsqueda. *a)* y ... corresponden al primer ... *b)* y *e)* corresponden ... segundo caso; *c)* y *d)* co... ...den al tercer caso. ...tado final del árbol.

... las flechas en color ... el elemento que quiere ...

FIGURA 6.24

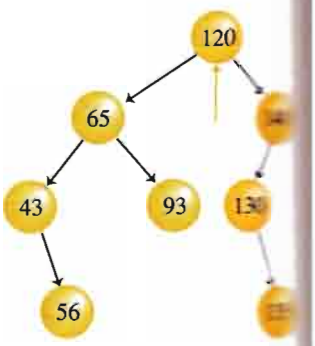
(*continuación*)

ELIMINACIÓN: CLAVE 87



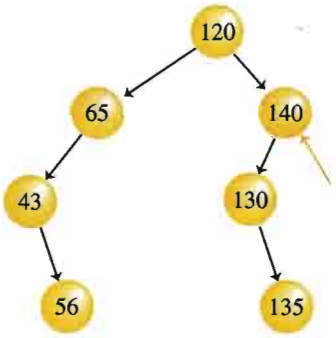
c)

ELIMINACIÓN: CLAVE



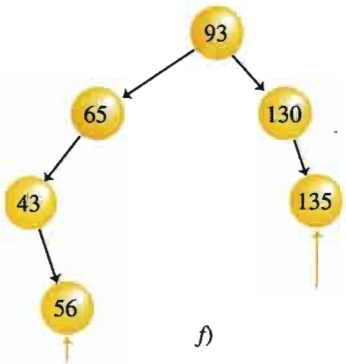
d)

ELIMINACIÓN: CLAVE 140

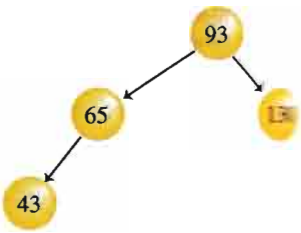


e)

ELIMINACIÓN: CLAVES 135 Y 56



f)



g)

A continuación se presenta el algoritmo correspondiente:

Algoritmo 6.9 Eliminación_ABB

Eliminación_ABB (APNODO, INFOR)

{El algoritmo realiza la eliminación de un elemento en un árbol binario de búsqueda. APNODO es una variable, por referencia, de tipo ENLACE. INFOR es un parámetro de tipo entero que contiene la información del nodo que se desea eliminar}
{AUX, AUXI y OTRO son variables auxiliares de tipo puntero. BO es una variable de tipo booleano}

```
1. Si (APNODO ≠ NIL)
     entonces
     1.1 Si (INFOR < APNODO^.INFO)
           entonces
              Regresar a Eliminación_ABB con APNODO^.IZQ e INFOR
           si no
        1.1.1 Si (INFOR > APNODO^.INFO)
              entonces
                 Regresar a Eliminación_ABB con APNODO^.DER e INFOR
              si no
                 Hacer OTRO ← NODO
              1.1.1.1 Si (OTRO^.DER = NIL)
                         entonces
                            Hacer APNODO ← OTRO^.IZQ
                         sino
                    1.1.1.1.1 Si (OTRO^.IZQ = NIL)
                                 entonces
                                    Hacer APNODO ← OTRO^.DER
                                 sino
                                    Hacer AUX ← APNODO^.IZQ y BO ← FALSO
                           1.1.1.1.1.A Mientras (AUX^.DER ≠ NIL) Repetir
                                          Hacer AUX1 ← AUX, AUX ← AUX^.DER
                                                y BO ← VERDADERO
                           1.1.1.1.1.B {Fin del ciclo del paso 1.1.1.1.1.A}
                                       Hacer APNODO^.INFO ← AUX^.INFO y
                                             OTRO ← AUX
                           1.1.1.1.1.C Si (BO = VERDADERO)
                                          entonces
                                             Hacer AUX1^.DER ← AUX^.IZQ
                                          sino
                                             Hacer APNODO^.IZQ ← AUX^.IZQ
                                 1.1.1.1.1.D {Fin del condicional del paso 1.1.1.1.1.C}
                              1.1.1.1.2 {Fin del condicional del paso 1.1.1.1.1}
                           1.1.1.2 {Fin del condicional del paso 1.1.1.1}
                                   Quitar (OTRO) {Se libera el espacio de memoria}
                     1.1.2 {Fin del condicional del paso 1.1.1}
               1.2 {Fin del condicional del paso 1.1}
               sino
                  Escribir "La información a eliminar no se encuentra en el árbol"
2.   {Fin del condicional del paso 1}
```

Ejemplo 6.15

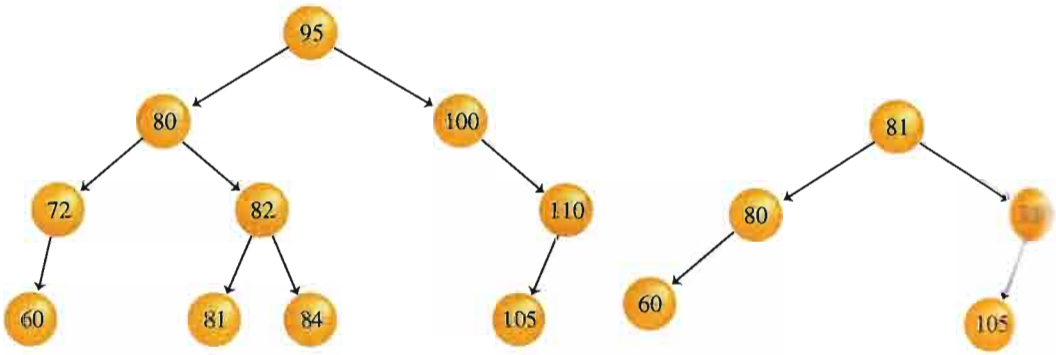
Dado el árbol de la figura 6.25*a*, verifique si queda igual al árbol de la figura 6.25*b* luego de eliminar, aplicando el algoritmo 6.9, las claves que se muestran a continuación:

95 - 72 - 84 - 100 - 82

FIGURA 6.25

Eliminación en un árbol binario de búsqueda. *a)* Antes de eliminar las claves. *b)* Después de eliminar las claves.

Nota: En el ejemplo, como el nodo a eliminar tiene dos descendientes, se sustituye por el nodo que se encuentra más a la derecha en el subárbol izquierdo.



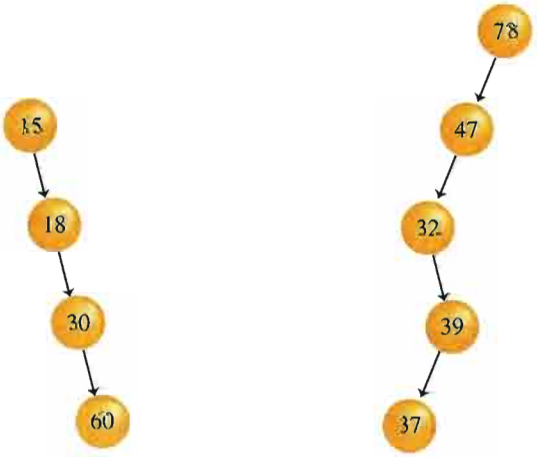
a) *b)*

6.4 ÁRBOLES BALANCEADOS

Cuando se estudiaron los árboles binarios de búsqueda se mencionó que es una estructura sobre la cual se pueden realizar eficientemente las operaciones de búsqueda, inserción y eliminación. Sin embargo, si el árbol crece o decrece descontroladamente, su rendimiento puede disminuir considerablemente. El caso más desfavorable se produce cuando se inserta un conjunto de claves ordenadas en forma ascendente o descendente, como se muestra en la figura 6.26.

FIGURA 6.26

Árboles binarios de búsqueda con crecimiento descontrolado.



Es de notar que el número promedio de comparaciones que se deben realizar para localizar una determinada clave en un árbol binario de búsqueda con crecimiento descontrolado es $N/2$, cifra que muestra un rendimiento muy pobre en la estructura.

Con el objeto de mantener la eficiencia en la operación de búsqueda surgen los **árboles balanceados**. La principal característica de éstos es la de realizar reacomodos o balanceos, después de inserciones o eliminaciones de elementos. Estos árboles también reciben el nombre de **AVL** en honor a sus inventores, dos matemáticos rusos, G. M. Adelson-Velskii y E. M. Landis.

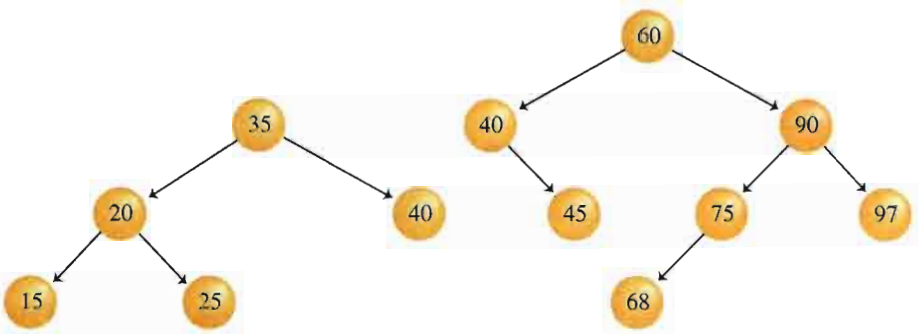
Formalmente se define un árbol balanceado como un árbol binario de búsqueda, en el cual se debe cumplir la siguiente condición: Para todo nodo *T* del árbol, la altura de los subárboles izquierdo y derecho no deben diferir en más de una unidad.

$$|H_{RI} - H_{RD}| \leq 1$$

donde H_{RI} es la altura de la rama o subárbol izquierdo y H_{RD} es la altura de la rama o subárbol derecho.

En la figura 6.27 se muestran dos ejemplos de árboles balanceados.

...URA 6.27
...boles balanceados. ...altura 3. *b*) Con ...



a) *b*)

Observe el lector que si se insertan las claves 10, 18 o 27 en el árbol balanceado de la figura 6.27*a*, éste pierde el equilibrio. Sin embargo, si se insertan las claves 37 o 45 en el mismo árbol, éste mantiene el equilibrio; más aún, lo mejora.

Ahora bien, con respecto a la eliminación, si a dicho árbol se le quitan las claves 15, 20 o 25 el árbol continúa siendo balanceado; incluso podrían quitársele las tres claves y el árbol no perdería el equilibrio. Sin embargo, si se elimina la clave 40 el árbol pierde el equilibrio y es necesario reestructurarlo.

Los árboles balanceados se parecen mucho, en su mecanismo de formación, a los números Fibonacci. El árbol de altura 0 es vacío, el árbol de altura 1 tiene un único nodo y, en general, el número de nodos del árbol con altura $h > 1$ se calcula aplicando la siguiente fórmula recursiva:

$$K_h = K_{h-1} + 1 + K_{h-2}$$ **Fórmula 6.2**

donde K indica el número de nodos del árbol y h la altura.

Por otra parte, algunos estudios demuestran que la altura de un árbol balanceado con n nodos nunca excederá de 1.44 * log n.

Ejemplo 6.16

Supongamos que se desea calcular el número de nodos de un árbol balanceado con altura 5. La forma en que se efectúa el cálculo es la siguiente:

$$\begin{aligned}
K_5 &= K_4 + 1 + K_3 \\
K_4 &= K_3 + 1 + K_2 \\
K_3 &= K_2 + 1 + K_1 \\
K_2 &= K_1 + 1 + K_0 \\
K_1 &= 1 \\
K_0 &= 0 \\
K_2 &= 2 \\
K_3 &= 4 \\
K_4 &= 7 \\
K_5 &= 12
\end{aligned}$$

6.4.1 Inserción en árboles balanceados

Al insertar un elemento en un árbol balanceado se deben distinguir los siguientes casos:

1. Las ramas izquierda (*RI*) y derecha (*RD*) del árbol tienen la misma altura ($H_{RI} = H_{RD}$), por lo tanto:
   - 1.1 Si se inserta un elemento en *RI*, entonces H_{RI} será mayor en una unidad a H_{RD}
   - 1.2 Si se inserta un elemento en *RD*, entonces H_{RD} será mayor en una unidad a H_{RI}

   Observe que en cualquiera de los dos casos mencionados (1.1 y 1.2), no se viola el criterio de equilibrio del árbol.

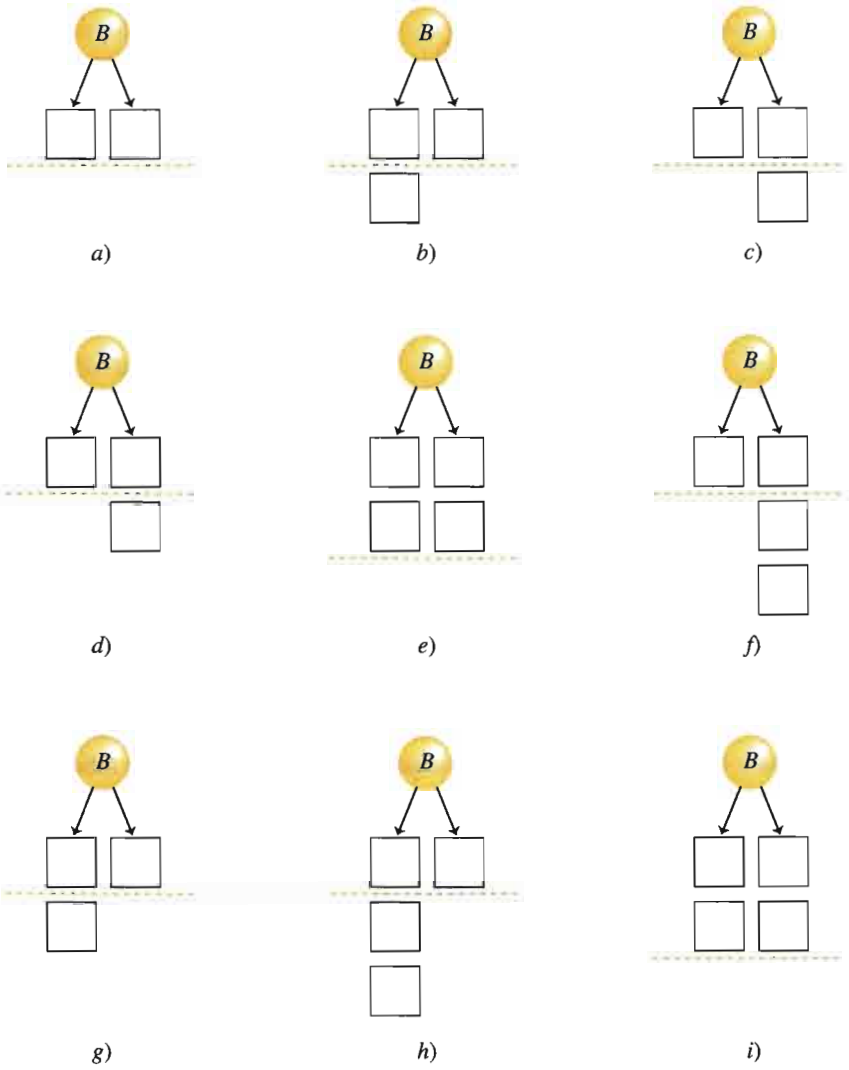
2. Las ramas izquierda (*RI*) y derecha (*RD*) del árbol tienen altura diferente ($H_{RI} \neq H_{RD}$):
   - 2.1 Supongamos que $H_{RI} < H_{RD}$:
     - 2.1.1 Si se inserta un elemento en *RI*, entonces H_{RI} será igual a H_{RD}. {Las ramas tienen la misma altura, por lo que se mejora el equilibrio del árbol}
     - 2.1.2 Si se inserta un elemento en *RD*, entonces se rompe el criterio de equilibrio del árbol y es necesario reestructurarlo.
   - 2.2 Supongamos que $H_{RI} > H_{RD}$:
     - 2.2.1 Si se inserta un elemento en *RI*, entonces se rompe el criterio de equilibrio del árbol y es necesario reestructurarlo.
     - 2.2.2 Si se inserta un elemento en *RD*, entonces H_{RD} será igual a H_{RI}. {Las ramas tienen la misma altura, por lo que se mejora el equilibrio del árbol}

En la figura 6.28 se muestran diagramas de los distintos casos que se presentan en la operación de inserción en árboles balanceados.

FIGURA 6.28

Diferentes casos de inserción en árboles balanceados. *a*) Caso 1. *b*) Caso 1.1. *c*) Caso 1.2. *d*) Caso 2.1. *e*) Caso 2.1.1. *f*) Caso 2.2. *g*) Caso 2.2. *h*) Caso 2.2.1. *i*) Caso 2.2.2.

La línea discontinua marca el equilibrio perfecto del árbol.



Ahora bien, para poder determinar si un árbol está balanceado o no, se debe manejar información relativa al equilibrio de cada nodo del árbol. Surge así el concepto de **factor de equilibrio** de un nodo (*FE*) que se define como la altura del subárbol derecho menos la altura del subárbol izquierdo.

$$FE = H_{RD} - H_{RI} \qquad \textbf{Fórmula 6.7}$$

Los valores que puede tomar *FE* son –1, 0, 1. Si *FE* llegara a tomar los valores –2 o 2, entonces debería reestructurarse el árbol. En la figura 6.29 se presenta un árbol balanceado con el correspondiente *FE* para cada nodo del árbol.

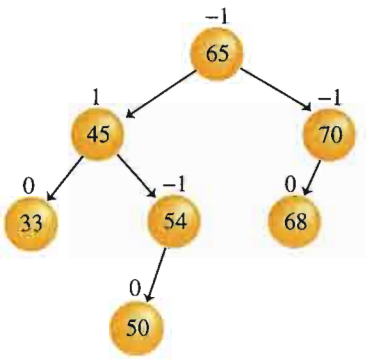


FIGURA 6.29
Árbol balanceado con el correspondiente *FE*.

Observe el lector que en la figura 6.29 el *FE* del nodo que almacena el 65 es –1 puesto que la altura del subárbol derecho es igual a 2 y la altura del subárbol izquierdo es igual a 3.

$$FE_{65} = 2 - 3 = -1$$

El *FE* de 45 se calcula como:

$$FE_{45} = 2 - 1 = 1$$

A continuación se presenta la definición de un árbol balanceado en lenguaje algorítmico.

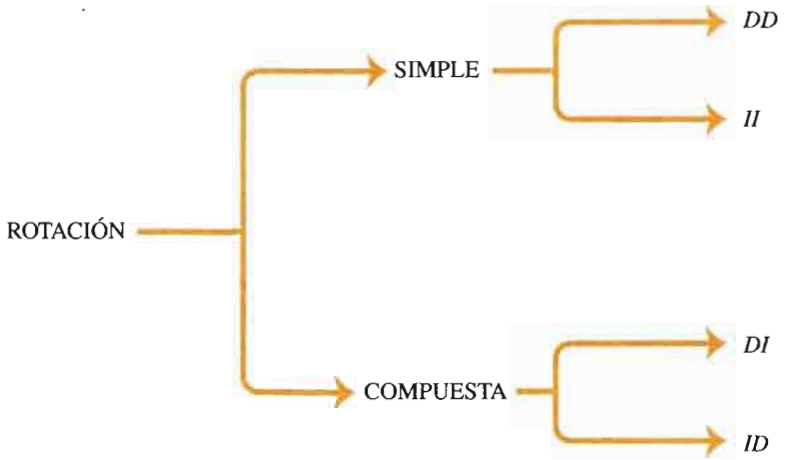
```
ENLACE = ^NODO
NODO   = REGISTRO
           IZQ, DER: tipo ENLACE
           INFO: tipo de dato
           FE: –1..1
{Fin de la definición}
```

6.4.2 Reestructuración del árbol balanceado

El proceso de inserción en un árbol balanceado es sencillo, sin embargo requiere de operaciones auxiliares que complican parcialmente el proceso. Primero se debe seguir el camino de búsqueda del árbol, hasta localizar el lugar donde hay que insertar el elemento. Luego se calcula su *FE*, que obviamente será 0, y regresamos por el camino de búsqueda calculando el *FE* de los distintos nodos visitados. Si en alguno de los nodos se viola el criterio de equilibrio entonces se debe reestructurar el árbol. El proceso termina

al llegar a la raíz del árbol, o cuando se realiza la reestructuración del mismo, en cuyo caso no es necesario determinar el *FE* de los nodos restantes.

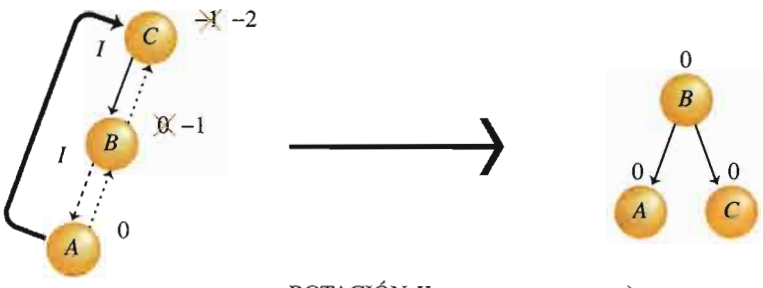
Reestructurar el árbol significa rotar los nodos del mismo para llevarlo a un estado de equilibrio. La rotación puede ser simple o compuesta. El primer caso involucra dos nodos y el segundo caso afecta a tres. Si la rotación es simple se puede realizar por las ramas derechas (***DD***) o por las ramas izquierdas (***II***). Si por otra parte la rotación es compuesta se puede realizar por las ramas derecha e izquierda (***DI***) o por las ramas izquierda y derecha (***ID***).



Ejemplo 6.17

En la figura 6.30 se pueden observar gráficamente los diferentes tipos de rotaciones. En la figura 6.30*a* se presenta la rotación *II*, en la figura 6.30*b* la rotación *DD*, en la figura 6.30*c* la rotación *DI* y en la figura 6.30*d* la rotación *ID*.

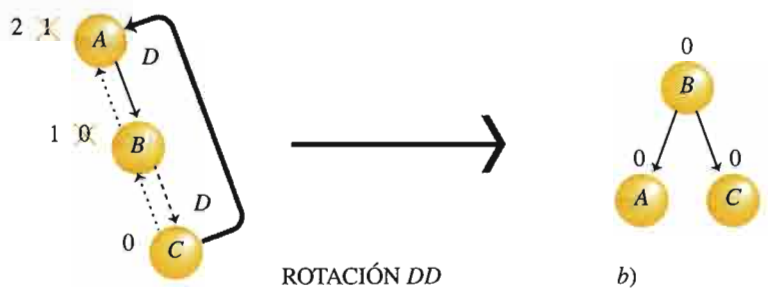
La línea continua (_____) marca el estado de los nodos del árbol antes de realizar la inserción. La línea discontinua (_ _ _ _) indica el nuevo elemento insertado. La línea con puntos (.....) marca el camino de regreso hasta que se detecta el desequilibrio del árbol. La línea gruesa (_____) indica el movimiento de los nodos en la rotación.

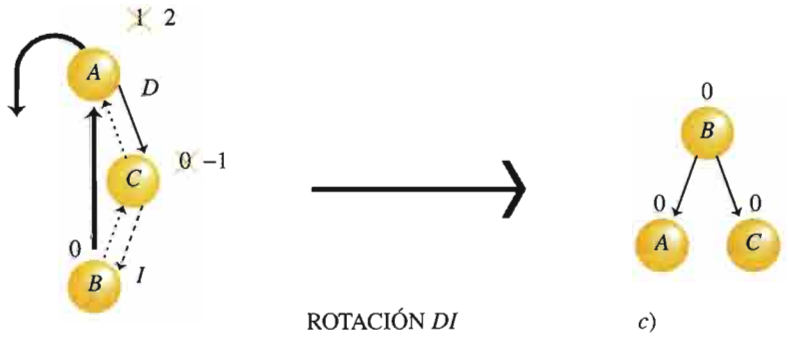


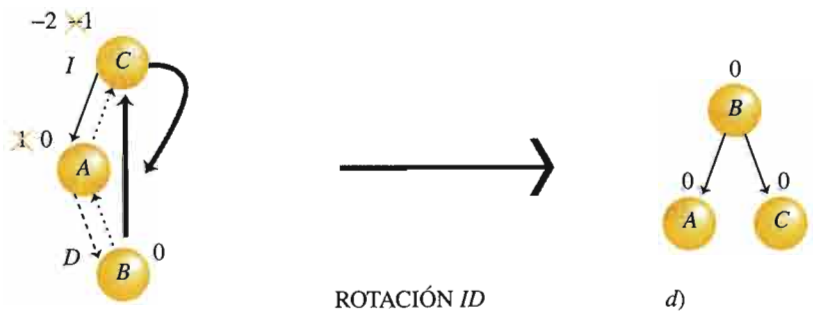
ROTACIÓN *II* *a*)

6.30
en árboles
a) Rotación *II*.
DD. *c*) Rotación
ID.

FIGURA 6.30
(*continuación*)







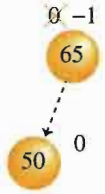
Ejemplo 6.18 Supongamos que se desea insertar las siguientes claves en un árbol balanceado que se encuentra vacío:

65 - 50 - 23 - 70 - 82 - 68 - 39

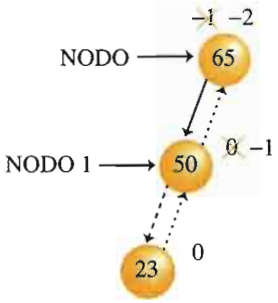
Las operaciones necesarias son las siguientes:

a) INSERCIÓN: CLAVE 65

b) INSERCIÓN: CLAVE 50



c) INSERCIÓN: CLAVE 23



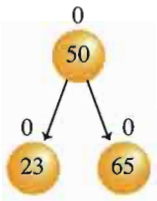
Al regresar, luego de insertar un nodo con el valor 23, siguiendo el camino de búsqueda se detecta que en la clave 65 se viola el criterio de equilibrio del árbol y se debe reestructurar. Se apunta con NODO la clave 65 y con NODO1 la rama izquierda de dicha clave. Luego se verifica el *FE* de NODO1; como en este caso es igual a –1 se puede realizar la rotación *II*. El movimiento de apuntadores para realizar la rotación *II* es el siguiente:

```
NODO^.IZQ  ← NODO1^.DER
NODO1^.DER ← NODO
NODO       ← NODO1
```

Respecto al *FE* de los nodos afectados, éste será siempre 0 en el caso de rotaciones simples.

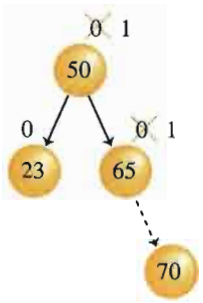
```
NODO^.FE  ← 0
NODO1^.FE ← 0
```

Luego de efectuar el reacomodo, el árbol queda así:

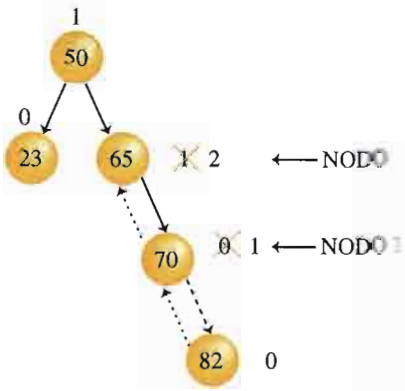


Al regresar siguiendo el camino de búsqueda se modifica el *FE* de los nodos 65 y 50, pero el equilibrio del árbol se mantiene.

d) INSERCIÓN: CLAVE 70



e) INSERCIÓN: CLAVE 82



ROTACIÓN *DD*

Al regresar, luego de insertar el valor 82, siguiendo el camino de búsqueda se observa una violación al criterio de equilibrio del árbol y se debe reestructurar. Se apunta con NODO la clave 65 y con NODO1 la rama derecha de dicha clave. Se verifica el *FE* de NODO1 y como en este caso es igual a 1 se puede realizar la rotación *DD*. El movimiento de apuntadores para realizar la rotación *DD* es:

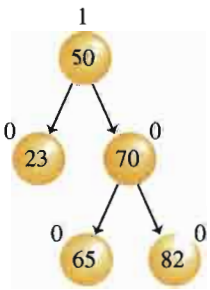
```
NODO^.DER  ← NODO1^.IZQ
NODO1^.IZQ ← NODO
NODO       ← NODO1
```

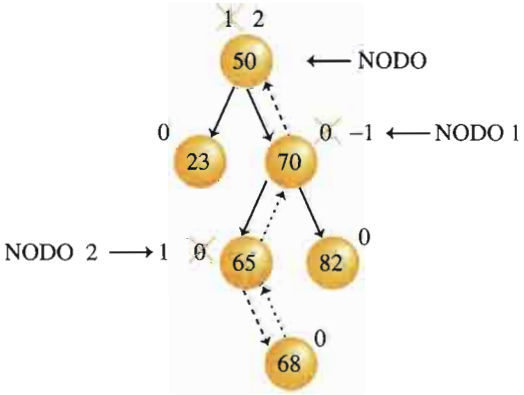
Respecto al *FE* de los nodos afectados, las asignaciones son las siguientes:

```
NODO^.FE  ← 0
NODO1^.FE ← 0
```

Luego de volver a equilibrarlo, el árbol queda de esta forma:

f) INSERCIÓN: CLAVE 68





ROTACIÓN *DI*

Luego de insertar la clave 68 y al regresar siguiendo el camino de búsqueda se advierte que en la clave 50 se rompe el equilibrio del árbol. Se apunta con NODO la clave 50 y con NODO1 su rama derecha. Se calcula el *FE* de NODO1 y como en este caso es igual a –1, se realiza la rotación *DI*. Se apunta entonces con NODO2 la rama izquierda de NODO1. El movimiento de apuntadores para realizar la rotación *DI* es:

```
NODO1^.IZQ  ← NODO2^.DER
NODO2^.DER  ← NODO1
NODO^.DER   ← NODO2^.IZQ
NODO2^.IZQ  ← NODO
NODO        ← NODO2
```

El *FE* de los nodos involucrados se asigna de acuerdo con los valores establecidos en la tabla 6.10.

TABLA 6.10
Factores de equilibrio en la rotación *DI*

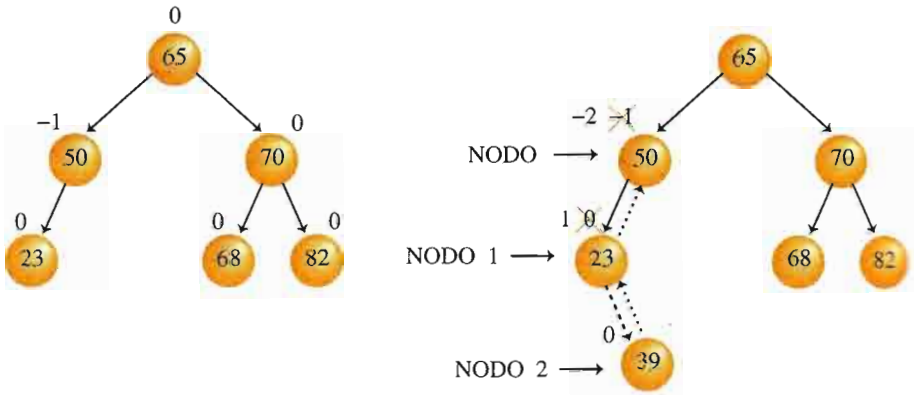
NODO2^.*FE* = –1	NODO^.*FE* = 0 NODO1^.*FE* = 1 NODO2^.*FE* = 0
NODO2^.*FE* = 0	NODO^.*FE* = 0 NODO1^.*FE* = 0 NODO2^.*FE* = 0
NODO2^.*FE* = 1	NODO^.*FE* = –1 NODO1^.*FE* = 0 NODO2^.*FE* = 0

Como en el ejemplo presentado el *FE* de NODO2 es igual a 1, se realizan las siguientes asignaciones:

```
NODO^.FE   ← –1
NODO1^.FE  ← 0
NODO2^.FE  ← 0
```

Luego de realizar el reacomodo, el árbol queda de la siguiente manera:

g) INSERCIÓN: CLAVE 39



ROTACIÓN *ID*

Regresando por el camino de búsqueda luego de insertar la clave 39, es evidente que en la clave 50 se rompe el equilibrio del árbol. Se apunta con NODO la clave 50 y con NODO1 su rama izquierda. Se verifica el *FE* de NODO1 y como en este caso es igual a 1, se realiza la rotación *ID*. Se apunta con NODO2 la rama derecha de NODO1. El movimiento de apuntadores para realizar la rotación *ID* es:

```
NODO1^.DER  ← NODO2^.IZQ
NODO2^.IZQ  ← NODO1
NODO^.IZQ   ← NODO2^.DER
NODO2^.DER  ← NODO
NODO        ← NODO2
```

El *FE* de los nodos involucrados se asigna de acuerdo con los valores establecidos en la tabla 6.11.

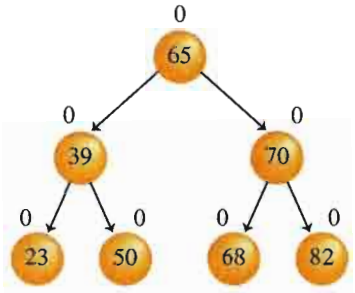
TABLA 6.11
Factores de equilibrio en la rotación *ID*

NODO2^.*FE* = −1	NODO^.*FE* = 1 NODO1^.*FE* = 0 NODO2^.*FE* = 0
NODO2^.*FE* = 0	NODO^.*FE* = 0 NODO1^.*FE* = 0 NODO2^.*FE* = 0
NODO2^.*FE* = 1	NODO^.*FE* = 0 NODO1^.*FE* = −1 NODO2^.*FE* = 0

Como en el ejemplo presentado el *FE* de NODO2 es igual a 0, se realizan las siguientes asignaciones:

NODO^.*FE* ← 0
NODO1^.*FE* ← 0
NODO2^.*FE* ← 0

Luego de volver a equilibrarlo, el árbol queda de la siguiente forma:



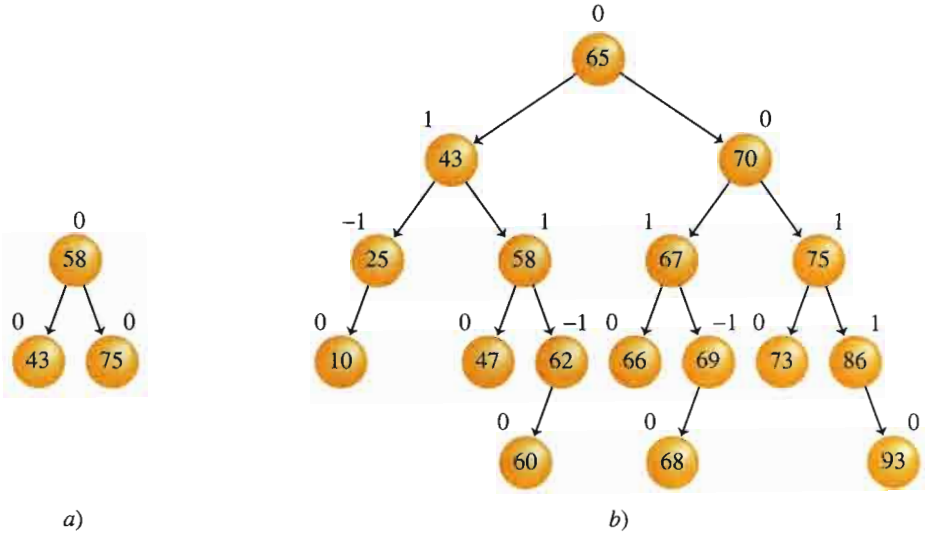
Nota: Observe que luego de realizar la inserción de un elemento y cuando se regresa por el camino de búsqueda, el *FE* del nodo visitado se incrementa en 1 si la inserción se hizo por su rama derecha y disminuye en 1 si la inserción se hizo por su rama izquierda.

Ejemplo 6.19

Dado como dato el árbol balanceado de la figura 6.31*a*, verifique si el mismo queda igual al de la figura 6.31*b* luego de insertar las siguientes claves:

86 - 65 - 70 – 67 - 73 - 93 - 69 - 25 - 66 - 68 - 47 - 62 - 10 - 60

FIGURA 6.31
Inserción en árboles balanceados. *a)* Antes de insertar las claves. *b)* Después de insertar las claves.



a) *b)*

A continuación se presenta el algoritmo de inserción en árboles balanceados.

Algoritmo 6.10 Inserta_balanceado

Inserta_balanceado (NODO, BO, INFOR)

```
{El algoritmo inserta un elemento en un árbol balanceado. NODO es un parámetro de tipo
puntero, por referencia. BO es un parámetro de tipo booleano, por referencia. BO se utiliza para
indicar que la altura del árbol ha crecido, su valor inicial es FALSO. INFOR es un parámetro
de tipo entero que contiene la información del elemento que queremos insertar}
{OTRO, NODO1 y NODO2 son variables auxiliares de tipo puntero}
1. Si (NODO ≠ NIL)
   entonces
1.1 Si (INFOR < NODO^.INFO)
      entonces
         Regresar a Inserta_balanceado con NODO^.IZQ, BO e INFOR
         {Llamada recursiva}
1.1.1 Si (BO = VERDADERO)
         entonces
1.1.1.1 Si (NODO^.FE)
            = 1: Hacer NODO^.FE ← 0 y BO ← FALSO
            = 0: Hacer NODO^.FE ← –1
            = –1: Hacer NODO1 ← NODO^.IZQ
            {Reestructuración del árbol}
1.1.1.1.1 Si (NODO1^.FE ≤ 0)
               entonces {Rotación II}
                  Hacer NODO^.IZQ ← NODO1^.DER,
                        NODO1^.DER, ← NODO,
                        NODO^.FE ← 0 y NODO ← NODO1
                  {Termina la rotación II}
               si no {Rotación ID}
                  Hacer NODO2 ← NODO1^.DER,
                        NODO^.IZQ ← NODO2^.DER,
                        NODO2^.DER ← NODO,
                        NODO1^.DER ← NODO2^.IZQ y
                        NODO2^.IZQ ← NODO1
1.1.1.1.1.A Si (NODO2^.FE = –1)
                  entonces
                     Hacer NODO^.FE ← 1
                  si no
                     Hacer NODO^.FE ← 0
1.1.1.1.1.B {Fin del condicional del paso 1.1.1.1.1.A}
1.1.1.1.1.C Si (NODO2^.FE = 1)
                  entonces
                     Hacer NODO1^.FE ← –1
                  si no
                     Hacer NODO1^.FE ← 0
1.1.1.1.1.D {Fin del condicional del paso 1.1.1.1.1.C}
               Hacer NODO ← NODO2
               {Termina la rotación ID}
```

```
1.1.1.1.2 {Fin del condicional del paso 1.1.1.1.1}
          Hacer NODO^.FE ← 0 y BO ← FALSO
1.1.1.2 {Fin del condicional del paso 1.1.1.1}
1.1.2 {Fin del condicional del paso 1.1.1}
si no
1.1.3 Si (INFOR > NODO^.INFO)
      entonces
         Regresar a Inserta_balanceado con NODO^.DER, BO e INFOR
         {Llamada recursiva}
1.1.3.1 Si (BO = VERDADERO)
        entonces
1.1.3.1.1 Si (NODO^.FE)
                = -1: Hacer NODO^.FE ← 0 y BO ← FALSO
                = 0: Hacer NODO1^.FE ← 1
                = 1: Hacer NODO1 ← NODO^.DER
                {Reestructuración de árbol}
1.1.3.1.1.A Si (NODO1^.FE ≥ 0)
            entonces {Rotación DD}
               Hacer NODO^.DER ← NODO1^.IZQ,
                     NODO1^.IZQ ← NODO,
                     NODO^.FE ← 0 y NODO ← NODO1
               {Termina la rotación DD}
            si no {Rotación DI}
               Hacer NODO2 ← NODO1^.IZQ,
                     NODO^.DER ← NODO2^.IZQ
                     NODO2^.IZQ ← NODO,
                     NODO1^.IZQ ← NODO2^.DER y
                     NODO2^.DER ← NODO1
               Si (NODO2^.FE = 1)
                  entonces
                     Hacer NODO^.FE ← -1
                  si no
                     Hacer NODO^.FE ← 0
               {Fin del condicional interno}
               Si (NODO2^.FE = -1)
                  entonces
                     Hacer NODO1^.FE ← 1
                  si no
                     Hacer NODO1^.FE ← 0
                  {Fin del condicional interno}
                  Hacer NODO ← NODO2
                  {Termina la rotación DI}
1.1.3.1.1.B {Fin del condicional del paso 1.1.3.1.1.A}
            Hacer NODO^.FE ← 0 y BO ← FALSO
1.1.3.1.2 {Fin del condicional del paso 1.1.3.1.1}
1.1.3.2 {Fin del condicional del paso 1.1.3.1}
      si no
         Escribir "La información ya se encuentra en el árbol"
1.1.4 {Fin del condicional del paso 1.1.3}
```

```
1.2 {Fin del condicional del paso 1.1}
    si no
        Crear (NODO)
        Hacer NODO^.INFO ← INFOR, NODO^.IZQ ← NIL, NODO^.DER ← NIL,
              NODO^.FE ← 0 y BO ← VERDADERO
2. {Fin del condicional del paso 1}
```

Eliminación en árboles balanceados

La operación de **eliminación en árboles balanceados** es más compleja que la operación de inserción, como normalmente ocurre en casi todas las estructuras de datos. Consiste en quitar un nodo del árbol sin violar los principios que definen un árbol balanceado. Recuerde que se definió como una estructura en la cual, para todo nodo del árbol, se debe cumplir que: **la altura del subárbol izquierdo y la altura del subárbol derecho no deben diferir en más de una unidad.**

Eliminar nodos en un árbol balanceado resulta difícil a pesar de que se utiliza el mismo algoritmo de eliminación, idéntico en lógica pero diferente en implementación, que en los árboles binarios de búsqueda y las mismas operaciones de reacomodo que se utilizan en el algoritmo de inserción en árboles balanceados.

En la operación de eliminación en árboles balanceados se deben distinguir los siguientes casos:

1. Si el elemento a eliminar es terminal u hoja, simplemente se suprime.
2. Si el elemento a eliminar tiene un solo descendiente, entonces se tiene que sustituir por ese descendiente.
3. Si el elemento a eliminar tiene los dos descendientes, entonces se tiene que sustituir por el nodo que se encuentra más a la izquierda en el subárbol derecho o por el nodo que se encuentra más a la derecha en el subárbol izquierdo.

Para eliminar un nodo en un árbol balanceado lo primero que se debe hacer es localizar su posición en el árbol. Se elimina siguiendo los criterios establecidos anteriormente y se regresa por el camino de búsqueda calculando el *FE* de los nodos visitados. Si en alguno de los nodos se viola el criterio de equilibrio, entonces se debe reestructurar el árbol. El proceso termina cuando se llega a la raíz del árbol. Cabe aclarar que mientras que en el algoritmo de inserción una vez efectuada una rotación se podía detener el proceso, en este algoritmo se debe continuar puesto que se puede producir más de una rotación en el camino hacia atrás. Para comprender mejor la operación de eliminación en árboles balanceados, observe el siguiente ejemplo.

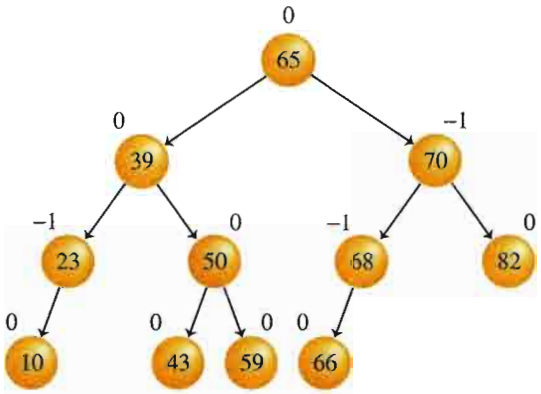
Ejemplo 6.20

Supongamos que se desea eliminar las siguientes claves del árbol balanceado de la figura 6.32:

82 - 10 - 39 - 65 - 70 - 23 - 50 - 59

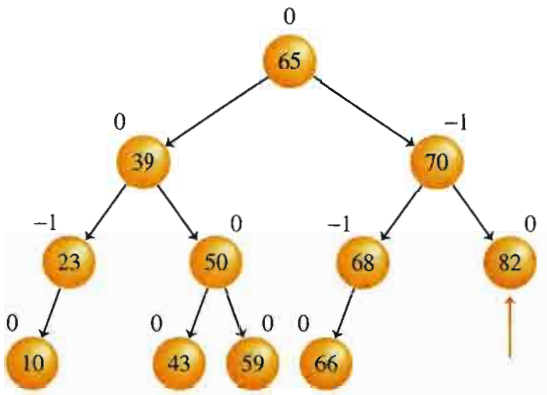
Cabe destacar que el movimiento de apuntadores y la reasignación de los *FE* no se presentarán en este ejemplo, porque son idénticos a los mostrados en el ejemplo 6.18.

FIGURA 6.32
Árbol balanceado.

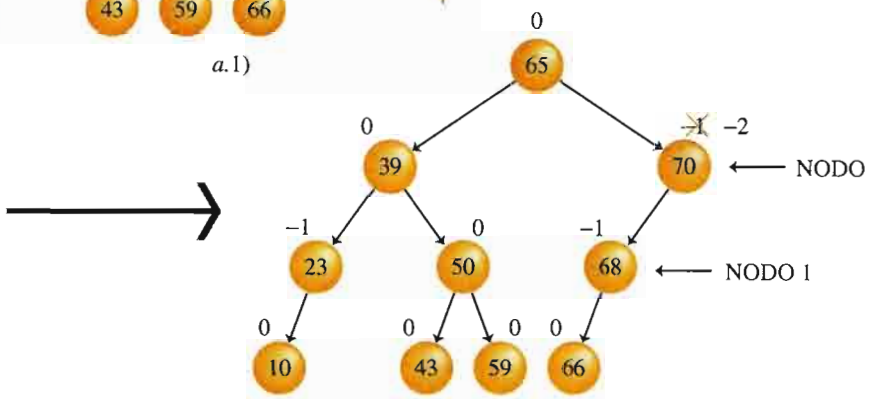


Las operaciones que se realizan son las siguientes:

a) ELIMINACIÓN: CLAVE 82

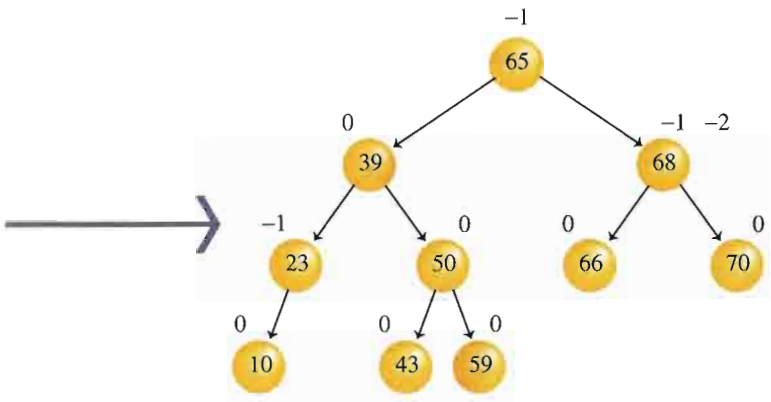


a.1)



a.2)

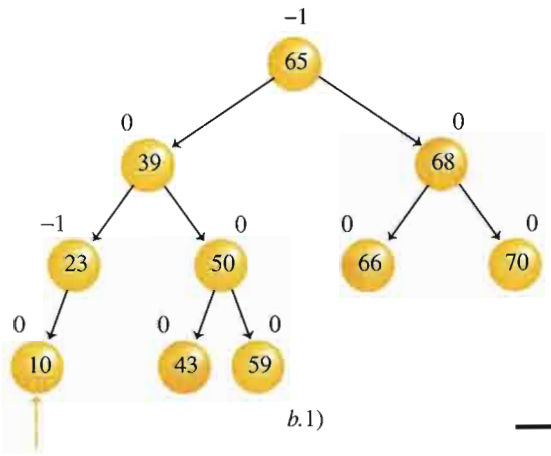
ROTACIÓN *II*



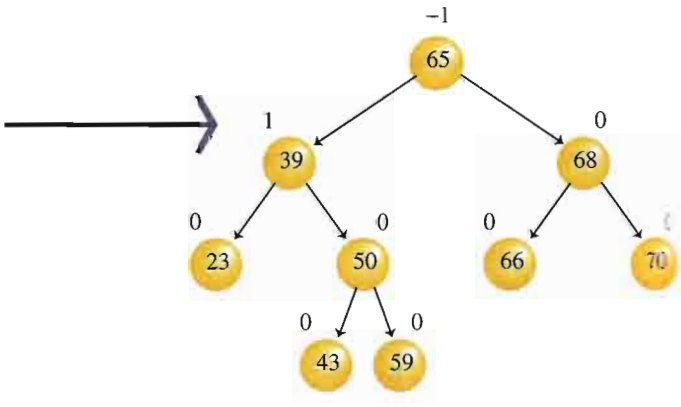
a.3)

La eliminación de la clave 82 es un proceso sencillo, ya que dicha clave no tiene descendientes (diagrama *a*.1). Al regresar siguiendo el camino de búsqueda es evidente que en la clave 70 se rompe el criterio de equilibrio y se debe reestructurar el árbol (diagrama *a*.2). Se apunta con NODO la clave 70 y con NODO1 la rama izquierda de NODO. Se verifica el *FE* de NODO1 y como éste es igual a –1, entonces se realiza la rotación *II*. Luego de la reestructuración, el árbol queda como en el diagrama *a*.3.

b) ELIMINACIÓN: CLAVE 10



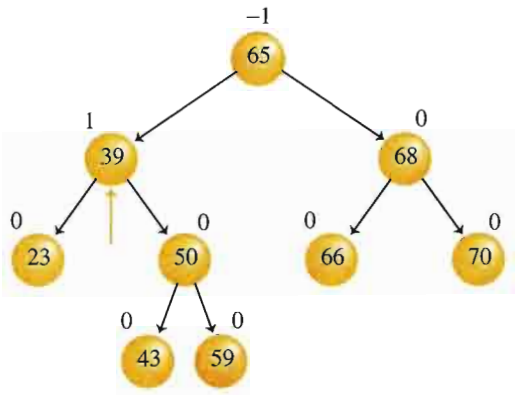
b.1)



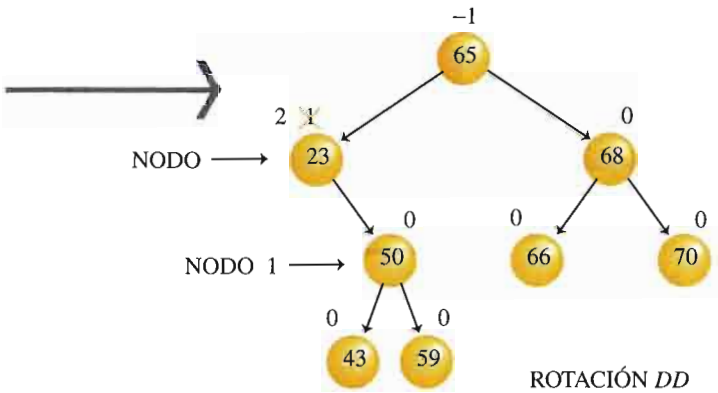
b.2)

La eliminación de la clave 10 es un proceso sencillo (diagrama *b*.1). No se debe reestructurar el árbol porque mantiene el equilibrio y sólo es necesario cambiar el *FE* de los nodos que almacenan al 23 y 39 (diagrama *b*.2).

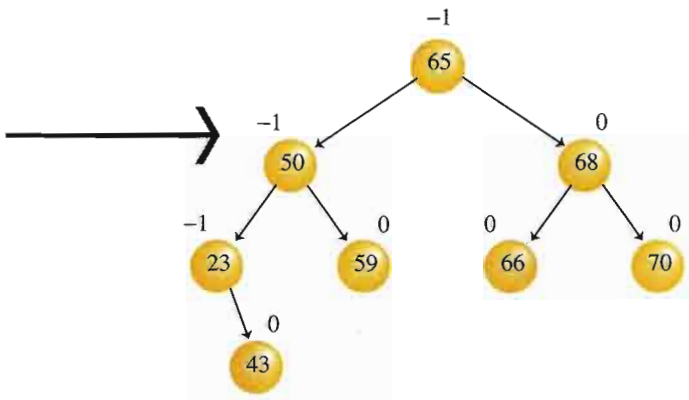
c) ELIMINACIÓN: CLAVE 39



c.1)



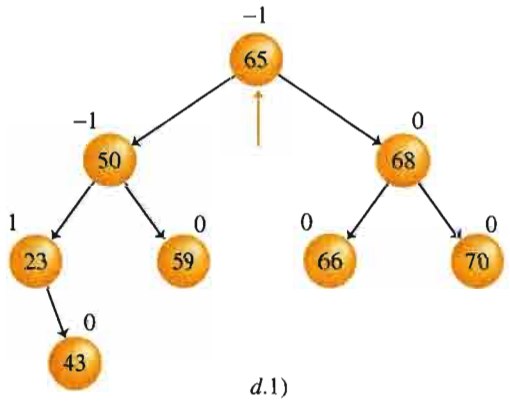
c.2)



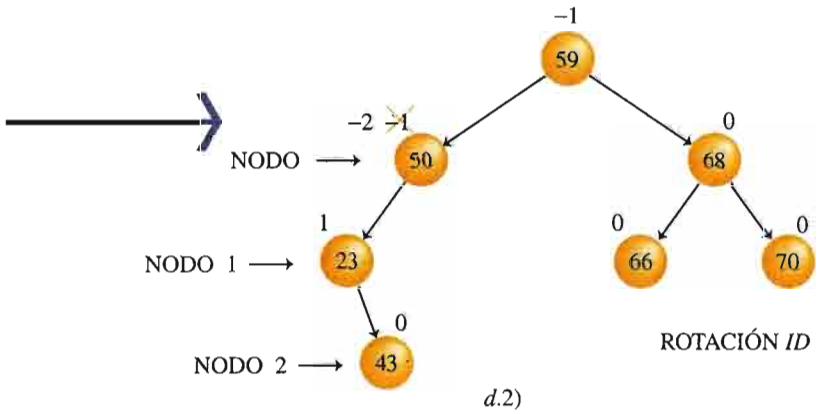
c.3)

Al eliminar la clave 39 se origina el caso más difícil de eliminación en árboles: la eliminación de una clave con dos descendientes (diagrama *c*.1). En este caso se opta por sustituir dicha clave por el nodo que se encuentra más a la derecha en el subárbol izquierdo (23). Luego de la sustitución, el árbol queda como se muestra en el diagrama *c*.2. Al realizar la sustitución se observa que en dicho nodo se viola el criterio de equilibrio y se debe reestructurar el árbol. Se apunta con NODO la clave 23 y con NODO1 la rama derecha de NODO. Se verifica el *FE* de NODO1 y como éste es igual a 0, se realiza la rotación *DD*. Luego del reacomodo, el árbol queda como en el diagrama *c*.3

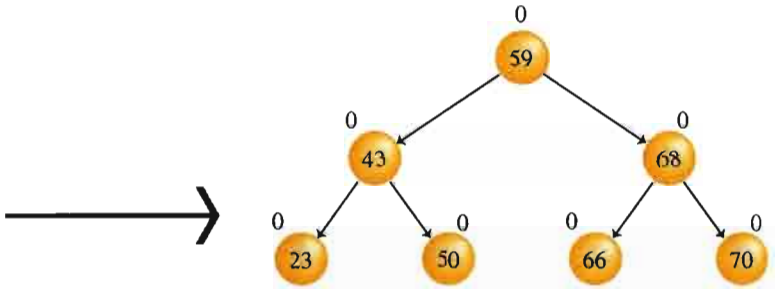
d) ELIMINACIÓN: CLAVE 65



d.1)



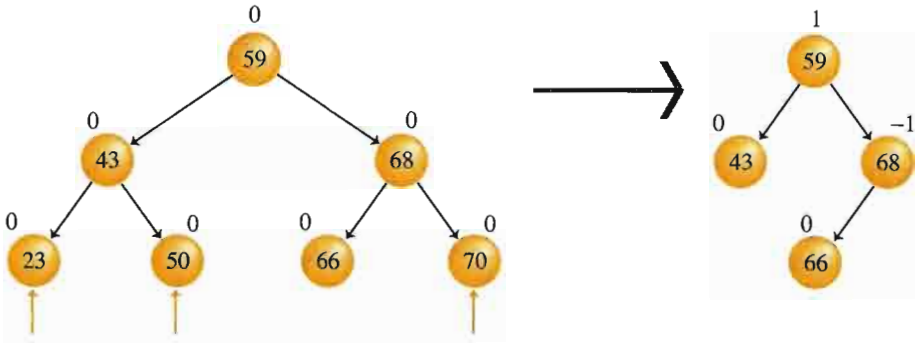
d.2)



d.3)

Al eliminar la clave 65 surge nuevamente el tercer caso de eliminación, que corresponde a una clave con dos descendientes (diagrama *d*.1). Se sustituye dicha clave por el nodo que se encuentra más a la derecha en el subárbol izquierdo (59). Luego de la sustitución, el árbol queda como se presenta en el diagrama *d*.2. Es evidente que, después de la sustitución, en el nodo con la clave 50 se viola el criterio de equilibrio y se debe reestructurar el árbol. Se apunta con NODO la clave 50 y con NODO1 la rama izquierda de NODO y se verifica el *FE*. Como en este caso es igual a 1, se apunta con NODO2 la rama derecha de NODO1 y se realiza la rotación *ID*. Luego de la reestructuración, el árbol queda como el presentado en el diagrama (*d*.3).

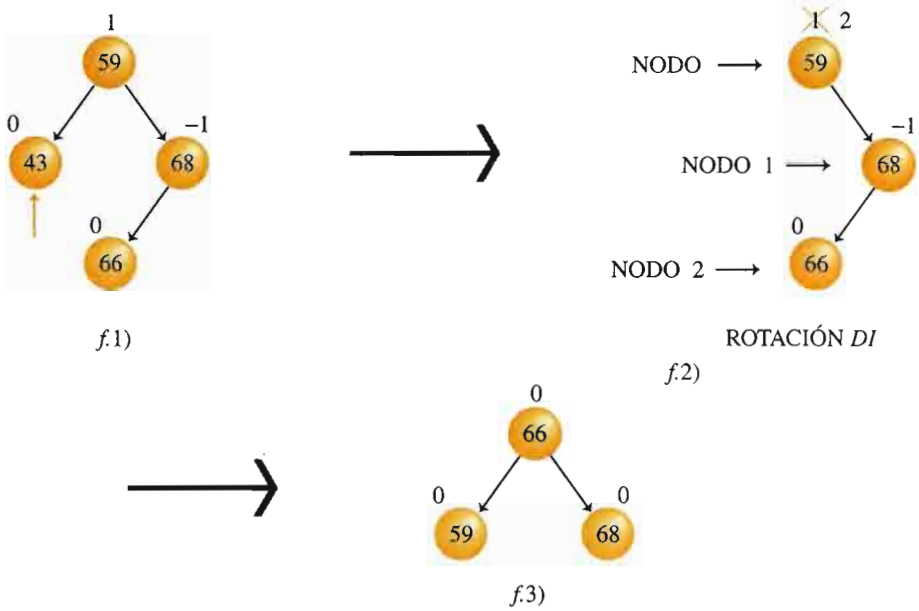
e) ELIMINACIÓN: CLAVES 70, 23 Y 50



e.1) *e*.2)

Luego de la eliminación de las claves, el árbol queda como en el diagrama *e*.2.

f) ELIMINACIÓN: CLAVE 43



f.1) ROTACIÓN *DI*

f.2)

f.3)

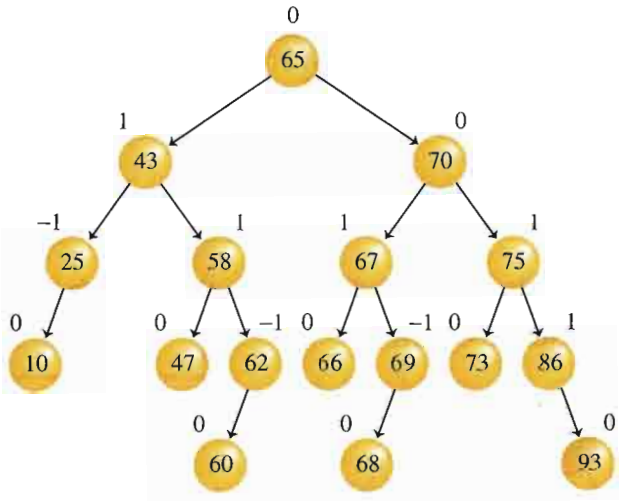
La eliminación de la clave 43 corresponde al primer caso de borrado en árboles, es el caso más simple (diagrama *f*.1). Sin embargo, al verificar el *FE* de la clave 59 se advierte que se rompe el equilibrio del árbol y se debe reestructurar (diagrama *f*.2). Se apunta con NODO la clave 59 y con NODO1 la rama derecha de NODO, y se verifica el *FE* de NODO1. Como éste es igual a –1, se apunta con NODO2 la rama izquierda de NODO1 y se realiza la rotación *DI*. Luego de la reestructuración, el árbol queda como en el diagrama *f*.3.

Observe cuidadosamente que luego de realizar la eliminación de un elemento y cuando se regresa por el camino de búsqueda, el *FE* del nodo visitado disminuye en 1 si la eliminación se hizo por su rama derecha y se incrementa en 1 si la eliminación se hizo por su rama izquierda.

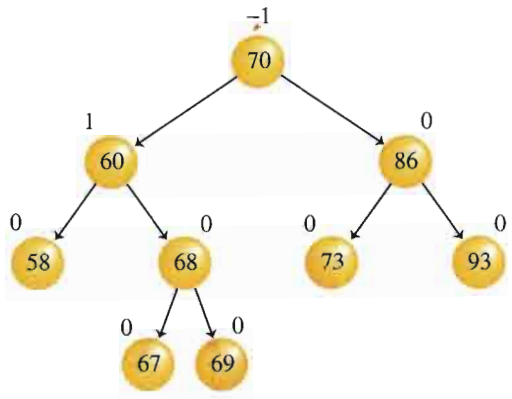
Ejemplo 6.21

Dado el árbol balanceado de la figura 6.33*a*, verifique si el mismo queda igual al de la figura 6.33*b* luego de eliminar las siguientes claves:

25 - 75 - 66 - 65 - 62 - 10 - 43 - 47



a)



b)

FIGURA 6.33
Eliminación en árboles balanceados. *a*) Antes de eliminar las claves. *b*) Después de eliminar las claves.

Con el fin de darle mayor modularidad al algoritmo de eliminación en árboles balanceados, se estudiarán dos algoritmos auxiliares. El primero, **Reestructura_izq**, se utiliza cuando la altura de la rama izquierda ha disminuido. El segundo, **Reestructura_der**, se emplea cuando la altura de la rama derecha ha disminuido.

Algoritmo 6.11 Reestructura_izq

Reestructura_izq (NODO, BO)

{Este algoritmo reestructura el árbol cuando la altura de la rama izquierda ha disminuido y el *FE* de NODO es igual a 1. NODO es un parámetro por referencia de tipo puntero. BO es un parámetro de tipo booleano, también por referencia. BO se utiliza para indicar que la altura de la rama izquierda ha disminuido}
{NODO1 y NODO2 son variables auxiliares de tipo puntero}

```
1. Si (BO = VERDADERO)
     entonces
   1.1 Si (NODO^.FE)
         = –1: Hacer NODO^.FE ← 0
         = 0: Hacer NODO^.FE ← 1 y BO ← FALSO
         = 1: {Reestructuración del árbol}
              Hacer NODO1 ← NODO^.DER
       1.1.1 Si (NODO1^.FE ≥ 0)
               entonces {Rotación DD}
                 Hacer NODO^.DER ← NODO1^.IZQ y NODO1^.IZQ ← NODO
         1.1.1.1 Si NODO1^.FE
                   = 0: Hacer NODO^.FE ← 1, NODO1^.FE ← –1 y
                        BO ← FALSO
                   = 1: Hacer NODO^.FE ← 0 y NODO1^.FE ← 0
         1.1.1.2 {Fin del condicional 1.1.1.1}
                 Hacer NODO ← NODO1
                 {Termina la rotación DD}
               si no {Rotación DI}
                 Hacer NODO2 ← NODO1^.IZQ, NODO^.DER ← NODO2^.IZQ,
                   NODO2^.IZQ ← NODO, NODO1^.IZQ ← NODO2^.DER y
                   NODO2^.DER ← NODO1
         1.1.1.3 Si (NODO2^.FE = 1)
                   entonces
                     Hacer NODO^.FE ← –1
                   si no
                     Hacer NODO^.FE ← 0
         1.1.1.4 {Fin del condicional 1.1.1.3}
         1.1.1.5 Si (NODO2^.FE = –1)
                   entonces
                     Hacer NODO1^.FE ← 1
                   si no
                     Hacer NODO1^.FE ← 0
```

```
1.1.1.6 {Fin del condicional 1.1.1.5}
        Hacer NODO ← NODO2 y NODO2^.FE ← 0
        {Termina la rotación DI}
  1.1.2 {Fin del condicional del paso 1.1.1}
1.2 {Fin del condicional del paso 1.1}
2. {Fin del condicional del paso 1}
```

Algoritmo 6.12 Reestructura_der

```
Reestructura_der

{Este algoritmo reestructura el árbol cuando la altura de la rama derecha ha disminuido y el
FE de NODO es igual a –1. NODO es un parámetro, por referencia, de tipo puntero. BO es un
parámetro de tipo booleano, también por referencia. BO se utiliza para indicar que la altura de
la rama derecha ha disminuido}
{NODO1 y NODO2 son variables auxiliares de tipo puntero}

1. Si (BO = VERDADERO) entonces
  1.1 Si NODO^.FE
        = 1 : Hacer NODO^.FE ← 0
        = 0 : Hacer NODO^.FE ← –1 y BO ← FALSO
        = –1: {Reestructuración del árbol}
              Hacer NODO1 ← NODO^.IZQ
      1.1.1 Si (NODO1^.FE ≤ 0)
            entonces {Rotación II)
                Hacer NODO^.IZQ ← NODO1^.DER y NODO1^.DER ← NODO
          1.1.1.1 Si NODO1^.FE
                = 0: Hacer NODO^.FE ← –1, NODO1^.FE ← 1 y
                        BO ← FALSO
                = –1: Hacer NODO^.FE ← 0 y NODO1^.FE ← 0
          1.1.1.2 {Fin del condicional del paso 1.1.1.1}
                Hacer NODO ← NODO1
                {Termina la rotación II}
            si no {Rotación ID)
                Hacer NODO2 ← NODO1^.DER, NODO^.IZQ ← NODO2^.DER,
                    NODO2^.DER ← NODO, NODO1^.DER ← NODO2^.IZQ y
                    NODO2^.IZQ ← NODO1
          1.1.1.3 Si (NODO2^.FE = –1)
                entonces
                    Hacer NODO^.FE ← 1
                si no
                    Hacer NODO^.FE ← 0
          1.1.1.4 {Fin del condicional del paso 1.1.1.3}
          1.1.1.5 Si (NODO2^.FE = 1)
                entonces
                    Hacer NODO1^.FE ← –1
```

```
                si no
                    Hacer NODO1^.FE ← 0
            1.1.1.6 {Fin del condicional del paso 1.1.1.5}
                Hacer NODO ← NODO2 y NODO2^.FE ← 0
                {Termina la rotación ID)
        1.1.2 {Fin del condicional del paso 1.1.1}
    1.2 {Fin del condicional del paso 1.1}
2. {Fin del condicional del paso 1}
```

A continuación se presenta el algoritmo de eliminación en árboles balanceados, el cual hará uso de los previamente explicados.

Algoritmo 6.13 Elimina_balanceado

Elimina_balanceado (NODO, BO, INFOR)

```
{El algoritmo elimina un elemento en un árbol balanceado. Utiliza dos algoritmos auxiliares
Reestructura_izq y Reestructura_der. NODO es un parámetro por referencia de tipo puntero.
BO es un parámetro de tipo booleano, también por referencia, y se utiliza para indicar que la
altura del árbol ha disminuido, su valor inicial es FALSO. INFOR es un parámetro de tipo
entero que contiene la información del elemento que se quiere eliminar}
{OTRO, AUX, AUX1 son variables auxiliares de tipo puntero. BOOL es una variable de tipo
booleano}
1. Si (NODO ≠ NIL)
   entonces
   1.1 Si (INFOR < NODO^.INFO)
       entonces
           Regresar a Elimina_balanceado con NODO^.IZQ, BO e INFOR
           Llamar al algoritmo Reestructura_izq con NODO y BO
       si no
       1.1.1 Si (INFOR > NODO^.INFO)
             entonces
                 Regresar a Elimina_balanceado con NODO^.DER, BO e INFOR
                 Llamar al algoritmo Reestructura_der con NODO y BO
             si no
                 Hacer OTRO ← NODO y BO ← VERDADERO
             1.1.1.1 Si (OTRO^.DER = NIL)
                     entonces
                         Hacer NODO ← OTRO^.IZQ
                     si no
                   1.1.1.1.1 Si (OTRO^.IZQ = NIL)
                             entonces
                                 Hacer NODO ← OTRO^.DER
                             si no
```

```
                    Hacer AUX ← NODO^.IZQ y BOOL ← FALSO
1.1.1.1.1.A Mientras (AUX^.DER ≠ NIL) Repetir
                        Hacer AUX1 ← AUX, AUX ← AUX^.DER
                          y BOOL ← VERDADERO
1.1.1.1.1.B {Fin del ciclo del paso 1.1.1.1.1.A}
                        Hacer NODO^.INFO ← AUX^.INFO y
                          OTRO ← AUX
1.1.1.1.1.C Si (BOOL = VERDADERO)
                        entonces
                            Hacer AUX1^.DER ← AUX^.IZQ
                        si no
                            Hacer NODO^.IZQ ← AUX^.IZQ
1.1.1.1.1.D {Fin del condicional del paso 1.1.1.1.1.C}
                        Llamar al algoritmo Reestructura_der
                          con NODO^.IZQ y BO
          1.1.1.1.2 {Fin del condicional del paso 1.1.1.1.1}
        1.1.1.2 {Fin del condicional del paso 1.1.1.1}
                    Quitar (OTRO) {Libera la memoria del nodo}
    1.1.2 {Fin del condicional del paso 1.1.1}
1.2 {Fin del condicional del paso 1.1}
 si no
    Escribir "La información no se encuentra en el árbol"
2. {Fin del condicional del paso 1}
```

El análisis matemático de los algoritmos de inserción —Inserta_balanceado— y eliminación —Elimina_balanceado— demuestra que es posible buscar, insertar y eliminar un elemento en un árbol balanceado de n nodos en $O(\log n)$ unidades de tiempo. Por otra parte, diversos análisis demuestran que son más frecuentes las rotaciones en las operaciones de inserción que en las de eliminación, ya que mientras se produce aproximadamente una rotación por cada dos inserciones, se produce una rotación por cada cinco eliminaciones.

Ejemplo 6.22

Supongamos que se desea eliminar la clave 65 del árbol balanceado de la figura 6.34:

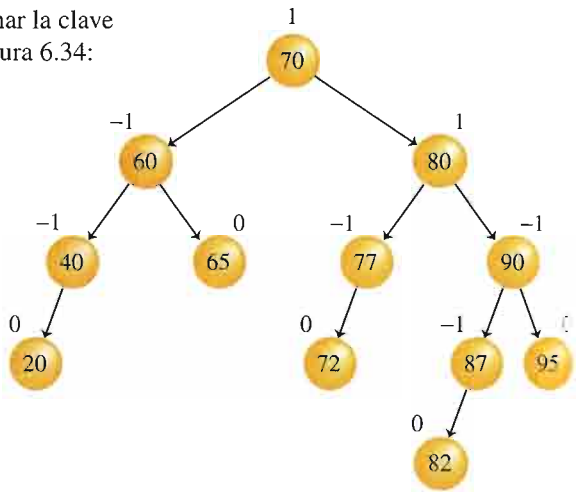
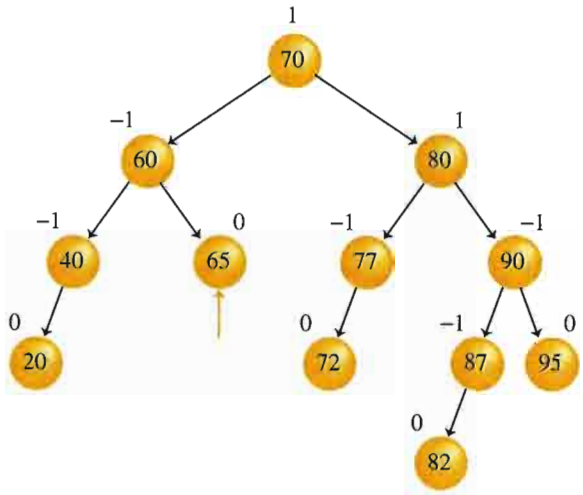


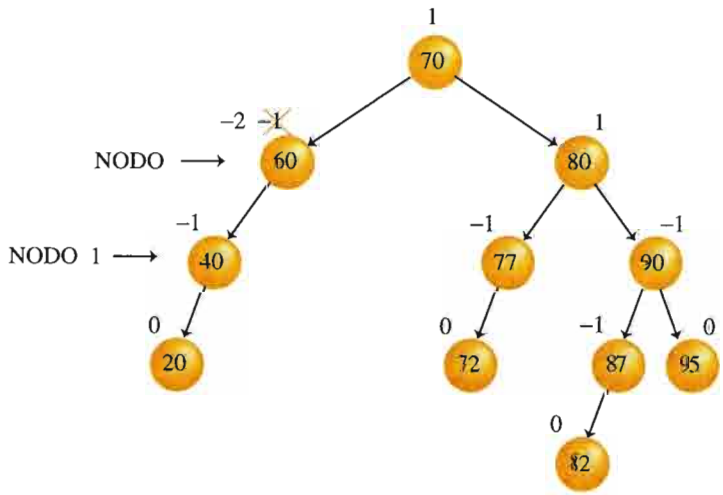
FIGURA 6.34
Árbol balanceado.

Las operaciones que se realizan son las siguientes:

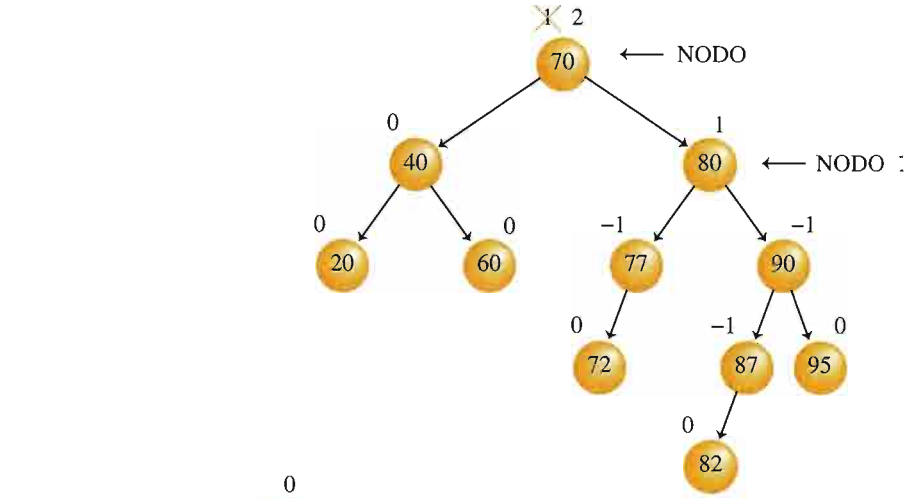
a) ELIMINACIÓN: clave 65



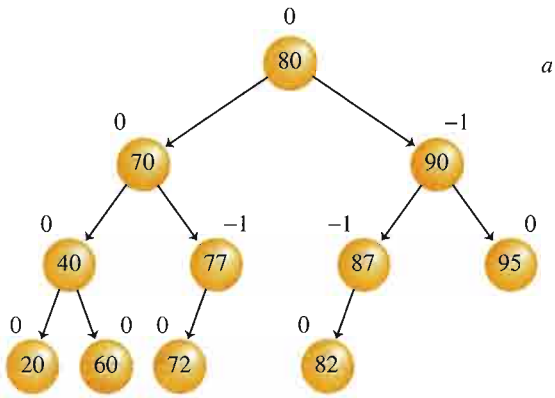
a.1)



a.2) ROTACIÓN *II*



a.3) ROTACIÓN *DD*



a.3)

Observe el lector que al eliminar la clave 65 se desbalancea el árbol y debemos efectuar la rotación *II*. Sin embargo, luego de balancear y modificar el factor de equilibrio del nodo que almacena la clave 70 nos damos cuenta que debemos efectuar un nuevo balanceo, ahora una rotación *DD*. Éste es un típico caso donde al eliminar una clave se produce una cadena de balanceos.

6.5 ÁRBOLES MULTICAMINOS

Los diferentes tipos de árboles binarios estudiados hasta el momento fueron desarrollados para funcionar en la memoria principal de la computadora. Sin embargo, existen muchas aplicaciones en las que el volumen de información es tal, que los datos no caben en la memoria principal y es necesario almacenarlos, organizados en archivos, en dispositivos de almacenamiento secundario. Esta organización de archivos debe ser suficientemente adecuada como para recuperar los datos en forma eficiente.

Es importante recordar que el tiempo necesario para localizar un registro en la memoria principal de la computadora se mide en microsegundos, mientras que el tiempo necesario para localizar una página (contiene varios registros) en memoria secundaria, por ejemplo disco, se mide en milisegundos. El tiempo de acceso, claro está, es miles de veces más rápido en la memoria principal que en la memoria secundaria.

Considere el caso de almacenar un árbol binario en disco. Se necesitará en promedio, para localizar alguno de los nodos, $\log_d n$ accesos a disco, donde n representa el número de nodos del árbol y del orden del mismo, que en este caso es igual a 2. Por ejemplo si el árbol contiene 1 000 000 de elementos, se necesitarían aproximadamente 20 accesos a disco. Ahora bien, si el árbol está organizado en páginas —nodos—, de tal manera que cada página contenga como mínimo 100 elementos, entonces se necesitarían como máximo tres accesos a disco ($\log_{100}$ 1 000 000). Note el lector que los accesos a disco disminuyen de modo considerable.

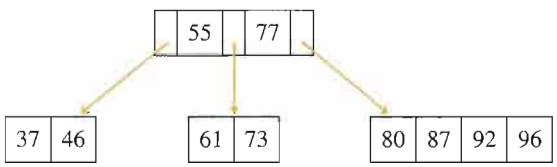
Existen diferentes técnicas para la organización de archivos indizados, sin embargo la organización en árboles-*B* y específicamente su variante, la organización en árboles-B^+, es la más utilizada.

6.5.1 Árboles-*B*

Los **árboles-*B*** son una generalización de los árboles balanceados. Éstos representan básicamente un método para almacenar y recuperar información en medios externos. Fueron propuestos por Bayer y McCreight en 1970. Su nombre árboles-*B* nunca fue explicado por los autores, aunque muchos sostienen que *B* proviene de Bayer, uno de sus inventores.

En este tipo de árboles, un grupo de nodos recibe el nombre de **página**. En cada página se almacena la información de un grupo de nodos y se identifica por medio de una clave o llave.

En general cada página de un árbol *B* de orden *d* contiene $2d$ claves como máximo y *d* claves como mínimo. Con esto se garantiza que cada página esté llena como mínimo hasta la mitad. Respecto al número de descendientes, cada página de un árbol-*B* de orden *d* tiene $2d + 1$ hijos como máximo y $d + 1$ hijos como mínimo, excepto la página raíz que puede contener como mínimo 1 dato y por consiguiente solamente 2 hijos. Las páginas en general son almacenadas en dispositivos de almacenamiento secundario, a excepción de la página raíz que es conveniente mantenerla en memoria principal. Cabe mencionar, que básicamente por cuestiones de espacio, en los ejemplos y figuras, en cada nodo se almacena solamente un dato, la clave con la cual vamos a trabajar. En la figura 6.35 se presenta un diagrama correspondiente a un árbol-*B* de orden 2.



URA 6.35
-B de orden 2.

En la figura 6.36 se observa una página de un árbol-*B* de orden *d*, con *d* claves y *d* + 1 hijos.

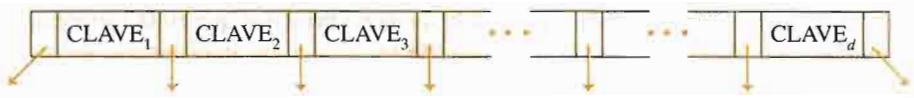


FIGURA 6.36
Página de un árbol-*B* de orden *d*.

Formalmente un árbol-*B* se define de la siguiente manera:

1. Cada página, excepto la raíz, contiene entre *d* y 2*d* elementos, siendo *d* el grado del árbol.
2. La raíz puede almacenar entre *1* y *2d* elementos.
3. Cada página, excepto la página raíz y las páginas hoja, tiene entre *d* + 1 y 2*d* + 1 descendientes. Se utilizará *m* para expresar el número de elementos por página.
4. La página raíz tiene al menos dos descendientes.
5. Las páginas hoja están todas al mismo nivel.

Ejemplo 6.23

Luego de analizar el árbol-*B* de la figura 6.37 se puede afirmar lo siguiente respecto a éste:

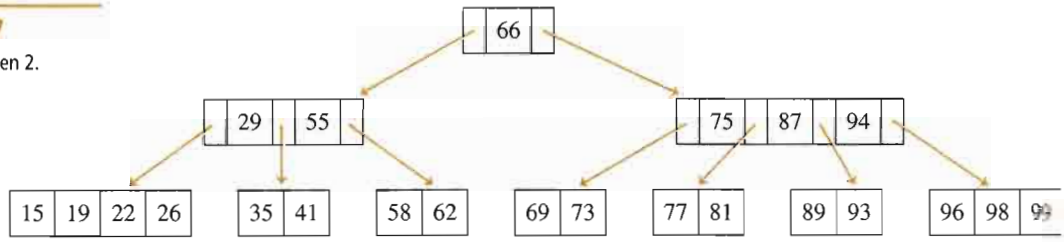


FIGURA 6.37
Árbol-*B* de orden 2.

- Orden del árbol: 2
- Altura del árbol: 3
- Todas las páginas contienen 2, 3 o 4 elementos, excepto la raíz que contiene 1.
- Los elementos dentro de la página se encuentran ordenados en forma creciente, de izquierda a derecha.
- Todas las hojas están al mismo nivel.
- Todas las páginas tienen 3 o 4 descendientes.

Búsqueda en árboles-*B*

El proceso de **búsqueda en árboles-*B*** es una generalización del proceso de búsqueda en árboles binarios de búsqueda. Los pasos necesarios para localizar una clave *X* en un árbol-*B* son los que se presentan a continuación. Se utiliza NIL para indicar que la página está vacía.

1. Se debe tener en memoria la página sobre la cual se quiere trabajar.
   1.1 *Si* (página ≠ NIL)
   entonces
   Se avanza hacia el paso 2

   si no
   Se avanza hacia el paso 3
   1.2 {Fin del condicional del paso 1.1}
2. Se debe verificar si la clave buscada se encuentra en dicha página. Si *m* es pequeña se utilizará búsqueda secuencial, de otra manera se podrá utilizar búsqueda binaria.
   2.1 *Si* (la clave se encuentra en la página)
   entonces {La operación de búsqueda concluye cuando se encuentra ¡ÉXITO! el dato en la página visitada}

   si no
   Se deben distinguir los siguientes casos:
   Si ($X < CL_{1)}$) *entonces*
   Se debe localizar PAG_0
   Si ($CL_i < X < CL_m$) *entonces*
   Se debe localizar PAG_i
   Si ($X > CL_m$) *entonces*
   Se debe localizar PAG_m
   2.2 {Fin del condicional del paso 2.1}
   2.3 Regresar al paso 1.

   Nota: Se utiliza el término *CL* para hacer referencia a las claves de una determinada página, *X* para indicar la clave que se busca y *PAG* para expresar la página que debe localizarse en memoria secundaria.
3. ¡FRACASO! La página que se desea localizar está vacía, por lo tanto el proceso de búsqueda se interrumpe y se informa que la clave no se encuentra almacenada en el árbol.

Inserción en árboles-*B*

El proceso de **inserción en árboles-*B*** es relativamente sencillo, aunque requiere cierto tratamiento especial debido a las características propias de estos árboles. Los árboles-*B* tienen un comportamiento típico, diferente al resto de los árboles estudiados anteriormente. Todas las hojas están al mismo nivel y por lo tanto cualquier camino desde la raíz hasta alguna de las hojas tiene la misma longitud. Por otra parte, los árboles-*B* tienen una forma extraña de crecer, lo hacen de abajo hacia arriba, es decir, desde las hojas hacia la raíz. Los pasos para llevar a cabo la inserción de un nodo en un árbol-*B* son los siguientes:

1. Localizar la página donde corresponde —por el valor, para no alterar el orden— insertar la clave.
2. *Si* ($m < 2d$) {El número de elementos de la página es menor a $2d$}

entonces
La clave se inserta en el lugar que le corresponde
{En la figura 6.38 se presenta un ejemplo de este caso}
si no {El número de elementos de la página es igual a $2d$}
La página afectada se divide en 2 y se distribuyen las $m + 1$ claves equitativamente entre las mismas. La clave del medio sube a la página antecesora
{En la figura 6.39 se presenta un ejemplo de este caso}

3. {Fin del condicional del paso 2}

Los pasos anteriores se repiten mientras sea necesario. Si alguna de las páginas antecesoras se desborda nuevamente, entonces hay que ordenar las claves en la página, aplicar partición y la clave del medio sube a la página antecesora. El proceso de propagación puede llegar incluso hasta la raíz, en dicho caso la altura del árbol se incrementa en una unidad.

{En la figura 6.40 se presenta un ejemplo de este caso}

FIGURA 6.38

Inserción de la clave 15 en un árbol-*B*. *a*) Antes de insertar la clave. *b*) Después de insertar la clave.

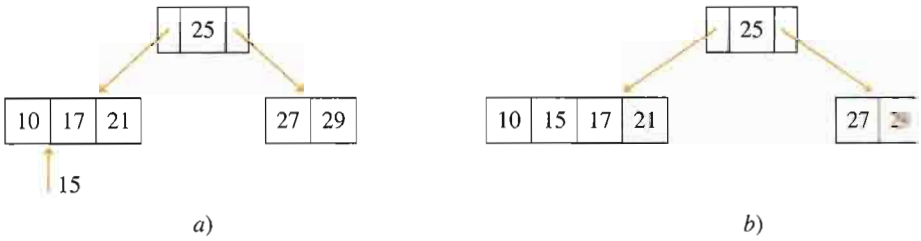


FIGURA 6.39

Inserción de la clave 13 en un árbol-*B*. *a*) Antes de insertar la clave. *b*) Después de insertarla.

Nota: Observe el lector que la inserción de la clave 13 provocó la división de la página *A* en dos páginas: *B* y *C*. Las claves se distribuyeron equitativamente entre las páginas citadas y la clave del medio (15) subió a la página antecesora.

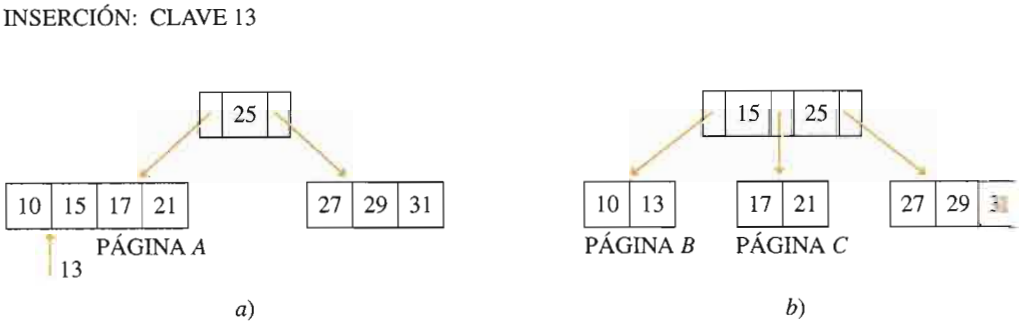
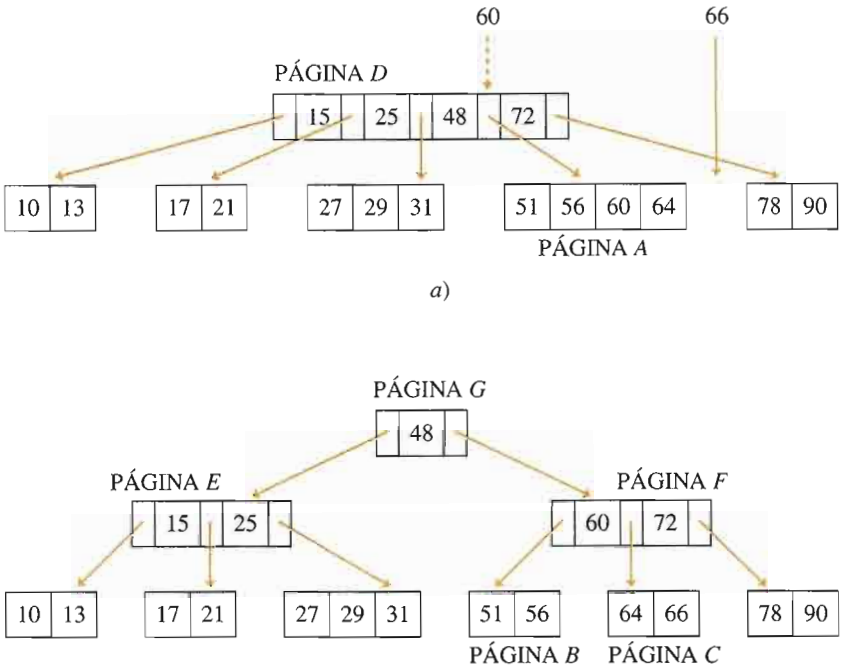


FIGURA 6.40
Inserción de la clave 66 en un árbol-*B*. *a*) Antes de insertar la clave. *b*) Después de insertarla.

Nota: Observe el lector que la inserción de la clave 66 provoca la división de la página *A* en dos páginas: *B* y *C*. Sin embargo, al subir la clave del medio (48) se produjo un nuevo desbordamiento que originó la partición de la página *D* en las páginas *E* y *F*. La clave 48 forma ahora parte de una nueva página (*G*) y representa la raíz del árbol.



Ejemplo 6.24

Supongamos que se desea insertar las siguientes claves en un árbol-*B* de orden 2 que se encuentra vacío:

10 - 27 - 29 - 17 - 25 - 21 - 15 - 31 - 13 - 51 - 20 - 24 - 48 - 19 - 60 - 35 - 66

Los resultados parciales que ilustran el crecimiento del árbol se presentan en los diagramas de la figura 6.41.

FIGURA 6.41
Inserciones en un árbol-*B* de orden 2.

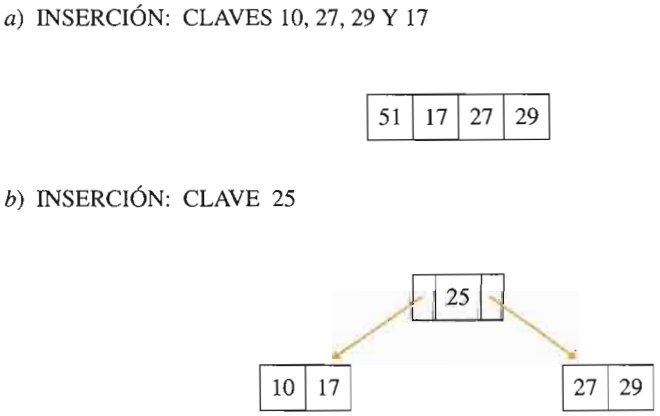
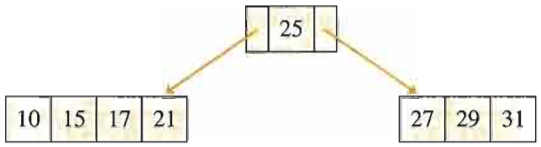
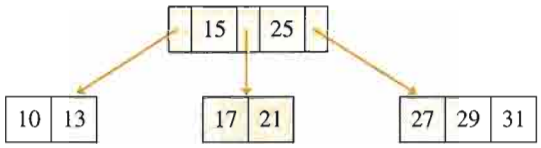


FIGURA 6.41
(*continuación*)

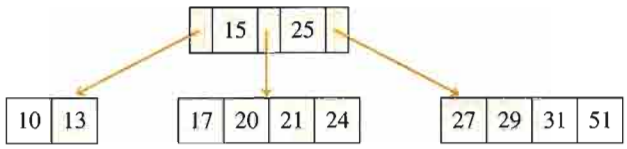
c) INSERCIÓN: CLAVES 21, 15 Y 31



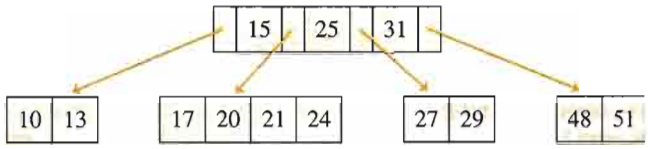
d) INSERCIÓN: CLAVE 13



e) INSERCIÓN: CLAVES 51, 20 Y 24



f) INSERCIÓN: CLAVE 48



g) INSERCIÓN: CLAVE 19

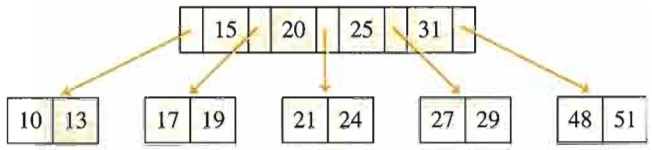
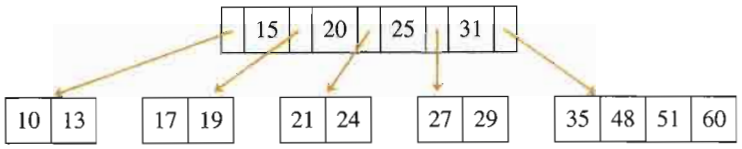
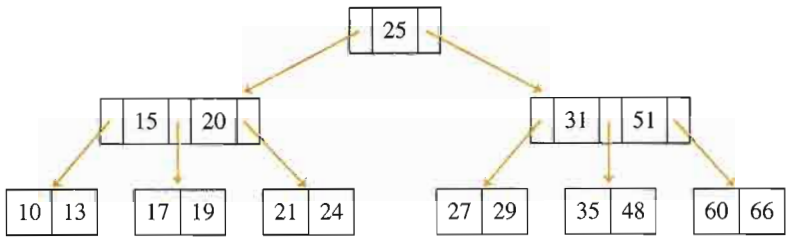


FIGURA 6.41
(continuación)

h) INSERCIÓN: CLAVES 60 Y 35



i) INSERCIÓN: CLAVE 66

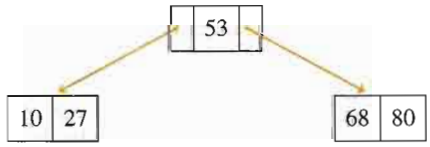


Ejemplo 6.25

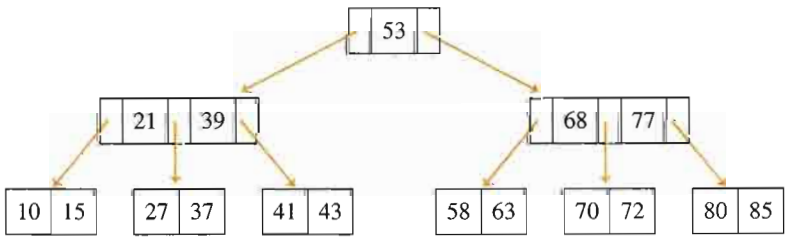
Dado como dato el árbol-*B* de orden 2 de la figura 6.42*a*, verifique si el mismo queda igual al de la figura 6.42*b* luego de insertar las siguientes claves:

43 - 21 - 77 - 58 - 63 - 15 - 37 - 41 - 72 - 39 - 95 - 70

FIGURA 6.42
Inserción en un árbol-*B* de orden 2. *a*) Antes de insertar las claves. *b*) Después de insertar las claves.



a)



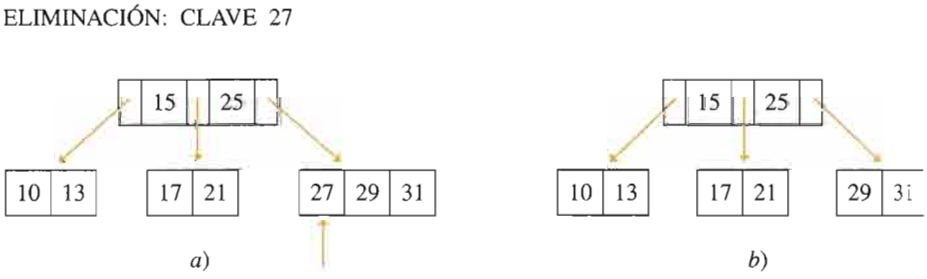
b)

Eliminación en árboles-*B*

La operación de **eliminación en árboles-*B*** es una operación más complicada que la inserción. Consiste en quitar una clave del árbol sin violar la condición de que en una página, excepto la raíz, no puede haber menos de d claves ni más de $2d$ claves, siendo d el orden del árbol. En la operación de borrado se deben distinguir los siguientes casos:

1. *Si* la clave a eliminar se encuentra en una página hoja *entonces* simplemente se suprime.
   1.1 *Si* ($m \geq d$)
   {Se verifica que el número de elementos en la página sea válido}
   entonces
   Termina la operación de borrado.
   {Se presenta un ejemplo de este caso en la figura 6.43}
   si no
   Se debe bajar la clave lexicográficamente adyacente de la página antecesora y sustituir esta clave por la que se encuentre más a la derecha en el subárbol izquierdo o por la que se encuentre más a la izquierda en el subárbol derecho. Con este paso se logra que m, en esta página, siga siendo $\geq d$.
   {Se presenta un ejemplo en las figuras 6.44*a* y 6.44*b*}
   Si esto no es posible, por las m de las páginas involucradas, se deben fusionar las páginas que son descendientes directas de la clave que se baja.
   {Se presenta un ejemplo de este caso en las figuras 6.44*c* y 6.44*d*}
   1.2 {Fin del condicional del paso 1.1}
2. {Fin del condicional del paso 1}
3. *Si* la clave a eliminar no se encuentra en una página hoja *entonces*
   Se debe sustituir por la clave que se encuentra más a la izquierda en el subárbol derecho o por la clave que se encuentra más a la derecha en el subárbol izquierdo.
   3.1 *Si* ($m \geq d$)
   {Se verifica que el número de elementos en la página sea válido}

FIGURA 6.43
Eliminación de la clave 27 en un árbol-*B* de orden 2. *a*) Antes de eliminar la clave. *b*) Después de eliminarla.



entonces
Termina la operación de borrado.
{Se presenta un ejemplo de este caso en la figura 6.45}
si no
Se debe bajar la clave lexicográficamente adyacente de la página antecesora y fusionar las páginas que son descendientes directas de dicha clave.
{En la figura 6.46 se presenta un ejemplo de este caso}
3.2 {Fin del condicional del paso 3.1}
4. {Fin del condicional del paso 3}

Cabe aclarar que el proceso de fusión de páginas se puede propagar incluso hasta la raíz, en cuyo caso la altura del árbol disminuye en una unidad. En la figura 6.47 se presentan dos ejemplos de este caso.

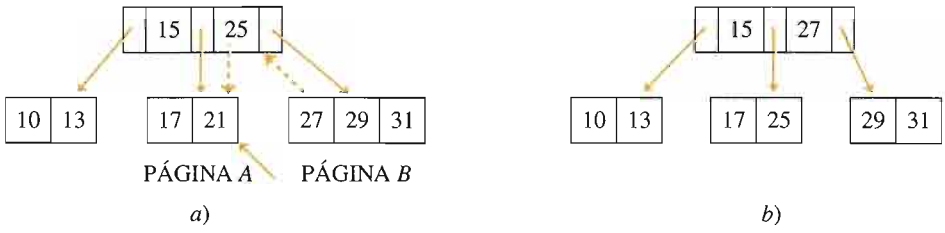


FIGURA 6.44
Eliminación de las claves 21 y 10 en un árbol-*B* de orden 2. *a*) Antes de eliminar la clave 21. *b*) Después de eliminarla. *c*) Antes de eliminar la clave 10. *d*) Después de eliminarla.

Notas: Al eliminar la clave 21 de la página *A*, baja la clave 25 de la página antecesora y esta es sustituida por la que se encuentra más a la izquierda en la página derecha; es decir, la clave 27 de la página *B*.

Al eliminar la clave 10 de la página *A*, baja la clave 15 de la página antecesora y se fusionan las páginas *A* y *B*.

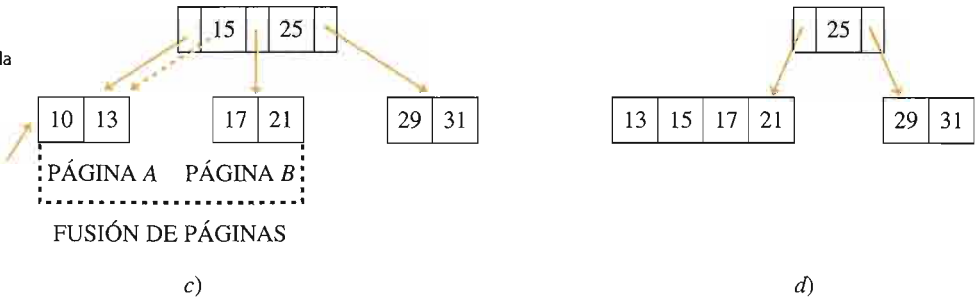


FIGURA 6.45
Eliminación de la clave 15 en un árbol-*B* de orden 2. *a*) Antes de eliminar la clave. *b*) Después de eliminarla.

Nota: Al eliminar la clave 15 se sustituye por la clave que se encuentra más a la izquierda en el subárbol derecho (17).

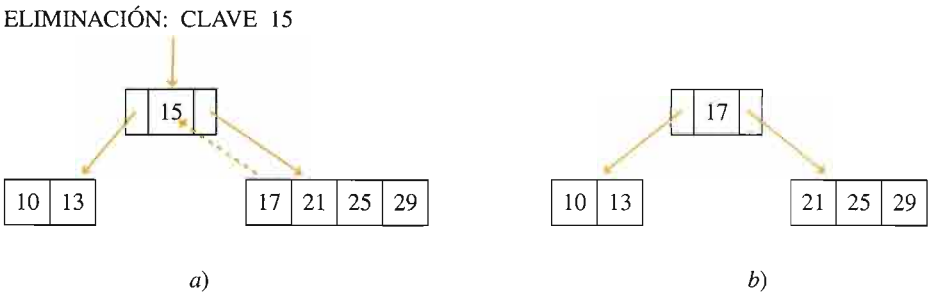


FIGURA 6.46

Eliminación de la clave 25 en un árbol-*B* de orden 2. *a*) Antes de eliminar la clave. *b*) Después de eliminarla.

Nota: Al eliminar la clave 25 se sustituye por la clave que se encuentra más a la derecha en el subárbol izquierdo (21). Sin embargo, al subir la clave 21, en la página *A*, *m* queda menor que *d*, por lo que es necesario realizar una fusión. Baja la clave correspondiente a la página antecesora (nuevamente 21), y se fusionan las páginas *A* y *B*.

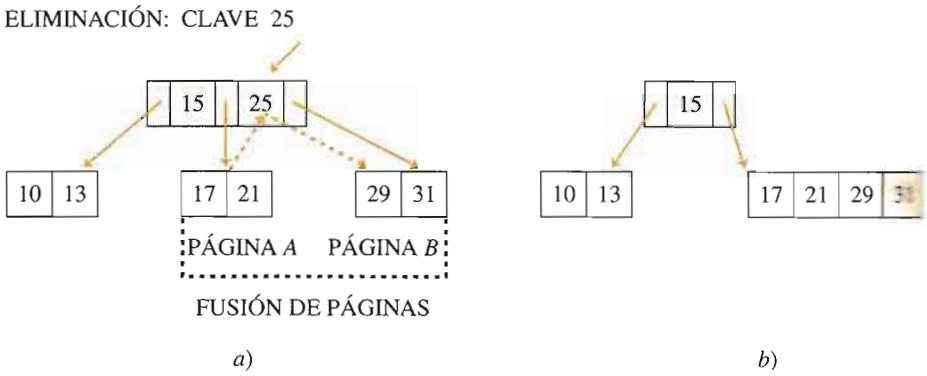


FIGURA 6.47

Eliminación de las claves 21 y 25 en un árbol-*B* de orden 2. *a*) Antes de eliminar la clave 21. *b*) Después de eliminarla.

Nota: Al eliminar la clave 21 de la página *A*, *m* queda menor a *d*, por lo que es necesario bajar la clave 20 de la página antecesora, produciéndose la fusión de las páginas *A* y *B*. Sin embargo, en la página *C* nuevamente *m* queda menor a *d*, por lo que es necesario bajar la clave 25 de la página *E*. Como esta página queda vacía, es necesaria entonces una nueva fusión, ahora de las páginas *C* y *D*. La altura del árbol disminuye en una unidad.

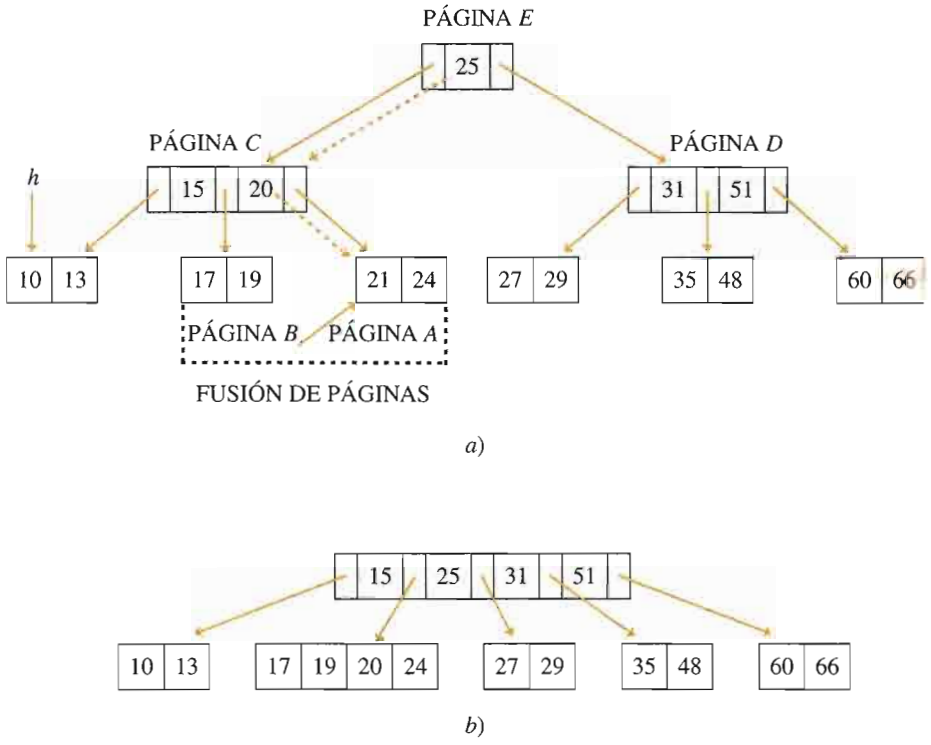
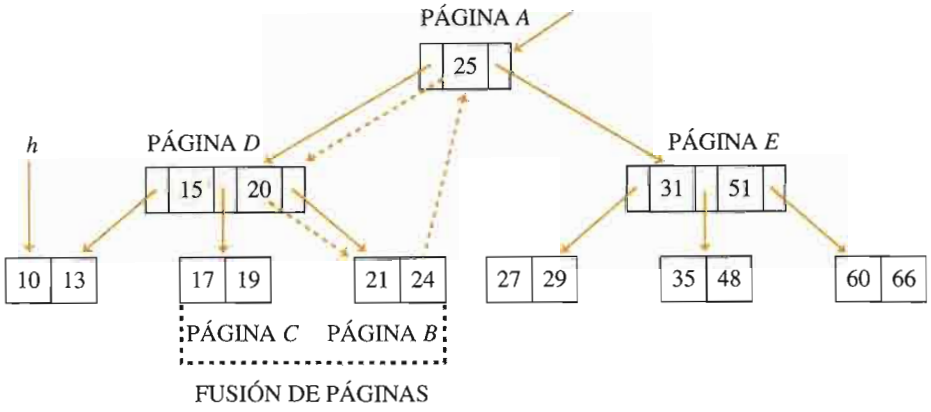


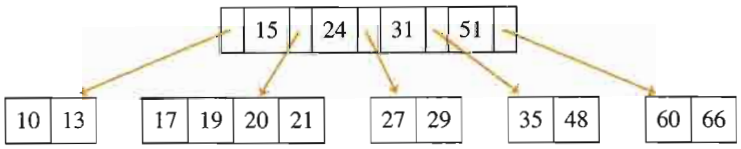
FIGURA 6.47
(continuación)
c) Antes de eliminar la clave 25. d) Después de eliminarla.

Nota: Al eliminar la clave 25 de la página A, se sustituye por la clave que se encuentra más a la derecha en el subárbol izquierdo (24 de la página B). Sin embargo, en la página B, m queda menor que d, por lo que es necesario bajar la clave 20 de la página D produciéndose la fusión de las páginas B y C. Nuevamente en la página D, m queda menor a d, por lo que ahora es necesario bajar la clave 24 de la página A. Como esta página queda vacía entonces necesita realizarse una fusión de las páginas D y E. La altura del árbol disminuye en una unidad.

ELIMINACIÓN: CLAVE 25



c)

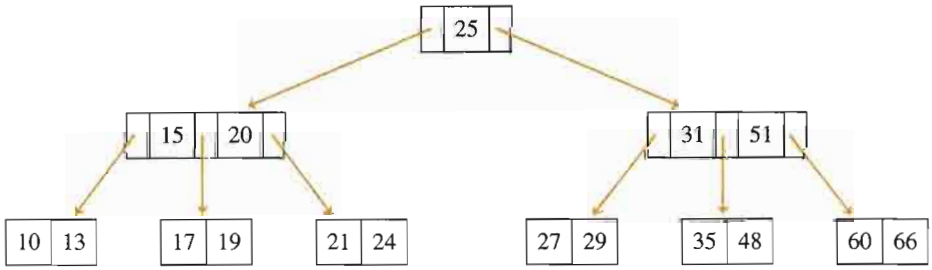


d)

Ejemplo 6.26

Supongamos que se desea eliminar las siguientes claves del árbol-*B* de orden 2 de la figura 6.48:

FIGURA 6.48
Árbol-*B* de orden 2.

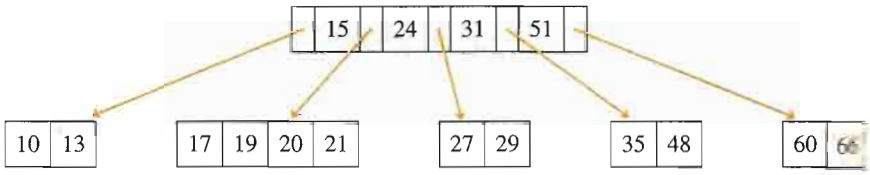


25 - 24 - 29 - 27 - 48 - 19 - 51 - 21 - 13 - 15 - 17 - 66 - 10

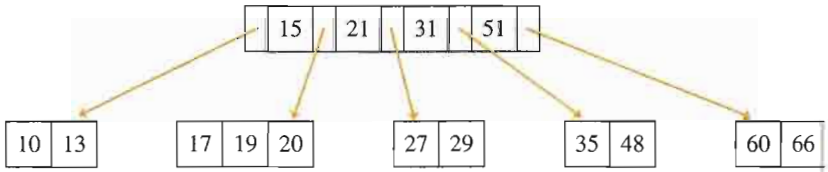
Los resultados parciales que ilustran cómo funciona el procedimiento se presentan en los diagramas de la figura 6.49.

FIGURA 6.49
Eliminaciones en un árbol-*B* de orden 2.

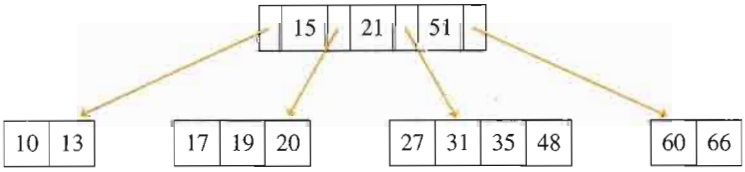
a) ELIMINACIÓN: CLAVE 25



b) ELIMINACIÓN: CLAVE 24



c) ELIMINACIÓN: CLAVE 29



d) ELIMINACIÓN: CLAVES 27, 48 Y 19

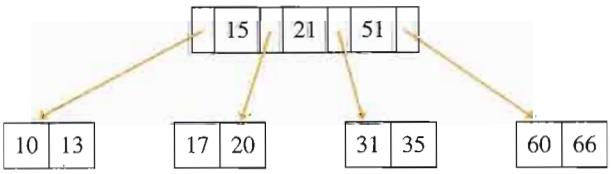
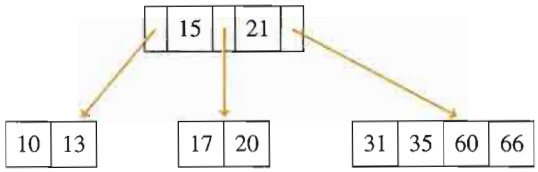
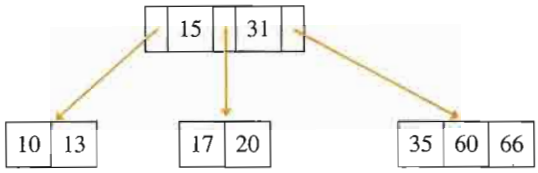


FIGURA 6.49
(continuación)

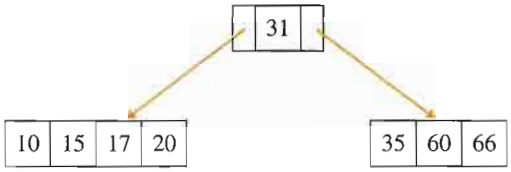
e) ELIMINACIÓN: CLAVE 51



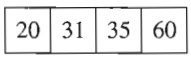
f) ELIMINACIÓN: CLAVE 21



g) ELIMINACIÓN: CLAVE 21



h) ELIMINACIÓN: CLAVES 15, 17, 66 Y 10

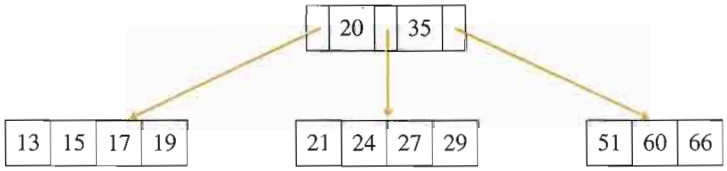


Ejemplo 6.27

Dado como dato el árbol-*B* de orden 2 de la figura 6.48, verifique si el mismo queda igual al de la figura 6.50, luego de eliminar las siguientes claves:

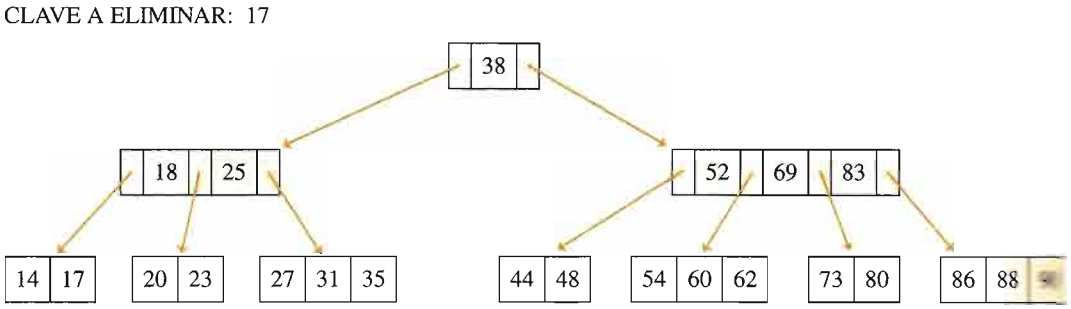
48 - 31 - 10 - 25

FIGURA 6.50
Árbol-*B* de orden 2 luego de eliminar las claves 48, 31, 10 y 25.



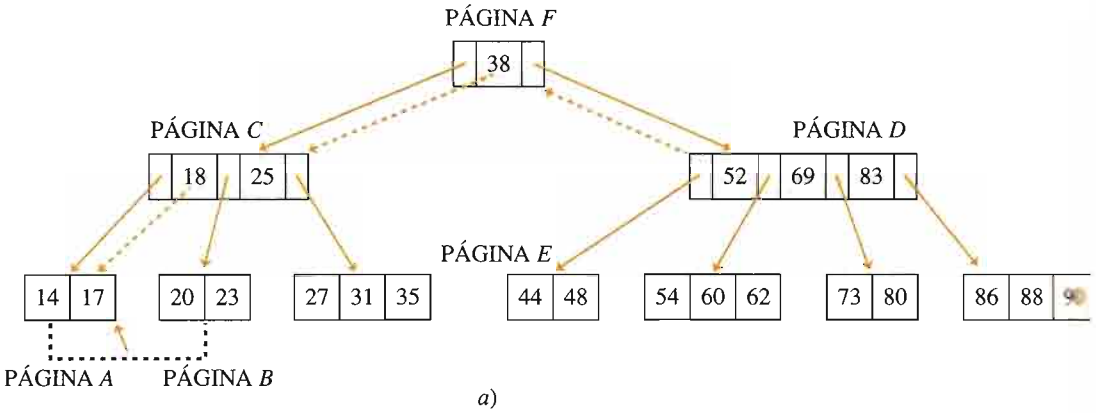
Ejemplo 6.28 Supongamos que se desea eliminar la clave 17 del árbol-*B* de orden 2 de la figura 6.51.

FIGURA 6.51
Árbol-*B* de orden 2.

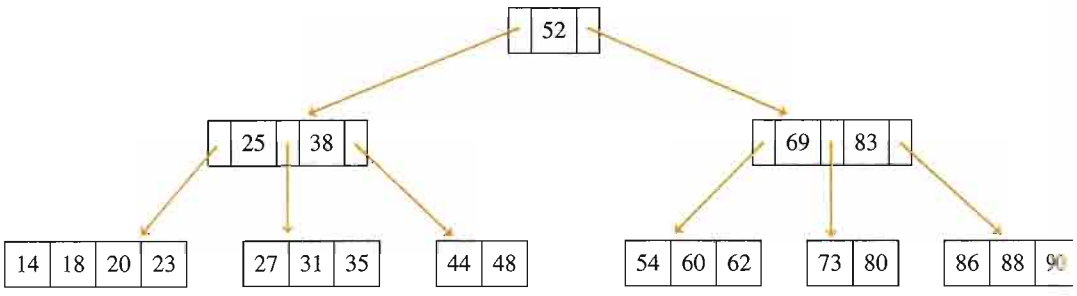


Las operaciones que se realizan son las siguientes:

a) ELIMINACIÓN: CLAVE 17



a)



b)

Nota: Al eliminar la clave 17 de la página *A*, *m* queda menor a *d*, por lo que es necesario bajar la clave 18 de la página *C*, produciéndose la fusión de las páginas *A* y *B*. Sin embargo, en la página *C* nuevamente *m* queda menor a *d*, por lo que es necesario bajar la clave 38 de la página *F*. Aquí es donde se produce uno de los casos más difíciles de borrado en árboles-*B*. En los ejemplos anteriores hacíamos fusión de las páginas *C* y *D*, disminuyendo la altura del árbol. Sin embargo, si hiciéramos esto *m* sería mayor a 2*d*, por lo que violaríamos los principios que definen un árbol-*B*. Es necesario entonces subir la clave 52 de la página *D* a la página *F*, y la página *E* pasa a ser el hijo derecho de la clave 38, ahora en la página *C*.

6.5.2 Árboles-B^+

Los **árboles-B^+** se han convertido en la técnica más utilizada para la organización de archivos indizados. La principal característica de estos árboles es que toda la información se encuentra en las hojas, mientras que los nodos raíz e interiores almacenan claves que se utilizan como índices. Debido a esta característica de los árboles-*B*, todos los caminos desde la raíz hasta cualquiera de los datos tienen la misma longitud. En la figura 6.52 presentamos un diagrama de un árbol-B^+ de orden 2.

Es de notar que los árboles-B^+ ocupan un poco más de espacio que los árboles-*B*, y esto ocurre al existir duplicidad en algunas claves. Sin embargo, esto es aceptable si el archivo se modifica frecuentemente, puesto que se evita la operación de reorganización del árbol que es tan costosa en los árboles-*B*.

Formalmente se define un árbol-B^+ de orden *d* de la siguiente manera:

1. Cada página, excepto la raíz, contiene *m* elementos, donde *m* es un valor entre *d* y 2*d*.
2. La raíz contiene de 1 a 2*d* elementos.
3. Cada página, excepto la raíz, tiene entre *d* + 1 y 2*d* + 1 descendientes.
4. La página raíz tiene al menos dos descendientes.
5. Las páginas hojas están todas al mismo nivel.
6. Toda la información, con las claves que las identifican, se encuentra en las páginas hoja.
7. Las claves almacenadas en las páginas raíz e interiores se utilizan como índices.

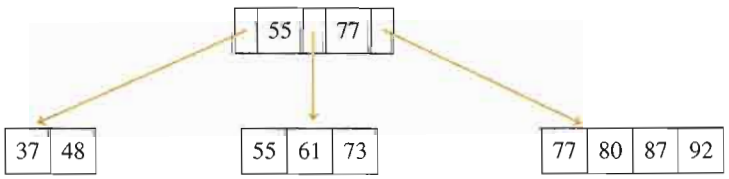


FIGURA 6.52
-B^+ de orden 2.

Búsqueda en árboles-B^+

La operación de **búsqueda en árboles-B^+** es similar a la operación de búsqueda en árboles-B. El proceso es simple, sin embargo puede suceder que al buscar una determinada clave la misma se encuentre en una página raíz o interior. En dicho caso no se debe detener el proceso porque en la página raíz o en las interiores sólo se almacenan claves que funcionan como índices. La búsqueda debe continuar en la página apuntada por la rama derecha de dicha clave.

Por ejemplo, al buscar la clave 55 en el árbol-B^+ de la figura 6.52 se advierte que ésta se encuentra en la página raíz. En este caso, se debe continuar el proceso de búsqueda en la página apuntada por la rama derecha de dicha clave.

Inserción en árboles-B^+

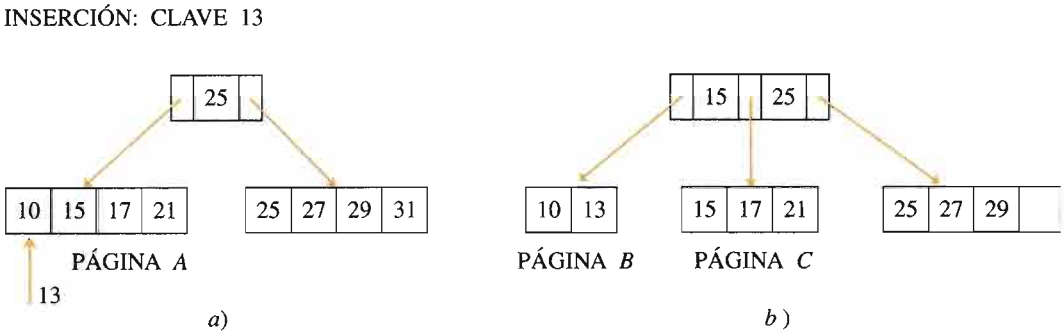
El proceso de **inserción en árboles-B^+** es relativamente simple, similar al proceso de inserción en árboles-B. La dificultad se presenta cuando se desea insertar una clave en una página que se encuentra llena ($m = 2d$). En este caso, la página afectada se divide en 2, distribuyéndose las $m + 1$ claves de la siguiente forma: "las d primeras claves en la página de la izquierda y las $d + 1$ restantes claves en la página de la derecha". Una copia de la clave del medio sube a la página antecesora.

En la figura 6.53 se muestran dos diagramas que ilustran cómo funciona este caso. Puede suceder que la página antecesora se desborde nuevamente, en dicho caso se debe repetir el proceso anterior. Es importante notar que el desbordamiento en una página que no es hoja no produce duplicidad de claves. El proceso de propagación puede llegar hasta la raíz, en cuyo caso la altura del árbol se puede incrementar en una unidad.

FIGURA 6.53

Inserción de la clave 13 en un árbol-B^+. *a*) Antes de insertar la clave. *b*) Después de insertarla.

Nota: Observe que la inserción de la clave 13 en la página *A* produce su división en dos páginas: *B* y *C*. Las *d* primeras claves se ubican en la página *B* (10 y 13). Las $d + 1$ claves restantes en la página *C* (15, 17 y 21). Una copia de la clave del medio (15) sube a la página antecesora.

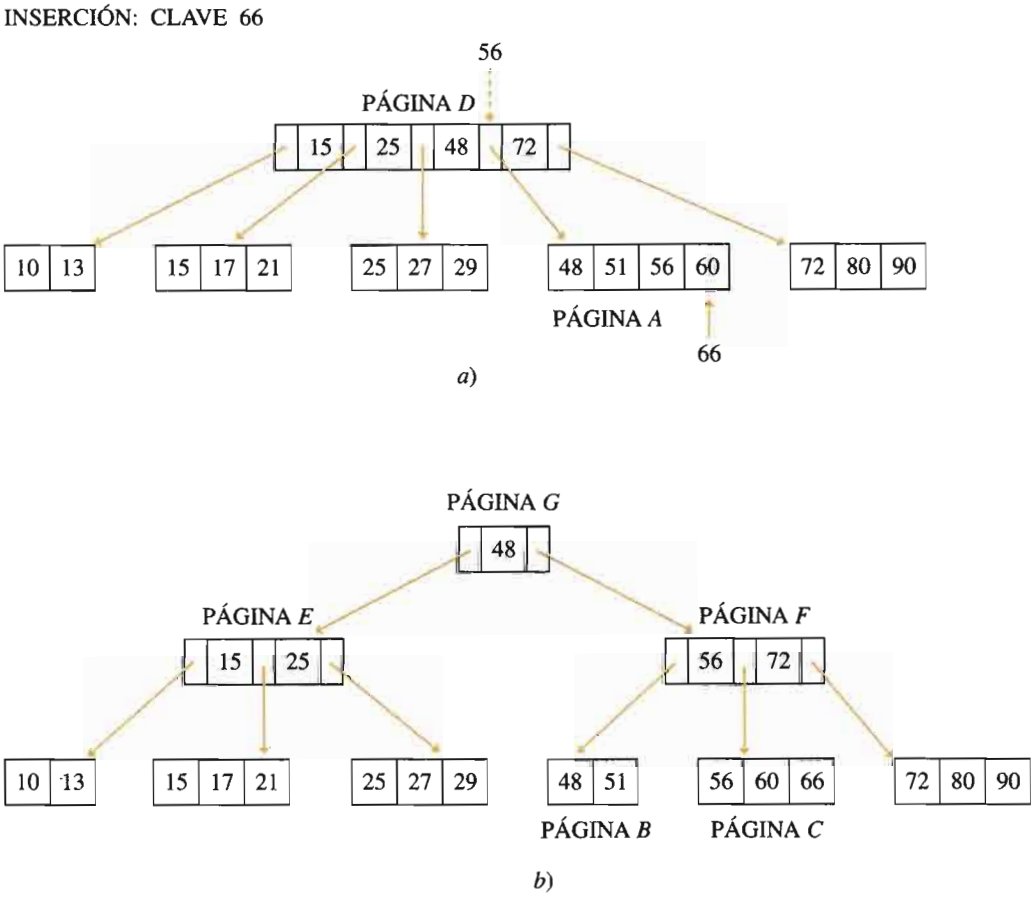


En la figura 6.54 se presentan dos diagramas que clarifican y resuelven este caso.

FIGURA 6.54

Inserción de la clave 66 en un árbol-B^+. *a*) Antes de insertar la clave. *b*) Después de insertarla.

Nota: La inserción de la clave 66 en la página *A* provocó la división de ésta en las páginas *B* y *C*. Sin embargo, al subir una copia de la clave del medio (56) se produce un nuevo desbordamiento en la página *D* que provoca su partición en las páginas *E* y *F*. La clave 48 forma ahora parte de la página *G* y representa la raíz del árbol. La altura del árbol se incrementa en una unidad.



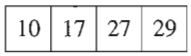
Ejemplo 6.29 Supongamos que se desea insertar las siguientes claves en un árbol-B^+ de orden 2 que se encuentra vacío:

10 - 27 - 29 - 17 - 25 - 21 - 15 - 31 - 13 - 51 -
20 - 24 - 48 - 19- 60 - 35 - 66

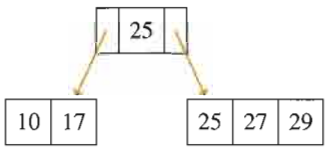
Los resultados parciales que ilustran el crecimiento del árbol se presentan en los diagramas correspondientes a la figura 6.55.

FIGURA 6.55
Inserciones en un árbol-B^+ de orden 2.

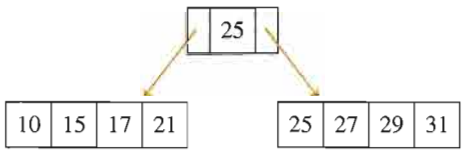
a) INSERCIÓN: CLAVES 10, 27, 29 Y 17



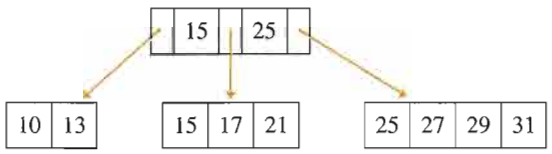
b) INSERCIÓN: CLAVE 25



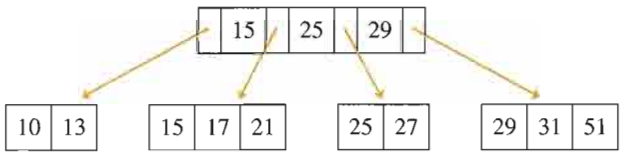
c) INSERCIÓN: CLAVES 21, 15 Y 31



d) INSERCIÓN: CLAVE 13



e) INSERCIÓN: CLAVE 51



f) INSERCIÓN: CLAVES 20, 24, 48 Y 19

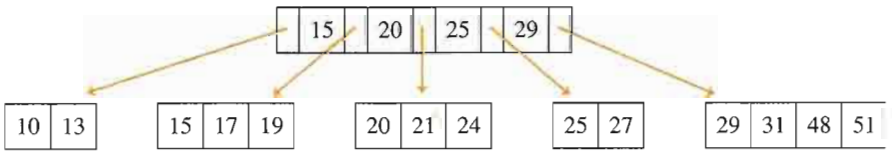
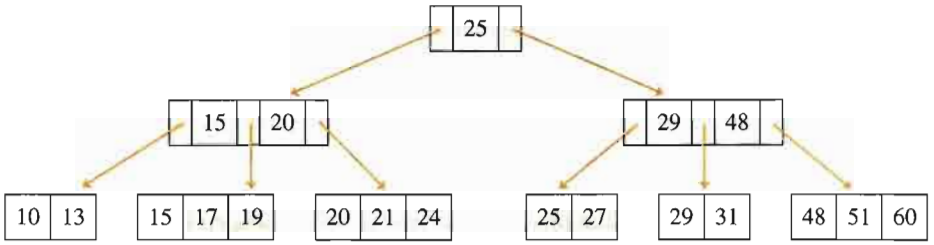
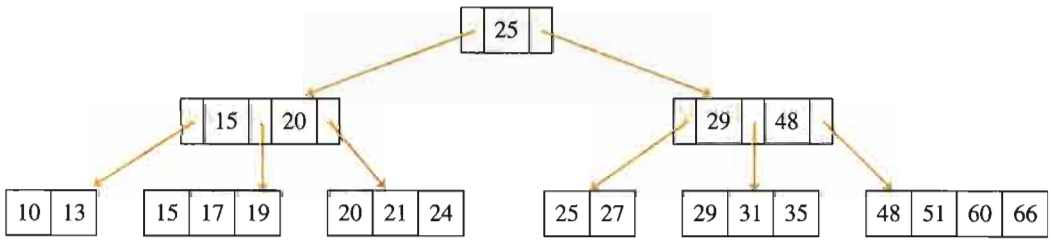


FIGURA 6.55
continuación)

g) INSERCIÓN: CLAVE 60



h) INSERCIÓN: CLAVES 35 Y 66

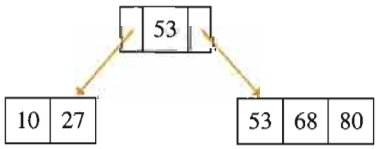


Ejemplo 6.30

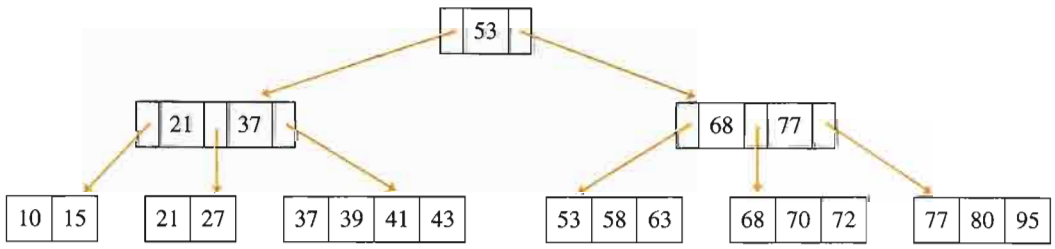
Dado como dato el árbol-B^+ de orden 2 de la figura 6.56*a*, verifique si el mismo queda igual al de la figura 6.56*b*, luego de insertar las siguientes claves:

43 - 21 - 77 - 58 - 63 - 15 - 37 - 41 - 72 - 39 - 95 - 70

FIGURA 6.56
...iones en un árbol-B^+ ... orden 2. *a*) Antes de ...tar las claves. ... Después de insertarlas.



a)



b)

Eliminación en árboles-B^+

La operación de **eliminación en árboles-B^+** es más simple que la operación de borrado en árboles-B. Esto ocurre porque las claves que se deben eliminar siempre se encuentran en las páginas hoja. En general se deben distinguir los siguientes casos:

1. *Si* al eliminar una clave m queda mayor o igual a d, *entonces* termina la operación de borrado. Las claves de las páginas raíz o internas no se modifican por más que sean una copia de la clave eliminada en las hojas. (Se presenta un ejemplo de este caso en la figura 6.57.)
2. *Si* al eliminar una clave m queda menor a d, *entonces* se debe realizar una redistribución de claves, tanto en el índice como en las páginas hojas. Cuando se cambia la estructura del árbol, se quitan aquellas claves que quedaron en los nodos interiores luego de haber eliminado su correspondiente información en los nodos hoja. Hay dos ejemplos que ilustran cómo funciona este caso en la figura 6.58.

Puede suceder que al eliminar una clave y al realizar una redistribución de las mismas, la altura del árbol disminuya en una unidad. En la figura 6.59 se presentan dos diagramas que corresponden a este caso.

FIGURA 6.57

Eliminación de la clave 25 de un árbol-B^+ de orden 2. *a*) Antes de eliminar la clave. *b*) Después de eliminarla.

Nota: Al eliminar la clave 25 de la página *A*, la página raíz *B* que contiene como índice a la clave eliminada no se modifica.

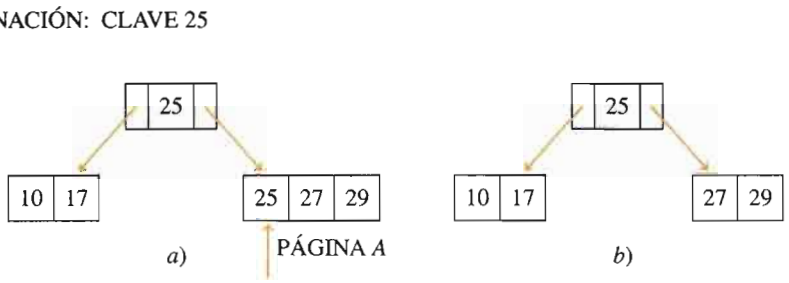


FIGURA 6.58

Eliminación de las claves 27 y 21 de un árbol-B^+ de orden 2. *a*) Antes de eliminar la clave 27. *b*) Después de eliminarla. *c*) Antes de eliminar la clave 21. *d*) Después de eliminarla.

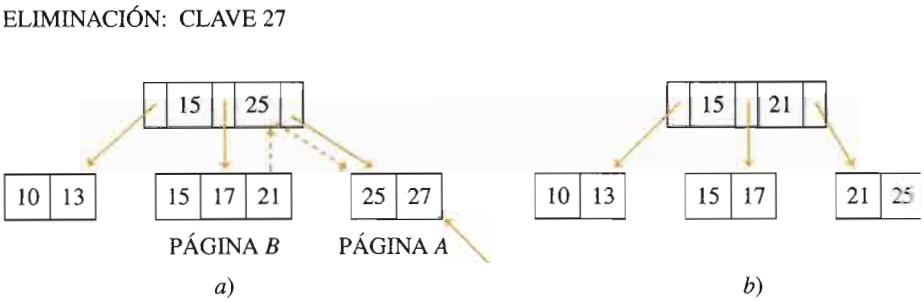


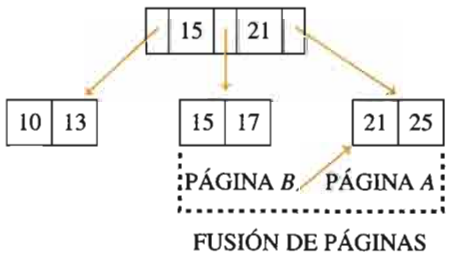
FIGURA 6.58

(continuación)

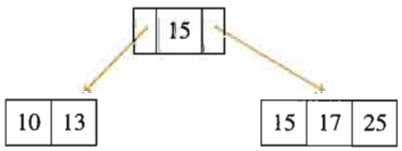
Notas: Al eliminar la clave 27 de la página *A*, *m* queda menor a *d*, por lo que debe realizarse una redistribución de claves. Se toma la clave que se encuentra más a la derecha en la rama izquierda de 25 (21 de la página *B*). Se coloca dicha clave en la página *A* y una copia de la misma, como índice, en la página *C*.

Al eliminar la clave 21 de la página *A*, *m* queda menor a *d*, por lo que debe realizarse una redistribución de claves. Como no se puede tomar una clave de la página *B* puesto que *m* quedaría menor a *d*, entonces se realiza una fusión de las páginas *A* y *B*.

ELIMINACIÓN: CLAVE 21



c)



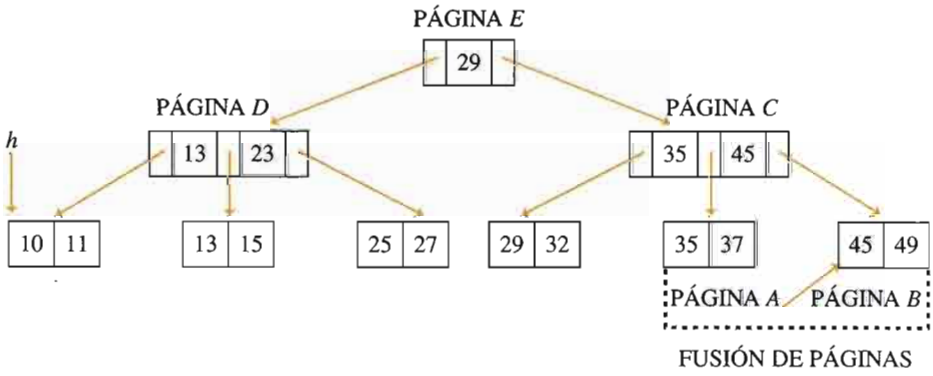
d)

FIGURA 6.59

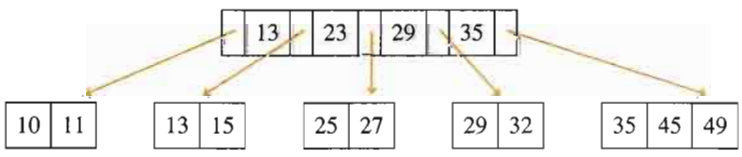
Eliminación de la clave 37 en un árbol-B^+ de orden 2. *a)* Antes de eliminar la clave. *b)* Después de eliminarla.

Notas: Al eliminar la clave 37 de la página *A*, *m* queda menor a *d*, por lo que debe realizarse una redistribución de claves. Como no puede tomarse una clave de la página *B*, puesto que *m* quedaría menor a *d*, entonces se realiza una fusión de las páginas *A* y *B*. Sin embargo, luego de esta fusión *m* queda menor a *d* en la página *C*, por lo que debe bajarse la clave 29 de la página *E* y realizarse una nueva fusión, ahora de las páginas *C* y *E*. La altura del árbol disminuye en una unidad.

ELIMINACIÓN: CLAVE 37



a)



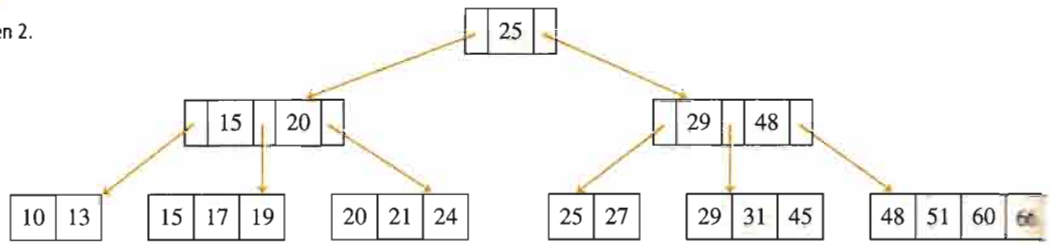
b)

Ejemplo 6.31

Supongamos que se desea eliminar las siguientes claves del árbol-B^+ de orden 2 de la figura 6.60.

15 - 51 - 48 - 60 - 31 - 20 - 66 - 29 - 10 - 25 - 17 - 24

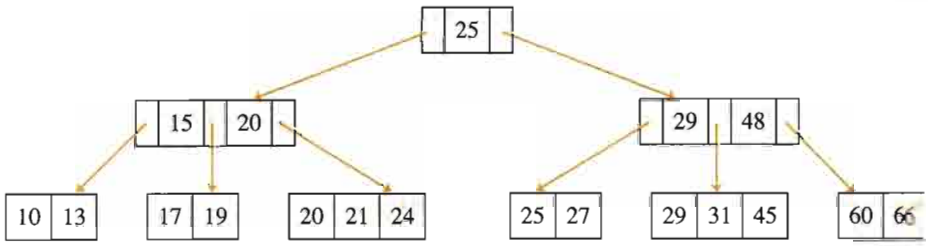
FIGURA 6.60
Árbol-B^+ de orden 2.



Los resultados parciales que ilustran cómo funciona el procedimiento se presentan en los diagramas de la figura 6.61.

FIGURA 6.61
Eliminaciones en un árbol-B^+ de orden 2.

a) ELIMINACIÓN: CLAVES 15, 51 Y 48



b) ELIMINACIÓN: CLAVE 60

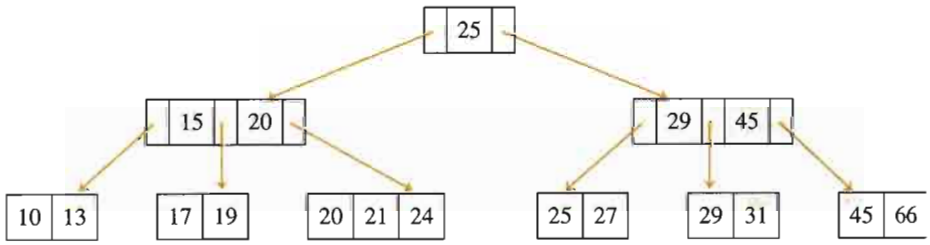
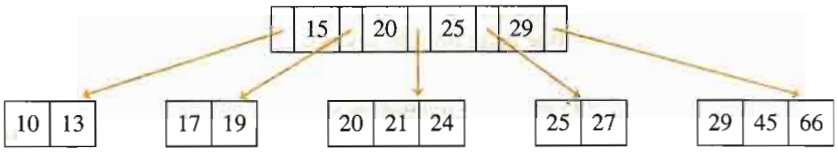
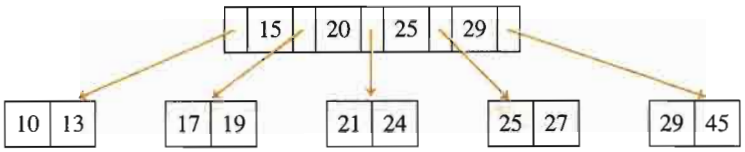


FIGURA 6.61
(continuación)

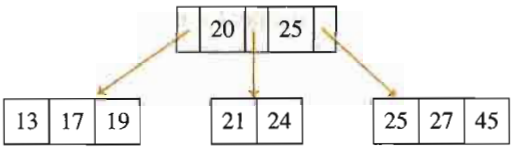
c) ELIMINACIÓN: CLAVE 31



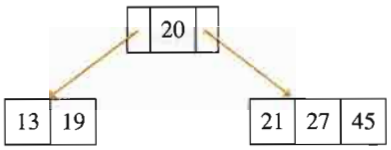
d) ELIMINACIÓN: CLAVES 20 Y 66



e) ELIMINACIÓN: CLAVES 29 y 10



f) ELIMINACIÓN: CLAVES 25, 17 y 24

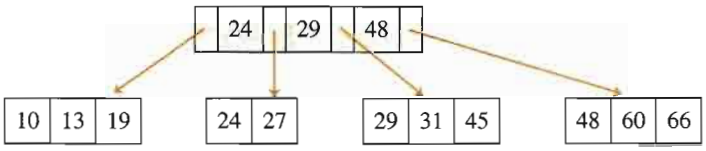


Ejemplo 6.32

Verifique si el árbol-B^+ de orden 2 de la figura 6.60 queda igual al de la figura 6.62, luego de eliminar las siguientes claves:

51 - 20 - 15 - 21 - 25 - 17

FIGURA 6.62
Árbol-B^+ de orden 2 luego de eliminar las claves 51, 15, 21, 25 y 17.



6.5.3 Árboles 2-4

Los **árboles 2-4** son una variante de los árboles multicaminos. Éstos se caracterizan porque cada uno de sus nodos puede tener máximo 4 hijos y todos los nodos externos —hojas— están al mismo nivel. Es decir, en estos árboles se debe garantizar el tamaño y la altura de los mismos.

Como en el caso de los árboles multicaminos analizados en las secciones previas, las operaciones de inserción y eliminación pueden ocasionar, respectivamente, la partición o fusión de los nodos con el objeto de mantener las propiedades enunciadas. Debido a que se llevan a cabo de manera similar a lo presentado, se deja al lector el desarrollo de los correspondientes algoritmos.

6.6 LA CLASE *ÁRBOL*

La **clase *Árbol*** tiene como atributo a la raíz de la estructura y como métodos a todas las operaciones analizadas, según el tipo de árbol que se esté representando. Gráficamente una clase *Árbol* —para árboles binarios— se puede ver como se muestra en la figura 6.63. En este caso, los métodos permiten llevar a cabo todas las operaciones presentadas previamente: los tres tipos de recorridos, búsqueda, inserción y eliminación.

Se tiene acceso a los miembros de un objeto de la clase *Árbol* por medio de la notación de puntos. Asumiendo que la variable AROBJ es un objeto de la clase *Árbol* previamente creado, se puede hacer:

AROBJ.Recorre_Preorden(argumento) para invocar el método que visita cada uno de los nodos del árbol siguiendo el recorrido preorden. En este método se requiere como argumento un puntero al nodo a visitar —la primera vez es la raíz—, ya que es un método recursivo.

AROBJ.Inserta(argumentos) para insertar un nuevo elemento en el árbol binario. En este método se requieren dos argumentos, uno para el nodo a visitar —la primera vez es la raíz— y otro para el dato a insertar.

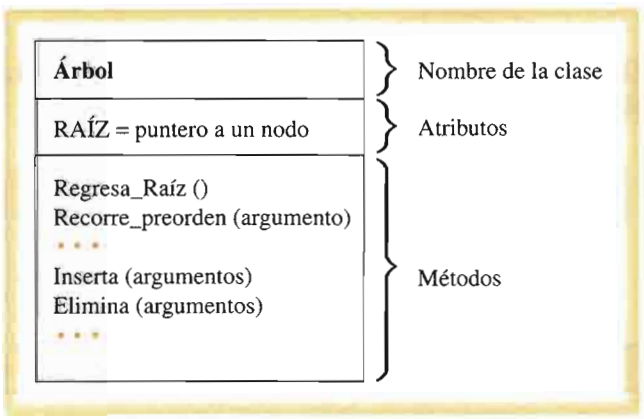
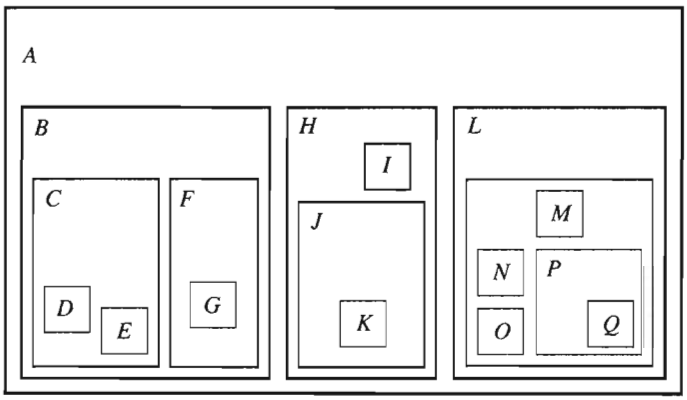


FIGURA 6.63
Clase *Árbol*.

EJERCICIOS

Árboles en general

1. Los árboles se pueden representar de diferentes formas. Dado el siguiente diagrama de Venn que corresponde a una estructura árbol, conviértalo a notación decimal de Dewey y notación indentada.



2. Dada la siguiente estructura del árbol representada como anidación de paréntesis:

$$(A(B(E(K), F), C(G(L, M(N))), D(H, I, (O, P, Q, R), J)))$$

Calcule lo siguiente:

a) Grado del árbol.
b) Grado del nodo *G*.
c) Altura del árbol.
d) Nodos terminales u hojas.
e) Nodos interiores.

3. Dada la siguiente estructura de árbol representada como notación decimal de Dewey:

1.*A*, 1.1.*B*, 1.1.1.*D*, 1.1.2.*E*, 1.1.2.1.*I*,1.1.2.2.J,
1.1.3.*F*, 1.1.4.*G*, 1.1.4.1.*K*, 1.1.4.1.1.*M*, 1.1.4.1.2.*N*,
1.2.*C*, 1.2.1.*H*, 1.2.1.1.*L*

Calcule las longitudes de camino interno y externo de dicho árbol.

4. Calcule cuál es el grado del nodo *T*, si *T* es padre del nodo *P* y éste tiene 4 hermanos.

Árboles binarios

5. Represente las siguientes expresiones algebraicas utilizando árboles binarios.

a) $((C * Y) \wedge 0.8 + (D/R)) * K$
b) $A = R - (C + L / D) * K$
c) $X = ((B / C * D) \wedge 0.5 + (B / K + P) \wedge 0.8) \wedge 0.5$

6. Dados los siguientes árboles binarios representados como anidación de paréntesis

a) (*K* (*B* (*A*, *F* (*D*)), *W* (*M* (*L*, *O* (*P*)), *Z*)))
b) (25(20(10(8)), 23(21)), 90(80(62(47(32))), 100))

Escriba los recorridos preorden, inorden y posorden de cada uno de ellos.

7. ¿Cuál es el máximo número de nodos de un árbol binario de altura *h*?

8. ¿Cuántos árboles binarios **distintos** se puede tener con 4 nodos? ¿Y cuántos con 7?

9. ¿Cuántos árboles binarios **similares** se puede tener con 4 nodos? ¿Y cuántos con 7?

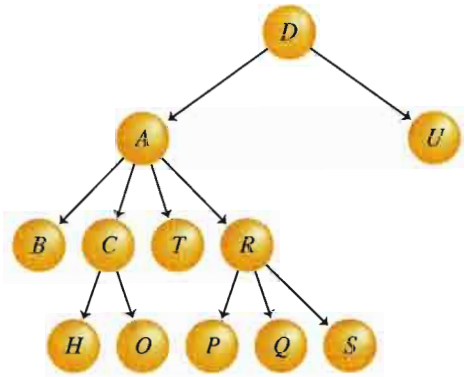
10. Dadas las siguientes secuencias de nodos obtenidas por los recorridos preorden, inorden y posorden, dibuje el correspondiente árbol binario.

- Preorden: *P - R - A - C - H - T - O - M*
- Inorden: *A - R - H - C - P - O - T - M*
- Postorden: *A - H - C - R - O - M - T - P*

Representación de árboles generales como árboles binarios

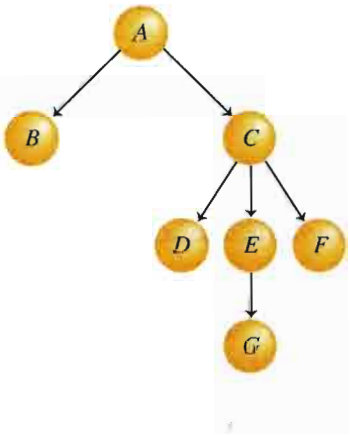
11. Dados los siguientes árboles generales, uno representado en forma de grafo, inciso *a*, y otro representado como anidación de paréntesis, inciso *b*), conviértalos a árboles binarios.

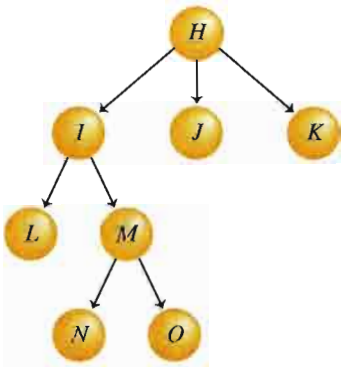
a)

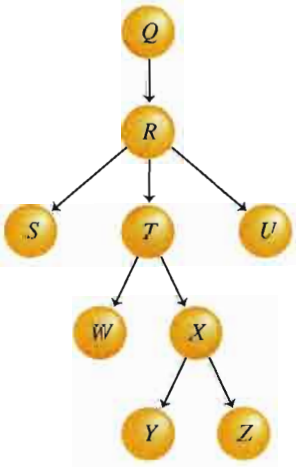


b) (*A* (*B* (*E*, *F* (*K*)), *C* (*G* (*L*, *M* (*Q*, *R*), *N*)), *D* (*H*, *I* (*O* (*S*), *P*))))

12. Dado el siguiente bosque, conviértalo en árbol binario.





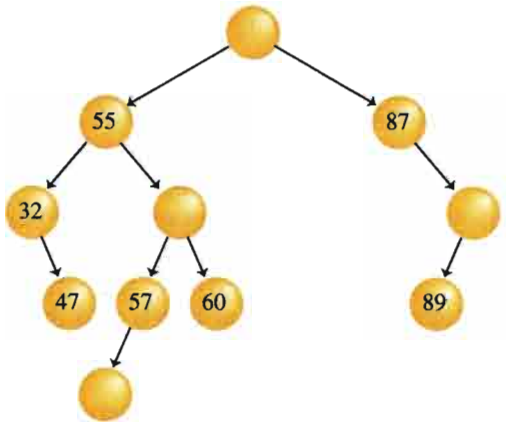


Árboles binarios de búsqueda

13. Dadas las siguientes claves que representan los signos del zodiaco, construya un árbol binario de búsqueda.

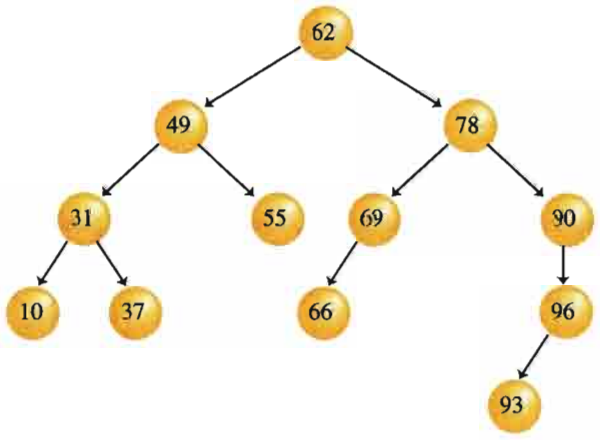
piscis - acuario - capricornio - cáncer - sagitario - virgo - leo - escorpión - libra - géminis - aries - tauro

14. Dado el siguiente árbol binario de búsqueda, complete los nodos en blanco de tal forma que no se violen los principios que definen justamente un árbol binario de búsqueda.



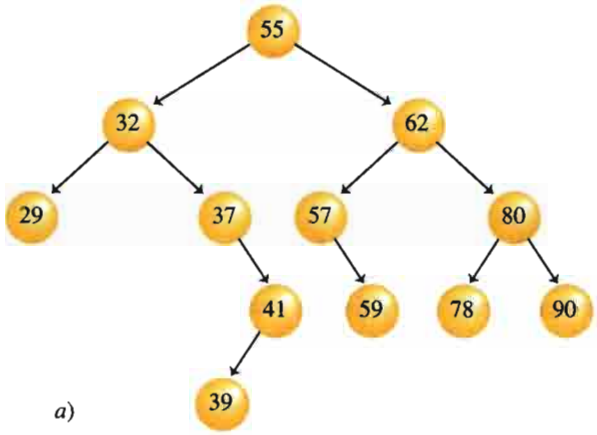
15. Dado el siguiente árbol binario de búsqueda, elimine las claves

49 - 37 - 62 - 90 - 78

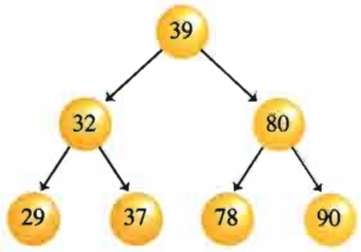


16. Verifique si el árbol binario de búsqueda del diagrama del inciso *a*) queda igual al del diagrama del inciso *b*), luego de eliminar las claves

55 - 62 - 57 - 59 - 41



a)



b)

Ejercicios de programación en árboles binarios

17. Escriba un programa que calcule e imprima cuántos nodos tiene un árbol binario.

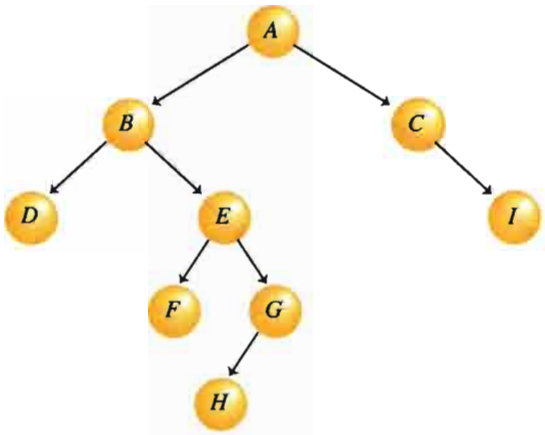
18. Escriba un programa que calcule e imprima el total de nodos internos que tiene un árbol binario.

19. Considerando que un árbol binario almacena números enteros, encuentre e imprima el máximo valor guardado en el árbol y el promedio de los mismos.

20. Escriba un procedimiento que realice lo siguiente:

a) Imprima la información almacenada en las hojas de un árbol binario.
b) Imprima la información almacenada en los nodos internos de un árbol binario.

21. Dado el siguiente árbol binario:



Escriba un programa que imprima los nodos del mismo de la siguiente forma:

- A - - B - - - D - - - E - - - - F - - - - G - - - - - H - - C - - - I

22. Dado el árbol binarlo del ejercicio 21, escriba un programa que imprima los nodos del mismo de la siguiente forma:

- A - - B - - C - - - D - - - E - - - I - - - - F - - - - G - - - - - H

23. Escriba un procedimiento que visite los nodos de un árbol binario de la siguiente forma:

- Raíz
- Rama derecha
- Rama izquierda

24. Escriba tres procedimientos que efectúen los recorridos en preorden, inorden y posorden en forma iterativa en lugar de recursiva, para lo cual se puede apoyar en una pila.

25. Escriba un procedimiento que elimine todas las hojas de un árbol binario.

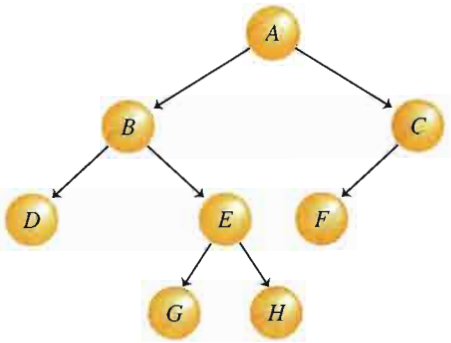
26. Escriba un programa que cargue los nodos de un árbol binario en un arreglo unidimensional. Cuide que se mantenga la relación padre-hijo entre los nodos.

27. Escriba una función que determine si dos árboles binarios son **similares**.

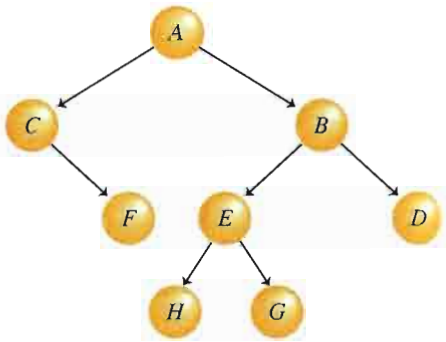
28. Escriba una función que determine si dos árboles binarios son **equivalentes**.

29. Escriba un procedimiento que intercambie los subárboles izquierdo y derecho de un árbol binario. Es de observar que este intercambio se debe realizar para todo nodo del árbol.

Ejemplo: Dado el árbol binario del diagrama del inciso *a*), el intercambio de ramas produce el árbol del diagrama del inciso *b*).

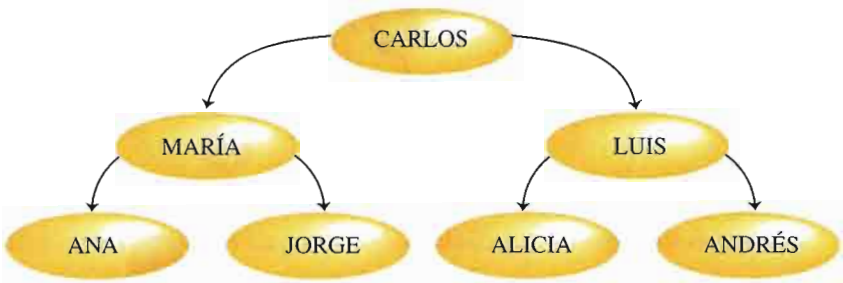


a)



b)

30. Se tiene almacenada toda la ascendencia de Carlos en un árbol binario. Se ha seguido el siguiente criterio para Carlos y todos sus progenitores: en la rama izquierda se ha guardado el nombre de la madre y en la rama derecha el nombre del padre. Observe la figura que se muestra a continuación.



Escriba un subprograma que imprima el nombre de todos los **progenitores femeninos** de Carlos.

31. Retome el problema anterior. Agregue una función que pueda insertar al árbol genealógico de Carlos tanto ascendientes femeninos como masculinos.

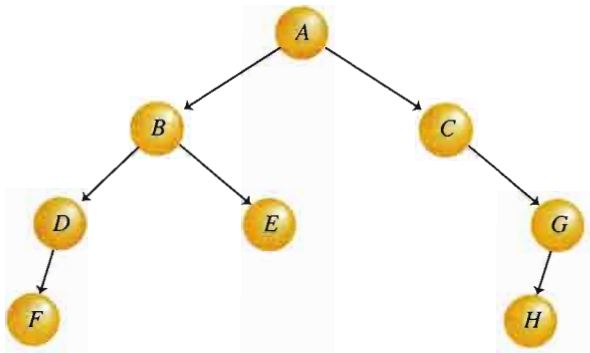
32. Defina la clase *Árbol binario* utilizando algún lenguaje de programación orientado a objetos, tomando como base para programar los métodos los algoritmos estudiados en este capítulo.

33. Retome el problema anterior. Agregue a la clase un método que muestre el contenido de un nodo.

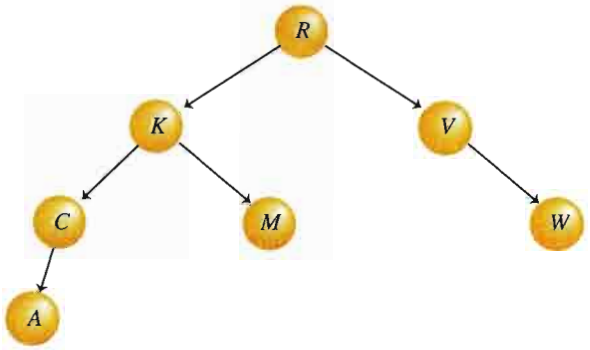
34. Escriba un programa de aplicación que dados dos objetos de la clase *Árbol binario*, previamente definida, imprima un mensaje adecuado según los mismos sean equivalentes o no. Determine si requiere definir nuevos métodos a la clase.

Árboles balanceados

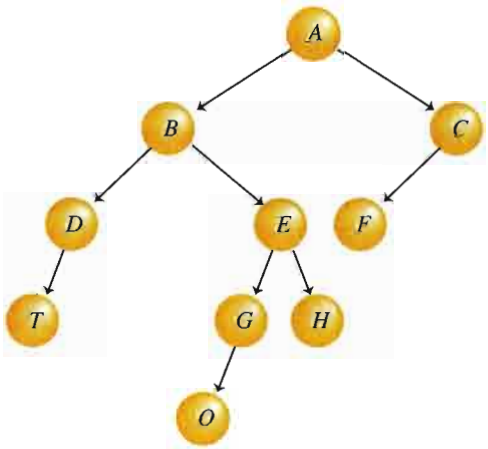
35. Determine si los siguientes árboles binarios son árboles balanceados.



a)



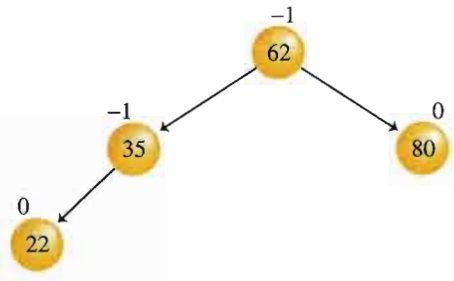
b)



c)

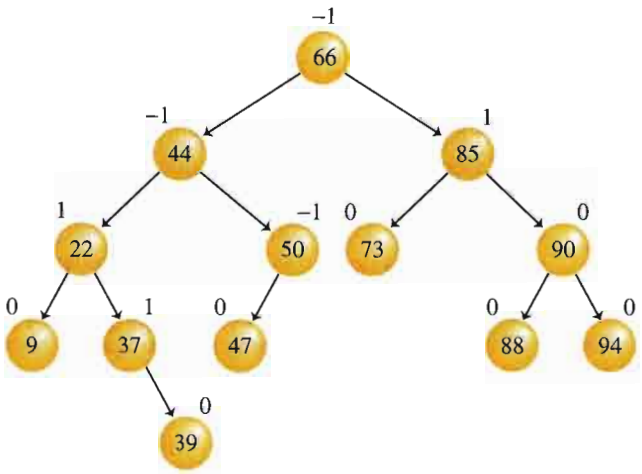
36. Calcule cuál es el máximo número de nodos de un árbol balanceado de altura 13. ¿Cuál es el mínimo?

37. Inserte las claves 10 - 47 - 38 - 06 - 55 - 90 - 49 en el árbol balanceado que se da a continuación.

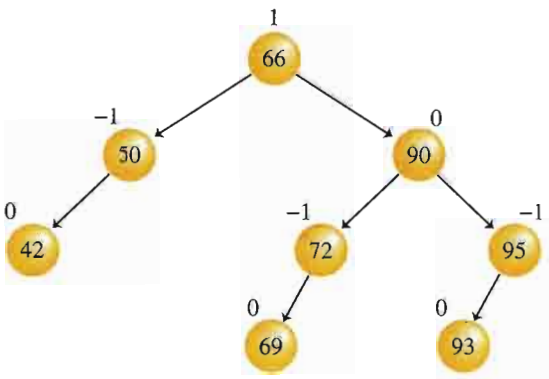


38. Elimine las siguientes claves del árbol balanceado del siguiente diagrama:

73 - 66 - 50 - 47 - 39 - 94



39. Escriba las instrucciones necesarias para equilibrar el árbol balanceado del siguiente diagrama, luego de eliminar la clave 50.



Árboles-B y árboles-B^+

40. ¿Cuál es el número de claves (m) que puede tener como máximo un árbol-B de orden 50 y tres niveles? ¿Cuál el de un árbol-B^+?

41. ¿Cuál es el número más pequeño de claves que, al ser insertadas, provocaría que un árbol-B de orden 50 tuviera tres niveles?

42. ¿Cuál es el número más pequeño de claves que, al ser insertadas, provocaría que un árbol-B de orden 100 tuviera cuatro niveles?

43. Un árbol-B de orden 100 y tres niveles tiene 5 800 000 claves. ¿Cuántas claves se podrían eliminar del árbol sin que éste tenga que disminuir su altura?

44. Realice los tres ejercicios anteriores, pero ahora aplicados a árboles-B^+.

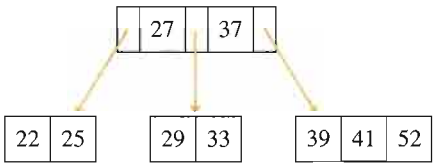
45. Inserte las siguientes claves:

08 - 77 - 36 - 68 - 90 - 37 - 41 - 52 - 57 - 13 - 30 - 28 - 24 - 86

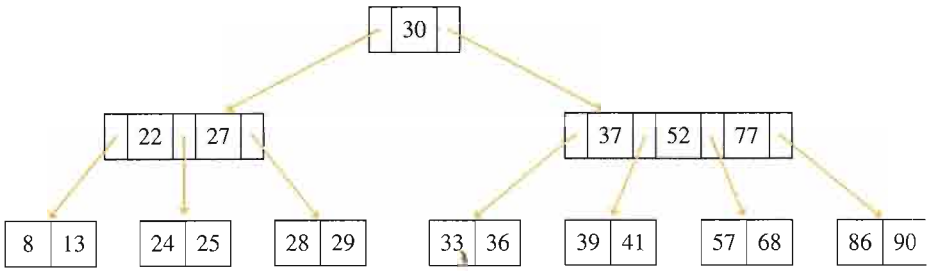
a) En un árbol-B de orden 2 que se encuentra vacío.
b) En un árbol-B^+ de orden 2 que se encuentra vacío.

46. Verifique si el árbol-B de orden 2 del diagrama del inciso *a*) queda igual al del diagrama del inciso *b*), luego de insertar las siguientes claves:

90 - 08 - 77 - 68 - 36 - 30 - 13 - 57 - 24 - 28 - 86



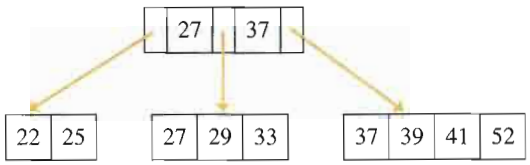
a)



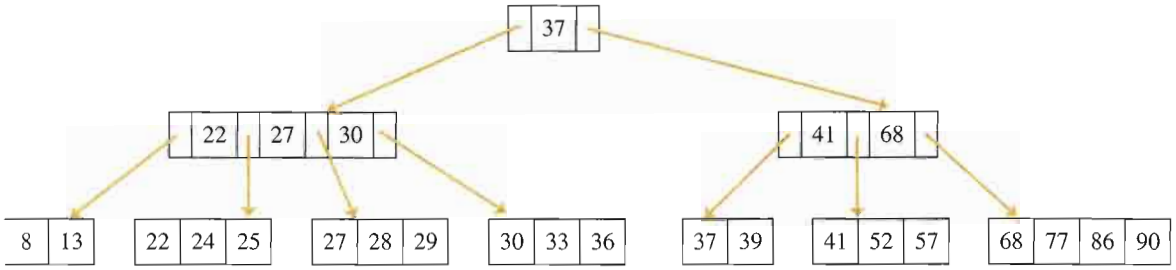
b)

47. Verifique si el árbol-B^+ de orden 2 del diagrama del inciso *a*) queda igual al del diagrama del inciso *b*), luego de insertar las siguientes claves:

90 - 08 - 77 - 68 - 36 - 30 - 13 - 57 - 24 - 28 - 86



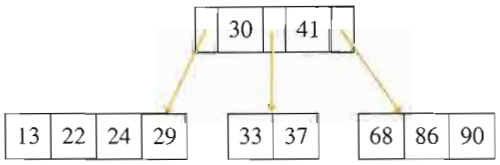
a)



b)

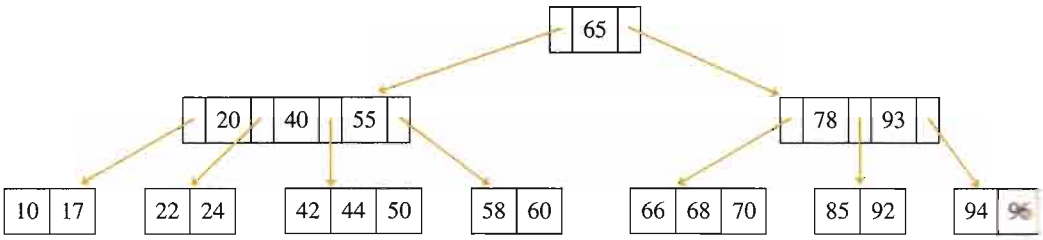
48. Verifique si el árbol-*B* de orden 2 del diagrama del inciso *b*) del ejercicio 46 queda igual al árbol del siguiente diagrama, luego de eliminar las claves:

52 - 77 - 36 - 08 - 57 - 39 - 28 - 25 - 27

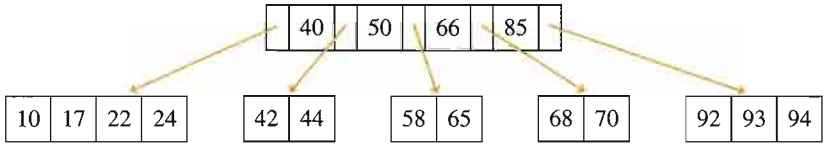


49. Verifique si el árbol *B* que se presenta en el inciso *a*) queda igual al árbol del diagrama del inciso *b*), luego de eliminar las siguientes claves:

96 - 55 - 60 - 20 - 78



a)



b)

Ejercicios de programación de árboles-*B* y árboles-*B*+

50. Escriba los subprogramas de inserción y eliminación en árboles-*B*.

51. Escriba los subprogramas de inserción y eliminación en árboles-B^+.