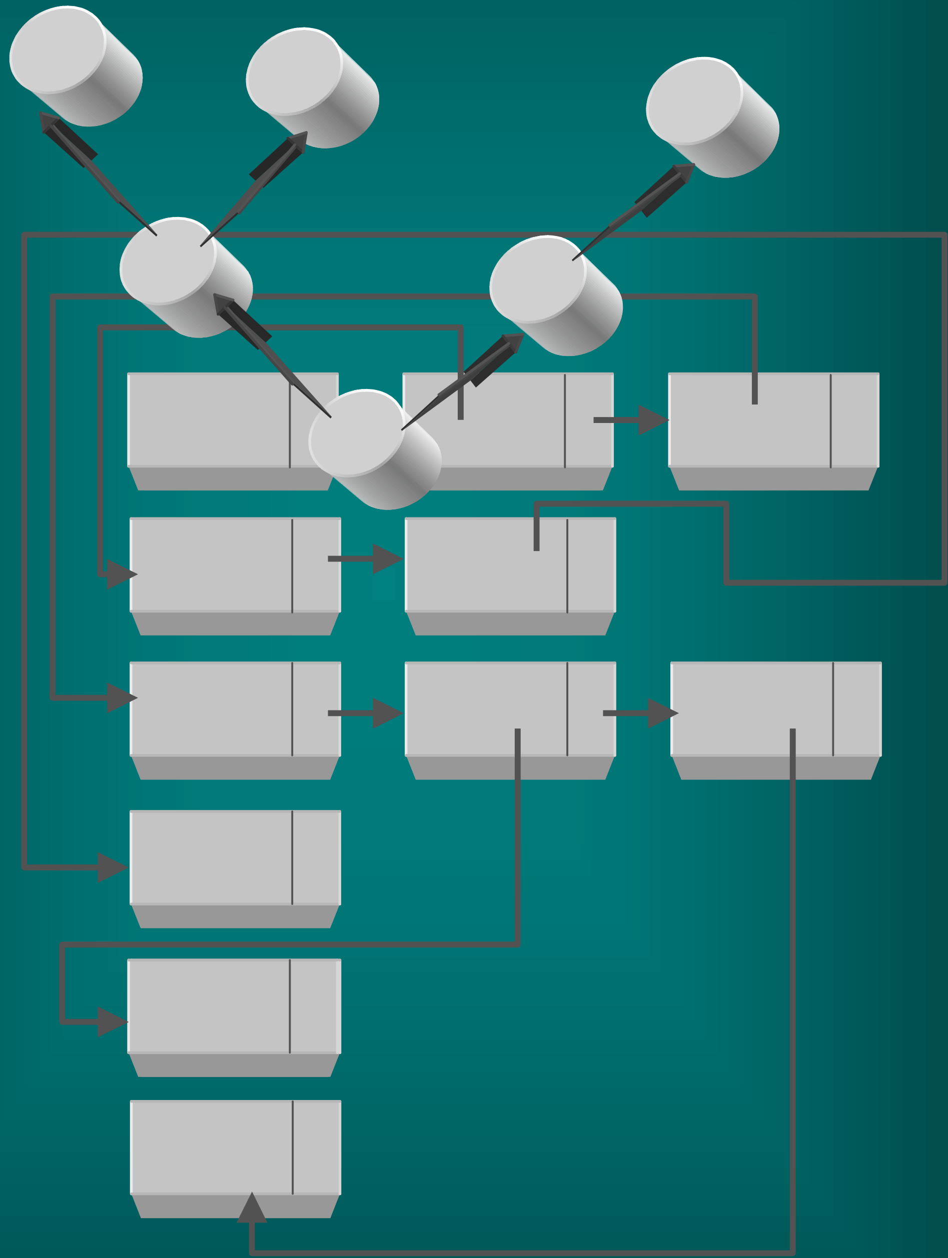


Estructura de Datos 2

Antología de la Materia

© Instituto Tecnológico Superior de Cajeme
Carretera Internacional Km. 2
Teléfono (644) 415 19 15, 415 19 14 y 415 18 80
E-mail: Itesca@itesca.edu.mx

Alrededor de un fanático de la programación,
existe una gran familia que ha tenido mucha paciencia
les dedico este libro a cada uno de los integrantes
y a todos lo que hiciera posible la creación del mismo

Índice

Agradecimientos i
Indice ii
Prefacio iv

CAPÍTULO 1

Árboles Binarios
Terminología 2
Longitud de Caminos 4
Longitud de Camino Interno 4
Longitud de Camino Externo 5
Árbol Binario 7
Representación grafica de un árbol general a binario. 9
Representación en Memoria 11
Recorrido de Árbol Binario 13
Recorrido Enorden 14
Recorrido Preorden 15
Recorrido Postorden 16
Árboles Hehebrados 16
Árboles en Montón 18
Creación de un Árbol Binario 20
Árbol Binario de Búsqueda 21
Operaciones en Árboles Binarios de Búsqueda 22
Búsqueda 22
Inserción 23
Eliminación 25
Ejercicios 28
Problemas 29

CAPÍTULO 2

Árboles Balanceados
Definición 32
Creación de un Árbol Equilibrado de N Claves 33

Factor de Equilibrio 34
Caso Para la Inserción de un Nuevo Nodo 36
Reestructuración del Árbol Balanceado (Rotación) 38
Formación de un Árbol Balanceado 40
Procedimiento de Inserción por Balanceo 48
Eliminación de un Nodo en un Árbol Balanceado 50
Ejercicios 59
Problemas 59

CAPÍTULO 3

Árboles Multicaminos
Árboles – B 63
  Representación de un árbol B de orden m 63
  Búsqueda 64
Proceso de Creación en un Árbol B 68
Operaciones de un árbol B 70
  Inserción 71
  Borrado 76
Árboles – B+ 84
Ejercicios 87
Problemas 87

CAPÍTULO 4

Grafos 89
Definición y Terminología 90
Representación de un grafo 94
  Matriz de Adyacencia 94
  Lista de Adyacencia 96
Operaciones de Grafo 98
Exploración de Grafos 102
Métodos de Recorrido de Grafos 103
  Recorrido de Profundidad (Depth-First) 104
  Recorrido Primero en Amplitud(Breadth-First) 107
Búsquedas 110
  Búsqueda en Profundidad 110
  Búsqueda en amplitud o anchura 111
Matriz de camino, Cierre transitivo 111
  Camino entre par de vértices 111
  Matriz de caminos 114

Cierre Transitivo 115
Árbol de cobertura Mínimo (MST) 116
Ejercicios 117
Problemas 118

Bibliografía

Prefacio

Este es una recopilación de una serie de autores de prestigiado renombre. Realizándose por la necesidad de que el estudiante tenga en sus manos la información necesaria para poder sobresalir en la materia de estructuras de datos.

En este podrá conocer todos lo referente a los diferentes tipos de estructuras de datos existentes (árboles y grafos), utilizando los diferentes tipos de algoritmos mostrados a lo largo de este libro.

Una recomendación para la persona que lea este, es que no debe esperar que el manual le proporcionara ungüentos milagrosos que por arte de magia traspase el conocimiento. El estudio de las diferentes estructuras a través de esta antología no será sino un examen de una particular forma de ver las cosas, en su representación y lo demás es tarea del programador. Es decir, uno puede a prender a construir un árbol o un grafo... pero lo interesante, de esto es que la persona sepa utilizar estas estructuras de una manera inteligente, para poder lograr que las aplicaciones tengan sentido, se vaya adquiriendo una serie de habilidades y experiencias para la programación respecto al lenguaje utilizado y de sus recursos. En fin el uso de este manual depende en su totalidad al alumno y no a nadie más.

Capítulo

1

Árboles Binarios

Esta unidad se centra la atención sobre una estructura de datos, *el árbol*, cuyo uso esta muy extendida es muy útil en numerosas aplicaciones. Se define formas de esta estructura de datos (árboles generales, árboles binarios y árboles binarios de búsqueda) y como se pueden representar en cualquier lenguaje como C ó Pascal, así como el método para su aplicación en la resolución de una amplia variedad de problemas. Al igual que a sucedido anteriormente con las listas, los árboles se tratan principalmente como estructuras de datos en lugar de cómo tipos de datos. Es decir, nos centramos principalmente en los algoritmos e implementaciones en lugar de en definiciones matemáticas.

Los árboles junto a los grafos constituyen estructuras no lineales. Las listas enlazadas tienen grandes ventajas o flexibilidad sobre la representación contigua de estructuras de datos(Arreglos), pero tiene gran debilidad: son listas secuénciales; es decir, están dispuestas de modo que es necesario moverse a través de ellas, una posición cada vez.

Los árboles Superan esa desventaja utilizando los métodos de puntero y listas enlazadas para su implementación. Las estructuras de datos organizadas como árboles serán muy valiosos en una gran gama grande de aplicaciones, sobre todo problema de recuperación de información.

Terminología

Un árbol es una estructura que organiza sus elementos, denominados nodos, formando jerarquías. Los científicos los utilizan los árboles generales para representar relaciones. Fundamentalmente, la relación clave es la de «padre-hijo» entre los nodos del árbol. Si existen una arista (rama) dirigida del nodo *n* al nodo *m*, entonces *n* es **padre** de *m* y *m* es **hijo** de *n*.

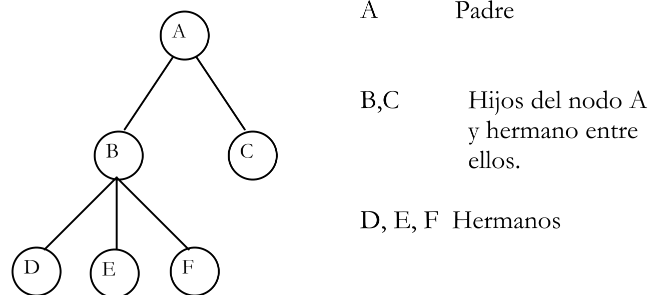


Figura 1. Árbol General.

Cada nodo de un árbol tiene al menos un padre y existe un único nodo, denominado **raíz** del árbol, que no tiene padre. El nodo A es la raíz del árbol. Un nodo que no tiene hijos se llama **hoja** (terminal) del árbol. Las hojas del árbol de la figura 1 son C,D,E,F. Todo nodo que no es raíz, ni hoja es conocido como **interior** como es B**.**

La relación Padre-Hijo entre los nodos se generaliza en las relaciones **ascendentes**(antecesor) y **descendiente**. En la Fig.1 A es un antecesor de D y por consiguiente D es un descendiente de A. Un **subárbol** de un árbol es cualquier nodo del árbol junto con todos sus descendientes como en la figura 1, B y sus descendientes son un subárbol. El **grado** (aridad) es él numero de hijos del nodo. La aridad de un árbol se define como el máximo de la aridad de sus nodos.

Se le denomina **camino** a la secuencia de nodos conectados dentro de un árbol, donde la **rama** es un camino que termina en una hoja. Para determinar el **nivel** de un nodo, se calcula por medio de los nodos que se encuentra entre él y la raíz, por consiguiente el nivel de un árbol es el numero de nodos que se encuentran entre la raíz y la hoja mas profunda; al máximo nivel de un árbol es también conocido como **Altura**.

Ejemplo 1.1

Dado el árbol general de la figura 1.2, se hace sobre el las siguientes consideraciones.

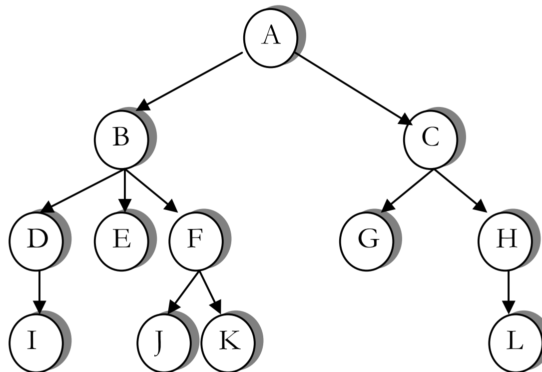


Figura 1.2 Árbol General

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
   H es hijo de L
3. A es padre de B
   B es padre de D
   D es padre de I
   C es padre de G
   L es padre de H
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 nodo interiores.
7. Los grados del nodo A es 2
   Los grados del nodo B es 3
   El grados del nodo C es 2
   El grados del nodo D es 1
   El grados del nodo E es 0

   El grados del árbol es 3
8. El nivel de D es 1
   El nivel de B es 2
   El nivel de D es 3
   El nivel de C es 2
   El nivel de L es 4
9. La altura del árbol es 4

Debido a su naturaleza jerárquica de los árboles se puede utilizar para representar en formaciones que sean jerárquicas por naturaleza, por ejemplo diagramas de organizaciones, árboles genealógicos, árbol de la especie humana, etc

Longitud de Camino

Se define la longitud de camino X como el numero de arcos que debe ser recorridos para llegar desde la raíz al nodo X. Por definición la raíz tiene longitud de 1, sus descendientes directos longitud de camino 2 y así sucesivamente. Considérese la figura 1.22 el nodo B tiene la longitud de camino 2, el nodo I longitud de camino 4 y el nodo Longitud de camino 3.

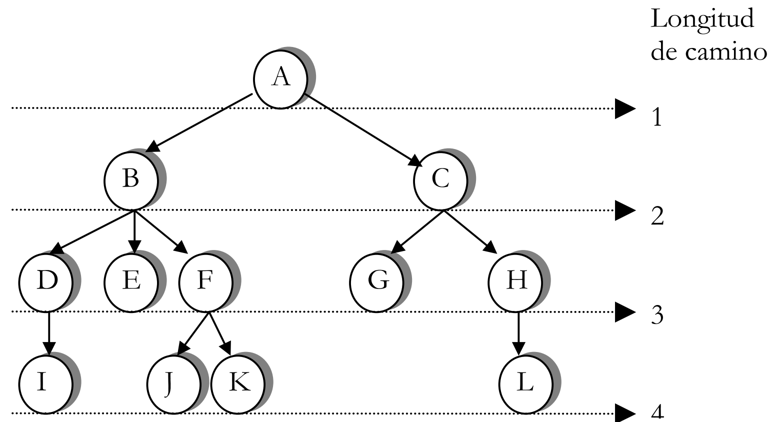


Figura 1.22 longitud de Caminos en un árbol

Longitud de Camino Interno

La longitud de camino interno es la suma de las longitudes de camino de todos los nodos del árbol. Puede calcularse por medio de las siguientes formula:

$$LCI = \sum_{i=1}^{h} n_i * i$$

donde i representa el nivel del árbol, h su altura y n_i el numero de nodos en el nivel i. La LCI del árbol de la figura 1.22 se calcula así:

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

Ahora bien, la media de la longitud de camino interno (LCIM) se calcula dividiendo la LCI entre el numero de nodos del árbol (n). Se expresa:

$$LCIM = \frac{LCI}{n}$$

y significa el numero de arcos que deben ser recorridos en un promedio para llegar, partiendo desde la raíz, a un nodo cualquiera del árbol.

La LCIM del árbol de la figura 1.22 se calcula mediante:

LCIM= 36/12=3

Longitud de Camino Externo

Para definir la longitud de camino externo es necesario primero definir los conceptos **árbol extendido y nodo especial.** Un árbol extendido es aquel en el que el numero de hijos de cada nodo es igual al grado del árbol. Si alguno de los nodos del árbol no cumple con esta condición entonces debe incorporársele al mismo nodo especial; tanto como sea necesario satisfacer la condición. Los nodos especiales tienen como objetivo remplazar las ramas vacías o nulas, no pueden tener descendientes y normalmente se representan con la forma de un cuadrado. En la figura 1.23 se presenta el árbol extendido de la figura 1.22.

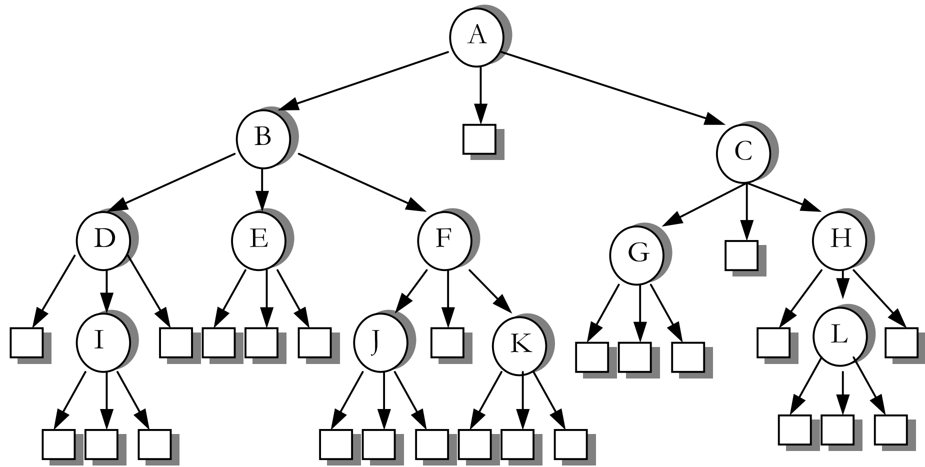


Figura 1.23 Árbol Extendido

El numero de nodos especiales de este nodo es de 25. se puede definir ahora la longitud de camino externo como la suma de las longitudes de caminos externo como la suma de las longitudes de caminos de todos los nodos especiales del árbol. Se calcula por medio de la siguiente formula:

$$LCE = \sum_{i=2}^{h+1} ne_i * i$$

donde *i* representa el nivel del árbol ,h su altura y ne_i el numero de nodos especiales en el nivel *i*. Obsérvese que *i* comienza desde 2, puesto que la raíz se encuentra en el nivel 1 y no puede ser nodo especial.

La LCE del árbol de la figura 1.23 se calcula de la siguiente manera:

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

Ahora bien la media de la longitud d camino externo (LCEM)se calcula dividiendo LCE entre el numero de nodos especiales del árbol (ne). Se expresa:

$$LCEM = \frac{LCE}{ne}$$

y significa el numero de arcos que debe ser recorridos en promedio para llegar partiendo de la raíz, a un nodo especial cualquiera del árbol.

$$LCEM=109/25=4.36$$

Ejemplo:

Dado el árbol general de la figura 1.24 y el árbol extendido de la figura 1.25 se calcula:

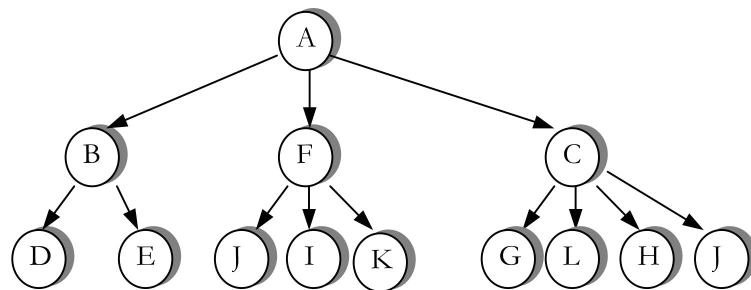


Figura 1.24 Árbol General

La longitud de camino interno:

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

La media de la longitud de camino interno:

$$LCIM = 34 / 13 = 2.61$$

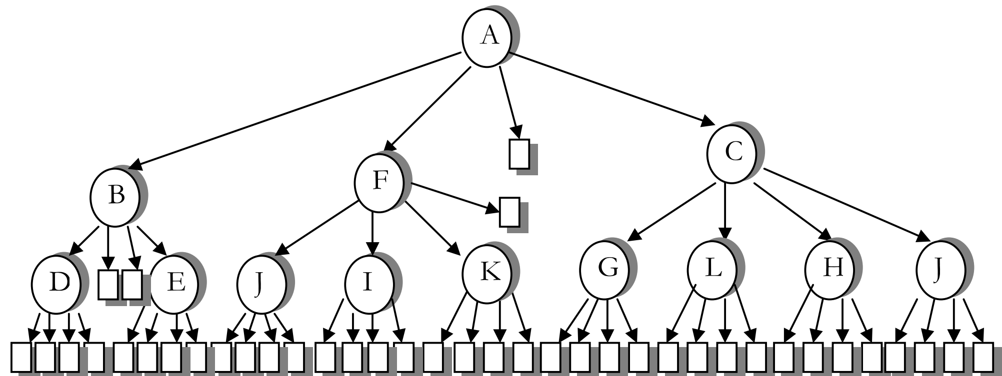


Figura 1.25 Árbol Extendido

La longitud de camino externo:

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

Y la media de la longitud de camino externo:

LCEM = 155 / 40 = 3.87

Árboles Binarios

Un árbol ordenado es aquel en el que las ramas de los nodos del árbol están ordenadas. Los árboles ordenados de grado 2 son de especial interés puesto que representan una de las estructuras de datos más importantes en computación, conocidas como **árboles binarios.**

En un árbol binario cada nodo puede tener como máximo dos subárboles y siempre es necesario distinguir entre el subárbol izquierdo y el subárbol derecho.

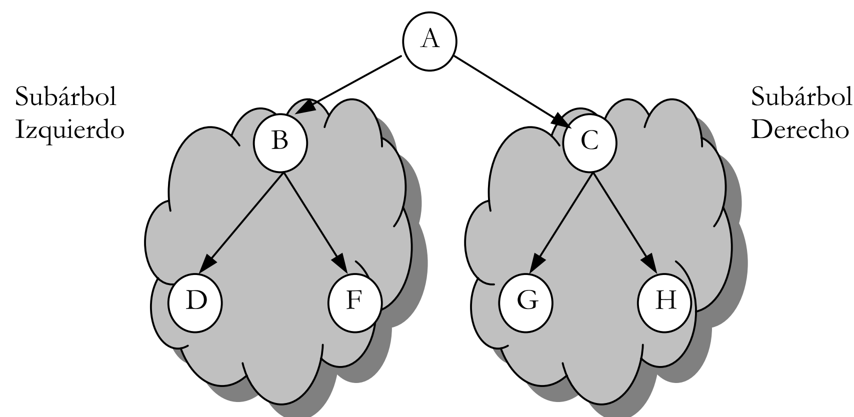


Figura 1.3 Subárboles de un árbol binario

Formalmente podemos definir un árbol binario de tipo T como una estructura homogénea que es la concatenación de un elemento de tipo T, llamada raíz, con dos árboles binarios disjuntos. Una forma particular de árbol binario puede ser la estructura vacía.

Los árboles binarios se clasifican en cuatro tipo que son : distintos, similares, equivalentes y completos. Cuando dos árboles binarios se dice que son **similares** si tiene la misma estructura y son **equivalentes** si son similares y contienen la misma información. En caso contrario se dice que estos árboles son **distintos.** Un árbol binario esta equilibrado si la altura de los dos subárboles de cada nodo del árbol se diferencia en una unidad como máximo.

Altura(Subárbol izquierdo) - Altura(Subárbol derecho) ≤ 1

El procedimiento de arbol binarios equilibrados es mas sencillo que los árboles no equilibrados.

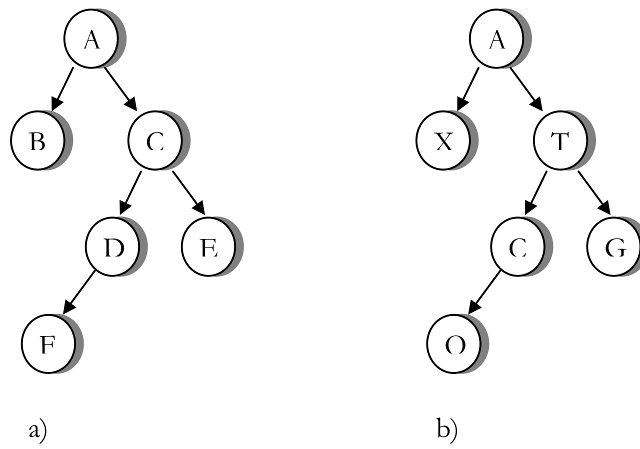
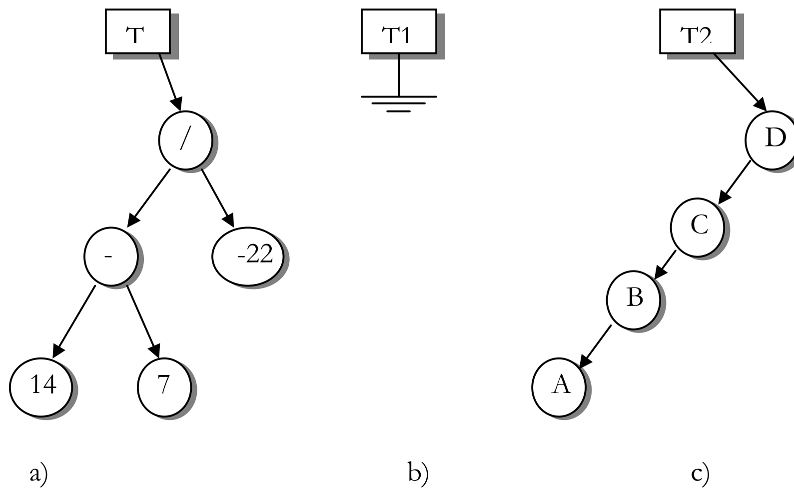
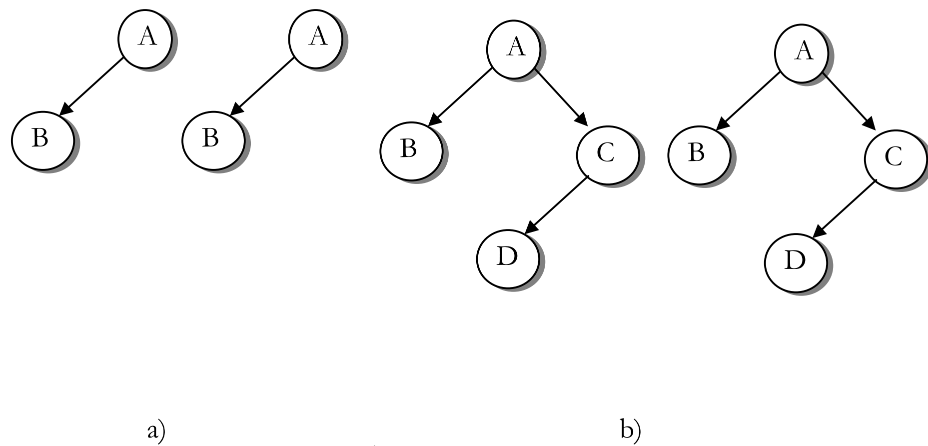


Figura 1.4 Árboles binarios Similares



a) b) c)
figura 1.5 Árboles binarios de diferentes alturas: (a) altura 3, (b) árbol vació altura 0, (c) altura 4



a) b)
Figura 1.6 Árboles binarios equivalentes

Se define un árbol **completo** (lleno) como un árbol en el que todos sus nodos, excepto los del ultimo nivel, tienen dos hijos ; el subárbol izquierdo y el subárbol derecho.

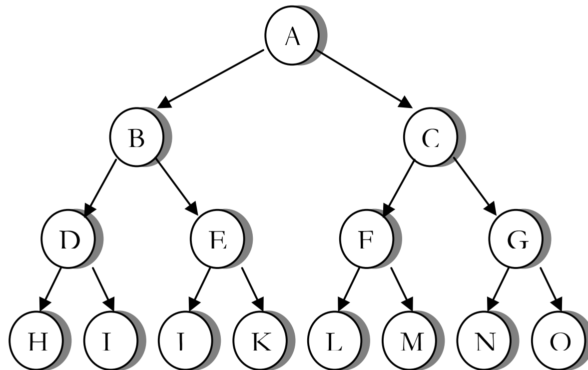


Figura 1.7 Árbol binario Completo de altura 4

Se puede calcular el numero de nodos de un árbol binario completo de altura h, aplicando la siguiente formula

$$\text{Numero de nodos}_{AB} = 2^h - 1$$

Representación grafica de un árbol general a binario.

Los árboles binarios representa una de las estructuras de datos mas importantes en la computación. Esto por su dinamismo, la no-linealidad entre sus elementos y por su sencilla programación. En lugar que en los árboles generales ya es imprescindible deducir cuantas ramas o caminos se desprenden de un nodo en un momento dado. Por ello y dado que los árboles binarios siempre se cuelgan como máximo de dos subárboles su programación es mas sencilla.

Afortunadamente existe una técnica sencilla para la conversión de un árbol general a formato de árbol binario. El algoritmo de conversión tiene tres pasos fáciles:

1. Debe enlazarse los hijos de cada nodo en forma horizontal(los hermanos)

2. Debe enlazarse en forma vertical el nodo padre con el hijo que se encuentra mas a la izquierda. Además debe eliminarse el vinculo de ese padre con el resto de sus hijos.

3. Debe rotarse el diagrama resultante, aproximadamente 45 grados a la izquierda y así se obtendrá el árbol binario correspondiente.

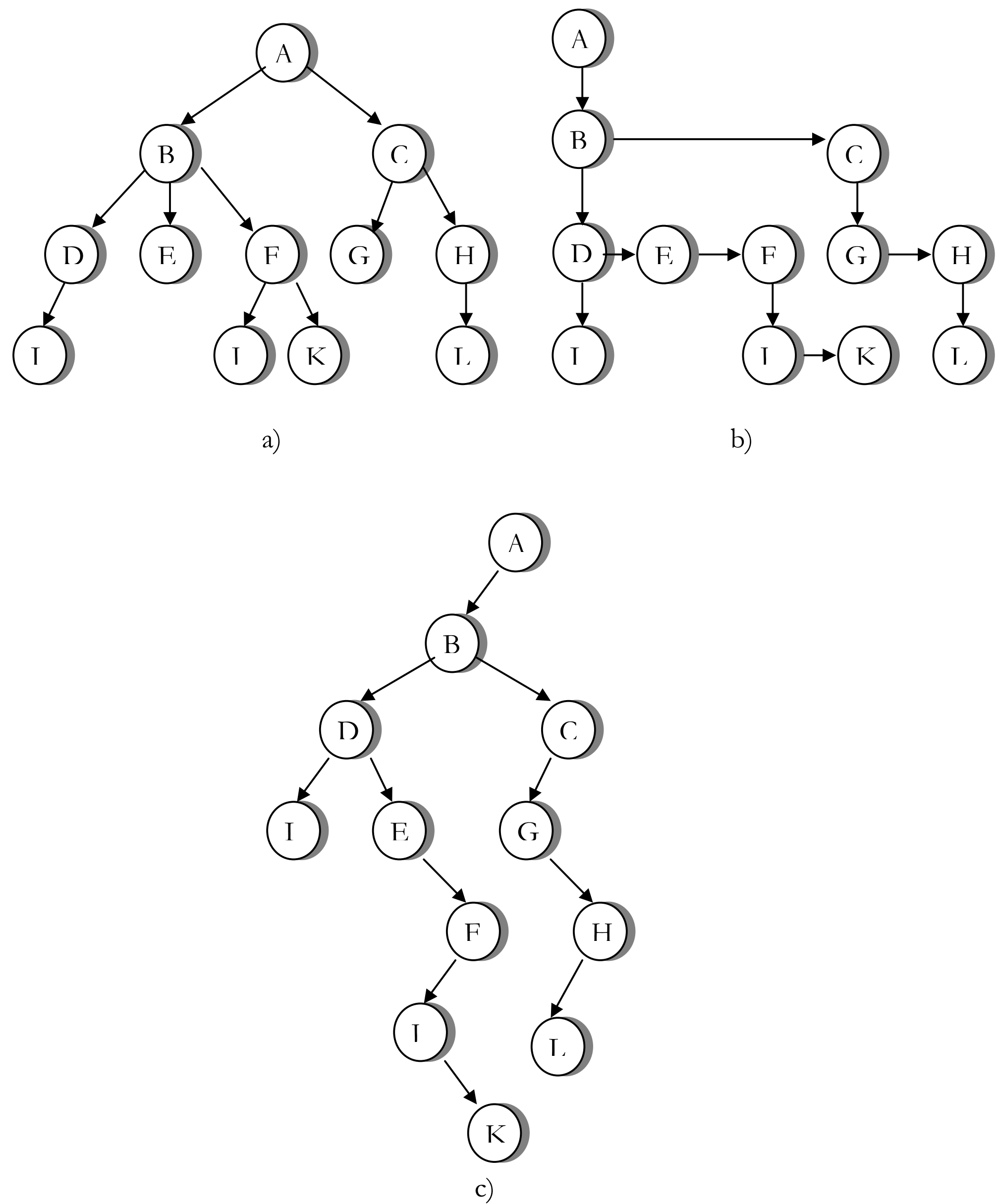


Figura 1.8 Conversión de un árbol general en un árbol binario. a) Árbol general. b) Árbol binario luego de aplicar los pasos 1 y 2. c) Árbol binario luego de aplicar el paso 3.

Representación en memoria

Los árboles binarios pueden ser representados de modos diferentes :

- Mediante punteros (memoria dinámica)
- Mediante arreglos (memoria estática)

Sin embargo se utilizará y se explicará la primera forma, puesto que es la mas natural para tratar este tipo de estructuras. los nodos del árbol binario serán representados como registros, que contendrán como mínimo tres campos. En un campo almacenara la información del nodo. Los dos restantes se utilizaran para apuntar a los subárboles. Dado el siguiente nodo T:

IZQ	INFO	DER

Este tiene tres campos:

IZQ: campo donde se almacena la dirección del subárbol izquierdo del nodo T.

INFO: campo donde se almacena la información de interés del nodo.

DER: campo donde se almacena la dirección del subárbol derecho del nodo T.

La definición del árbol binario en lenguaje algorítmico es como sigue:

```
Enlace = ^nodo
Nodo = registro
            Izq: tipo enlace
            Info: tipo Dato
            Der: tipo enlace
 {fin de la definición}
```

Nota:

Se utiliza el símbolo ^para representar el concepto de dato tipo puntero

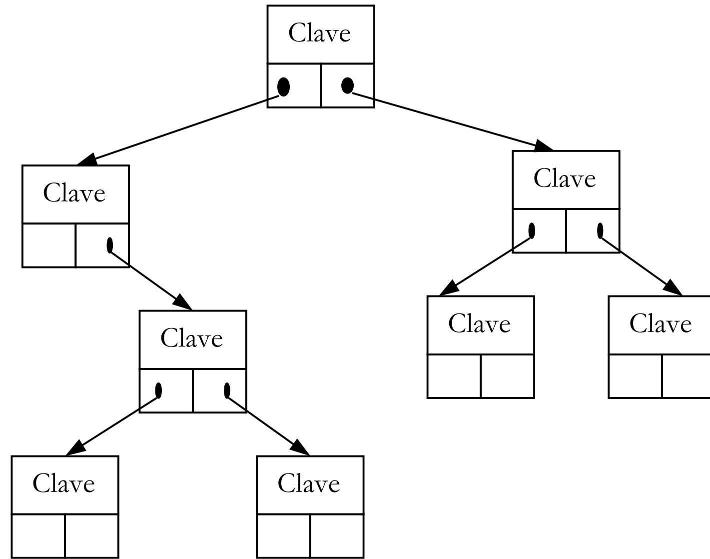


Figura 1.12 Representación de un árbol de punteros.

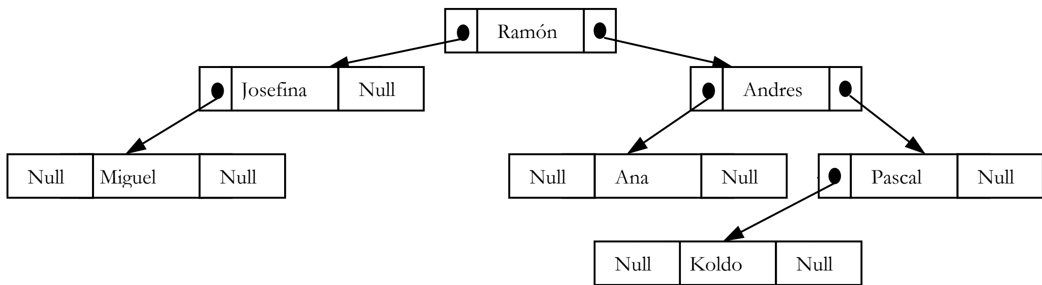


Figura 1.13 Representación de un árbol con una estructura de punteros

Otro método de representación , y uno de los mas fáciles es mediante tres arreglos lineal es paralelos que contemplan el campo de la información y los dos punteros de ambos subárboles. Utilizando el mismo método anterior

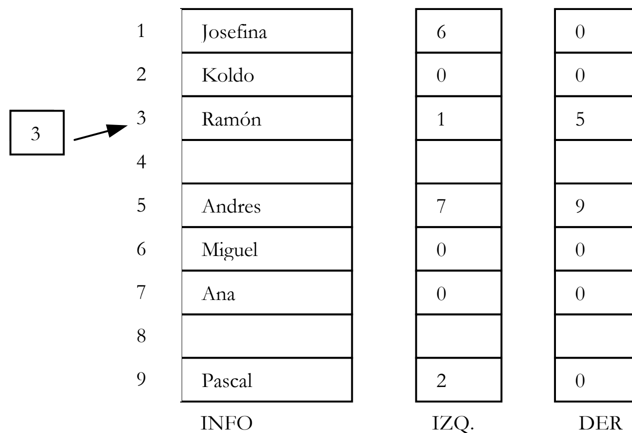


Figura 1.14 árbol binario como arreglo

Otra manera representar por medio de arreglos es utilizando el algoritmo siguiente:

La raíz del árbol se guarda en ARBOL[1]
Si un nodo n esta en ARBOL[i] entonces
*Su hijo izquierdo se pone en ARBOL[2*i]*
*Y su hijo derecho en ARBOL[2*i+1]*
Si un subárbol esta vació, se le da el valor de NULL

La raíz del árbol se guarda en ARBOL[1]Este sistema requiere mas posición de memoria que nodos tiene el árbol. Así la transformación necesita un arreglo con 2^{h+2} elementos, si el árbol tiene una profundidad h. Utilizando el ejemplo anterior que tiene una altura de 3requerirá 32 posiciones.

Arreglo
Árbol

1	Ramón
2	Josefina
3	Andrés
4	Miguel
5	NULL
6	Ana
7	Pascal
8	NULL
9	NULL
10	
11	
12	NULL
13	NULL
14	Koldo
15	NULL

16	
17	
18	
....	
28	NULL
29	NULL
30	
31	
32	

Figura 1.15 árbol binario con un solo arreglo

Recorrido de un Árbol Binario

Una de las operaciones más importantes a realizar en un árbol binario es el recorrido de los mismos. Recorrer significa visitar los nodos del árbol en forma sistemática; de tal manera que todos los nodos del mismo sean visitados una sola vez. Existen tres formas diferentes de efectuar el recorrido de un árbol binario:

1. visitar el nodo (N)

2. Recorrer el subárbol izquierdo(I)

3. Recorrer el subárbol derecho(D)

Según sea la estrategia a seguir los recorridos se conocen como **enorden** (Inorder), **preorden** (preorder) y **postorden** (postorder). Cada uno de estos son de naturaleza recursiva.

preorden (nodo-izdo-dcho) (**NID**)

enorden (izdo-nodo-dcho) (**IND**)

postorden (izdo-dcho-nodo) (**IDN**)

Recorrido enorden

Si el árbol no esta vacío, el método implica los siguientes pasos:

1. Recorre el subárbol izquierdo (I)
2. visita el nodo raíz (N)
3. Recorre el subárbol derecho (D)

El algoritmo correspondientes es .

```
ENORDEN(NODO)
Inicio
    Si NODO ≠ NULL entonces
        ENORDEN( NODO^.IZQ)
        Visualizar NODO^.INFO
        ENORDEN( NODO^.DER)
    FIN SI
FIN
```

El árbol de la figura 1.16los nodos se han numerado en el orden en el que son visitados durante el recorrido enorden. El primer subárbol recorrido es el izquierdo del nodo raíz. Este subárbol consta de los nodos B, D y E, que es a su vez otro árbol con el nodo B como raíz, por lo que siguiendo el orden IND ese visita primero D, a continuación B y por ultimo E (derecha).después de visitar el subárbol izquierdo se visita el nodo raíz A y por ultimo se visita el subárbol derecho que consta de los nodos C, F y G. Y continua con el orden IND para el subárbol derecho.

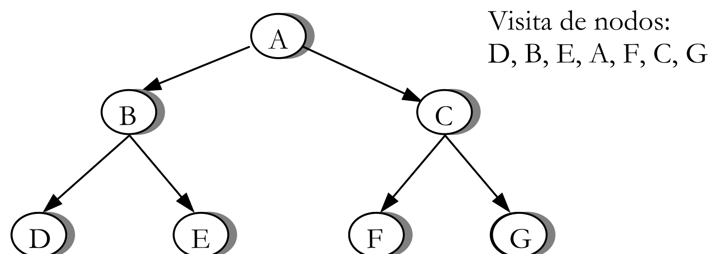


Figura 1.16. Recorrido de ENORDEN árbol binario

Recorrido preorden

El recorrido preorden (NID) conlleva los siguientes pasos:

1. visitar el nodo (N)
2. Recorrer el subárbol izquierdo(I)
3. Recorrer el subárbol derecho(D)

El algoritmo correspondiente es:

```
PREORDEN(NODO)
Inicio
    Si NODO ≠ NULL entonces
        Visualizar NODO^.INFO
        PREORDEN( NODO^.IZQ)
        PREORDEN( NODO^.DER)
    FIN SI
FIN
```

Si utilizamos el recorrido preorden del árbol de la figura 1.17 se visita primero el raíz (nodo A). A continuación se visita el subárbol, que consta de los nodos B, D y E. Dado que el subárbol es a su vez un árbol, se visitan los nodos utilizando el orden NID. Por consiguiente, se visitan primero el nodo B, después D (izquierdo) y por ultimo E (derecho).

A continuación se visita subárbol derecho A, que es un árbol que contiene los nodos C, F y G. De nuevo siguiendo el orden NID, se visita primero el nodo C, a continuación F y por ultimo G.

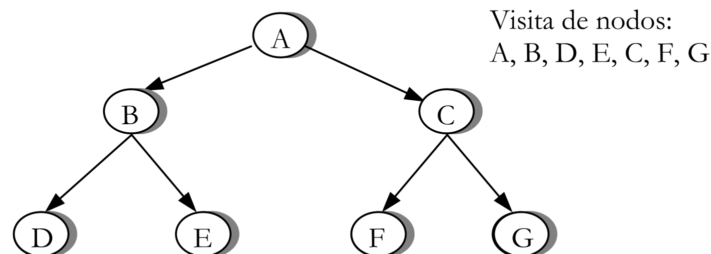


Figura 1.17. Recorrido de PREORDEN árbol binario

Recorrido postorden

El recorrido del postorden (IDN) realiza los pasos siguientes:

1. Recorrer el subárbol izquierdo(I)
2. Recorrer el subárbol derecho(D)
3. visitar el nodo (N)

El algoritmo correspondiente es:

```
POSTORDEN(NODO)
Inicio
    Si NODO ≠ NULL entonces
        POSTORDEN ( NODO^.IZQ)
        POSTORDEN ( NODO^.DER)
        Visualizar NODO^.INFO
    FIN SI
FIN
```

Si se utiliza el recorrido postorden del árbol de la figura 1.18, se visitara primero el subárbol izquierdo A. Este subárbol consta de los nodos B, D y E y siguiendo el orden IDN, se visitara primero D(izquierdo), luego E (derecho) y por ultimo B(nodo). A continuación se visitara el subárbol derecho que consta con los nodos C, F y G. Siguiendo el orden IDN para este árbol, se visita primero F, después a G y por ultimo C. Finalmente se visita el raíz A.

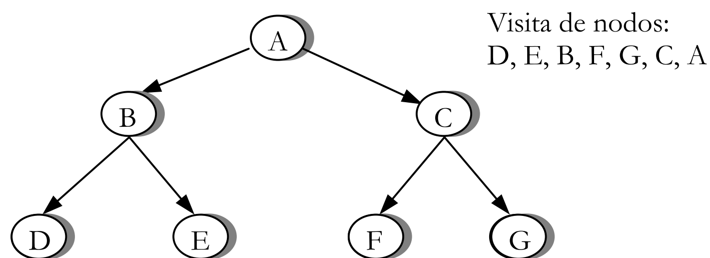
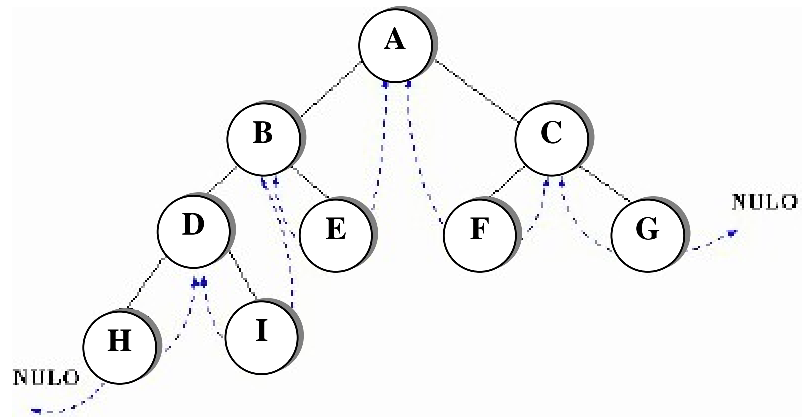


Figura 1.18. Recorrido de POSTORDEN árbol binario

Árboles Enhebrados

Al estudiar la representación enlazada de un árbol binario es fácil observar que existen muchos enlaces nulos. De hecho, existen mas enlaces nulos que punteros con valores reales. En concreto, para un árbol con n nodos, existen n+1 enlaces nulos de los 2n enlaces existentes en la representación (mas de la mitad). Como el espacio de memoria ocupada por los enlaces nulos es el mismo que el

ocupado por los no nulos, podría resultar conveniente utilizar estos enlaces nulos para almacenar alguna información de interés para la manipulación del árbol binario. Una forma de utilizar estos enlaces es sustituir por punteros a otros nodos del árbol. En particular, los enlaces nulos situados en el subárbol derecho de un nodo se suelen reutilizar para apuntar al sucesor de ese nodo en un determinado recorrido del árbol, por ejemplo enorden, mientras que los enlaces nulos en el mismo tipo de recorrido. Si para algunos nodos no existe predecesor (porque es el primero del recorrido) o su sucesor (porque es el ultimo), se mantiene con valor nulo el enlace correspondiente.



La ventaja de este tipo de representación no es sólo el mejor aprovechamiento de la memoria disponible, sino por la posibilidad de un acceso rápido al sucesor (o al predecesor) de un nodo, que como hemos visto anteriormente, puede ser una operación frecuentemente necesaria. En general, los algoritmos que impliquen recorrer el árbol se podrán diseñar de una manera más eficiente.

Para poder manejar correctamente toda la información de la que se dispone en la representación **enhebrada** o **hilvanada** (con hilos) del árbol binario, es necesario poder distinguir entre lo que son punteros normales, que representan las relaciones reales entre los nodos, y lo que son **hilos**. Esto se puede hacer añadiendo dos campos boléanos (en el lenguaje C por valores que similares) a la representación de los nodos del árbol. Estos nuevos campos indicarán si los enlaces izquierdo y derecho son hilos o no.

La estructura de un nodo, siguiendo la representación utilizada en lenguaje C, vendría dada por la siguiente declaración:

```
Type
   Struct Nodo
   {
           Int Info;
           Struct Nodo *Izq, *Der;
           Int HiloIzq, HiloDer;
   }
   typedef nodo arbol;
Var
   Árbol raiz;
```

Con el objeto de no mantener absolutamente ningún enlace nulo y para facilitar el

recorrido del árbol, se suele añadir a la estructura un nodo raíz que no contiene información real (como se ha visto en las listas). El nodo raíz que representa el árbol vacío tendrá la estructura que se muestra en la figura:

Hilo izq	Izq	Info	Der	Hilo Der
Cierto/falso	-	X	-	Cierto/falso

En general, la utilización de enlaces hilos simplifica los algoritmos de recorrido del árbol, por lo que resultan recomendables cuando el recorrido del árbol (o los movimientos parciales dentro del mismo) es una operación frecuente. Sin embargo, desde el punto de vista de la manipulación general del árbol, hay que tener en cuenta que la inserción de nuevos nodos debe mantener en todo momento esta estructura de enlaces con los nodos sucesor y predecesor, y que cada vez que se inserte un nodo se deberá comprobar si existen enlaces de este tipo que deban ser modificados o creados, además de los punteros normales que relacionan los nodos del árbol.

Árboles en Montón

Un bosque representa un conjunto normalmente ordenado de uno o mas árboles generales. Es posible utilizar el algoritmo de conversión analizado en el punto anterior con algunas modificaciones, para representar un bosque en un árbol binario.

Considérese el bosque, constituido por tres árboles generales. Los pasos que deben aplicarse para lograr la conversión del bosque a un árbol binario son los siguientes:

1. Debe enlazarse en forma horizontal las raíces de los distintos árboles generales.

2. Debe enlazarse los hijos de cada nodo en forma horizontal (los hermanos)

3. Debe enlazarse en forma vertical el nodo padre con el hijo que se encuentra más a la izquierda. Además debe eliminarse el vinculo de ese padre con el resto de sus hijos.

4. Debe rotarse el diagrama resultante, aproximadamente 45 grados a la izquierda y así se obtendrá el árbol binario correspondiente.

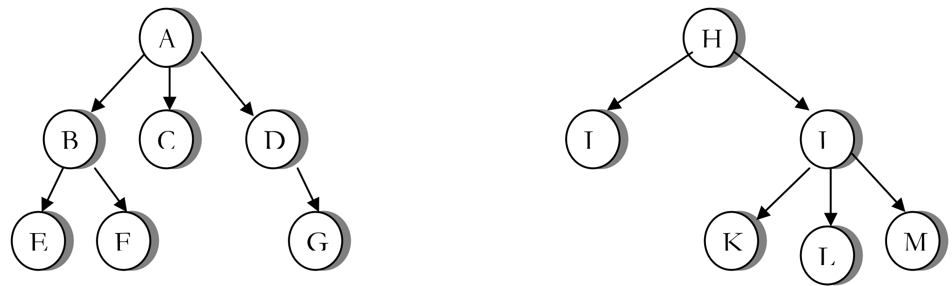


Figura 1.9 Bosque de árboles generales

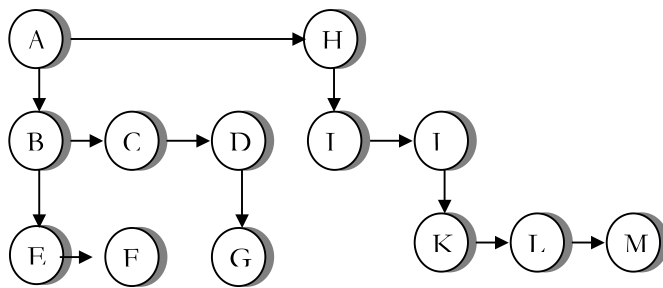


Figura 1.10 Conversión de un bosque en un árbol binario luego de aplicar los pasos 1,2 y 3

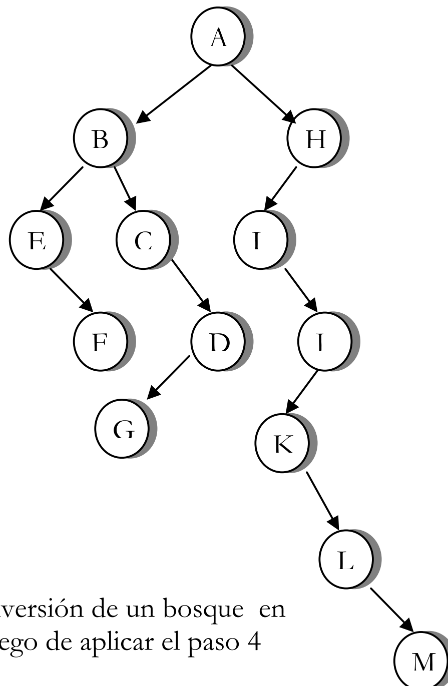


Figura 1.11 conversión de un bosque en árbol binario luego de aplicar el paso 4

Creación de un Árboles Binarios

Hasta el momento se analizado los distintos tipos de recorrido de un árbol binario. Sin embargo, debe recordarse que antes de recorrer un árbol, debe de crearse el mismo en memoria. Un algoritmo muy simple que crea los nodos de un árbol binario en memoria es el siguiente. El siguiente algoritmo como los mostrados anteriormente utiliza recursividad para la inserción de un nuevo nodo, donde la variable Nodo es de tipo puntero. La primera vez Nodo es creado en el programa principal :

```
Carga( Nodo )
puntero Q
Inicio
    Leer (INFOR)
    Nodo^.Info=INFOR
    Escribir "Desea crear nodo por la Izquierda"
    Leer (Respuesta)
    Si  Respuesta entondes
            Crear(Q)
            Nodo^.Izq=Q
            Carga(Nodo^.Izq)
    Sino
            Nodo^.Izq=NULL
    Fin Si
    Escribir "Desea crear nodo por la Derecho"
    Leer (Respuesta)
    Si  Respuesta entondes
            Crear(Q)
            Nodo^.Der=Q
            Carga(Nodo^.Der)
     Sino
            Nodo^.Izq=NULL
     Fin Si
Fin
```

Árboles Binarios de Búsqueda

Los árboles vistos hasta el momento no tienen orden definido; sin embargo los árboles binarios ordenados tiene sentido. Estos árboles se denominan árboles binarios de búsqueda, debido a que se pueden buscar en ellos un termino utilizando un algoritmo de búsqueda binaria similar al empleado en los arreglos estructuras de datos 1.

Un árbol binario de búsqueda es un árbol binario, que puede estar vacío, y que si es no vacío cumple con las siguientes propiedades:

- Todos los nodos están identificados por una clave y no existen dos elementos con la misma clave.
- Las claves de los nodos del subárbol izquierdo son menores que la clave del nodo raíz.
- Las claves de los nodos del subárbol derecho son mayores que la clave del nodo raíz.
- Los subárboles izquierdo y derecho son también árboles binarios de búsqueda.

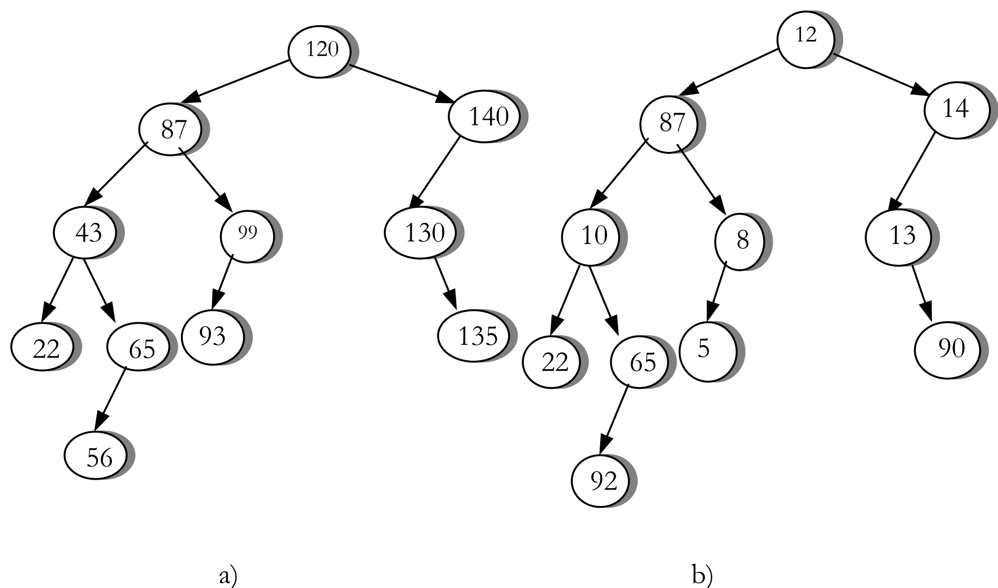


Figura 1.19 a) Árbol binario de búsqueda b) no es árbol binario de búsqueda.

En las figura 1.19 se muestran dos árboles, en donde el primero de ellos es un árbol binario de búsqueda ya que cada nodo esta ordenado dependiendo de su valor. Observe que en el nodo 120 al lado izquierdo se encuentra los valores menores que el, y al lado derecho los mayores. Estas mismas reglas se cumplen para cada uno de sus nodos.

Operaciones en Árboles Binarios de Búsqueda

De lo expuesto hasta ahora se deduce que los árboles binarios tienen naturaleza recursiva y en consecuencia las operaciones sobre los árboles son recursivas, si bien siempre tenemos la opcion de realizarlas de forma iterativa. Estas operaciones son:

- Búsqueda de un nodo.
- Inserción de un nodo.
- Supresión de un nodo.

Búsqueda

La definición de árbol binario de búsqueda especifica un criterio en la estructuración del árbol en función de las claves de los nodos. En este caso, existe un criterio de ordenación de los nodos. Por lo tanto, será bastante simple describir un método eficiente de búsqueda que explote esta ordenación.

Suponer que se busca un elemento que posea la clave x. La búsqueda comenzará por el nodo raíz del árbol. La clave de ese nodo informará por dónde debe continuar la búsqueda, ya no es necesario recorrer exhaustivamente todos los nodos del árbol. Si la clave del nodo es igual a x, la búsqueda finaliza con éxito. Si la clave es menor que x, se sabe que si existe un nodo en el árbol que posea como clave el valor x deberá estar en el subárbol derecho, por lo tanto la búsqueda deberá continuar por esa parte del árbol. Si, por el contrario, la clave del nodo es mayor que x, entonces la búsqueda deberá continuar por el subárbol izquierdo. El proceso continuará hasta que se encuentre un nodo con clave igual a x o un subárbol vacío, en cuyo caso se puede asegurar que no existe ningún nodo con clave x en el árbol. Este método de búsqueda sugiere seguir un esquema recursivo.

```
Busca(Nodo, infor)
Inicio
  SI Nodo = NULO entonces
     Regresa(NULO)
  Sino
     Si  infor =  Nodo^.info entonces
         Regresa(Nodo)
     Sino
        Si infor < Nodos^.info entonces
            Regresa(Busca(Nodo^.Izq,infor))
        Sino
            Regresa(Busca(Nodo^.Der, infor))
        Fin Si
     Fin Si
  Fin Si
Fin
```

En otro caso de búsqueda, la recursividad puede ser fácilmente sustituida por un esquema iterativo mediante la utilización de un bucle de repetición "mientras". En ese caso, el algoritmo de búsqueda iterativo sería:

```
Búsqueda(Nodo,infor)
Inicio
  Aux = Nodo
  Enc = FALSO
  Mientras (aux <> NULL) Y (Enc = FALSO) hacer
     Si Aux^.info = x entonces
        Enc = CIERTO
     Sino
        Si x < Aux^.info entonces
          Aux = Aux^.Izq
        Sino
          Aux = Aux^.Der
     Fin Si
  Fin Mientras
  Regresa(Enc)
Fin
```

El método de búsqueda se asemeja mucho al método de búsqueda binaria sobre arreglos ordenados, tras la comparación con un elemento se puede decidir en qué región de la estructura se puede encontrar la información buscada, descartándose el resto de los elementos de la estructura. De esta forma, se reduce considerablemente el número de comparaciones necesarias para localizar el elemento.

Inserción

La inserción de un nuevo nodo en un árbol binario de búsqueda debe realizarse de tal forma que se mantengan las propiedades del árbol. De modo que lo primero que hay que hacer es comprobar que en el árbol no existe ningún nodo con clave igual a la del elemento que se desea insertar. Si el elemento no existe la inserción se realiza en un nodo en el que al menos uno de los dos punteros Izq o Der tengan valor NULL.

Para realizar la condición anterior se desciende en el árbol a partir del nodo raíz , dirigiéndose de izquierda a derecha de un nodo, según el valor a insertar sea inferior o superior al valor del campo Info de este nodo. Cuando se alcanza un nodo del arbolen que no se pueda continuar, el nuevo elemento se engancha a la izquierda o derecha de este nodo en función de su valor sea inferior o superior al del nodo alcanzado.

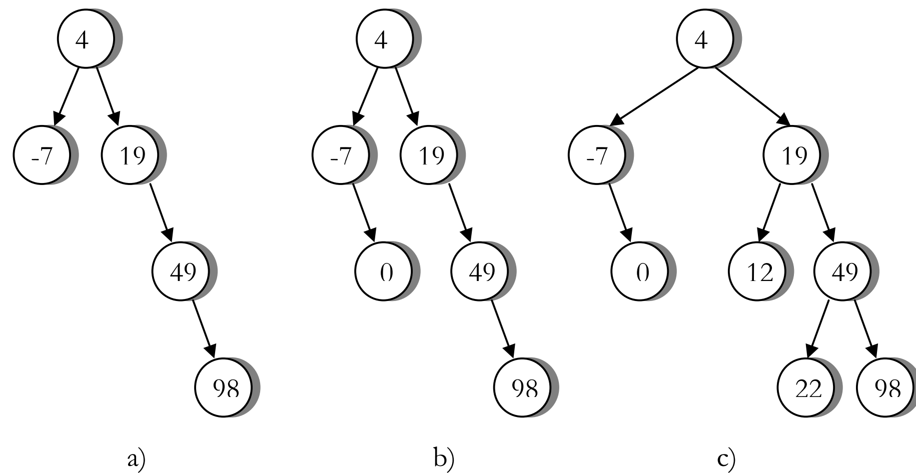


Figura 1.20 Inserción de un árbol de búsqueda binaria: (a) inserta 100, (b) inserta 0, (c) inserta 22 y 12

En este manual se manejan dos algoritmos de inserción, dando la opción al usuario de escoger entre uno u otro.

```
INSERCIÓN(Nodo,Infor)
Inicio
  Si infor < Nodo^.info entonces
     Si Nodo^.Izq=NULL entonces
        Crea(Otro)
        Otro^.Izq = NULL
        Otro^.Der = NULL
        Otro^.Info = Infor
        Nodo = Otro
     Sino
        INSERCIÓN(Nodo^.Izq,Infor)
     Fin Si
  Sino
    Si infor > Nodo^.info entonces
       Si Nodo^.Der=NULL entonces
          Crea(Otro)
          Otro^.Izq = NULL
          Otro^.Der = NULL
          Otro^.Info = Infor
          Nodo = Otro
       Sino
          INSERCIÓN(Nodo^.Der,Infor)
       Fin Si
    Sino
       Escribir "El Nodo ya se encuentra en el árbol"
    Fin Si
  Fin Si
Fin
```

```
INSERCIÓN(Nodo,Infor)
Inicio
 Si Nodo<>NULL Entonces
   Si Infor < Nodo^.Info entonces
     INSERCIÓN(Nodo^.Izq,Infor)
   Sino
     Si Infor > Nodo^.info entonces
       INSERCIÓN(Nodo^.Der,Infor)
     Sino
       Escribir "El nodo ya se encuentra en el árbol"
     Fin Si
   Fin Si
 Sino
   Crea(Otro)
   Otro^.Izq = NULL
   Otro^.Der = NULL
   Otro^.Info = Infor
   Nodo = Otro
 Fin Si
Fin
```

Eliminación

La operación de eliminación de un nodo es también una extensión de la operación de búsqueda, si buen es mas compleja que la inserción debido a que el nodo a suprimir puede ser cualquiera y la operación de su supresión deba mantener la estructura de árbol binario de búsqueda después de la eliminación de datos. Los pasos a seguir son:

1. buscar en el árbol para encontrar la posición de nodo a eliminar.

2. reajustar los punteros de sus antecesores si el nodo a suprimir tiene menos de dos hijos, o subir a la posición que este ocupa el nodo descendiente con la clave inmediatamente superior o inferior con objeto de mantener la estructura del árbol binario.

La eliminación de una clave y su correspondiente nodo, presenta dos casos claramente diferenciados. En primer lugar en un nodo hoja o tiene un único descendiente, resulta una tarea fácil, ya que lo único que hay que hacer es asignar el enlace desde el nodo padre (según el camino de búsqueda) el descendiente del nodo a eliminar (o bien NULL). En segundo lugar, que el nodo tenga los dos descendientes. Para mantener la estructura del árbol de búsqueda tenemos dos alternativas, remplazar la clave a eliminar por la mayor de las claves menores, o bien remplazar por la menor de las mayores. Se elige la primera opción, lo que se supone bajar por la derecha de la rama izquierda del nodo a eliminar hasta llegar al nodo hoja, que será el que este mas a la derecha dentro del subárbol izquierdo de la clave a borrar.

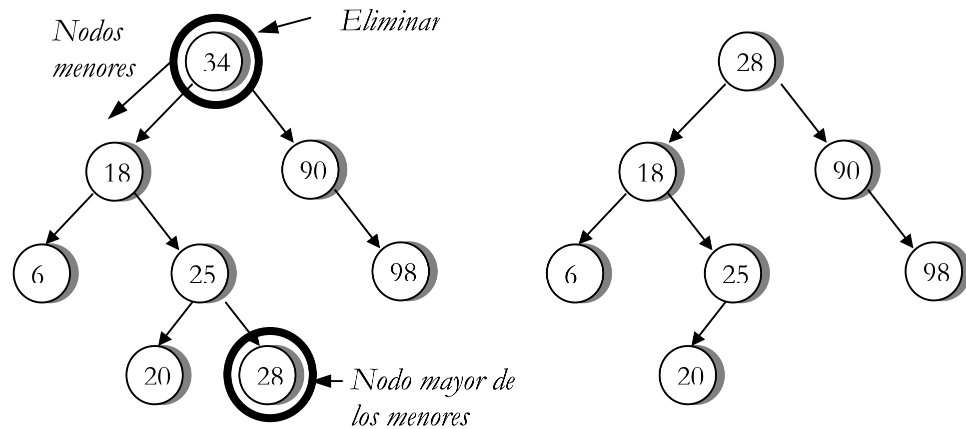


Figura 1.21 Eliminación de un árbol de búsqueda binaria

La figura 1.20 nos muestra la eliminación del nodo 34, lo primero que se debe de hacer es localizar el nodo que se desea eliminar, posteriormente se toma como raíz y para elimínalo se puede hacer de dos formas. El primer método se va por el lado izquierdo donde se encuentran los nodos menores a la raíz, encontrándose el nodo 18 como el primer nodos, a continuación se toma el nodo mas mayor de los menor que vienen siendo el nodo mas a la derecha, o sea que se van pasando por el nodo 25, y después al mas a la derecha el 28. se puede cambiar ese nodo por el 34 y no cambia nada el árbol. El siguiente algoritmo muestra la

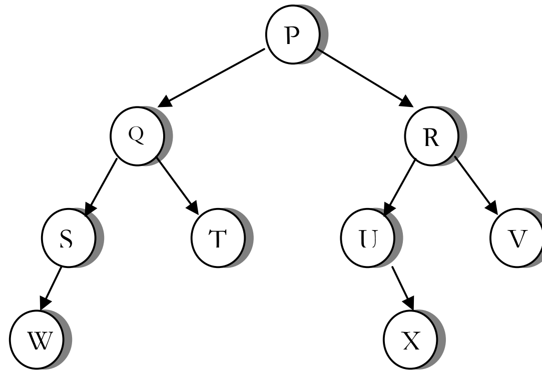
```
ELIMINACIÓN(Nodo,Infor)
Inicio
 Si Nodo<>NULL Entonces
   Si Infor < Nodo^.Info Entonces
     ELIMINACION(Nodo^.Izq,Infor)
   Sino
     Si Infor > Nodo^.info Entonces
       ELIMINACION(Nodo^.Der,Infor)
     Sino
       Otro=Nodo
       Si Otro^.Der=NULL Entonces
          Nodo=Otro^.Izq
       Sino
          Si Otro^.Izq=NULL Entonces
             Nodo=Otro^.Der
          Sino
             Aux=Otro^.Izq
             Aux1=Aux
             Mientras Aux^.Der<>NULL Hacer
                 Aux1=Aux
                 Aux=Aux^.Der
             Fin Mientras
              Otro^.info=Aux^.Info
              Otro=Aux
```

```
                    Aux1^.Der=Aux^.Izq
               Fin Si
           Fin Si
       Fin Si
     Fin Si
     QUITA(Otro)
  Sino
      Escribir "El nodo no se encuentra en el árbol"
  Fin Si
Fin
```

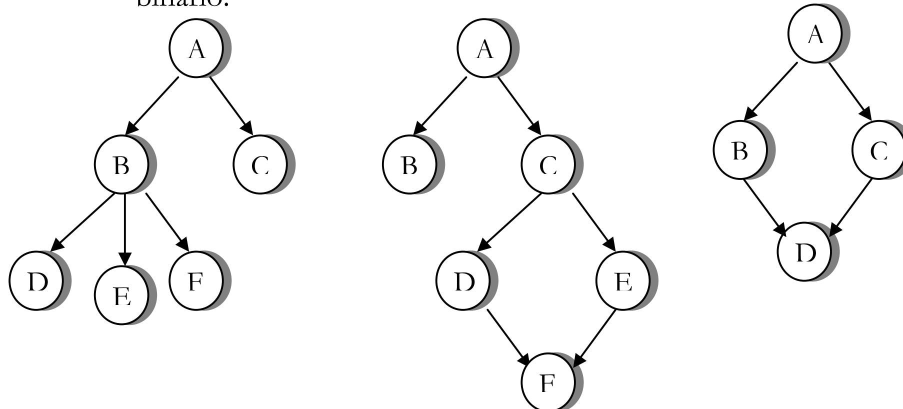
Ejercicios

1.1. Considerando el árbol siguiente:

a) ¿Cuál es su altura?
b) ¿Esta el árbol equilibrado?¿Porque?
c) Lista todos los nodos hoja
d) ¿Cuál es el predecesor inmediato (padre)del nodo U?
e) Lista los hijos del nodo R
f) Lista los sucesores del nodo R



1.2. Explica porque cada una de las siguientes estructuras no es un árbol binario:



1.3. Para cada una de las siguientes listas de letras:

1. M, Y, T, E, R.
2. R, E, M, Y, T.
3. T, Y, M, E, R.
4. C, O, R, N, F, L, A, K, E, S.

a) Dibuja el árbol de búsqueda que se construye cuando las letras se insertan en el orden dado.
b) Realiza recorridos enorden, preorden y postorden del árbol y mostrar la secuencia de letras que resultan en cada caso.

1.4. Para el árbol del ejercicio 1.1, recorrer cada árbol utilizando los ordenes siguientes: NDI, DNI, DIN.

1.5. Dibujar los árboles binarios que representan las siguientes operaciones:
   a) (A + B) / (C – D)
   b) A + B + C / D
   c) A - (B - (C – D) / (E + F))
   d) (A + B) * ((C + D) / (E + F))
   e) (A - B) / ((C * D) - (E / F))

1.6. El recorrido preorden de un cierto árbol binario produce
   ADFGHKLPQRWZ
   Y en recorrido enorden produce
   GFHKDLAWRQPZ
   Dibujar el árbol binario.

1.7. Escribir una función recursiva que cuete las hojas de un árbol binario.

1.8. escribir un programa que procese un árbol binario cuyos nodos contengan caracteres y a partir del siguiente menú de opciones:

I	(seguido de un carácter)	: Insertar un carácter
B	(seguido de un carácter)	: Buscar un carácter
RE		: Recorrido en orden
RP		: Recorrido en preorden
RT		: Recorrido en postorden
SA		: Salir

1.9. Escribir una función que tome un árbol como entrada y devuelva el numero de hijos del árbol.

1.10. Escribir una función booleana a la que se le pase un puntero a un árbol binario y devuelva verdadero si el árbol es completo y falso en caso contrario.

1.11. Diseñar una función recursiva que devuelva un puntero a un elemento en un árbol binario de búsqueda.

1.12. Diseñar una función iterativa que encuentre un elemento en un árbol binario de búsqueda.

Problemas

1.13. Escribir un programa que lea un texto de longitud indeterminada y que produzca como resultado la lista de todas las palabras diferentes contenidas en el texto, así como su frecuencia de aparición.

1.14. Se dispone de un árbol binario de elementos de tipo entero. Escribir funciones que calculen:
   a) La suma de sus elementos
   b) La suma de sus elementos que son múltiplos de 3.

1.15. Escribir una función booleana IDÉNTICOS que permita decir si dos árboles binarios son iguales.

1.16. Diseñar un programa interactivo que permita dar de alta, baja, listar, etc., en un árbol binario de búsqueda.

1.17. Construir un procedimiento recursivo para encontrar una determinada clave en un árbol binario.

1.18. Dados dos árboles binarios de búsqueda indicar mediante un programa si los árboles tiene o no elementos comunes.
1.19. Dado un árbol binario de búsqueda construir su árbol espejo (árbol espejo es aquel que se construye a partir de uno dado, convirtiendo el subárbol derecho en izquierdo y viceversa).

Capítulo

2

Árboles Balanceados

Se utilizan los árboles binarios de búsqueda para almacenar datos organizados jerárquicamente. Sin embargo en muchas ocasiones la inserción y eliminación de los elementos en el árbol no ocurren en el orden predecible; es decir, los datos no esta organizados jerárquicamente.

Aquí se estudiara el tipo de árboles adicionales : árboles balanceados (equilibrados) o árboles AVL, como también se conocen, que ayudan eficientemente a resolver estas situaciones.

Árboles Balanceados

Al manejarse los árboles binarios de búsqueda se menciono que es una estructura sobre la cual se puede realizar eficiente mente las operaciones de búsqueda, inserción y eliminación. Sin embargo, si el árbol crece o decrece descontroladamente, el rendimiento puede disminuir considerablemente. El caso mas desfavorable se produce cuando se inserta un conjunto de claves ordenadas de forma ascendente o descendente, como se muestra en la figura 2.1

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

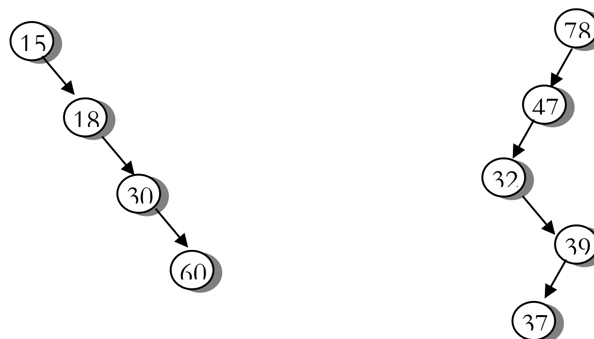


Figura 2.1 árboles binaros de búsqueda con crecimiento descontrolado

Con el objeto de mejorar el rendimiento de la búsqueda surgen los árboles balanceados (equilibrados). La idea central de estos es la de realizar reacomodos o balanceos, después de la inserción o eliminación de los elementos. Estos árboles también reciben el nombre de árboles 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 las siguientes condiciones : "Para todo nodo T del árbol , la altura de los subárboles izquierdo y derecho no debe diferir en mas de una unidad ". En la figura 2..2 que muestran dos diagramas con árboles balanceados.

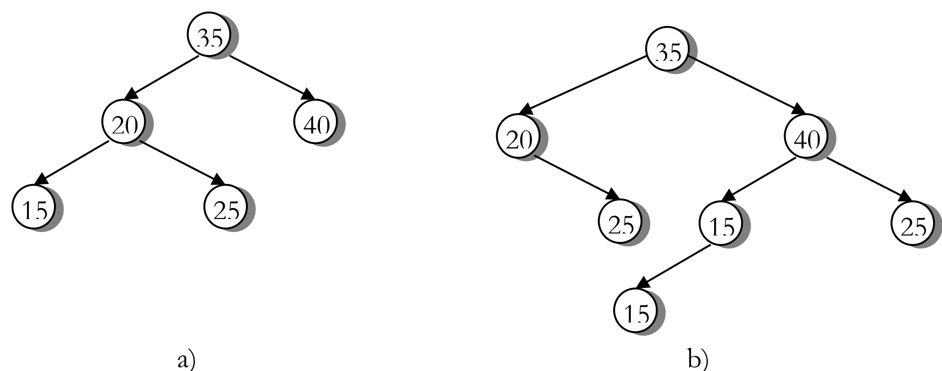


Figura 2.2 Dos árboles balanceados a) con altura 3. b)Con altura 4

Creación de un Árbol Equilibrado de N Claves

Considerando la formación de los árboles balanceados, se parece mucho a la secuencia de los números Fibonacci:

$$a(n) = a(n-2) + a(n-1)$$

de igual forma, un árbol de fibonacci (árbol equilibrado) puede definirse:

1. Un árbol vació es un árbol de fibonacci de altura 0.
2. Un nodo único es un árbol de fibonacci de altura 1.
3. A_{h-1} y A_{h-2} son árboles de fibonacci de altura h-1 y h-2 entonces $A_h = <A_{h-1}, X, A_{h-2}>$ es un árbol de fibonacci de altura h.

El numero de nodos de A_h viene dado por la sencilla relación recurrente.

$N_0=0$

$N_1=1$

$N_h=N_{h-1}+1+N_{h-2}$

Para conseguir un árbol AVL, con un numero dado N, de un nodo hay que distribuir equitativamente los nodos a la izquierda y a la derecha de un nodo dado. En definitiva, es seguir la relación de recurrencia anterior que podemos expresar recursivamente:

1. Crea nodo raíz.
2. Genera un subárbol izquierdo con $n_i=n/2$ nodos del nodo raíz utilizando la misma estrategia.
3. Genera un subárbol derecho con $n_d=n/2$ nodos del nodo raíz utilizando la misma estrategia.

El Pseudocódigo del programa GeneraBalanceado implementa la creación de un árbol equilibrado de n claves.

```
/* Pseudocódigo Genera Balanceado */

Ptrae = ^Nodo  /* Puntero  de tipo  registro Nodo*/
Nodo  = Record
  Clave : integer
  Izq, Der :Nodo
Fin Record

R : Ptrae
N : Integer
```

```
Modulo ÁrbolEq ( N: integer ) : Ptrae
Nuevo: Ptrae
Niz, Ndr : Integer
Inicio
  Si N=0 Entoces
    Regresa (NULL)
  Sino
    Niz=n div 2
    Ndr=N-Niz-1
    Crear(Nuevo)
    Escribe"Clave"
    Leer Nuevo^.Clave
    Nuevo^.Izdo=Arboleq(Niz)
    Nuevo^.Drch=Arboleq(Ndr)
   Regresa Nuevo
 Fin  Si
Fin

Modulo Dibujaarbol ( R : Ptrae, H: integer )
I : Integer
Inicio
  Si  R< > NULL Entonces
    Dibuja árbol(R.^.Izdo, H+1)
    Para I=1 to H Hacer
       Escribe " "
    Fin Para
     Escribe R.^.Clave
     Dibuja árbol(R.^.Drch, H+1)
  Fin  Si
Fin

Inicio
  Escribir "¿Numeros de nodos  del árbol ?"
  Leer N
  R = Arboleq(N)
  Dibujaarbol(R,0)
Fin
```

Factor de Equilibrio

Para poder comenzar en el proceso de equilibrio de un árbol binario que fue afectado por una inserción se debe conocer que es el factor de equilibrio (FE). Cada nodo, además de la información que se pretende almacenar, debe tener los dos punteros a los árboles derecho e izquierdo, igual que los ABB, y además un miembro nuevo: el factor de equilibrio.
El factor de equilibrio es la diferencia entre las alturas del árbol derecho y el izquierdo:

$$FE = H_{RD} - H_{RI}$$

Por definición, para un árbol AVL, este valor debe ser -1, 0 ó 1. Utilizando las figuras 2.1 y 2.2 se mostrara el FE de cada uno de los nodos.

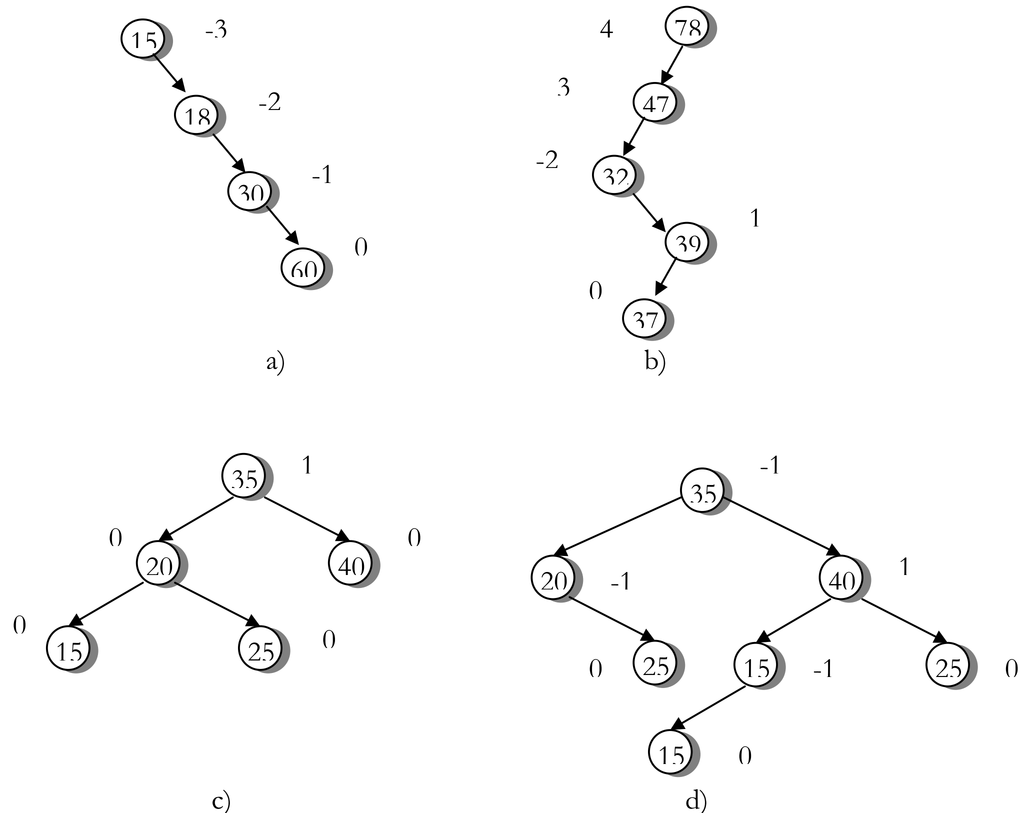


Figura 2.3 Los árboles a) y b) no son árboles equilibrados. Los árboles c) y d) son árboles equilibrados

Obsérvese que en la figura 2.3 d) el FE del nodo 40 es de –1, puesto que la altura del subárbol derecho es igual a 1 y la altura del subárbol izquierdo 2.

$$FE_{40} = 2 - 1 = 1$$

El FE de 20 se calcula como:

$$FE_{20} = 0 - 1 = -1$$

Cada uno de los nodos de los árboles de la figura 2.3 muestra el FE de cada nodo. y como se menciono anteriormente el FE debe de cumplir que para ser un árbol equilibrado todos los árboles del nodo deben de tener un FE entre –1 a 1. por tal motivo los árboles a)y b) no son árboles equivalentes porque contiene nodos con mayor FE de 1 o menor que-1. en cambio los árboles c) y d) se encuentran en el rango permitido. Cada uno de los nodos debe de guarda su FE, o sea que la estructura del nodo cambiaria de la siguiente manera:

```
Enlace = ^Nodo
Nodo = registro
            Info : tipo de dato
            FE : integer
            Izq, Der : enlace
Fin registro
```

Y en lenguaje C se mostraría :

```
Struct Nodo
{
            int Info;
            int FE;
            Struct  Nodo *Izq, Der;
}
```

Casos Para la Inserción de un Nuevo Nodo

Ahora se va a insertar un nuevo nodo , y como ocurre con las inserciones , como nodo hoja. Al insertar un elemento (nodo) en un árbol balanceado debe distinguirse los siguientes casos:

1. La rama izquierda (RI) y derecha (RD) del árbol tiene la misma altura (H_{RI}= H_{RD}), por lo tanto:

   1.1. Si se inserta un elemento en RI entonces H_{RI} será mayor a H_{RD}.

   1.2. Si se inserta un elemento en RD entonces H_{RD} será mayor a H_{RI}.

   Observe que en cualquiera de los dos caso mencionados no se viola el criterio de equilibrio.

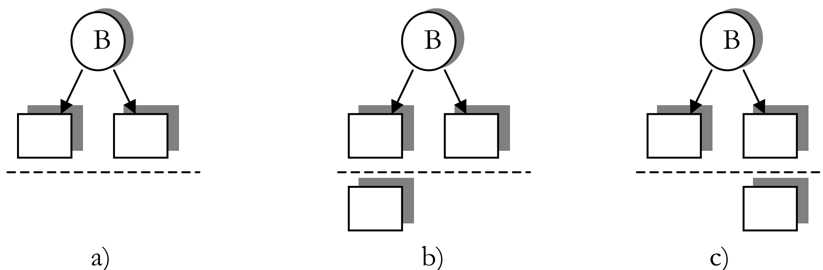


Figura 2.4 Los diferentes situaciones de inserción de la primera caso a) Caso 1 , b) Caso 1.1 y c) Caso 1.2

2. La rama izquierda (RI) y la derecha (RD) del árbol tienen altura diferente($H_{RI} = H_{RD}$):

 2.1. Supóngase 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 tiene la misma altura, por lo que será mejor el equilibrio).

  2.1.2. Si se inserta un elemento en RD entonces se rompe el criterio de equilibrio del árbol y es necesario reestructurarlo.

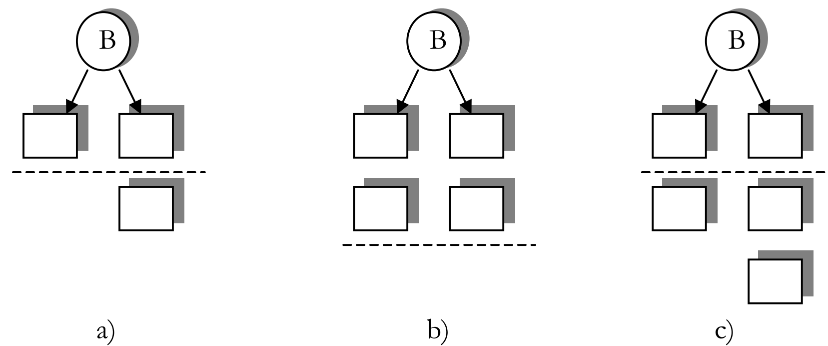


Figura 2.5 Los diferentes situaciones de inserción de la primera caso a) Caso 2.1, b) Caso 2.1.1 y c) Caso 2.1.2

 2.2. Supóngase 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 estructurarlo.

  2.2.2. Si se inserta un elemento en RD entonces H_{RD} será igual a H_{RI}. (las ramas tiene la misma altura, por lo que será mejor el equilibrio).

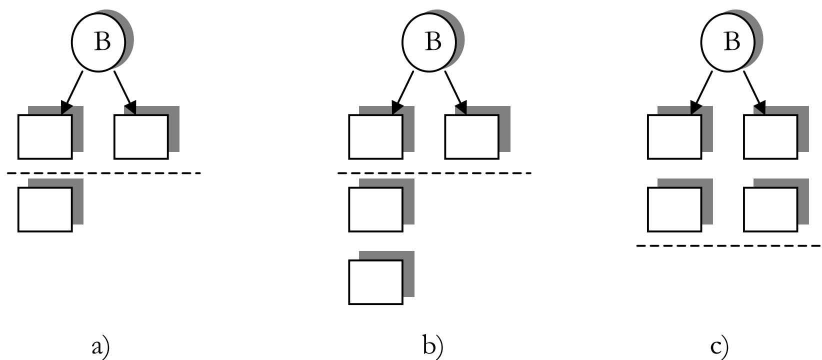


Figura 2.6 Los diferentes situaciones de inserción de la primera caso a) Caso 2.2. , b) Caso 2.2.1 y c) Caso 2.2.2

Reestructuración del Árbol Balanceado (Rotación)

En todo este proceso hay que recordar que estamos trabajando con árboles binarios de búsqueda con un campo adicional que es el factor de equilibrio (FE), por lo que el árbol resultante debe seguir siendo un árbol de búsqueda .

Para añadir un nuevo nodo al árbol, primero se baja por el árbol siguiendo el camino de búsqueda determinado por el valor de la clave, hasta llegar al lugar donde hay que insertar el nodo. Este nodo se inserta como hoja del árbol, por lo que su FE será 0.

Una vez hecho esto, se regresa por el camino de búsqueda marcado, calculando el factor de equilibrio de los distintos nodos que forman el camino. Este calculo hay que hacerlo porque al añadir el nuevo nodo al menos una rama de un subárbol a aumentado en altura. Puede ocurrir que el nuevo FE de un nodo viole el criterio de balanceo. Si es así, debe de reestructurar el árbol (subárbol) de raíz dicho nodo.

El proceso de regreso termina cuando se llega a la raíz del árbol o cuando se realiza la reestructuración en un nodo del mismo; en cuyo caso no es necesario determinar el FE de los resultantes nodos, debido a que dicho factor queda como el que tenia antes de la inserción. Pues el efecto de reestructuración hace que no aumente la altura.

Las violaciones del FE de un nodo pueden darse de cuatro maneras distintas. El equilibrio o reestructuración se realizan con el desplazamiento particular de los nodos implicados, rotando los nodos. 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 puede realizarse por las ramas derechas (DD) o por las ramas izquierdas (II). Si la rotación es compuesta puede realizarse por las ramas derecha e izquierda (DI) o por las ramas izquierda y derecha (ID).

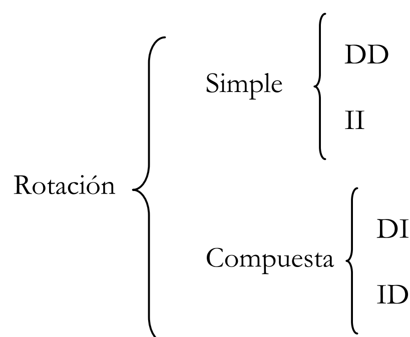


Figura 2.7 Diagrama de los tipos de Rotaciones

Los cuatro casos de violación del balanceo de un nodo se muestran en la siguientes figuras, 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 de 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, así como la reestructuración que equilibra el árbol:

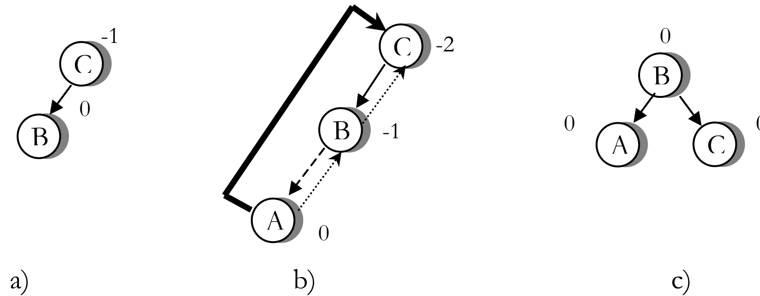


Figura 2.8. Rotación Simple II de Árbol Balanceado

En la figura 2.8 a) es el árbol original, en el b) se inserta el nodo A (Siguiendo el camino de búsqueda) y cambia el factor de equilibrio. En el c) el nodo C se ha rotado el criterio de equilibrio, la reestructuración consiste en rotación a la izquierda, izquierda (rotación simple II).

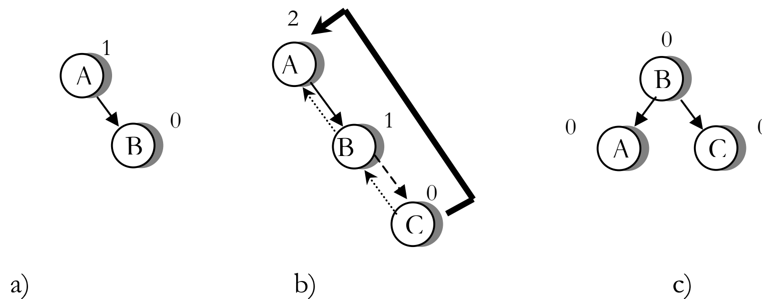


Figura 2.9. Rotación Simple DD de Árbol Balanceado

En la figura 2.9 a) es el árbol original, en el b) ahora se inserta el nodo C (Siguiendo el camino de búsqueda) y cambia el factor de equilibrio. En el c) el nodo A se ha rotado el criterio de equilibrio, la reestructuración consiste en rotación a la derecha, derecha (rotación simple DD).

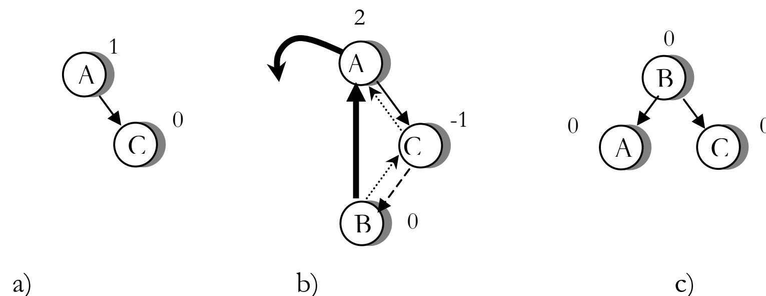


Figura 2.10. Rotación Compuesta DI de Árbol Balanceado

En la figura 2.10 a) es el árbol original, en el b) ahora se inserta el nodo B (Siguiendo el camino de búsqueda) y cambia el factor de equilibrio. En el c) el nodo A se ha rotado el criterio de equilibrio, la reestructuración consiste en rotación a la derecha, izquierda (rotación simple DI).

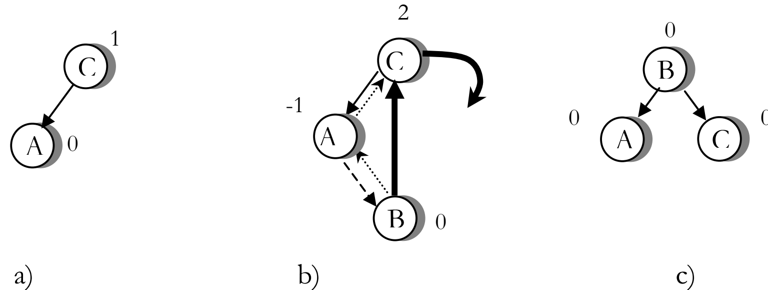


Figura 2.11. Rotación Compuesta ID de Árbol Balanceado

En la figura 2.11 a) es el árbol original, en el b) ahora se inserta el nodo B (Siguiendo el camino de búsqueda) y cambia el factor de equilibrio. En el c) el nodo C se ha rotado el criterio de equilibrio, la reestructuración consiste en rotación a la izquierda, derecha (rotación simple ID).

Formación de un Árbol Balanceado

Se simulara la inserción de los nodos en un árbol de búsqueda balanceado, partiendo del árbol vació. Por comodidad que el campo clave es entero. El nodo se representara como una estructura, donde contendrá cada uno de los campos representados anteriormente. Los punteros N, N1 y N2 referencian al nodo que viola las condiciones de equilibrio y a los descendientes en el camino de búsqueda.

Supóngase que se va a insertar las siguientes claves en un árbol balanceado:

Claves : 65, 50, 23, 70, 82, 68, 39.

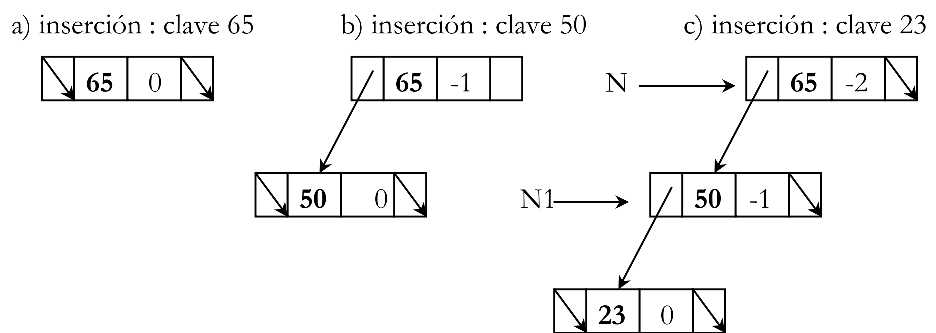


Figura 2.12. Seguimiento de la inserción en el árbol

Una vez insertado el nodo con la clave 23, al regresar por el camino de búsqueda cambia los factores de equilibrio, así el del nodo 50 pasa a –1 y en el nodo 65 se hace pasa a –2. se ha roto el criterio de equilibrio y debe de reestructurarse. Se apunta con N a la clave 65 y con N1 la rama izquierda de dicha clave. Luego de verificar los

factores de equilibrio –1 y –2 deben de realizarse una rotación de los nodos II para rehacer el equilibrio. Los movimientos de los punteros para realiza esta rotación II es el siguiente:

```
1) N^.Izq = N1^.Der
2) N1^.Der = N
3) N = N1
```

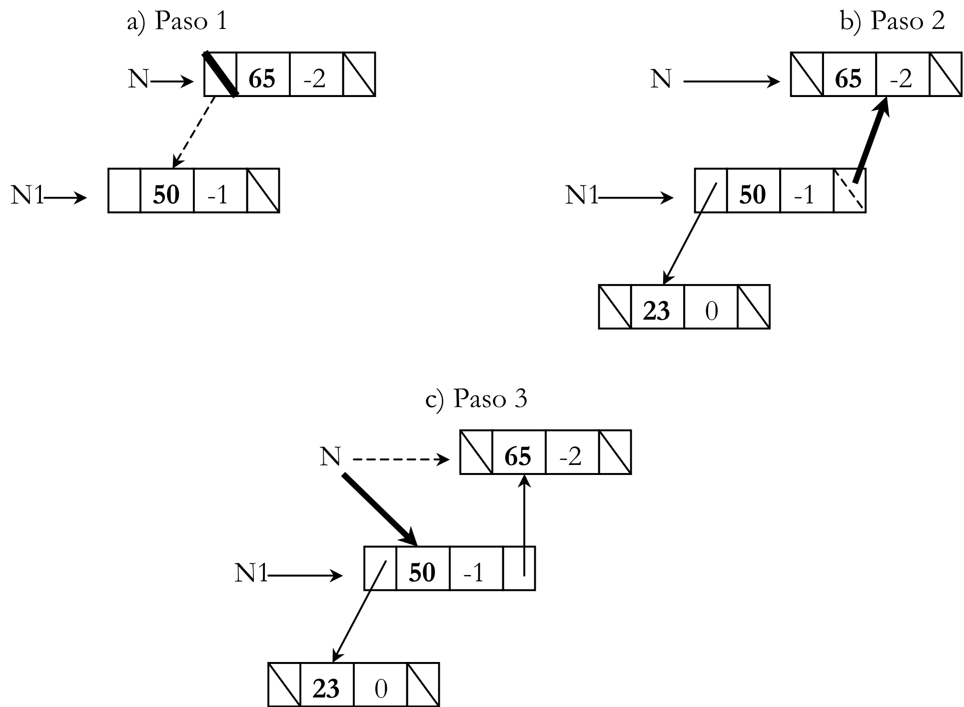


Figura 2.13. Seguimiento del algoritmo de rotación II.

El orden que llevan cada uno de los pasos es el mismo que lleve el algoritmo, con la línea discontinua (- - - -) se puede apreciar el valor que se tenia antes de cada el paso. Con la línea normal (_____) son los valores que no se afectan en ese paso y con la línea gruesa (━━━) se muestra el paso después de elaborarse.

Respecto al FE de los nodos afectados el reacomodo, este será siempre 0 (cero) en el caso de la **rotaciones simples**

```
N^.FE = 0
N1^.FE = 0
```

Luego de efectuar el reacomodo, el árbol queda así:

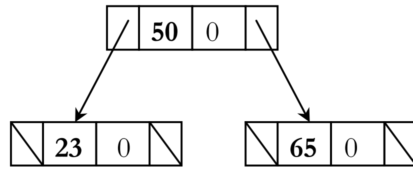


Figura 2.14. Árbol reestructurado por II

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

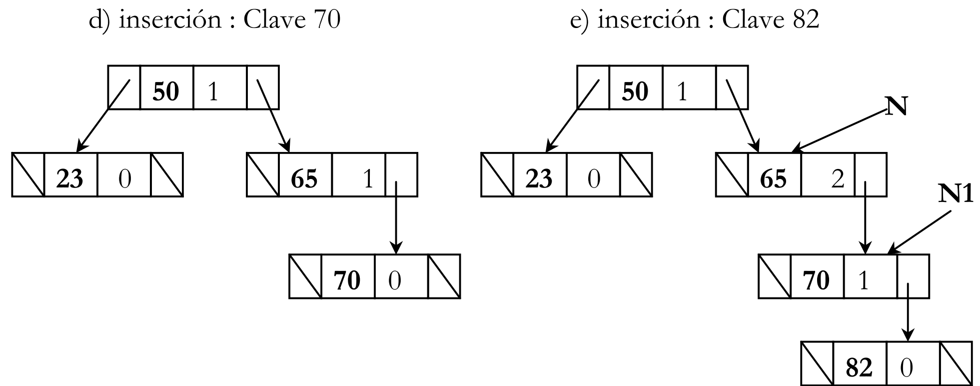
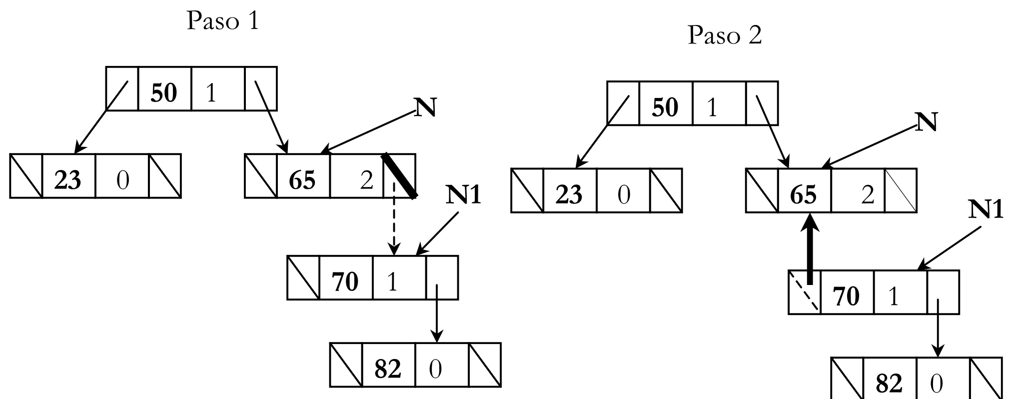


Figura 2.14. continuación del seguimiento de la inserción de Árbol

Una vez insertado el nodo con la clave 82 a la derecha del nodo 70 , y regresar por el camino de búsqueda para así calcular los nuevos factores de equilibrio, se observa que dichos factor queda incrementado en 1 pues la inserción ha sido por la derecha. En el nodo con la clave 65 queda roto el equilibrio. Para reestructurar se realizo una rotación DD. Los movimientos de los puntos para realizar esta rotación DD son:

1) N^.Der = N1^.Izq
2) N1^.Izq = N
3) N = N1



Paso 3

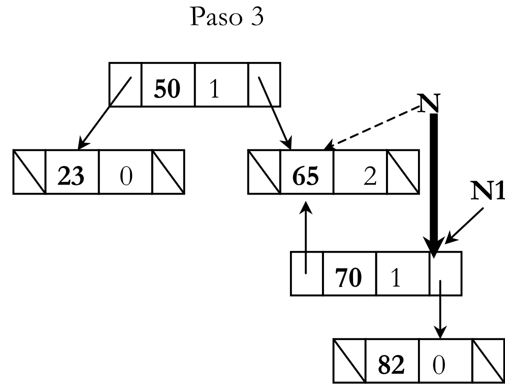


Figura 2.15. Seguimiento del algoritmo de rotación DD.

Como el anterior, una vez realizada la rotación, los factores de equilibrio de los nodos implicados serán 0 como ocurre en las rotaciones simples, y el árbol que da de la siguiente forma:

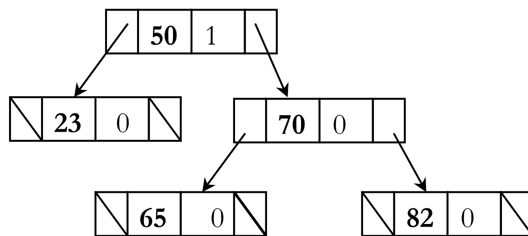


Figura 2.16. árbol reestructurado

f) inserción : clave 68

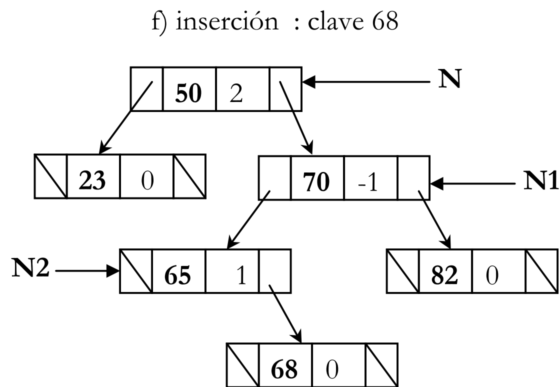


Figura 2.17. Continuación del seguimiento de la inserción de Árbol

Para insertar el nodo 68 se sigue el camino : derecha de 50, izquierda 70 y se inserta ala derecha del nodos 65, al regresar por el camino de búsqueda los factores de equilibrio en 1 si se fue por la rama derecha, se decrementa en 1 si se fue por la

rama izquierda. En el nodo 50, el balanceo se ha roto. La rotación de los nodos para restablecer el equilibrio de DI. Los movimientos de los punteros para realizar esta rotación DI son :

```
1)  N1^.Izq = N2^.Der
2) N2^.Der = N1
3) N^.Der =N2^.Izq
4) N2^.Izq = N
5) N = N2
```

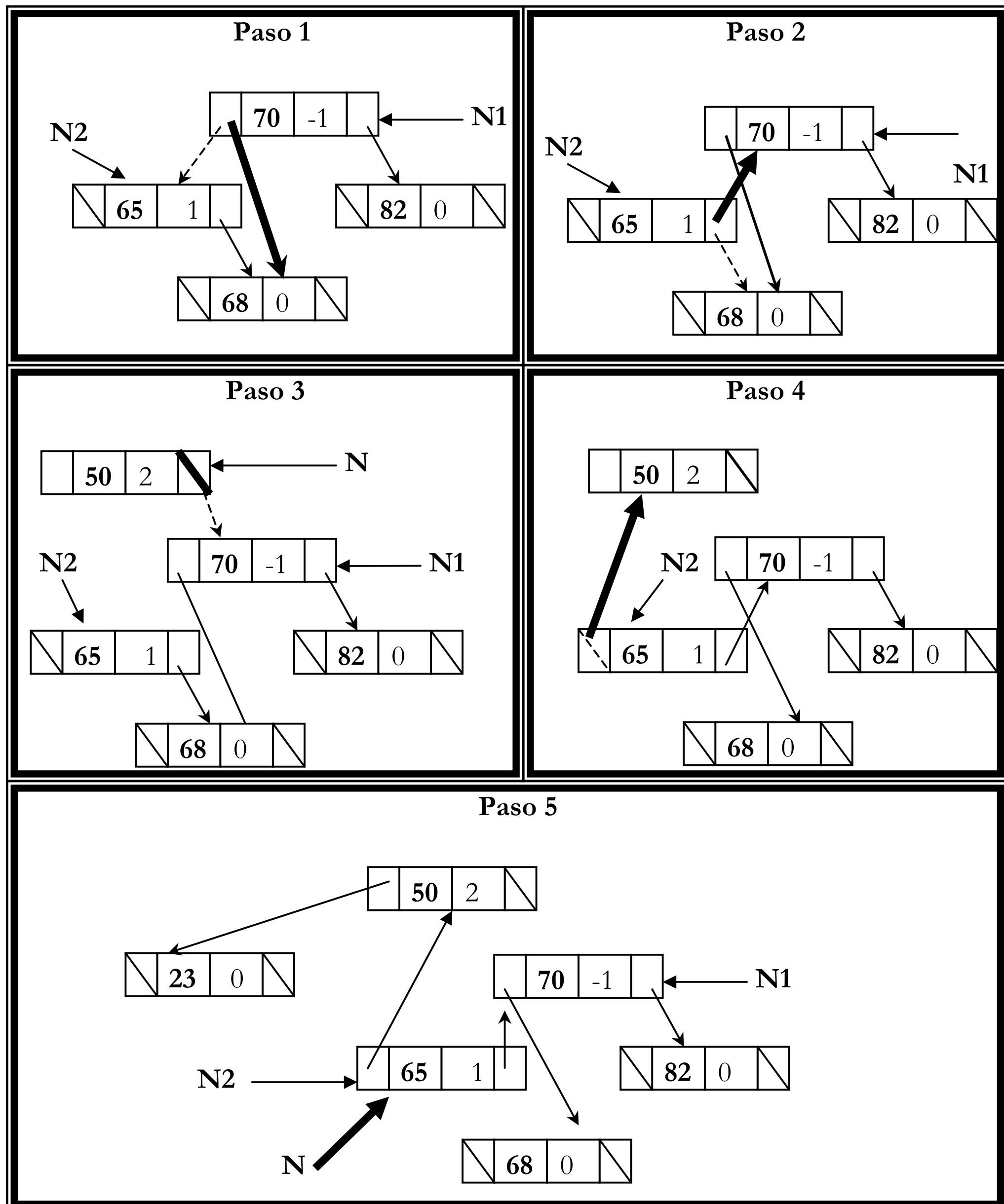


Figura 2.18. Seguimiento del algoritmo de rotación DI

El FE de los nodos involucrados se decide en base a la siguiente tabla:

N2^.FE=-1	N2^.FE=0	N2^.FE=1
N^.FE=0	N^.FE=0	N^.FE=-1
N1^.FE=1	N1^.FE=0	N1^.FE=0
N2^.FE=0	N2^.FE=0	N2^.FE=0

Como el ejemplo presentado el FE de N2 es igual a 1, se realiza las siguientes asignaciones:

N^.FE = -1
N1^.FE = 0
N2^.FE = 0

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

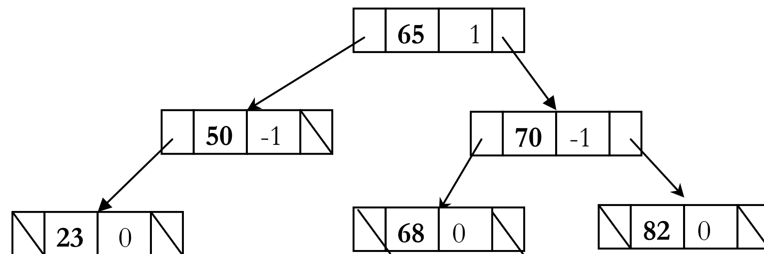


Figura 2.19. árbol reestructurado por DI

g) Inserción : Clave 39

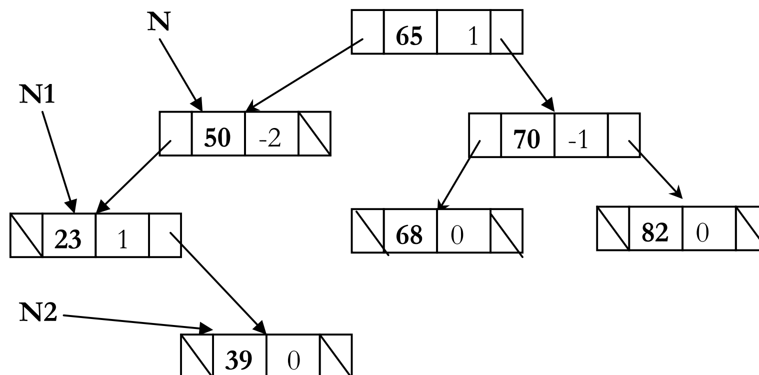
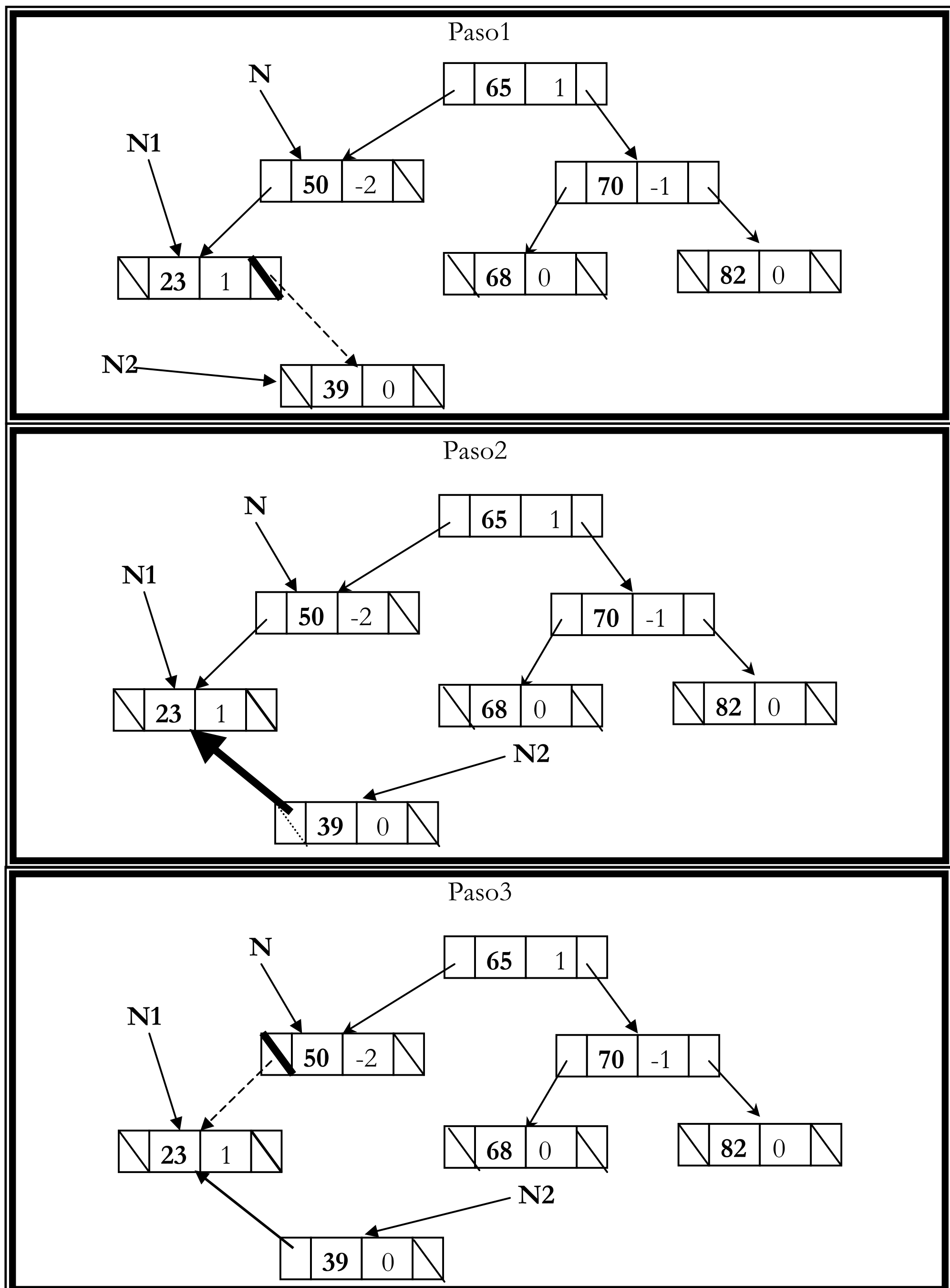


Figura 2.20. Continuación del seguimiento de la inserción de Árbol

El camino seguido para insertar el nodo con la clave 39 ha seguido el camino de izquierda 65, izquierda de 50, derecha de 23. al regresar por el camino de búsqueda, el factor de equilibrio del nodo 29 se incrementa en 1 por seguir el camino de la rama derecha, el del nodo 45 se decrementa en 1 por seguir la rama izquierda y pasa a ser –2 se ha roto el criterio de equilibrio. La rotación de los nodos para reestablecer el equilibrio es ID.

```
1) N1^.Der = N2^.Izq
2) N2^.Izq = N1
3) N^.Izq =N2^.Der
4) N2^.Der = N
5) N = N2
```



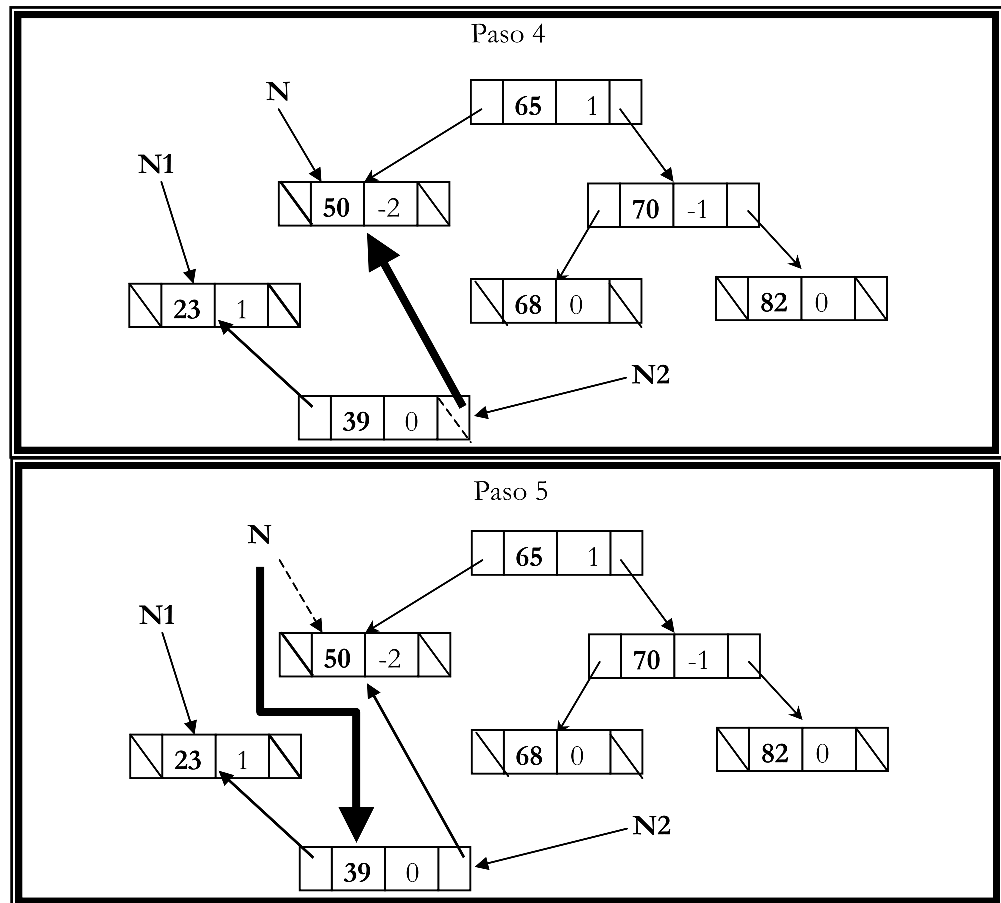


Figura 2.18. Seguimiento del algoritmo de rotación ID

El FE de los nodos involucrados se decide en base a la siguiente tabla:

N2^.FE=-1	N2^.FE=0	N^.FE=1
N^.FE=0	N^.FE=0	N^.FE=-1
N1^.FE=1	N1^.FE=0	N1^.FE=0
N2^.FE=0	N2^.FE=0	N2^.FE=0

Como en el ejemplo presentado el FE de N2 es igual a 0, se realizan las siguientes asignaciones:

```
N^.FE =  0
N1^.FE = 0
N2^.FE = 0
```

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

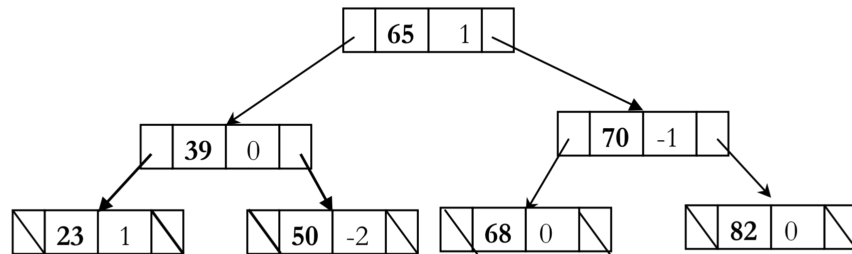


Figura 2.22. Árbol Reestructurado por ID

Procedimiento de Inserción por Balanceo

El algoritmo inserta un elemento en un árbol balanceado. N es una variable de tipo puntero (por referencia). BO es una variable de tipo booleano (por referencia). BO se utiliza para indicar que la altura del árbol ha crecido, su valor inicial es falso. Infor es una variable de tipo entero que contiene la información del elemento que queremos insertar.

```
INSERTABALENACEADO(N,BO, Infor)
N1,N2 : Puntero
Inicio
  Si N<>NULL Entonces
     Si Infor < N^.Info Entonces
        INSERTABALENACEADO(N^.Izq,BO, Infor)
        Si BO Entonces
           Selecciona N^.FE donde
                Caso 1 : N^.FE=0
                         BO=Falso
                Caso 0 : N^.FE=1
                Caso -1:  N1=N^.Izq
                          Si N^.FE<='0' Entonces
                             N^.Izq=N1^.Der
                             N1^.Der = N
                             N^.FE=0
                             N=N1
                          Sino
                             N2=N1^.Der
                             N^.Izq=N2^.Der
                             N2^.Der=N
                             N1^.Der=N2^.Izq
                             N2^.Izq=N1
                             Si N2^.FE=-1 Entonces
                                N^.FE = 1
                             Sino
                                N^.FE = '0'
                             Fin Si
```

```
                      Si N2^.FE=1 Entonces
                         N1^.FE = -1
                      Sino
                         N1^.FE = 0
                      Fin Si
                      N=N2
                   Fin Si
                   N^.FE=0
                   BO=Falso
         Fin Selección
      Fin Si
   Sino
      Si Infor > N^.Inf Entonces
         INSERTABALENACEADO(N^.Der,BO, Infor)
         Si BO Entonces
            Selecciona N^.FE donde
               Caso '-1' : N^.FE=0
                           BO=Falso
               Caso '0' : N^.FE=1
               Caso '1':  N1=N^.Der
                          Si N^.FE>=0 Entonces
                             N^.Der=N1^.Izq
                             N1^.Izq = N
                             N^.FE=0
                             N=N1
                          Sino
                             N2=N1^.Izq
                             N^.Der=N2^.Izq
                             N2^.Izq=N
                             N1^.Izq=N2^.Der
                             N2^.Der=N1
                             Si N2^.FE=1 Entonces
                                N^.FE = -1
                             Sino
                                N^.FE = 0
                             Fin Si
                             Si N2^.FE=-1 Entonces
                                N1^.FE = 1
                             Sino
                                N1^.FE = 0
                             Fin Si
                             N=N2
                          Fin Si
                          N^.FE=0
                          BO=Falso
            Fin Selección
         Fin Si
      Sino
         Escribir " El nodo ya se encuentra en el árbol"
   Fin  Si
Sino
   Crear (N)
```

```
    N^.Info=Infor
    N^.Izq=NULL
    N^.Der=NULL
    N^.FE=0
    BO=Verdadero
  Fin Si
Fin
```

Eliminación de un Nodo en un Árbol Balanceado

La operación de borrado en un árboles balanceados es un poco mas compleja que la operación de inserción. Consiste en quitar un nodo del árbol sin violar los principios que definen justamente un árbol balanceado. Recuérdese 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 mas de una unidad".

La complejidad en la operación de borrado resulta ser cierta a pesar de que se utiliza el mismo algoritmo de borrado (idéntico en lógica pero diferente en implementación)que en los árboles binarios de búsqueda y la misma operación de reacomodo que se utiliza en el algoritmo de inserción en árboles balanceados.

Recuerde que la operación de borrado en árbol balanceado deben distinguirse los siguientes casos:

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

Para eliminar un nodo en un árbol balanceado lo primero que debe hacerse es localizar su posición en el árbol. Se elimina siguiendo los criterios establecidos anteriormente y se regresa por el camino de búsqueda calculando FE de los nodos visitados. Si en alguno de los nodos se viola el criterio de equilibrio, entonces debe de reestructurase el árbol. El proceso termina cuando se llega hasta la raíz del árbol. Cabe aclarar que mientras en el algoritmo de inserción una vez que era efectuada una rotación podía detenerse el proceso, en este algoritmo debe continuarse puesto que se puede producir mas de una rotación en el camino hacia atrás.

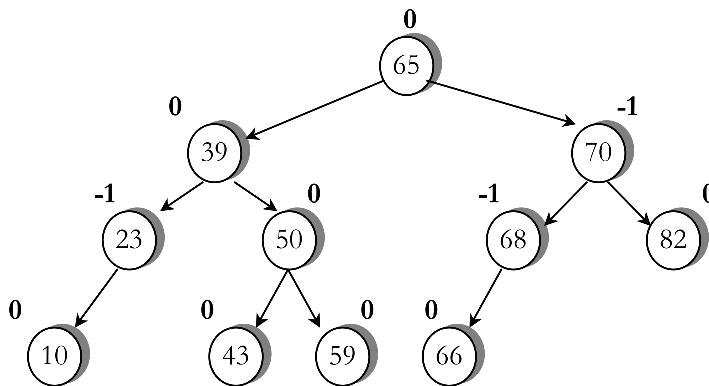
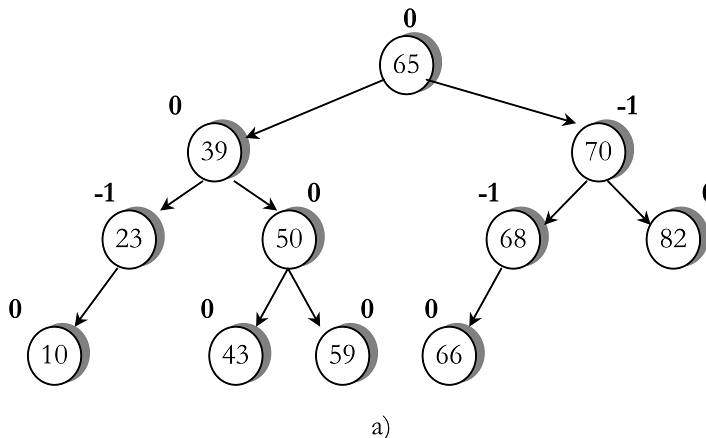
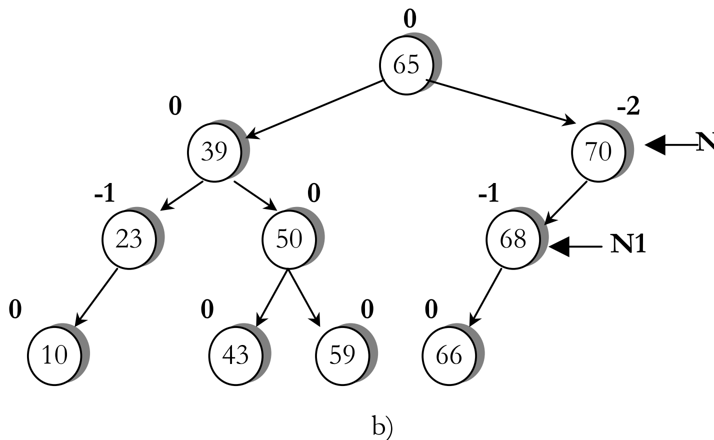


Figura 2.23. Árbol Balanceado.

En el árbol de la figura 2.23 se va eliminando el nodo con clave 82: al ser un nodo hoja es borrado es simple, se suprime el nodo (Figura 2.24 a). Al volver por el camino de búsqueda para determinar FE, resulta que el FE del nodo 70 pasaría a ser -2 ya que ha decrementado la altura de la rama derecha, es violado el criterio de equilibrio (Figura 2.24 b). Sea punta con N la clave 70 y con N1 la rama izquierda de N. Se verifica el FE de N1 y como es igual a –1, entonces se realiza la rotación II. Luego de la reestructuración de el árbol en la raíz 68(Figura 2.24 c).



a)



b)

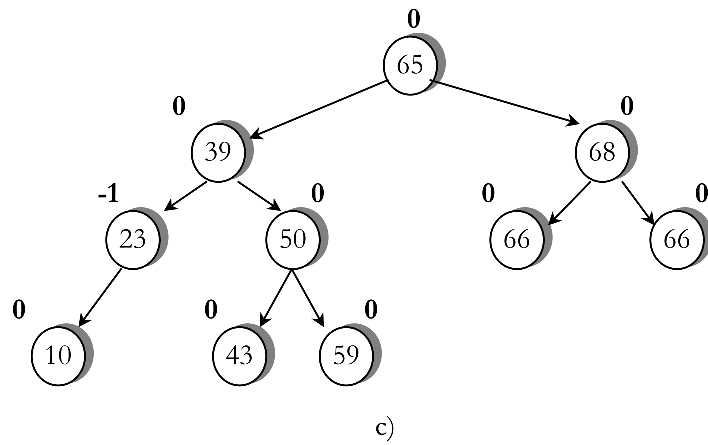


Figura 2.24. Eliminación de un nodo con el método II

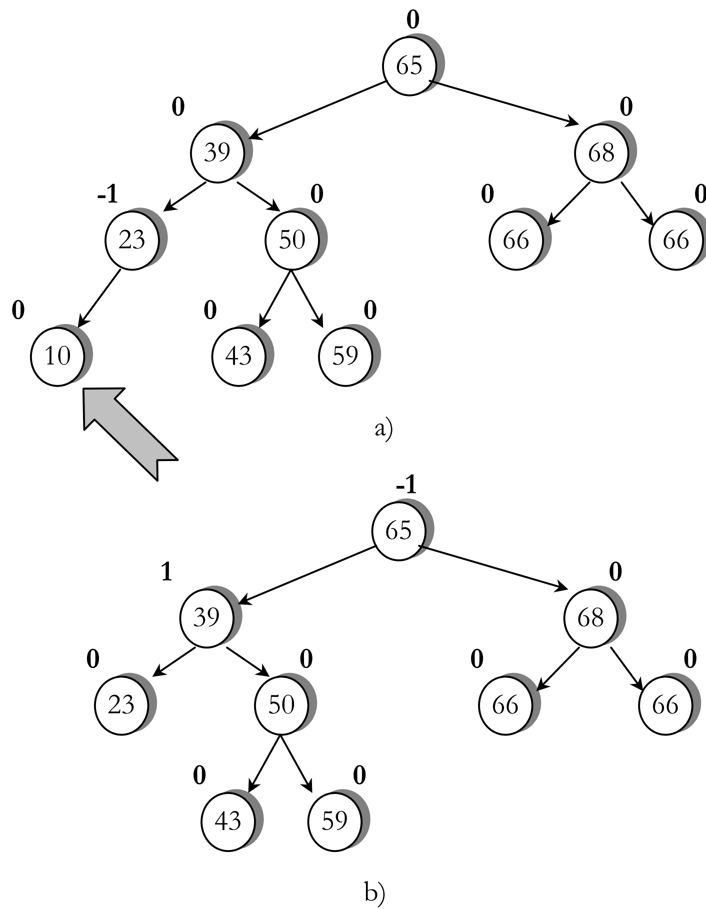
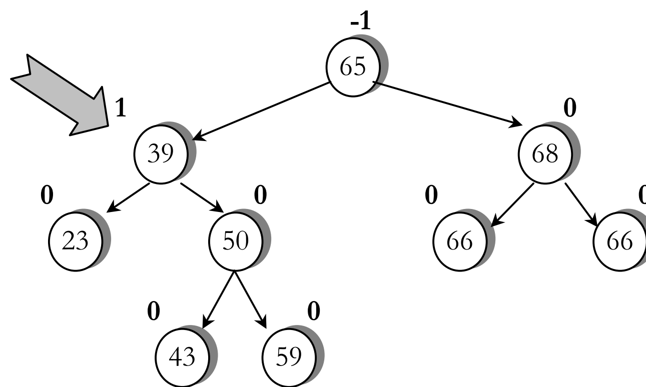
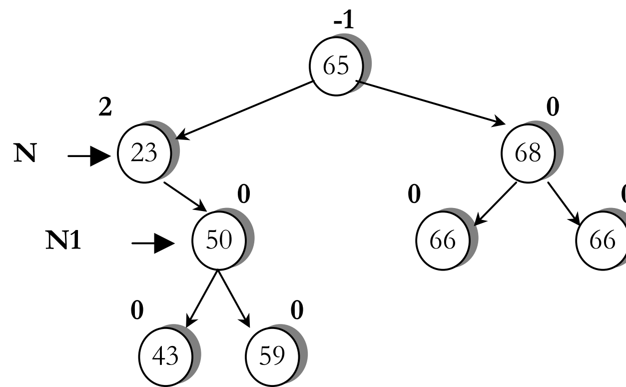


Figura 2.25. Eliminación de un nodo hoja

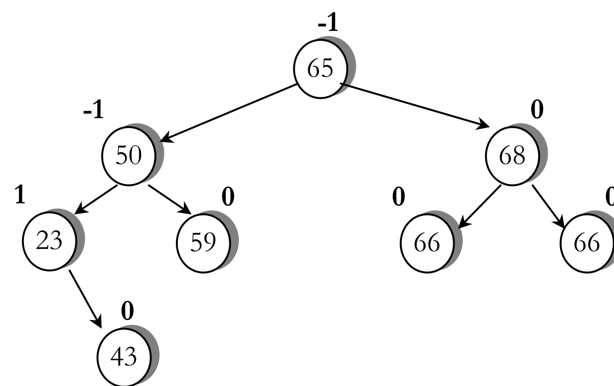
La eliminación de la clave 10 es un proceso sencillo (figura 2.25 a) no debe reestructurar el árbol y solo es necesario cambiar el FE de los nodos 23 y 39 (figura 2.25 b).



a)



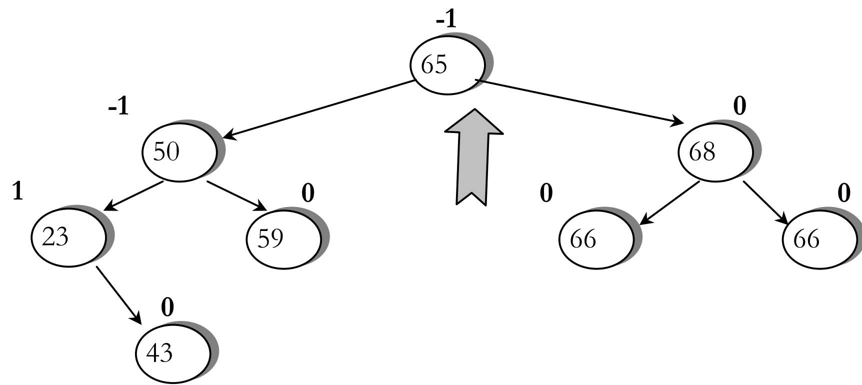
b)



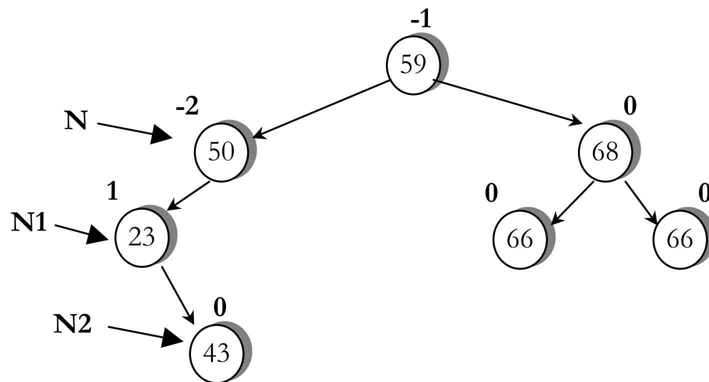
c)

Figura 2.26. Eliminación de un nodo con el método DD

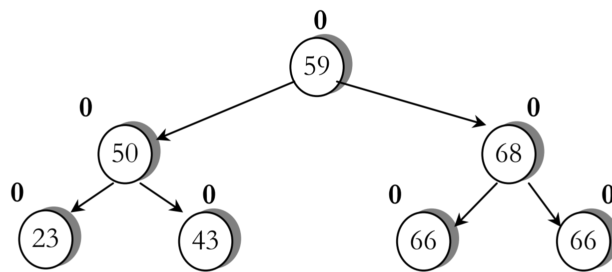
Al eliminar la clave 39 se origina el caso mas difícil de borrado en árboles: la eliminación de una clave con dos descendientes (fig. 2.26 a) se opta por sustituir dicha clave por el nodo que se encuentra mas a la derecha en el subárbol izquierda (23). Luego de la sustitución se observa que en dicha clave se viola el criterio de equilibrio y debe reestructurarse el árbol. Se apunta el con N a la clave 23 y con N1 la rama derecha de N. Se verifica el FE de N1 y como este es igual a 0, se realiza la rotación DD. Luego del reacomodo el árbol queda como en la figura 2.26 c.



a)



b)



c)

Figura 2.27. Eliminación de un nodo con el método ID

Al eliminar la clave 65 surge nuevamente el tercer caso de borrado , que corresponde a una clave con dos descendientes (figura 2.27a). Se sustituye dicha clave por el nodo que se encuentra mas a la derecha en el subárbol izquierdo (59). Es evidente que depuse de la sustitución, en la clave 50 se viola el criterio de equilibrio y que debe reestructurarse el árbol. Se apunta con N a la clave 50 y N1 la rama izquierda N y se verifica el FE. Como en este caso es igual a 1, se apunta N2 la rama derecha de N1 y se realiza la rotación ID, posteriormente se reestructura (Fig. 2.27 c).

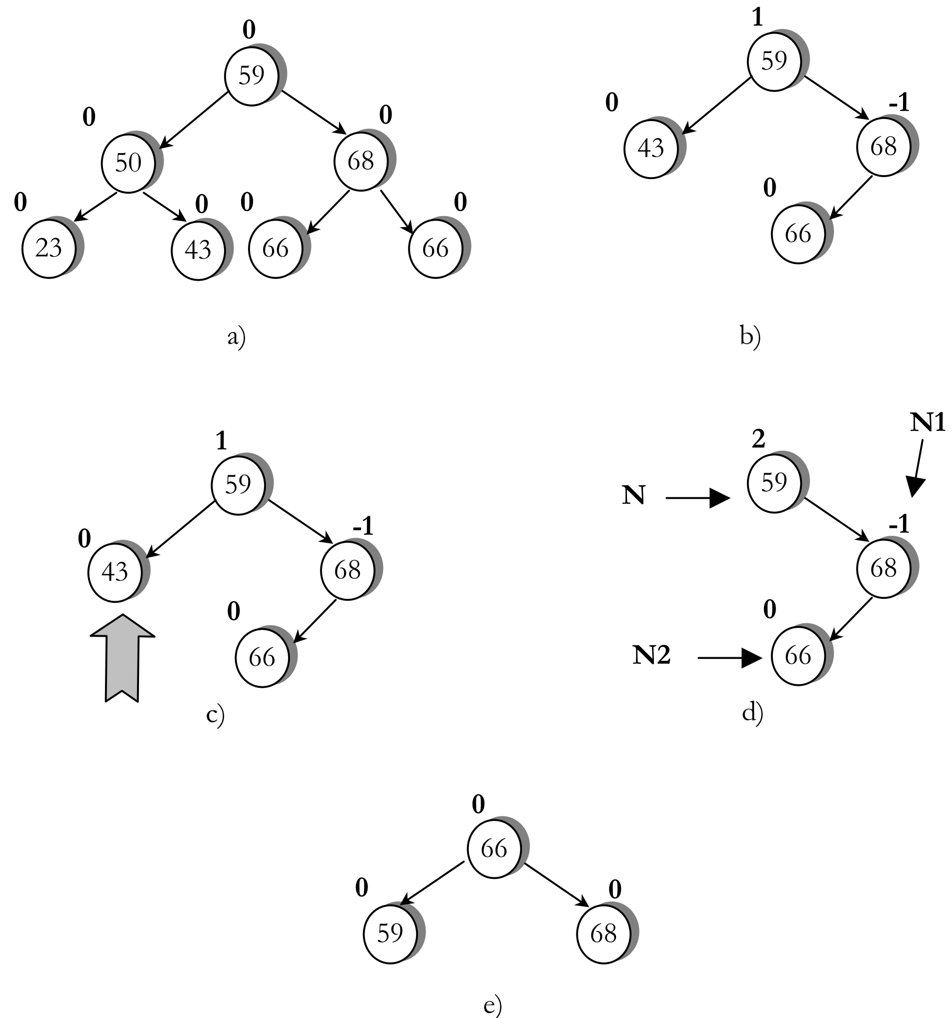


Figura 2.28. Eliminación de un nodo con el método DI

Luego eliminación de las claves 70, 23 y 50, el árbol queda como en la figura 2.28 b. La eliminación de la clave 43 corresponde al primer caso de borrado en árboles, es el caso mas simple (figura 2.28 c). Sin embargo, al verificar el FE de la clave 59 se advierte que se rompe el equilibrio del árbol y debe reestructurarse (figura 2.28 d). Se apunta con N la clave 59 y con N1 la rama derecha de N, y se verifica el FE de N1. Como es igual a –1, se apunta con N2 la rama izquierda de N1 y se realiza la rotación DI. Luego de la reestructuración, el árbol queda como en la figura 2.28 e.

Con el fin de darle mayor modularidad al algoritmo de eliminación en árboles balanceados, se estudiaran tres algoritmos auxiliares. El primero **reestructura1,** se utiliza cuando la altura de la rama izquierda ha disminuido. El segundo, **reestructura2** se emplea cuando la altura de la rama derecha ha disminuido. El ultimo **borrado** se utiliza en el caso mas difícil de borrado en árboles , con el objeto de sustituir el elemento que desea eliminarse por el que se encuentra mas a la derecha en el subárbol izquierdo.

```
REESTRUCTURA1(N,BO)
N1,N2 : Puntero
Inicio
   Si BO Entonces
       Selecciona N^.FE donde
           Caso '1' : N^.FE=0
           Caso '0' : N^.FE=1
                      BO=Falso
           Caso '-1':  N1=N^.Der
                       Si N1^.FE>=0 Entonces
                              N^.Der=N1^.Izq
                              N1^.Izq = N
                              Selecciona N1^.FE donde
                                         Caso '0' : N^.FE=1
                                                    N1^.FE=-1
                                                    BO=Falso
                                         Caso '1':  N^.FE=0
                                                    N1^.FE=0
                              Fin Selecciona
                              N=N1
                        Sino
                              N2=N1^.Izq
                              N^.Der=N2^.Izq
                              N2^.Izq=N
                              N1^.Izq=N2^.Der
                              N2^.Der=N1
                              Si N2^.FE=1 Entonces
                                  N^.FE = -1
                               Sino
                                  N^.FE = 0
                               Fin Si
                               Si N2^.FE=-1 Entonces
                                  N1^.FE = 1
                               Sino
                                  N1^.FE = 0
                               Fin Si
                               N=N2
                               N2^.FE=0
                           Fin Si
       Fin Seleccionar
    Fin Si
Fin
```

```
REESTRUCTURA2(N,BO)
N1,N2 : Puntero
Inicio
  Si BO Entonces
     Selecciona N^.FE donde
        Caso '1' : N^.FE=0
        Caso '0' : N^.FE=-1
                   BO=Falso
        Caso '-1':  N1=N^.Izq
                    Si N1^.FE<=0 Entonces
                         N^. Izq =N1^.Der
                         N1^.Der = N
                         Selecciona N1^.FE donde
                                 Caso '0' : N^.FE=-1
                                            N1^.FE=1
                                            BO=Falso
                                 Caso '-1':  N^.FE=0
                                            N1^.FE=0
                         Fin Selecciona
                         N=N1
                     Sino
                         N2=N1^.Der
                         N^.Izq=N2^.Der
                         N2^.Der=N
                         N1^.Der=N2^.Izq
                         N2^.Izq=N1
                         Si N2^.FE=-1 Entonces
                             N^.FE = 1
                          Sino
                             N^.FE = 0
                          Fin Si
                          Si N2^.FE=-1 Entonces
                             N1^.FE = -1
                          Sino
                             N1^.FE = 0
                          Fin Si
                          N=N2
                          N2^.FE=0
                       Fin Si
     Fin Seleccionar
   Fin Si
Fin
```

```
BORRADO(Aux,Otro,BO)
Inicio
  Si Aux^.Der<>NULL Entonces
    BORRADO(Aux^.Der,Otro,BO)
    Reestructura2(Aux,BO)
  Sino
    Otro^.Info=Aux^.Info
    Aux=Aux^.Izq
    BO=Verdadero
  Fin Si
Fin
```

A continuación se muestra el algoritmo de eliminación en árboles balanceados.

```
EliminaBalanceado(N,BO,Infor)
Otro : Puntero
Inicio
  Si N<> NULL Entonces
    Si Infor<N^.Info Entonces
      EliminaBalanceado(N^.Izq,BO,Infor)
      Reestructura1(N,BO)
    Sino
      Si Info> N^.Info Entonces
        EliminaBalanceado(N^.Der,BO,Infor)
        Reestructura2(N,BO)
      Sino
        Otro= N
        Si Otro^.Der=NULL Entonces
          N=Otro^.Izq
          BO=Verdadero
        Sino
          Si Otro^.Izq=NULL Entonces
            N=Otro^.Der
            BO=Verdadero
          Sino
            Borrado(Otro^.Izq,Otro,BO)
            Restructura1(N,BO)
            Quita(Otro)
          Fin Si
        Fin Si
      Fin Si
    Fin Si
  Sino
    Escribir “El nodo no se encuentra en el arbol”
  Fin Si
Fin
```

Ejercicios

2.1 Dada la secuencia de claves enteras:100, 29, 71, 82, 48, 39, 101, 22, 46, 17, 3, 20, 25, 10.Representar gráficamente el árbol AVL correspondiente. Elimine claves consecutivamente hasta encontrar un desequilibrio y dibuje la estructura del árbol tras efectuarse la oportuna restauración.

2.2 Obtener la secuencia de rotaciones resultante de la inserción del conjunto de elementos {1,2,3,4,5,6,7,15,14,13,12,11,10,9,8} en un árbol AVL.

2.3 Inserte las claves en el orden indicado a fin de incorporarlas a un árbol AVL.

a. 10,100,20,80,40,70.
b. 5,10,20,30,40,50,60.

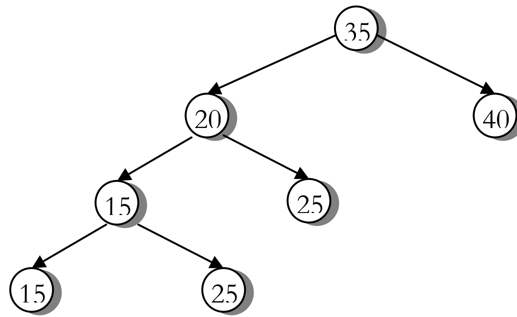
Problemas

2.4 Escribir un algoritmo de inserción eficiente para un árbol de búsqueda binaria con el fin de incluir un registro nuevo cuya llave se sabe que no existe en el árbol.
2.5 Muestre que es posible un árbol de búsqueda binaria en el cual solo existe una hoja única incluso si los elementos del árbol no están insertados en orden estrictamente ascendente o descendente.
2.6 verificar por simulación que si están no presentes registros para el algoritmo de búsqueda e inserción en el árbol binario por orden aleatorio, la cantidad de comparaciones de llaves es O(Log n).
2.7 Muestre que cada árbol de búsqueda binaria de n nodos no tiene la misma probabilidad (suponiendo que los elementos se insertan en orden aleatorio) y que los árboles AVL son mas probables que los árboles en línea recta.
2.8 Escriba un algoritmo para suprimir un nodo de un árbol binario que sustituye el nodo con su antecesor de orden inmediato y no con su sucesor de orden inmediato.
2.9 Escribir un algoritmo que busque y suprima un registro con la llave a partir de un árbol de búsqueda binaria.

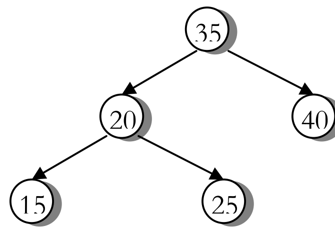
K	R	LEFT	RIGHT

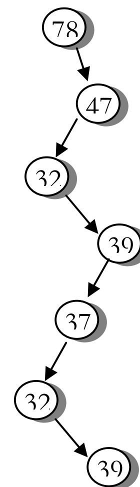
2.10 Escribir un algoritmo que elimine todos los registros entre un rango de llaves de un árbol de búsqueda binaria cuyo no es similar al registro anterior

2.11 considere los siguientes árboles de búsqueda



a)





b) c)

a) Cuantas permutaciones de los enteros del 1 al 7 producirán los de búsqueda binaria.
b) Cuantas permutaciones de los enteros del 1 al 7 producirán los de búsqueda binaria similares
c) Cuantas permutaciones de los enteros del 1 al 7 producirán los de búsqueda binaria con la misma cantidad de nodos en cada nivel iguales a los árboles de las figuras.

d) Encontrar una asignación de probabilidades para los primeros 7 enteros positivos como algoritmos de búsqueda que hacen optimo cada uno de los árboles.

2.12 Muestra que el árbol Fibonacci de orden h+1, es un árbol de altura balanceada h y que tiene menos nodos que cualquier otro árbol de altura balanceada h.

Capítulo

3

Árboles Multicaminos

Las variantes de los árboles binarios hasta el momento fueron desarrolladas para funcionar en 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 del mismo en forma eficiente. Debe recordarse además, 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 pagina (contiene varios registros) en memoria secundaria, por ejemplo un disco, se mide en milisegundos. El tiempo de acceso claro esta es miles de veces mas rápido en la memoria principal que en la memoria secundaria.

Considérese el caso de almacenar un árbol binario en disco. Se necesitara en promedio, para localizar alguno de los nodos, $log_d\ n$ accesos a disco, donde n representa el numero de nodos del árbol y d el orden del mismo, que en este caso es igual a 2. por ejemplo si un árbol contiene 1 000 000 elementos , se necesitar aproximadamente 20 accesos a disco. Ahora bien, si el árbol esa organizado en paginas, de tal manera que cada pagina contiene como mínimo 100 elementos, entonces se necesitarían como máximo 3 accesos s disco (log_{100} 1 000 000). Nótese que el acceso a disco disminuyen de modo considerable.

Existen diferentes técnicas para la organización de archivos indexados, sin embargo la organización en árboles-B y específicamente su variante, árboles B+ son mas utilizadas.

Árboles-B

El Árbol-B tiene una corta pero importante historia. A finales de los años 60, fabricantes de computadoras y grupos de investigación independientes, compitieron en el desarrollo de sistemas de archivo de propósito general y los llamados "métodos de acceso" para sus máquinas. La Sperry Univac Corporation (junto con la Case Western Reserve University), H. Chiat, M. Schwartz, y otros desarrollaron e implementaron un sistema el cual llevaba a cabo las operaciones de inserción y búsqueda de una manera afín al método del árbol-B que describiremos brevemente. Independientemente, B. Cole, S. Radcliffe, M. Kaufman, y otros desarrollaron un sistema similar al de Control Data Corporation (junto con la Stanford University).R. Bayer y E. McCreight, entonces en los Boeing Scientific Research Labs, propusieron un mecanismo externo para el índice con un costo relativamente bajo para la mayoría de las operaciones; lo llamaron árbol-B.

Esta sección presenta las estructuras de datos básicas del árbol-B y los algoritmos de mantenimiento como una generalización de los árboles de búsqueda binaria en la cual salen más de dos caminos de un mismo nodo; la siguiente sección discute sobre lo costos de cada operación.

Es costumbre denominar a los nodos de un árbol B, paginas. Cada nodo, cada pagina, es una unidad a la que se accede en bloque.

La estructura de los datos que representa un árbol B de orden *m* tiene las siguientes características:

- Todas las hojas están en el mismo nivel.
- Todas las paginas internas, menos la raíz, tiene a lo sumo *m* ramas (no vacías) y con mínimo m/2 (redondeando al máximo entero) ramas.
- El numero de claves en cada pagina interna es uno menos que el numero de sus ramas, y estas claves dividen las de las ramas a manera de un árbol de búsqueda.
- La raíz tiene como máximo *m* ramas, puede llegar a tener hasta 2 y ninguna si el árbol consta de la raíz solamente.

Representación de un árbol B de orden m

Para representar una pagina o nodo del árbol B tenemos que pensar en almacenar la dirección de las ramas que cuelgan de los nodos. Para ello se utiliza dos vectores y un campo adicional que en todo momento contendrá el numero de claves de la pagina.

La declaraciones siguientes se refieren a un árbol B de orden 5.

```
Max=4
Min=2
Pagina  Record
    Cuenta : integer
    Info : arreglo[1..Max] de Entero
    Ramas :  arreglo[0...Max] de Pagina
Fin Record
```

La razón de indexar ramas desde 0 esta en que cada pagina tiene un mismo numero de ramas igual al de claves mas 1, y que es la forma mas fácil de acceder a la rama más a la izquierda.

Búsqueda

Hay que tener en cuenta que en un árbol binario de búsqueda, la rama elegida en un nodo depende del resultado de una comparación de la clave buscada y de la clave almacenada en el nodo. Si la clave buscada es menor que la clave almacenada, se elige la rama de la izquierda; si es mayor, se sigue por la rama de la derecha. La Figura 3.1 muestra parte de un árbol utilizado para almacenar los números de empleados, y el camino elegido para la clave de búsqueda 15.

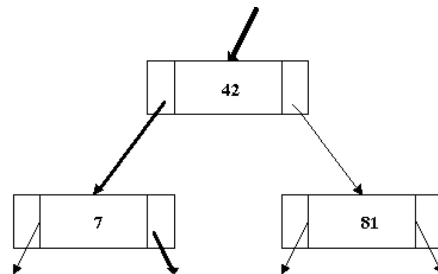


Figura 3.1. Parte de un árbol binario de búsqueda para números de empleado. El camino seguido para la búsqueda "15" está sobreimpreso.

Veamos ahora la Figura 3.2 en la que se muestra un árbol de búsqueda modificado con dos claves almacenadas en cada nodo. La búsqueda se realiza mediante la elección de uno de los tres caminos de cada nodo. En la figura, la clave buscada es 15, es menor que 42 por lo que tomaremos el nodo más a la izquierda de los de la raíz. Para las claves entre 42 y 81 se elegir el descendiente central, mientras que elegiremos el más a la derecha para aquellas claves mayores de 81. El procedimiento de decisión se repite en cada nodo hasta que se encuentra una clave exactamente igual (éxito) o hasta que se llega a una hoja (error).

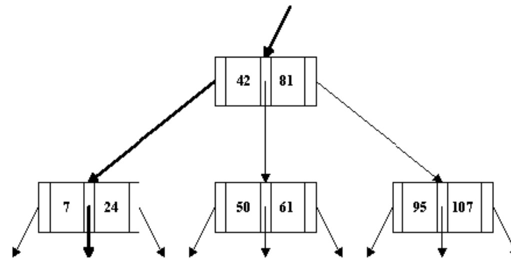


Figura 3.2. Un árbol de búsqueda con 2 claves y 3 ramas por nodo. El camino para la búsqueda '15' está sobreimpreso.

En general, cada nodo en un árbol-B de orden d contiene como mucho 2d claves y 2d+1 punteros, como se observa en la Figura 3.3. De hecho, el número de claves puede variar de un nodo a otro, pero cada uno debe tener al menos d claves y d+1 punteros. Como consecuencia, cada nodo está completo a la mitad de su capacidad. En la implementación más usual un nodo forma un registro del fichero índice, tiene una longitud fija capaz de almacenar 2d claves y 2d+1 punteros, y contiene información adicional, que nos informa de cuantas claves están almacenadas correctamente en el nodo.

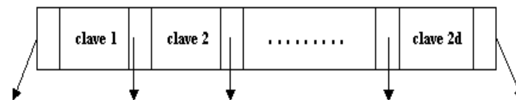


Figura 3.3. Un nodo en un árbol-B de orden d con 2d claves y 2d + 1 punteros.

Normalmente, los nodos con gran número de claves no pueden ser almacenados en memoria principal y requieren accesos a memoria secundaria cada vez que es necesario consultarlos. Más adelante, veremos como, bajo un criterio de coste, mantener más de una clave por nodo disminuye el coste de las operaciones de inserción, y borrado.

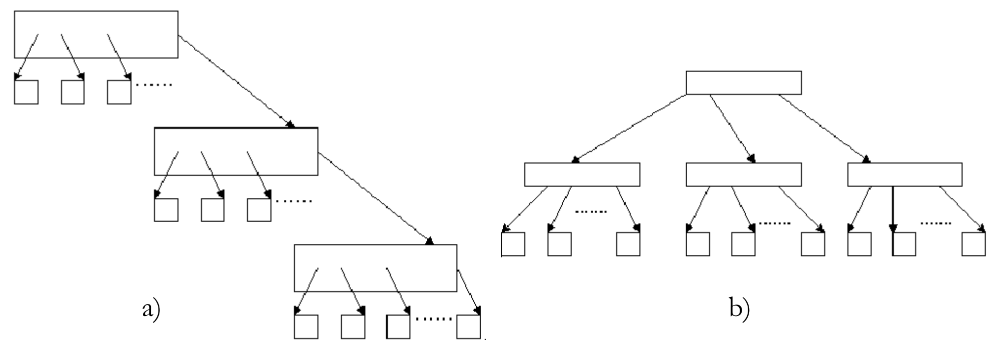


Figura 3.4.(a) Un árbol no balanceado con muchos caminos largos y (b) un árbol balanceado con todos los caminos hasta las hojas exactamente de la misma longitud.

El algoritmo sigue la misma estrategia que la búsqueda en un árbol binario de búsqueda. Salvo que en los árboles B cuando estamos posicionados en una pagina (nodo) hay que inspeccionar las claves de que consta. La inspección de cómo resultado la posición de la clave, o bien el camino a seguir en la búsqueda.

La operación es realizada por el procedimiento que tiene como entrada la raíz del árbol B y la clave a localizar. Devuelve un valor lógico (verdadero si se encuentra la clave), la dirección de la pagina que contiene a la clave y la posición dentro del nodo. El examen de cada pagina se realiza en el procedimiento auxiliar *Buscanodo*.

```
Buscarnodo(Infor,N,Enc,K)
Inicio
   Si Infor< N^.info[1] Entonces
      Enc =falso
      K=0
   Sino
      K=N^.Cuenta
      Mientras  infor <N^.info[k] and k>1 hacer
            K=K-1
      Si  Infor=N^.clave[k] Entonces
         Enc =verdadero
      Fin Si
   Fin Si
Fin
```

```
Buscar(Infor,Raiz,Enc,N,K)
Inicio
   Si Raiz =NULL Entonces
      Enc =falso
   Sino
      Buscarnodo(Infor,Raiz,Enc,K)
      Si  Enc Entonces
         N=Raiz
      Sino
         Buscar(Infor,Raiz^.ligas[k],Enc,N,K)
      Fin Si
   Fin Si
Fin
```

La belleza de los árboles-B reside en los métodos para insertar y borrar registros, que dejan siempre el árbol balanceado. Al igual que en el caso de los árboles binarios de búsqueda, inserciones aleatorias de registros dentro de un fichero pueden dejar el árbol sin balancear. Mientras que un árbol no-balanceado como el que se muestra en la Figura 3.4. a, tiene algunas ramas más largas que otras, un árbol balanceado, como el mostrado en la Figura 3.4. b, tiene todas las hojas a la misma

profundidad. Intuitivamente, los árboles-B tiene una forma como la que se muestra en la Figura 3.5. El camino más largo en un árbol-B de n claves contiene como mucho unos $\log_d$ (n) nodos, siendo d el orden del árbol-B. Una operación de búsqueda puede visitar n nodos en un árbol no balanceado indexando un fichero de n registros, pero en un fichero igual con estructura de árbol-B de orden d no visitaría mas de 1+logd(n) nodos. Puesto que cada visita requiere un acceso al dispositivo de almacenamiento secundario, el balanceo del árbol tiene un gran ahorro potencial. Se han propuesto varios esquemas para árboles balanceados

Cada esquema requiere un tiempo de cómputo para realizar el balanceo, por lo que el ahorro en las operaciones de recuperación debe ser mayor que el tiempo de balancearlo. El esquema de balanceo del árbol-B restringe los cambios en el árbol a una sola rama desde las hojas al raíz, así que no se puede producir una 'fuga' en la raíz. Además el mecanismo de balanceo, utiliza almacenamiento extra para reducir los costes de balanceo, (supuestamente el almacenamiento secundario es barato comparado con el tiempo de búsqueda).

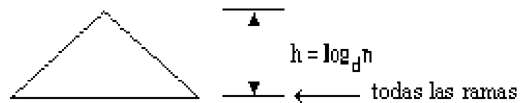


Figura 3.5. La forma de un árbol-B de orden d como índice de un archivo con n registros

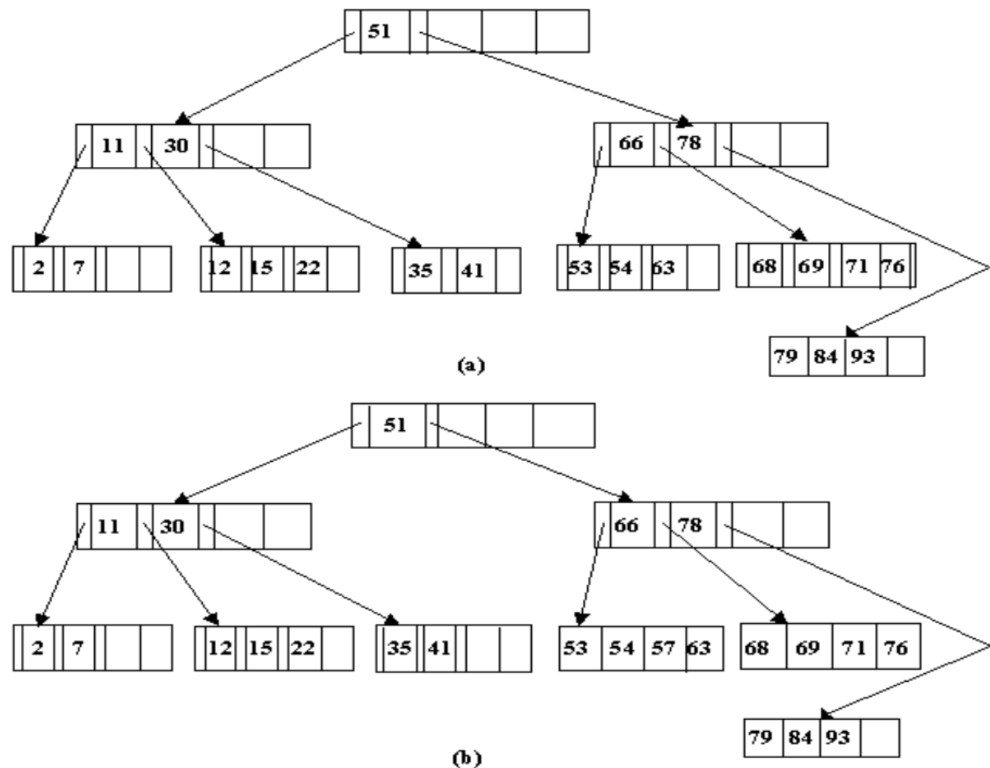


Figura 3.6. (a) Un árbol-B de orden 2, y (b) el mismo árbol, tras la inserción de la clave "57"

Nótese que el número de claves en el nodo raíz puede ser menos que d, el orden del árbol-B. Todos los demás nodos tienen como mínimo d claves. Por lo tanto, los árboles-B tienen las ventajas del esquema de un árbol balanceado, mientras evita algunos de los costes (en tiempo) de mantenimiento.

Proceso de Creación en un Árbol B

La condición de que todas las hojas de un árbol B se encuentren en el primer nivel impone el comportamiento característico de los árboles B: crecer hacia arriba, crecen en la raíz.

El método que se sigue para añadir un nuevo elemento a un árbol B es el siguiente:

- Primero se busca si la clave a insertar esta ya en el árbol, para lo cual se sigue el camino de búsqueda .
- En caso de que la clave no este en el árbol, la búsqueda termina en un nodo hoja. Entonces la nueva clave se inserta en el nodo hoja. Mas bien se intenta insertar en el nodo hoja.
- De no estar lleno el nodo hoja, la inserción es posible en dicho nodo y termina la inserción de la clave.
- El comportamiento característicos de los árboles B se pone de manifiesto ahora. De esta la hoja llena, la inserción no es posible en dicho nodo, entonces se divide el nodo (incluyendo virtualmente la nueva clave) en dos nodos en el mismo nivel del árbol, excepto la clave media que no se incluye en ninguno de los dos nodos, si no que sube en el árbol por el camino de búsqueda para que a su vez inserte en el nodo antecedente. Es por esto por lo que se dice que el árbol crece hacia arriba. En esta ascensión de claves medianas puede ocurrir que llegue al nodo raíz, entonces este se divide en dos nodos y la clave enviada hacia arriba se convierte en una nueva raíz. Esta es la forma de que el árbol B crezca en altura.

Un seguimiento de la creación de un árbol B de orden 5 con la clave entero sin signo. Al ser de orden 5 el numero máximo de claves en cada nodo será 4. las claves que se insertaras son:

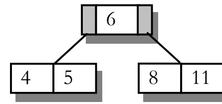
6 11 5 4 8 9 12 21 14 10 19 28 3 17 32 15 16 26 27

- Con las cuatro primeras se a completa el primer nodo, eso si ordenadas crecientemente a la manera de un árbol de búsqueda.

4	5	6	11

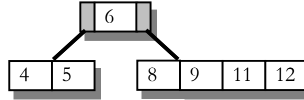
a)

- La clave siguiente es 8 encuentra el nodo ya lleno. La clave mediana de las 5 es 6. el nodo lleno se divide en dos, excepto la clave mediana que sube y se convierte en nueva raíz.



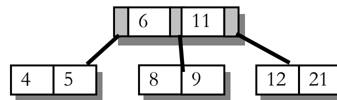
b)

- Las siguientes claves 9,12 se insertan siguiendo el criterio de búsqueda, en el nodo rama derecha de la raíz por ser ambas claves mayores que 6.



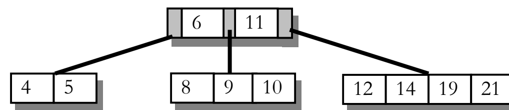
c)

- La clave 21 sigue por el camino de búsqueda, derecha, del nodo raíz. El nodo donde debe insertarse esta lleno, se parte en dos nodos y la clave mediana 11 asciende por el camino de búsqueda para ser insertada en el nodo raíz.



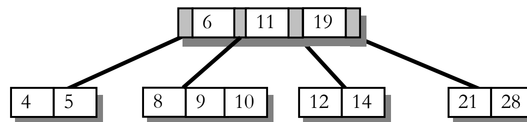
d)

- Las siguientes claves 14,10,19 son insertadas en los nodos de las rama que se corresponden con el camino de búsqueda.



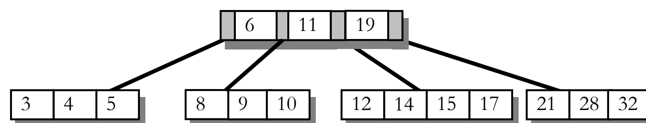
e)

- La insertar la clave 28 de nuevo se va a producir el proceso de división del nodo derecho del ultimo nivel y ascensión por el camino de búsqueda de la clave mediana 19.



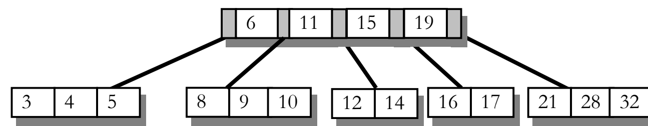
f)

- continuando con la inserción de las claves 3, 17, 32, 15 son insertadas en los nodos de hojas siguientes.



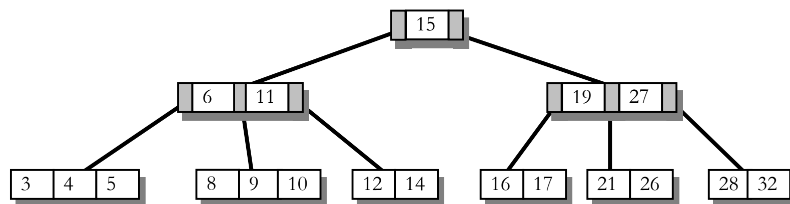
g)

- Al llegar la clave 16, baja por el camino de búsqueda (rama derecha de clave 11 en nodo raiz). El nodo donde va ser insertado esta lleno. Se produce la división en dos nodos y la ascensión de la clave mediana 15.



h)

- Por ultimo, al insertar las claves 26 y 27, primero se llena el nodo mas a la derecha del ultimo nivel con la clave 26. Con la clave 27, que sigue el mismo camino de búsqueda, ocurre que provoca que se divida en dos el nodo y ascienda el nodo padre de la clave 27. Ahora bien ocurre que el nodo padre también esta lleno por lo que se divide en dos, sube el nodo 15, pero como el nodo dividido es la raíz, con esta clave se forma un nuevo nodo raíz: el árbol a crecido en altura. El árbol crece hacia arriba, hacia la raíz.



i)
Figura 3.7 Proceso de Creación de un árbol B

En esta formación del árbol B que hemos ido realizando podemos observar estos dos hechos relevantes.

- Una división de un nodo, prepara a la estructura para inserciones simples de nuevas claves.

- Siempre es la clave media la que sube al nodo antecedente. La mediana no tiene porque coincidir con la clave que se esta insertando. Por lo que podemos afirmar que no importa el orden en que lleguen las claves en el balanceo del árbol.

Operaciones de Un árbol B

Los árboles B mas utilizados cuando se manejan memoria principal son los de orden 5; un orden mayor aumentaría considerablemente la complejidad de los algoritmos y un orden menor disminuiría la eficiencia de la localización de la clave.

La operaciones que se manejan en los árboles B son similares a la que se vieron en los capítulos anteriores que fueron Inserción y Eliminación solo que los

algoritmos que se presentan en cada una de las operaciones manejan datos en memoria principal.

Inserción

Para ilustrar cómo se mantiene el balanceado durante la inserción, considérese la Figura 3.6. a la cual muestra un árbol-B de orden 2. Puesto que cada nodo en un árbol-B de orden d contiene entre d y 2d claves, cada nodo del ejemplo tiene entre 2 y 4 claves. En cada nodo debe existir un indicador (que no está reflejado en la figura) para informar sobre el número de claves que tiene el nodo. La inserción de un nuevo nodo requiere un proceso de dos pasos. Primero, un proceso de búsqueda desde la raíz para localizar la hoja apropiada para la inserción. Entonces se realiza la inserción, y se restablece el balanceado del árbol mediante un procedimiento, el cual se aplica desde la hoja hacia la raíz. Refiriéndonos a la Figura 3.6. a, uno puede ver que cuando inserta la clave "57" la búsqueda termina sin éxito en la cuarta hoja. Ya que la hoja puede alojar otra clave, se inserta la clave nueva simplemente, dando lugar al árbol que se muestra en la Figura 3.6.b . Si la clave "72" fuera insertada, sin embargo, habría complicaciones porque la hoja correspondiente ya está llena. Cuando quiera que una clave necesite se insertada en un nodo que ya está lleno, ocurre una división: se divide el nodo como se muestra en la

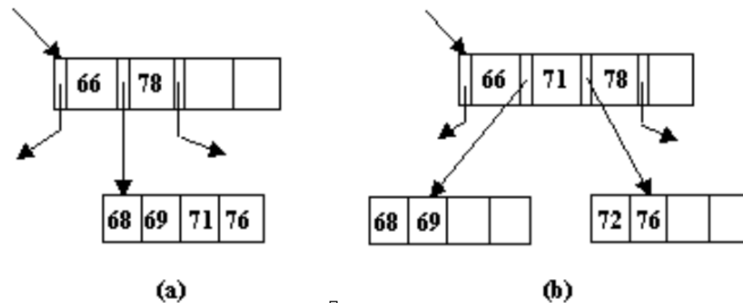


Figura 3.8. (a) Una hoja y su antecesor en un árbol-B, y (b) el mismo árbol tras la inserción de la clave "72". cada nodo conserva entre 2 y 4 claves (d y 2d).

La siguiente clave siempre está en la hoja más a la izquierda del subárbol apuntado por el puntero derecho de la posición vacía.

Figura 3.8. De las 2d+1 claves, se pone el d más pequeño en un nodo, el d más grande se pone en otro nodo, y se asciende el valor restante al nodo padre donde sirve como un separador. Normalmente el nodo padre alojará una clave adicional y el proceso de inserción termina. Si ocurriera que el nodo padre estuviera lleno también, entonces se aplicaría el mismo proceso de división de nuevo. En el peor de los casos, la división se propaga todo el camino hasta la raíz y el árbol aumenta en altura un nivel. De hecho, un árbol-B sólo incrementa en altura debido a una división hacia la raíz.

Una formulación recursiva va a ser la adecuada para reflejar el proceso de propagación en la división de las paginas, debido a que se realiza en el sentido inverso por el camino de búsqueda.

El diagrama de bloques del proceso de inserción se muestra en la siguiente figura.

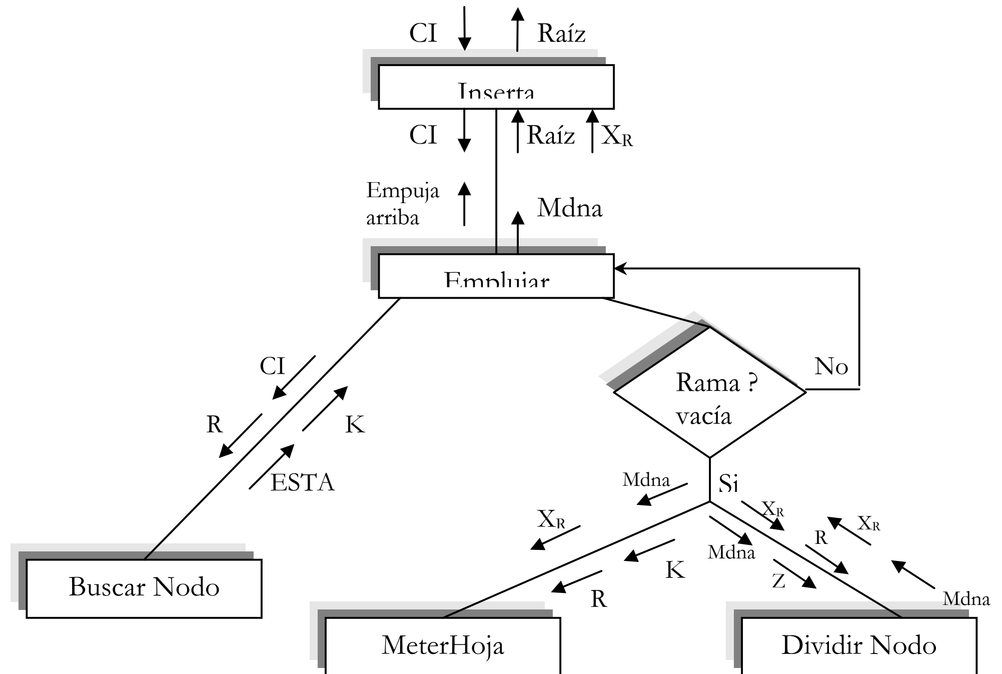


Figura 3.9. Diagrama de bloques del proceso de Insercion

El procedimiento de inserción recibe la clave nueva y la raíz del árbol B. Este procedimiento pasa control al procedimiento empujar que baja por el camino de búsqueda determinado por Buscanodo, hasta llegar a un nodo hoja. El procedimiento Meterhoja realiza la acción de añadir la clave en el nodo. Si el nodo tiene ya el máximo de claves, es llamado el procedimiento dividirnodo para que a partir del nodo lleno se formen dos nodos y ascienda la clave mediana.

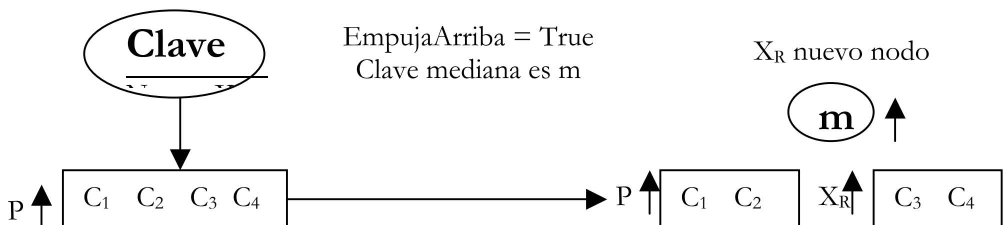


Figura 3.10. Proceso de división de nodo

La propagación hacia arriba del proceso de división puede llegar hasta la raíz, por tanto aumentar la altura del árbol. La variable EmpujaArriba detecta este hecho en el procedimiento Insertar.

```
Inserción (CI,Raiz)
EmpujarArriba : Booleano
X : Tipo clave
Xr, P : Puntero
Inicio
    Empujar (CI, Raiz, EmpujaArriba, X, Xr)
    Si EmpujaArriba Entonces
        Crear(P)
        P^.Cuenta=1
        P^.Claves[1]=X
        P^.Ramas[0]=Raiz;
        P^.Ramas[1]=Xr;
        Raiz=P
    Fin Si
Fin
```

En el procedimiento Empujar controla la realización de las tareas mas importantes en el proceso de inserción de una clave. Lo primero que hace es bajar por el camino de búsqueda hasta llegar a una rama vacía. Cuando esto ocurre se devuelve la clave y se activa el interruptor que indica que la clave "sube". Al subir por el camino de búsqueda primero se encuentra con el nodo hoja. Si hay huecos, se inserta la clave y el proceso termina. De estar lleno, contador de claves cuenta tiene el máximo de ellas, el procedimiento Dividirnodo crea un nuevo nodo (Xr tiene su dirección) que junto al nodo original reparte las claves, y la clave mediana sigue subiendo por el camino de búsqueda. La forma mas fácil de codificar el hecho de bajar y luego subir por el camino de búsqueda es mediante llamadas recursivas a Empujar, de tal forma que cuando las llamadas retornan control se esta volviendo por el camino de búsqueda, con el swith EmpujaArriba se detecta si ha habido división de nodo y por tanto hay una clave mediana.

```
Empujar (CI,R,EmpujaArriba,Mdna,Xr)
Esta : Booleano
K : Entero
Inicio
    Si R=NULL Entonces
        EmpujaArriba=True
        Mdna=CI
        Xr=NULL
    Sino
        Buscarnodo(CI,R,Esta,K)
        Si Esta Entonces
            Escribir  CI +"No permitido Claves Repetidas"
        Fin Si
        Empujar(CI,R^.Ramas[k], EmpujarArriba,Mdna,Xr)
        Si EmpujaArriba  Entonces
            Si R^.Cuenta<Max Entonces
                EmpujaArriba=False
                MeterHoja(Mdna,Xr,R,K)
            Fin Si
        Sino
            EmpujaArriba=True
            DividirNodo(Mdna, Xr, R, K, Mdna, Xr)
        Fin Si
    Fin Si
Fin
```

En el Procedimiento MeterHoja es llamado cuando ya se ha determinado que hay hueco para añadir a la pagina una nueva clave. Sele pasa como argumentos la clave, la dirección de la pagina, la dirección de la rama con el nodo sucesor y la posición a partir de la cual se inserta.

```
MeterHoja (X,Xder,P,K)
I : Posicion
Inicio
    Para I=P^.Cuenta Hasta K+1 Hacer
        P^.Claves[I+1]=P^.Claves[I]
        P^.Ramas[I+1]=P^.Ramas[I]
     Fin Para
     P^.Claves[K+1]=X
     P^.Ramas[K+1]=Xder
     P^.Cuenta= P^.Cuenta+1
Fin
```

El ultimo de los procedimientos el de dividir nodo. El nodo donde hay que insertar la clave X y la rama correspondiente esta lleno. A nivel lógico es dividido el nodo en dos nodos y la clave mediana enviada hacia arriba para una reinserción posterior.

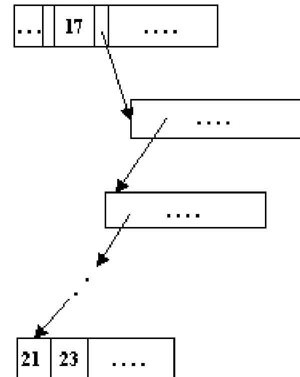
Para ello lo que se hace es crear un nuevo nodo donde se llevan las claves mayores de la clave mediana. Dejando en el nodo original las claves menores. Primero se determina la posición que ocupa la clave mediana, teniendo en cuenta la posición central del nodo y la posición donde debería de insertase la nueva clave. A continuación se divide el nodo, se inserta la nueva clave en el nodo que le corresponde y por ultimo se extrae la clave mediana.

```
DividirNodo(X,Xder,P,K,Mda,Mder)
I, Posmda : Entero
Inicio
    Si K<=Min entonces
        Posmda=Min
    Sino
        Posmda=Min+1
    Fin Si
    Crear(Mder)
    Para I=Posmda+1 hasta Max hacer
        Mder^.Clave[I-Posmda]=P^.Clave[I]
        Mder^.Ramas[I-Posmda]=P^.Ramas[I]
    Fin Para
    Mder^.Cuenta=Max-Posmad
    P^.Cuenta=Posmad
    Si K<=Min Entonces
        MeterHoja(X,Xder,P,K)
    Sino
        MeterHoja(X,Xder,Mder,K-Posmda)
    Fin Si
    Mda=P^.Claves[P^.Cuenta]
    Mder^.Ramas[0]=P^.Ramas[P^.Cuenta]
    P^.Cuenta= P^.Cuenta+1
Fin
```

Borrado

El borrado en un árbol-B también requiere una operación de búsqueda para localizar el nodo exacto. Hay entonces dos posibilidades: la clave que se quiere borrar reside en una hoja, o la clave reside en un nodo interno. Un borrado de un nodo interno requiere que se encuentre una clave adyacente y cambiarla en la posición que se dejó vacante de manera que este dispuesto para trabajar correctamente.

Figura 3.11. El borrado de la clave "17" requiere encontrar la siguiente clave en secuencia, "21", e intercambiarla con la posición vacante.



Localizar una clave adyacente en secuencia de claves solicita, una búsqueda por la hoja más a la izquierda en el subárbol derecho del ahora slot vacío. Como en un árbol binario de el valor requerido siempre reside en una hoja. La Figura 3.11. demuestra estas relaciones.

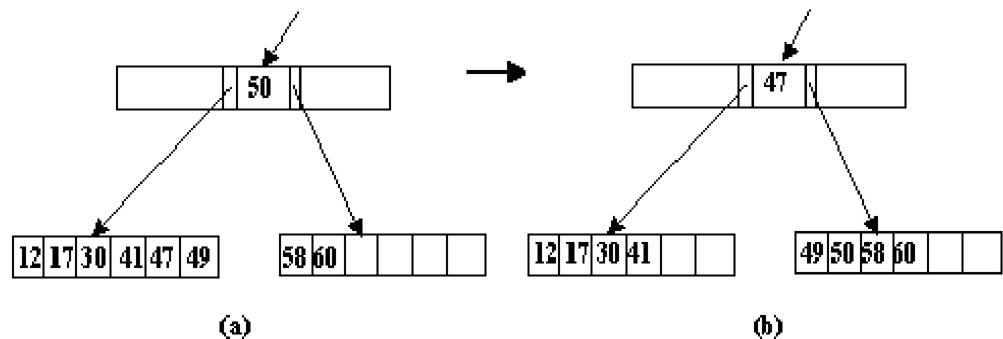


Figura 3.12. (a) Parte de un árbol-B antes, y (b) después de la redistribución de nodos entre dos vecinos. Nótese que la posición final de la clave intermedia, "50". La redistribución en nodos de igual tamaño ayuda a evitar 'underflow' en sucesivos borrados.

Una vez que el slot vacío ha sido "movido" a una hoja, debemos verificar para ver que por lo menos quedan d claves. Si menos de d claves ocupan la hoja, entonces se dice que ocurre un underflow, y una redistribución de las claves se vuelve necesaria. Para restaurar el equilibrio (y la propiedad de los árboles-B de que cada nodo tiene por lo menos d claves) sólo se necesita una clave que se puede

obtener pidiéndola prestada de una hoja vecina. Pero ya que la operación requiere por lo menos dos accesos a almacenamiento secundario, una mejor redistribución dividiría uniformemente las llaves restantes entre los dos nodos vecinos, disminuyendo el coste de borrados sucesivos del mismo nodo. La Redistribución viene ilustrada por la Figura 3.12.

Por supuesto, la distribución de claves entre dos vecinos bastar sólo si hay por lo menos 2d claves para distribuir. Cuando quedan menos de 2d valores, debe ocurrir un encadenamiento. Durante un encadenamiento, se combinan las claves simplemente en uno de los nodos, y se desecha el otro (nota ese encadenamiento es lo inverso de fraccionamiento). Ya que queda un único nodo, la clave que separa los dos nodos en el antecesor ya no es necesaria; también es añadida a la única hoja restante. La Figura 3.13. muestra un ejemplo de encadenamiento y la situación final de la clave separadora.

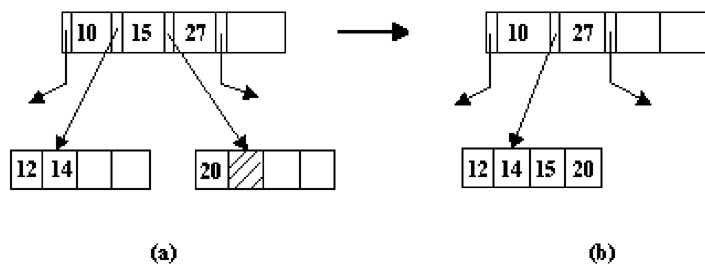
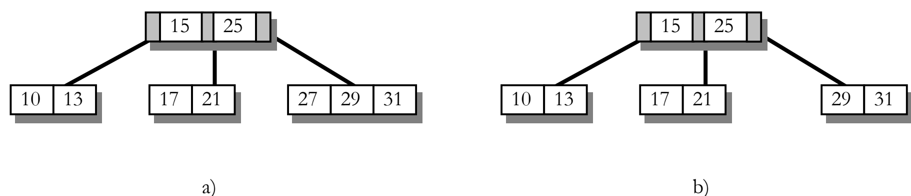


Figura 3.13. (a) Borrado que origina una concatenación, y (b) el árbol re-balanceado.

Cuando algún nodo pierde una clave separadora debido a un encadenamiento de dos de su hijos, también puede causar underflow y requiere redistribución de uno de su vecinos. El proceso de encadenamiento puede forzar encadenamiento en el próximo nivel más alto, y así sucesivamente, hasta el nivel raíz. Finalmente, si se encadenan los descendientes de la raíz, forman una nueva raíz, decrementando la altura del árbol-B en 1. Se encuentran algoritmos de inserción y borrado.

Para probar cada una de las reglas antes mencionadas se presentara un ejemplo que mostrara el proceso de eliminación.



a) b)

Figura 3.14. Eliminación de la clave 27 en un árbol B de orden 2. a) antes de eliminar la clave. b) Después de eliminarla.

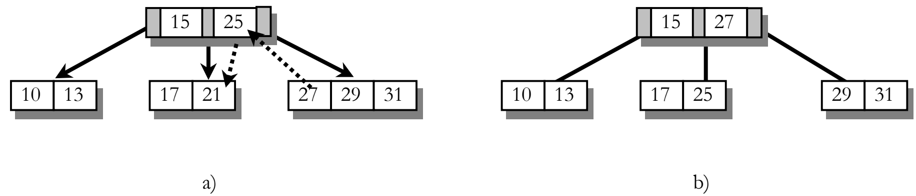


Figura 3.15. Eliminación de la clave 21 en un árbol B de orden 2. a) antes de eliminar la clave. b) Después de eliminarla.

Al eliminar la clave 21 de la pagina A, baja la clave 25 de la pagina antecedente y esta es sustituida por la que se encuentra mas a la izquierda en la pagina derecha, la clave 27 de la pagina B.

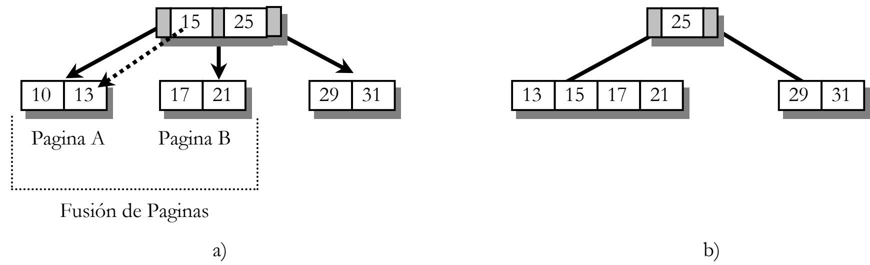


Figura 3.16. Eliminación de la clave 10 en un árbol B de orden 2. a) antes de eliminar la clave. b) Después de eliminarla.

Al eliminar la clave 10 de la pagina A, baja la clave 15 de la pagina antecesora y se fusionan las paginas A y B.

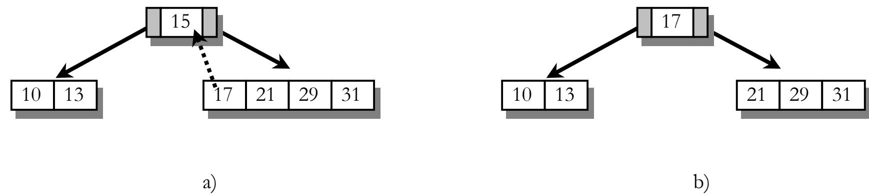


Figura 3.17. Eliminación de la clave 15 en un árbol B de orden 2. a) antes de eliminar la clave. b) Después de eliminarla.

Al eliminar la clave 15 se sustituye por la clave que se encuentra más a la izquierda en el subárbol derecho.

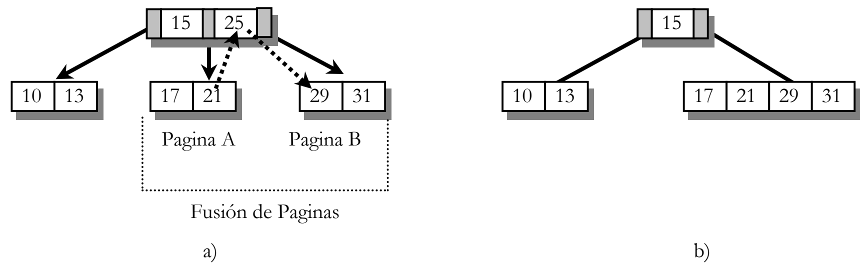


Figura 3.18. Eliminación de la clave 25 en un árbol B de orden 2. a) antes de eliminar la clave. b) Después de eliminarla.

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 pagina A, m queda menor que d por lo que necesita realizarse una fusión. Baja la clave correspondiente ala pagina antecesora(nuevamente 21) y se fusionan las paginas A y B.

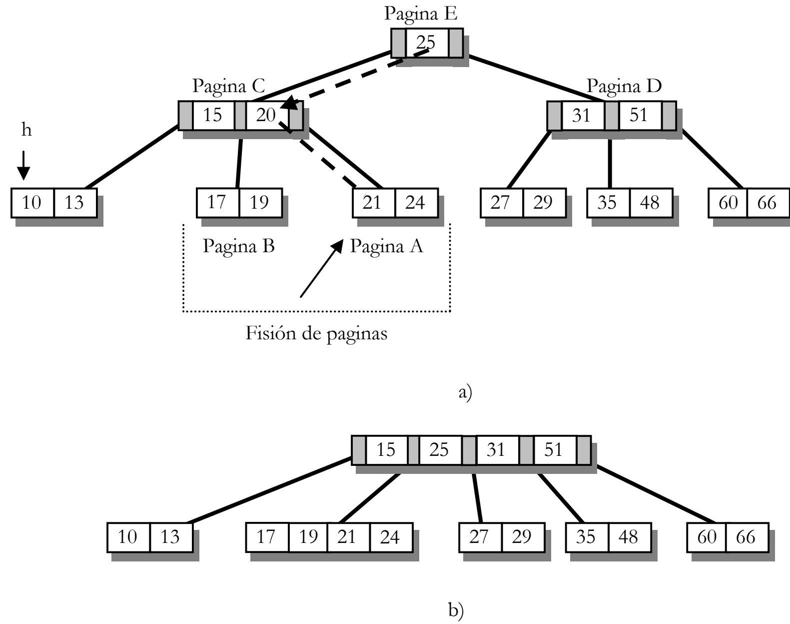
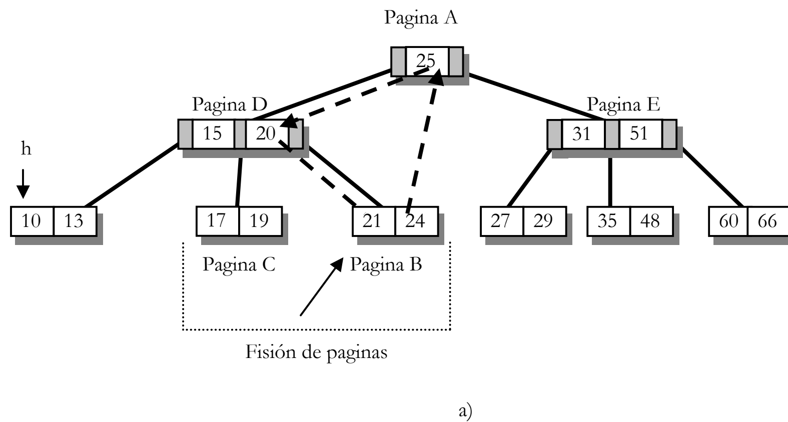


Figura 3.19. Eliminación de la clave 21 en un árbol B de orden 2. a) antes de eliminar la clave. b) Después de eliminarla.

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



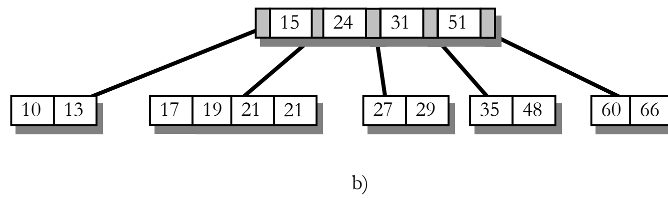


Figura 3.21. Eliminación de la clave 25 en un árbol B de orden 2. a) antes de eliminar la clave. b) Después de eliminarla.

Al eliminar la clave 25 de la pagina A, se sustituye por la clave que se encuentra mas a la derecha en el subárbol izquierdo (24 de la pagina B). Sin embargo, en la pagina B, *m* queda menor a *d*, por lo que es necesario bajar la clave 20 de la pagina D produciéndose la fisión de la pagina B y C. Nuevamente en la pagina D, *m* queda menor a *d*, por lo que ahora es necesario bajar la clave 24 de la pagina A. Como esta pagina queda vacía entonces necesita realizarse una fisión entre las paginas D y E. La altura del árbol disminuirá en una unidad.

El proceso empieza con la llamada al procedimiento eliminar, este pasa control a eliminarRegistro que es el encargado de controlar todo el proceso.

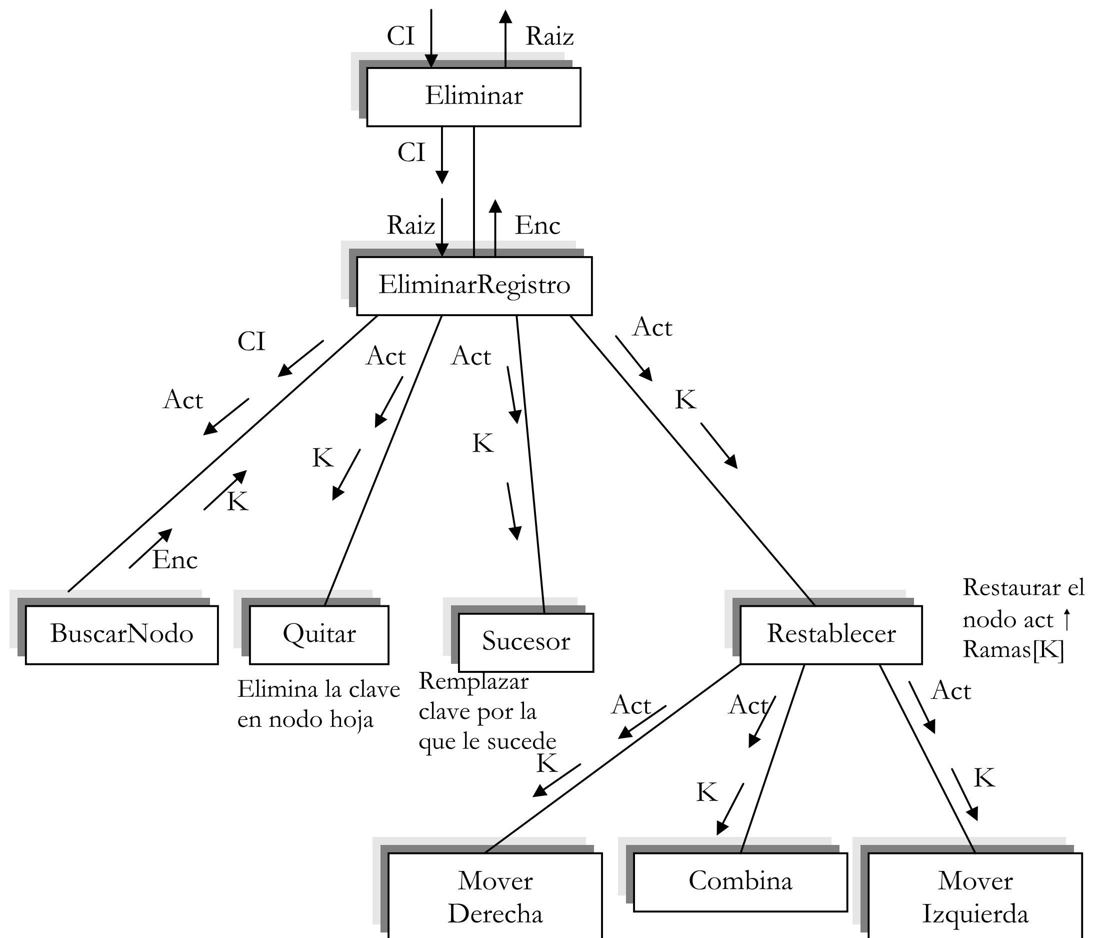


Figura 3.22 Diagrama de bloques del proceso de eliminación

El procedimiento Eliminar tiene la misma estructura que insertar. Tiene como misión principal devolver la nueva raíz si la raíz inicia se ha quedado sin clave.

```
Elimiar (CI,Raiz)
Enc : Booleano
P : Puntero
Inicio
    EliminarRegistro(CI, Raiz, Enc)
    Si Not Enc Entonces
        Escribir "Error : la clave no esta en el arbol"
    Sino
        Si Raiz^.Cuenta =0 Entonces
            P=Raiz
            Raiz=Raiz^.Ramas[0]
            Destruir (P)
        Fin Si
    Fin Si
Fin
```

En el procedimiento EliminarRegistro controla el proceso de borrado de una clave. Primero busca el nodo donde se encuentra la clave a eliminar. Si el nodo es una hoja, llama a la rutina elimina. De no ser hoja es hallado el inmediato sucesor de la clave, se coloca en el nodo donde se encuentra la clave; después de suprimir la clave sucesor en el nodo hoja.

```
ElimiarRegistro (CI,R,Enc)
K : Puntero
Inicio
  Si R=NULL Entonces
    Enc=Falso
  Sino
    BuscarNodo(CI,R,Enc,K)
    Si Enc Entonces
      Si R^.Ramas[K-1]=NULL Entonces
        Quitar (R,K)
      Sino
        Sucesor(R,K)
        EliminaRegistro(R^.Claves[K],R^.Ramas[K],Enc)
        Si Not Enc Entonces
          Escribir "Error en el Proceso"
        Fin si
    Sino
      EliminaRegistro(CI, R^.Ramas[K], Enc)
      Si R^.Ramas[K]<>NULL Entonces
        Si R^.Ramas [K]^.Cuenta<Min Entonces
          Resableces(R,K)
        Fin Si
      Fin Si
    Fin Si
Fin
```

La forma mas fácil de expresar este proceso es la recursividad; al retornar la llamada recursiva debe de comprobar si hay numero de claves suficientes, mayor o igual que el mínimo necesario (2 en caso de árbol B de orden 5), de no ser así hay que mover claves al nodo para alcanzar el numero necesario de claves.

El procedimiento Quitar es una rutina que recibe la dirección del nodo y la posición de la clave a eliminar. Elimina la clave junto a la rama que le corresponde.

```
Quitar(P,K)
J : Entero
Inicio
   Para j=k+1 Hasta P^.Cuenta Hacer
      P^.Clave[J-1] = P^.Claves[J]
      P^.Ramas[J-1]=P^.Ramas[J]
   Fin Para
   P^.Cuenta=P^.Cuenta-1
Fin
```

El procedimiento Sucesor e una rutina que busca la clave inmediatamente sucesor de la clave *K*, que por la propiedad de los árboles B esta en una hoja y esta reemplaza a la clave *K*.

```
Sucesor(P,K)
Q : Puntero
Inicio
   Q = P^.Ramas[K]
   Mientras Q^.Ramas[0]<>NULL Hacer
      Q=Q^.Ramas[0]
   Fin Mientras
   P^.Claves[K]=Q^.Claves[1]
Fin
```

En el procedimiento restablecer se realizan las acciones mas complejas del proceso. Restaura el nodo P^.Ramas[K] el cual se ha quedado con un numero de claves menor que el mínimo.

```
Restablecer(P,K)
Inicio
  Si K >0 Entonces
    Si P^.Ramas[K-1]^.Cuenta>Min Entonces
      MoverDer (P,K)
    Sino
      Combina(P,K)
    Fin Si
  Sino
    Si P^.Ramas[1]^.Cuenta>Min Entonces
      MoverIzq (P,K)
    Sino
      Combina(P,1)
    Fin Si
  Fin Si
Fin
```

El procedimiento siempre toma una clave a partir del hermano de la izquierda (clave menores), solo utiliza el hermano derecho cuando no hay hermano izquierdo. Puede ocurrir que no pueda tomas claves y entonces llamar a combina para formar un nodo con dos nodos.

El procedimiento MoverDer mueve una clave de nodo antecedente (P) al nodo que se esta restaurando. A su vez, asciende la clave mayor del hermano izquierdo al nodo antecedente.

```
MoverDer(P,K)
J : Entero
Inicio
  Para J=Cuenta Hasta 1 Hacer
    P^.Ramas[K]^.Claves[j+1] = P^.Ramas[K]^.Claves[J]
    P^.Ramas[K]^.Ramas[j+1] = P^.Ramas[K]^.Ramas[J]
  Fin Para
  P^.Ramas[K]^.Cuenta= P^.Ramas[K]^.Cuenta+1
  P^.Ramas[K]^.Ramas[1]= P^.Ramas[K]^.Ramas[0]
  P^.Ramas[K]^.Claves[1]=P^.Claves[K]
  P^.Claves[K] = P^.Ramas[K-1]^.Claves[Cuenta];
  P^.Ramas[K]^.Ramas[0] = P^.Ramas[K-1]^.Ramas[Cuenta]
  P^.Ramas[K-1]^.Cuenta = P^.Ramas[K-1]^.Cuenta - 1
Fin
```

El Procedimiento MoverIzq Realiza la misma acción que el procedimiento MoverDer, salvo que ahora la clave que asciende al nodo antecedente es la clave menor (izquierda) del nodo restaurar.

```
MoverIzq(P,K)
J : Entero
Inicio
  P^.Ramas[K-1]^.Cuenta = P^.Ramas[K-1]^.Cuenta + 1
  P^.Ramas[K-1]^.Claves[Cuenta] = P^.Claves[K]
  P^.Ramas[K-1]^.Ramas[Cuenta] = P^.Ramas[K]^.Ramas[0]
  P^.Claves[K] = P^.Ramas[K]^.Claves[K]
  P^.Ramas[K]^.Ramas[0]= P^.Ramas[K]^.Ramas[1]
  P^.Ramas[K]^.Cuenta= P^.Ramas[K]^.Cuenta - 1
  Para J=1 Hasta P^.Ramas[K]^Cuenta Hacer
    P^.Ramas[K]^.Clave[J] = P^.Ramas[K]^.Clave[J+1]
    P^.Ramas[K]^.Ramas[J] = P^.Ramas[K]^.Ramas[J+1]
  Fin Para
Fin
```

El ultimo Procedimiento es Combina, esta encargado de formar un solo nodo con dos nodos. Combina el nodo que esta en le rama K con el nodo que se encuentra en el nodo antecedente. Una vez finalizado el proceso, libera el nodo obsoleto.

```
Combina(P,K)
J : Entero
Q :Puntero
Inicio
   Q = P^.Ramas[K]
   P^.Ramas[K-1]^.Cuenta = P^.Ramas[K-1]^.Cuenta + 1
   P^.Ramas[K-1]^.Claves[Cuenta] = P^.Claves[K]
   P^.Ramas[K-1]^.Ramas[Cuenta] = Q^.Ramas[0]
   Para J=1 Hasta Q^.Ramas[0] Hacer
      P^.Ramas[K-1]^.Cuenta = P^.Ramas[K-1]^.Cuenta + 1
      P^.Ramas[K-1]^.Claves[Cuenta] = Q^.Claves[J]
      P^.Ramas[K-1]^.Ramas[Cuenta] = Q^.Ramas[J]
   Fin Para
   Para J=K Hasta P^.Cuenta – 1 Hacer
      P^.Claves[J] = P^.Claves[J+1]
      P^.Ramas[J] = P^.Ramas[J+1]
   Fin Para
Fin
```

Árboles-B+

En un árbol-B+, todas las claves están en las hojas. Los niveles más altos, que están organizados como un árbol-B, constan sólo de un índice, un mapa para permitir una rápida localización de las bifurcaciones de índices y claves. La Figura 13 muestra la separación lógica de las bifurcaciones de índices y claves. Naturalmente los nodos índices y nodos hoja pueden tener diferentes formatos o incluso diferentes tamaños. En particular, los nodos hoja son unidos juntos de izquierda a derecha como se muestra. La lista de uniones de hojas esta relacionada como un conjunto sucesión. Las uniones del conjunto sucesión permiten un procesamiento secuencial fácil.

Para apreciar totalmente un árbol-B+, uno debe entender las implicaciones de tener un índice independiente y un conjunto sucesión. Considere por un momento la operación encontrar. Buscando proceder desde la raíz de un árbol-B+ a través del índice a una hoja. Desde todas las claves incluidas en las hojas, no es un problema qué valores se encuentren conforme avanza la búsqueda con tal que se siga la trayectoria correcta a la hoja.

Durante el borrado en un árbol-B+, la habilidad para dejar los valores sin clave en la parte de índices como separadores simplifica el proceso. La clave a eliminar debe estar siempre en una hoja de tal modo que su eliminación sea simple. Mientras la hoja se quede al menos medio llena, el índice no necesita cambiarse, incluso si una copia de la clave ha sido propagada en él. La Figura 14 muestra como la copia de una clave borrada puede todavía dirigir búsquedas a la hoja correcta. Por supuesto, si surge una condición de no desequilibrio, la redistribución o procedimientos de concatenación pueden requerir valores ajustados en el índice y también en las hojas.

Las operaciones de inserción y búsqueda en un árbol-B+ son procesadas casi idénticamente a las operaciones de inserción y de búsqueda en un árbol-B.

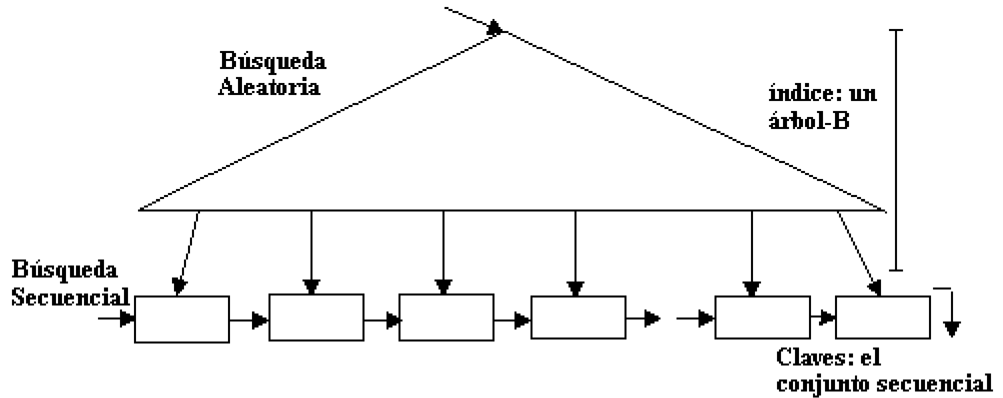


Figura 13. Un árbol-B+ con las partes índice y claves separadas.

Las operaciones 'por clave empiezan en la raíz como en un árbol-B; el proceso secuencial empieza en la hoja más a la izquierda.

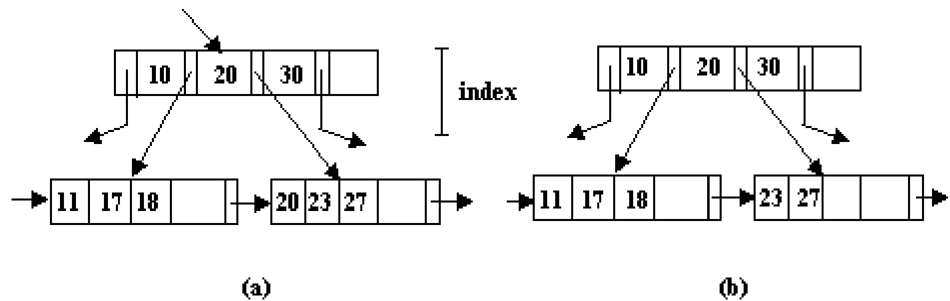


Figura 14. (a) Un árbol-B+ y (b) un árbol-B+ después del borrado de la clave "20". Incluso después del borrado, la clave "20" vale todavía como valor separador en la parte índice.

Cuando una hoja se divide en dos en vez de ascender la clave intermedia, el algoritmo asciende una copia de la clave, reteniendo la clave real en la hoja derecha. Las operaciones de búsqueda difieren de las que se realizan en los árboles-B en que la búsqueda no para si una clave en el índice es igual al valor por el que se pregunta. En cambio, se sigue el puntero situado más a la derecha, y se procede de esta forma todo el camino hasta la hoja. Hemos visto que un árbol-B que soporta un bajo coste en operaciones de búsqueda, inserción y borrado puede necesitar log n accesos a almacenamiento secundario para proceder a la siguiente operación. La implementación de árboles-B+ contiene las propiedades de coste algorítmico para operaciones por clave, pero tiene la ventaja de requerir como máximo un acceso para satisfacer la siguiente operación. Por otra parte, durante el procesamiento secuencial de un fichero, ningún nodo ser accedido más de una vez, así que se necesita disponer de espacio para un sólo nodo en memoria principal. Así, los árboles-B+ son adecuados para aplicaciones que supongan procesamiento aleatorio y secuencial.

B+ Árboles Prefijo.

La separación de los índices y conjunto de secuencias en los árboles-B+ es intuitivamente atrayente. Tener en cuenta que parte de los índices sirven simplemente como un mapa de caminos para guiar la búsqueda de la hoja correcta; no necesita contener todas las claves actuales. Cuando las claves consisten en una tira de caracteres hay buena razón para no usar las claves actuales como separadores: las claves actuales requieren demasiado espacio. Bayer y Unterauer consideran como alternativa el B+ árbol prefijo.

Supongamos que la secuencia alfabética de claves, 'binary', 'computer', 'electronic', 'program' y 'system' están asignadas o localizadas en un árbol-B como el mostrado en la Figura 15. El separador valora o señala entre las claves 'computer', 'electronic', las cuales no necesitan estar íntegramente: basta con alguna cadena de caracteres. Por ejemplo, algunos de estas tiras, 'elec', 'e' ó 'd' servirían perfectamente. Ya que no se hace diferencia durante el proceso de recuperar, el más corto separador debería ser usado para así guardar espacio. Como los requerimientos de espacio pueden llegar a ser muy pequeños, muchas claves pueden estar situadas en cada nodo, aumentando así los elementos y disminuyendo la altura del árbol. Ya que los árboles son más cortos tiene un coste menor la búsqueda, usando separadores cortos disminuirá el tiempo de acceso así como guardamos espacio.

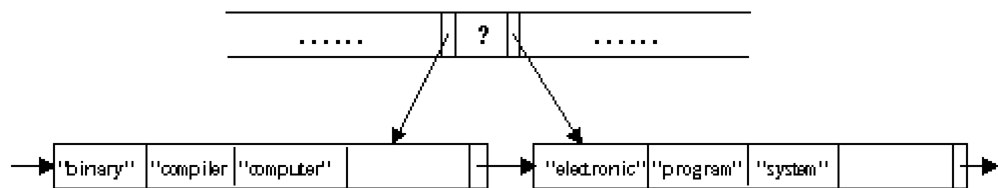


Figura 15. Parte de un árbol-B Prefijo. La entrada índice "e" es suficiente para diferenciar "computer" de "electronic"

La técnica más simple para elegir el único prefijo más corto de la clave para servir como separador da resultado. En el ejemplo, el prefijo más corto de 'electronic' es suficiente para distinguirlo de 'computer'. Algunas veces, sin embargo, la técnica del prefijo no funciona bien: eligiendo el prefijo más corto de 'programers' para distinguirlo de 'programer' no resulta del todo correcto. En cada uno de los casos, Bayer y Unterauer sugieren explora unas pequeñas claves cercanas para obtener una buena pareja para el algoritmo de separación. Mientras esto pueda dejar los nodos desigualmente cargados, teniendo no mucha cantidad de claves extra en uno de los nodos, no afectará al coste global.

Ejercicios

3.1. Dada la secuencia de claves enteras:190,57,89,90,121,170,35,48, 91,22,126,132 y 80;dibuje el árbol B de orden 5 cuya raíz es R, que se corresponde con dichas claves.

3.2. En el árbol R del problema anterior, elimine la clave 91 y dibuje el árbol resultante. Elimine ahora la clave 48.Dibuje el árbol resultante,¿ha habido reducción en el número de nodos?

3.3. Dada la siguiente secuencia de claves:7,25,27,15,23,19,14,29,10, 50,18,22,46,17,70,33 y 58;dibuje el árbol B+ de orden 5 cuya raíz es R, que se corresponde con dichas claves.

3.4. Construir cada uno de los B-árboles que se van generando conforme se van insertando los números 1,9,32,3,53,43,44,57,67,7,45,34,23,12,23,56,73,65,49,85,89, 64,54,75,77,49, en un B-árbol de orden 5.

3.5. Supongamos que se insertan un conjunto de elementos en un B-árbol en un determinado orden.¿La altura del B-árbol resultado es independiente del orden en que se han insertado los elementos?.Razónese la respuesta.

Problemas

3.6 Muestre como utilizar un árbol B y B+ para implementar una cola de prioridad. Muestre que cualquier de n operaciones de inserción y un mínimo de operaciones de supresión puede ejecutarse en O(n Log n)pasos. Escribir un algoritmo para la inserción y supresión de una cola.

3.7 Seleccione cualquier párrafo grande de un libro. Inserte cada palabra del párrafo en orden en el árbol de búsqueda multicaminos de arriba hacia abajo inicialmente básico de orden 5 omitiendo cualquier palabra repetida. Haga lo mismo para un árbol B de orden 5 y un árbol B+ de orden 5.

3.8 Escribir un algoritmo y una rutina de cualquier lenguaje para suprimir un registro de un árbol de búsqueda multidireccional (multicaminos)de arriba hacia debajo de orden n.

3.9 Escribir un algoritmo y una rutina de cualquier lenguaje para suprimir un registro de un árbol B de orden n.

3.10 Escribir un algoritmo para crear un árbol B compacto a partir de ingresos en orden. Use el algoritmo para escribir una rutina con el propósito de producir un árbol B compacto a partir de un árbol B ordinario.

3.11 Escribir un algoritmo y rutina en algún lenguaje para búsqueda de un árbol B.

3.12 Escribir un algoritmo y rutina en un lenguaje:

a. Inserción de un árbol B+
b. Inserción de un árbol B*
c. Eliminación de un árbol B+
d. Eliminación de un árbol B*

Capítulo

4

Grafos

Con los árboles se han representado relaciones entre objetos en las que existen una jerarquía. Con frecuencia, es necesario representar relaciones arbitrarias entre objetos de datos. Los grafos se clasifican en dirigidos y no dirigidos y son modelos naturales de tales relaciones. Así, los grafos se usan para representar redes de alcantarillado, redes de comunicación, circuitos eléctricos, etc. Una vez modelado el problema mediante un grafo se pueden hacer estudios sobre diversas propiedades. Para ello se utilizan algoritmos concretos que resuelven ciertos problemas.

La teoría de grafos a sido aplicada en le estudio de problemas que surgen en áreas diversas de las ciencia, como la química, la ingeniería eléctrica o la investigación operativa. El primer paso siempre será representar el problema con un grafo. En esta representación cada elemento, cada objeto del problema, forma un nodo. La relación, comunicación o conexión entre los objetos da lugar a una arista, que puede ser dirigida o bidireccional (no dirigida)

Definición y Terminología

Un grafo, G, es un par, compuesto por dos conjuntos V y A. Al conjunto V se le llama conjunto de vértices o nodos del grafo. A es un conjunto de pares de vértices, estos pares se conocen habitualmente con el nombre de arcos o ejes del grafo. Se suele utilizar la notación G = (V, A) para identificar un grafo. Los grafos representan un conjunto de objetos donde no hay restricción a la relación entre ellos. Son estructuras más generales y menos restrictivas.

Se representa el conjunto de vértices de un grafo dado G por V_G y el conjunto de arcos por A_G. Por ejemplo en el grafo de la figura 3.1

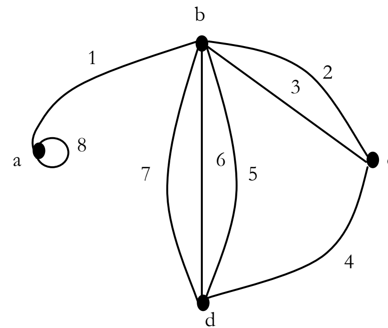


Figura 4.1. Grafo

Una arista se representa por los vértices que conecta. La arista 3 conecta a los vértices b y c, y se representa A(b,c). Algunos vértices pueden conectar un nodo consigo mismo por ejemplo el vértice 8 tiene le formato V(a,a). Esta arista se denomina bucle o lazos.

Un grafo se denomina **sencillo** si cumple las siguientes condiciones:

- No tiene lazos, no existen un arco A de la forma (V,V) donde esta en V.
- No existe mas de un arco para unir dos nodos, es decir no existe mas de un arco (V_1,V_2) para cualquier par de vértices V_1 y V_2.

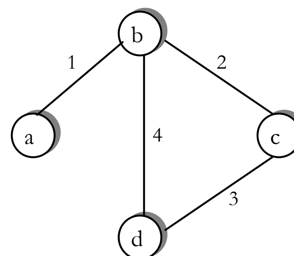


Figura 4.2 Grafo sencillo

un grafo no sencillo se denomina grafo múltiple. Como se ve en la figura 3.3 al principio ya existía la arista 2 que comunicaba a los nodos b y c, pero al existir la nueva arista 5 dejo de ser grafo simple para convertirse en múltiple.

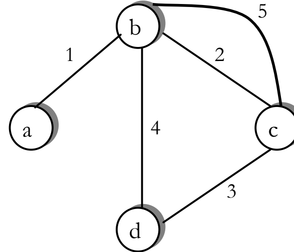


Figura 4.3 Grafo Múltiple.

Un **camino** es una secuencia de uno o mas arcos que conectan dos nodos. Representaremos por $C(V_1,V_2)$ un camino que conecta los vértices V_1 y V_2.

La **longitud** de un camino es el numero de arcos comprende. En el grafo de la figura 3.2 existen los siguientes caminos entre los nodos b y d.

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

En algunos grafos se dan arcos desde un vértice a si mismo(V,V), el camino $v - v$ se denomina bucles. No es frecuente encontrarse con grafos que tengan bucles. Un camino es simple si todos los nodos que forman el camino son distintos, pudiendo ser iguales v_0, v_n (los extremos del camino). En la figura siguiente se muestra por medio de las flechas un camino el cual seria C(4,7)=(4,8)(8,9)(9,7), donde en muchos libros se maneja P =(4,8,9,7).

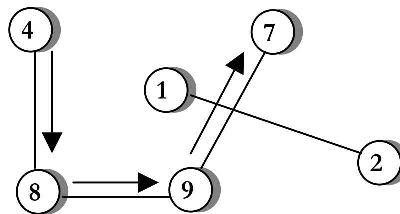


Figura 4.4 . Camino simple de un Grafo no dirigido

Un ciclo es un camino simple cerrado, $v_0 = v_n$ de longitud ≥ 2, en definitiva compuesto al menos por 3 nodos.

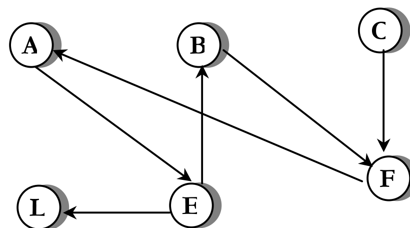


Figura 4.5 . Ciclo de Camino simple de un Grafo

Los vértices (A, E)(E, B)(B, F)(F, A) en este grafo dirigido forman un ciclo de longitud 4. un ciclo de longitud *k* se denomina *k-ciclo.*

Dos vértices se dicen **adyacentes** si hay un arco que los une. Así V_i y V_j son adyacentes si existen en un camino si los une. Esta definición es muy general y normalmente se particulariza; si existe un camino desde A a B, decimos que A es adyacente a B y B es adyacente desde A.

Podemos clasificar los grafos en dos grupos: dirigidos y no dirigidos. En un grafo no dirigido el par de vértices que representa un arco no está ordenado(u—v). Por lo tanto, los pares (v1, v2) y (v2, v1) representan el mismo arco. En un grafo dirigido(digrafo) cada arco está representado por un par ordenado de vértices(u→v) , de forma que y representan dos arcos diferentes. ejemplos de grafos (dirigidos y no dirigidos):

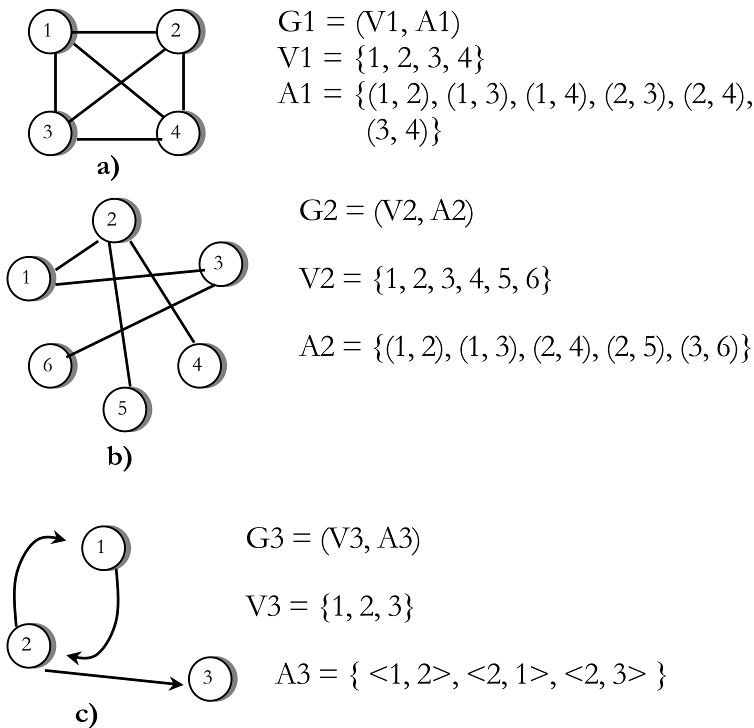


Figura 4.6. Tipos de grafos a) Grafo no dirigido; b) Grafo no dirigido; c) Grafo no dirigido.

Un grafo no dirigido es conexo si existe un camino entre cualquier par de nodos que forman el grafo. En el caso de ser un grafo dirigido podemos distinguir entre fuertemente conexo y conexo. Un grafo dirigido fuertemente conexo si existe un camino entre cualquier par de nodos que forman el grafo; un grafo dirigido es conexo si existe una cadena que une par de vértices. Un grafo completo es aquel que tiene un arco para cualquier par de vértices.

Los grafos son modelos matemáticos de numerosas situaciones reales: un mapa de carreteras, la red de ferrocarriles, el plano de un circuito eléctrico, el esquema de la red telefónica de una compañía, etc.

El número de distintos pares de vértices (v(i), v(j)), con v(i) <> v(j), en un grafo con n vértices es n*(n-1)/2. Este es el número máximo de arcos en un grafo no dirigido de n vértices. Un grafo no dirigido que tenga exactamente n*(n-1)/2 arcos se dice que es un grafo completo. En el caso de un grafo dirigido de n vértices el número máximo de arcos es n*(n-1).

Algunas definiciones básicas en grafos:

Orden de un grafo es el número de nodos (vértices) del grafo.

Grado es una cualidad que se refiere a los nodos de un grafo. En un grafo no dirigido el grado de un nodo *v* , grado(*v*), es el numero de aristas que contiene a *v* . En un grafo dirigido se distinguen entre grafos de entrada y grado de salida; grado de entrada de un nodo *v,* gradent(*v*) el numero de arcos que llega a *v* . grado de salida de *v*, gradSal(*v*) , es el numero de arcos que salen de *v* . A veces no se sabe distinguir entre arco y arista, la diferencia esta en que arista son los arcos hacia los dos sentidos. Por ejemplo en la figura 3.4b el grado(2)= 3 y en la figura 3.4c el gradent(2)=1 y el gradsal(2) = 1.

Grafo simétrico: es un grafo dirigido tal que si existe la relación entonces existe , con u, v pertenecientes a V.

Grafo no simétrico: es un grafo que no cumple la propiedad anterior.

Grafo reflexivo: es el grafo que cumple que para todo nodo u de V existe la relación (u, u) de A.

Grafo transitivo: es aquél que cumple que si existen las relaciones (u, v) y (v, z) de A entonces existe (u, z) de A.

Grafo completo: es el grafo que contiene todos los posibles pares de relaciones, es decir, para cualquier par de nodos u, v de V, (u <> v), existe (u, v) de A.

Camino euleriano: camino simple que contiene todos los arcos del grafo.

Grafo euleriano: es un grafo que tiene un camino euleriano cerrado.

Grafo conexo: es un grafo no dirigido tal que para cualquier par de nodos existe al menos un camino que los une.

Grafo fuertemente conexo: es un grafo dirigido tal que para cualquier par de nodos existe un camino que los une.

Punto de articulación: es un nodo que si desaparece provoca que se cree un grafo no conexo.

Componente conexa: subgrafo conexo maximal de un grafo no dirigido (parte más grande de un grafo que sea conexa).

Representación de un grafo

Al pensar en los tipos de datos para representar un grafo en memoria, se debe de tener en cuenta que se debe representar un numero (infinito)de vértices y de arcos que unen dos vértices. Se puede elegir una representación secuencial, mediante arreglo; o bien una representación dinámica, mediante una estructura multienlazada. La representación mediante arreglos se conoce como matriz de adyacencia, la representación dinámica denomina lista de adyacencia. La elección de una manera u otra dependerá de las operaciones que se apliquen sobre los vértices y arcos.

Matriz de Adyacencia

La matriz de adyacencia M es un arreglo de dos dimensiones que representa las conexiones entre pares de verticales. Sea un grafo G con un conjunto de nodos V_Gy un conjunto de aristas A_G. Supongamos que el grafo es de orden N, donde N>=1. la matriz de adyacencia M se representa por un arreglo N x N. Donde :

$$M(i, j) = \begin{cases} 1 \text{ si existe una arista } (V_i, V_j) \text{en } A_G, V_i \text{ es adyacente a } V_j \\ 0 \text{ en caso contrario} \end{cases}$$

la columna y la fila de la matriz representa los vértices del grafo. Si existe una arista desde i a j (esto es, el vértice i es adyacente a j), y el coste o peso de la arista de i a j se introduce si no existe la arista, se introduce un 0; lógicamente , los elementos de la diagonal principal son todos ceros, ya que el coste de la arista i a i es 0. utilizando la figura

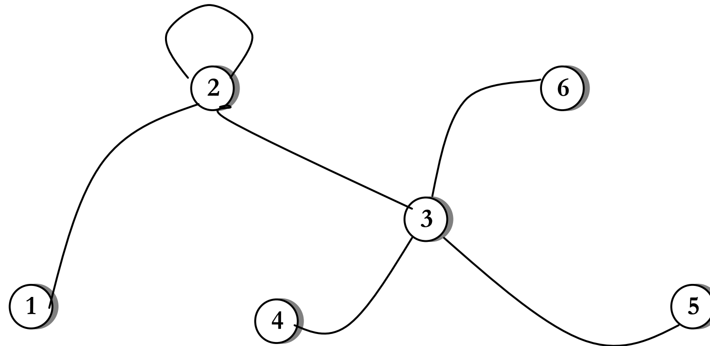


Figura 4.7 grafo no dirigido

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

Figura 4.8 Matriz de adyacencia

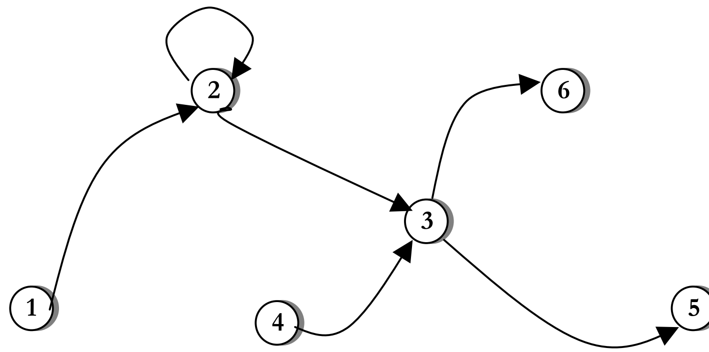


Figura 4.9 grafo no dirigido

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

Figura 4.10 Matriz de adyacencia

Podemos observar que es una matriz simétrica . En los grafos no dirigidos la matriz de adyacencia siempre será simétrica, ya que cada arco no dirigido (v_i, v_j) se corresponde con los arcos dirigidos (v_i, v_j), (v_j, v_i).

Los grafos con factor de peso, grafos valorados pueden representarse de tal forma que si existe arco, el elemento a_{ij} es el factor de peso; la no existencia de arcos supone que a_{ij} es 0 o infinito (esto solo si el factor de peso no puede ser cero). A esta matriz se le denomina matriz valorada.

El grafo valorado de la figura es un grafo dirigido con valor e peso.

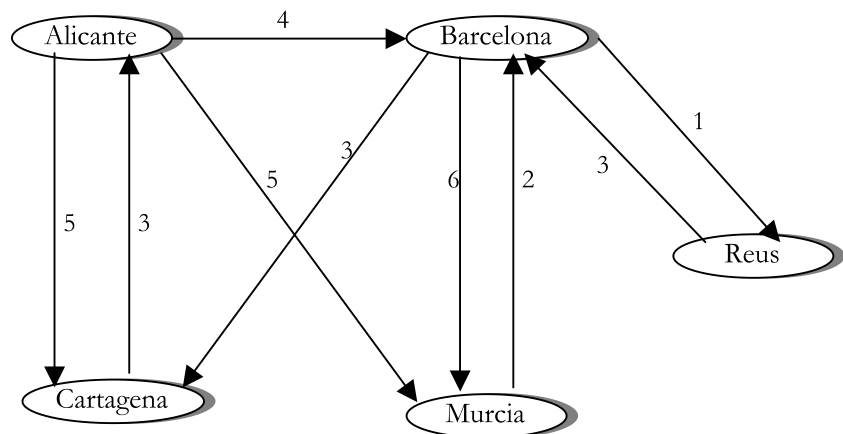


Figura 4.11 Grafo valorado

i \ j					
A	0	4	5	5	0
B	0	0	3	6	3
C	3	0	0	0	0
M	0	2	0	0	0
R	0	1	0	0	0

Figura 4.12 Matriz de adyacencia

Lista de Adyacencia

La representación de un grafo con matriz de adyacencia puede resultar poco eficiente en algunos casos. Así, cuando e numero de vértices varia a lo largo el proceso, dándose el caso de que el numero de ellos sea mayor que el previsto. También cuando el grafo es disperso, es decir tiene pocos arcos, y por tanto la matriz de adyacencia tiene muchos ceros (matriz sparce), resulta poco eficiente la matriz de adyacencia ya que el espacio que ocupa es el mismo que si el grafo tuviera muchos arcos. Cuando se da esta ineficiencias se representa un grafo mediante listas enlazadas que se denominan listas de adyacencia.

La lista de adyacencia son una estructura multienlazada formada por una lista de directorios cuyos nodos representan los vértices del grafo y del que además emergen una lista enlazada cuyos nodos, a su vez, representan los arcos con vértice origen el del nodo de la lista directorio. Una lista de adyacencias del grafo de la figura 3. se da en la siguiente figura

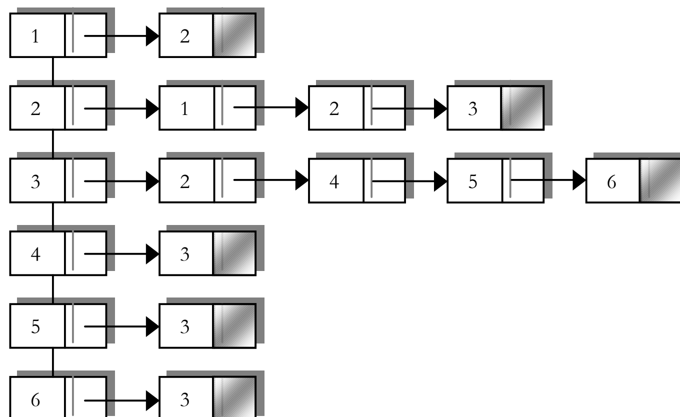


Figura 4.13 lista de adyacencia

Un grafo no dirigido de orden N con A arcos requiere N entradas en el directorio y 2 * A entradas de listas enlazadas, excepto si existen bucles que reducen el numero de listas enlazadas en 1.

Un grafo dirigido de orden N con A arcos requiere N entradas en el directorio y A entradas de listas enlazadas.

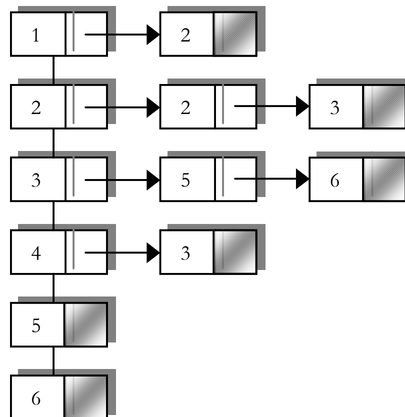
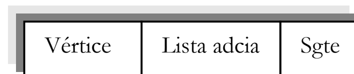


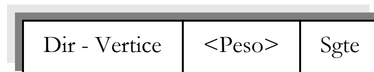
Figura 4.14 lista de adyacencia

Representación

Cada nodo en la lista directorio tiene que guardar el vértice que representa, la dirección de acceso a la lista de adyacencia (arco que sale he dicho vértice) y además la dirección del nodo siguiente de la lista directorio Gráficamente:



Cada nodo de la lista de adyacencia de un vértice del grafo almacena la dirección del vértice (en la lista directorio)con el que forma el arco, en caso de ser un grafo valorado también el factor de peso y como en todas las listas la dirección del nodo siguiente. Gráficamente :



En pseucódigo seria:

```
Nododir = record
    Vert : cadena
    Lady : Nodoly
    Sgte  : Nododir
Fin record
```

```
Nodoly = record
     PtrV : Nododir
     Peso : Real
     Sgte  : Nodoly
Fin record
```

Operaciones de Grafo

Los grafos a l igual que las pilas, colas ... son tipos abstractos de datos. Ya hemos establecidos los datos y su representación, ahora definimos las operaciones básicas sobre estos datos y su realización que será dependiente de las representación de los datos.

La operación unión (X,Y) añade un arco(X,Y) al grafo. En caso de ser un grafo valorado se define la operación union_peso(X,Y,W) que añade el arco (X,Y) cuyo factor de peso es W.

Operación Borr_arco(X,Y) elimina del grafo el arco(X,Y).

Operación Adyacente(X,Y) funcion logica que debe devolver true si forman un arcolos vértices (X,Y).

Operación Nuevo_vértice(X) añade el vértice X al grfo G.

Operación Borra_vértice(X) elimina del grafo G el vértice X.

Las operaciones básicas con lista se implementan con variables punteros y variables dinámicas. Así la operación auxiliar que crea el vértice es :

```
CrearVt(Info)
V : Nododir
Inicio
   Crea(V)
   V^.Vert=info
   V^.Lady=NULL
   V^.Sgte=NULL
   Regresa (V)
Fin
```

En la operación Unión dado el grafo y dos vértices que forman un arco se añade el arco en la correspondiente lista de adyacencia. La relación de Union_peso solo presenta la diferencia de asignar el factor de peso al crear el nodo de la lista adyacente.

```
Union(G,V1,V2)
P,Q: Nododir
Inicio
   P= dirección (G,V1)
   Q= dirección (G,V2)
   Si P<>NULL  y  Q<>NULL Entonces
      Si P^.Lady =NULL Entonces
         P^.Lady=Creanodoady(Q)
      Sino
         Ultimo(P^.Lady)^.Sgte= Creanodoady(Q)
      Fin Si
   Fin Si
Fin
```

Surgen operaciones auxiliares como Dirección y ultimo. La primera operación devuelve un puntero al nodo de la lista directorio donde se encuentra una vértice; la segunda operación devuelve un puntero al ultimo nodo de la lista. Los códigos de ambas funciones son:

```
Directorio(G,V)
P,D :Nododir
Inicio
   P=NULL
   D=G
   Mientras P=NULL y D<>NULL Entonces
      Si D^.Vert =V Entonces
         P=D
      Sino
         D=D^.Sgte
      Fin Si
      Regresa(P)
Fin
```

```
Ultimo(L)
Inicio
   Si L<>NULL Entonces
      Mientras L^.Sgte=NULL Entonces
          L=L^.Sgte;
      Fin Mientras
   Fin Si
    Regresa(L)
Fin
```

La operación elimina arco una vez encontrada la lista de adyacencia se precede a eliminar el nodo que representa el arco.

```
Borr_arco(G,V1,V2)
P,Q : Nododir
R,W : Nodoly
Sw : Booleano
Inicio
  P = Dirección(G,V1)
  P = Dirección(G,V2)
  Si P<>NULL Y Q<>NULL Entonces
    R=P^.Lady
    W=NULL
    Sw=False
    Mientras R<> NULL Y Not Sw Hacer
      Si R^.PtrV=Q Entonces
        Si W =NULL Entonces
          P^.Lady=R^.Sgte
        Sino
          W^.Sgte = S^.Sgte
        Fin Si
        Destruir(R)
        Sw = Verdadero
      Sino
        W=R
        R=R^.Sgte
      Fin Si
    Fin Mientras
  Fin Si
Fin
```

```
Adyacente(G,V1,V2)
P,Q : Nododir
R,W : Nodoly
Sw : Booleano
Inicio
  P = Dirección(G,V1)
  P = Dirección(G,V2)
  Si P<>NULL Y Q<>NULL Entonces
    R=P^.Lady
    Sw=False
    Mientras R=NULL Y Not Sw Entonces
      Si R^.PtrV=Q Entonces
          Sw=Verdadero
      Sino
          Sw=Falso
          R=R^.Sgte
      Fin Si
    Fin Mientras
    Regresar(Sw)
  Sino
    Regresar(Falso)
  Fin Si
Fin
```

La operación nuevo vértice(X) añade un nuevo nodo al grafo a la lista directorio. Siempre se añade como ultimo nodo.

```
Nuevo_vértice (G,V1,V2)
Inicio
  Si G<>NULL Entonces
    Ultimo(G)^.Sgte.=CrearVt(X)
  Sino
    G = CrearVt(X)
  Fin Si
Fin
```

La operación elimina vértice del grafo supone elimina los arcos que va a el y después suprimirlo de la lista directorio. Se utiliza operaciones auxiliares : anterior(G,P) devuelve un puntero a la dirección del nodo anterior u libera que libera memoria de cada nodo de la lista de adyacencia.

```
Borra_Vertice (G, Infor)
P,Q : Nododir
R,W : Nodoly
Sw : Booleano
Inicio
  P = Dirección(G,Infor)
  Si P<>NULL Entonces
    Q=G
    Mientras Q<>NULL Hacer
      Borr_Arco(G,Q^.Vert,X)
      Q=Q^.Sgte
    Fin Mientras
    Si G=P Entonces
      G=G^.Sgte
    Sino
      Anterior(G,P)^.Sgte= P^.Sgte
    Fin Si
    Liberar(P^.Lady)
    Destruir(P)
  Fin Si
Fin
```

Exploración de grafos

A la hora de explorar un grafo, nos encontramos con dos métodos distintos. Ambos conducen al mismo destino (la exploración de todos los vértices o hasta que se encuentra uno determinado), si bien el orden en que éstos son "visitados" decide radicalmente el tiempo de ejecución de un algoritmo, como se verá posteriormente.

En primer lugar, una forma sencilla de recorrer los vértices es mediante una función recursiva, lo que se denomina búsqueda en profundidad. La sustitución de la recursión (cuya base es la estructura de datos pila) por una cola nos proporciona el segundo método de búsqueda o recorrido, la búsqueda en amplitud o anchura.

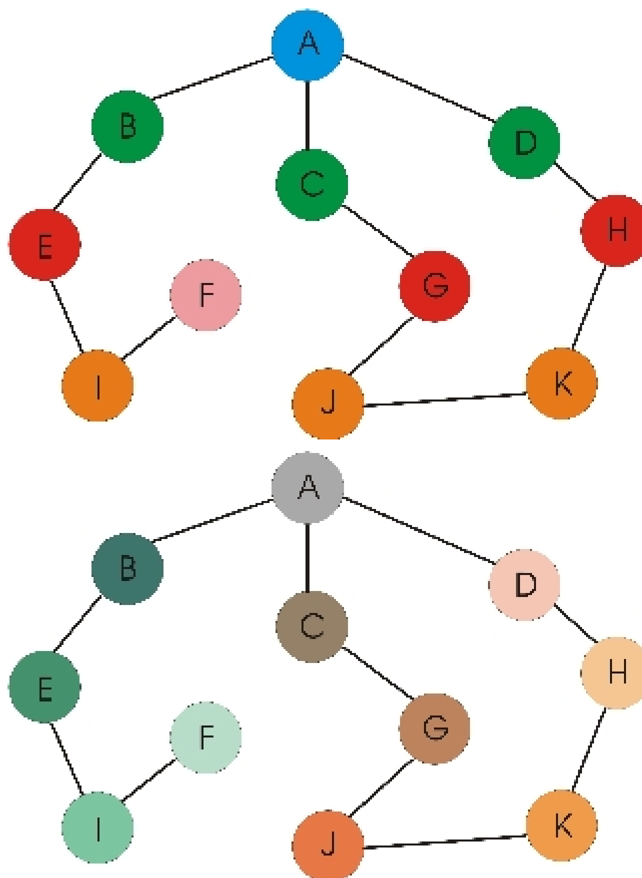


Figura 4.15. Grafo Explorado

Suponiendo que el orden en que están almacenados los nodos en la estructura de datos correspondiente es A-B-C-D-E-F... (el orden alfabético), tenemos que el orden que seguiría el recorrido en profundidad sería el siguiente:

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

En un recorrido en anchura el orden seria, por contra:

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

Es decir, en el primer caso se exploran primero los verdes y luego los marrones, pasando primero por los de mayor intensidad de color. En el segundo caso se exploran primero los verdes, después los rojos, los naranjas y, por último, el rosa.

Es destacable que el nodo D es el último en explorarse en la búsqueda en profundidad pese a ser adyacente al nodo de origen (el A). Esto es debido a que primero se explora la rama del nodo C, que también conduce al nodo D.

En estos ejemplos hay que tener en cuenta que es fundamental el orden en que los nodos están almacenados en las estructuras de datos. Si, por ejemplo, el nodo D estuviera antes que el C, en la búsqueda en profundidad se tomaría primero la rama del D (con lo que el último en visitarse sería el C), y en la búsqueda en anchura se exploraría antes el H que el G.

Métodos de Recorridos para Grafos

Con frecuencia se desea recorrer una estructura de datos, es decir, visita cada uno de los elementos de una manera sistemática. Ya se ha visto técnicas de recorrido para árboles y listas; examinaremos ahora técnicas de recorrido para grafo.

Los elementos de un grafo debe de ser visitados son, por lo regular, los nodos del grafo. Siempre es posible recorrer un grafo de manera eficiente visitando los nodos del grafo. De un amanera que depende de la implantación elegida. Por ejemplo si un grafo con n nodos se representa, mediante una matriz de adyacencia o un arreglo de lista de adyacencia, el simple listado de elementos de 0 a n – 1, recorre el grafo. De manera similar, si los nodos del grafo se guardan en una lista ligada, un árbol de búsqueda, una tabla de dispersión o alguna otra estructura, el recorrido de la estructura adyacente podría considerarse un recorrido del grafo. Sin embargo es de mayor interés un recorrido que corresponda a la estructura del grafo del objetivo y no a una para la estructura de implantación subadyacente. Es decir la secuencia en la cual se visitan los nodos deberían relacionarse con la estructura de adyacencia de grafo.

La definición de un recorrido relacionado a la estructura de un grafo es mas compleja que para una lista o un árbol por tres razones:

1. En general no hay un primer nodo natural de un grafo a partir del cual deba empezar el recorrido como hay un primer nodo en una lista o en

un nodo raíz de un árbol. Además una vez determinado un nodo inicial y visitado todos los nodos que puede alcanzar a partir de ese nodo, pues queda otro nodo en el grafo que no sea visitado debido a que no son alcanzables partiendo del nodo inicial. Esto es también improbable en una lista o en un árbol, donde todo nodo es alcanzable de la cabecera o de la raíz. Así una vez que a sido visitado todos los nodos alcanzables en un grafo, el algoritmo de recorrido enfrenta otra vez el problema de seleccionar otro nodo.

2. No hay orden natural entre los sucesores de un nodo particular. Así no hay un orden a priori en el cual deba visitarse los sucesores de un nodo particular.

3. a diferencia de un nodo de una lista o de un árbol, el nodo de un grafo puede tener mas de un predecesor . si el nodo x es un sucesor de los nodos *y* y *z*, *x* puede ser visitado después de *y* pero antes de *z*. Por lo tanto es posible que un nodo sea visitado antes de sus antecesores. De hecho, si un grafo es cíclico, cualquier recorrido posible tiene que incluir algún nodo que se visite antes que uno de sus antecesores.

Recorrido de Profundidad (Depth-First)

La técnica de recorrido de profundad se define mejor usando el algoritmo *Depthfirst(s)* que sisita todos los nodos alcanzables por *s*. Suponemos un algoritmos *visi(nd)* que visite n nodo nd y una función *visited(nd)* que da como resultado verdadero si nd a sido visitado y falso en caso contrario. Como mejor se implanta esto es un indicador en cada nodo. *visit* hace el campo igual a verdadero. Para ejecutar el recorrido del campo se hace primero igual a falso para todos los nodos. A continuación presentamos un algoritmo recursivo para *Depthfirst(s):*

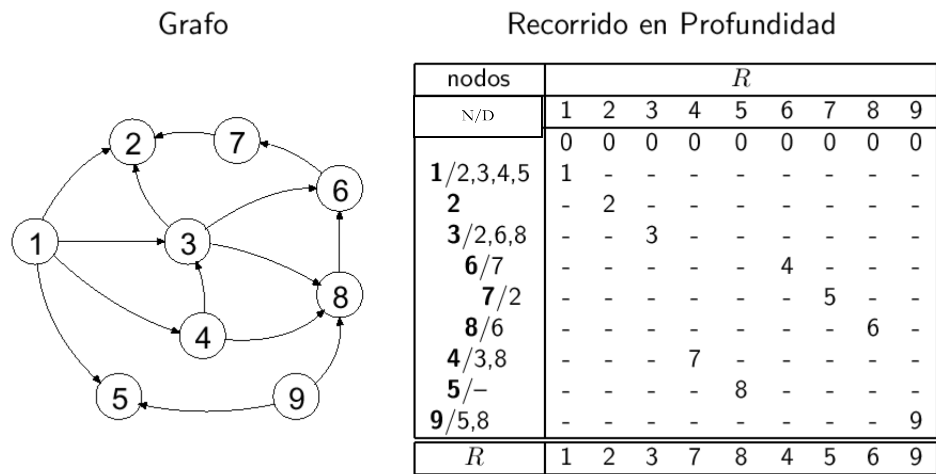
```
Depthfirst(n)
D,A: puntero
Inicio
   A= n^.lady
   D=A^.ptrV
   Mientras ( A!=NULL) Hacer
      Si Not Visitado(n) Entonces
            Visit(n)
            Buscar profundidad (D)
      Sino
            A=A^.Sgte
            D=A^.PtrV
      Fin Si
   Fin Mientras
Fin
```

Un recorrido depth first, como lo indica el nombre, recorre un solo camino del grafo tan lejos como pueda ir (es decir, hasta visitar un nodo que no tenga

sucesor o un nodo cuyo sucesor ya hayan sido visitados). Después continua con el ultimo nodo del camino que acaba de recorrer que tenga un sucesor no visitado y comienza a recorrer un nuevo camino que sale de ese nodo. El recorrido primero por profundidad también es llamado a veces **búsqueda primero en profundidad**.

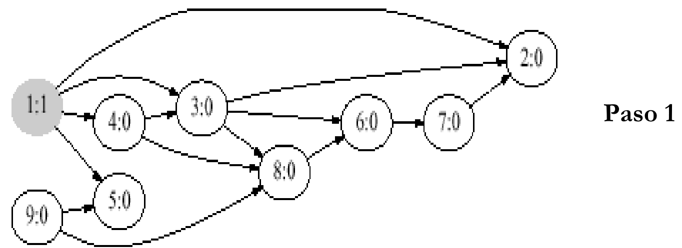
Este recorrido igual que cualquier otro método que cree un bosque extensivo, puede usarse para determinar si un grafo esta conectado y para determinar los componentes conexas de un grafo no dirigido. También puede ser utilizado para determinar si un grafo es aciclico. Tanto en un grafo dirigido como en un grafo no dirigido, existen un ciclo si y solo si existe una arista trasera en un bosque extensivo primero por profundidad.

Recorrido en Profundidad: Ejemplo

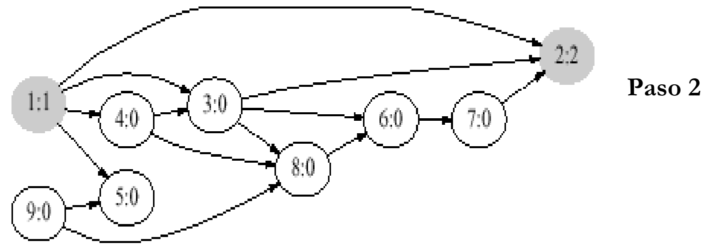


nodos	R								
N/D	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0
1/2,3,4,5	1	-	-	-	-	-	-	-	-
2	-	2	-	-	-	-	-	-	-
3/2,6,8	-	-	3	-	-	-	-	-	-
6/7	-	-	-	-	-	4	-	-	-
7/2	-	-	-	-	-	-	5	-	-
8/6	-	-	-	-	-	-	-	6	-
4/3,8	-	-	-	7	-	-	-	-	-
5/–	-	-	-	-	8	-	-	-	-
9/5,8	-	-	-	-	-	-	-	-	9
R	1	2	3	7	8	4	5	6	9

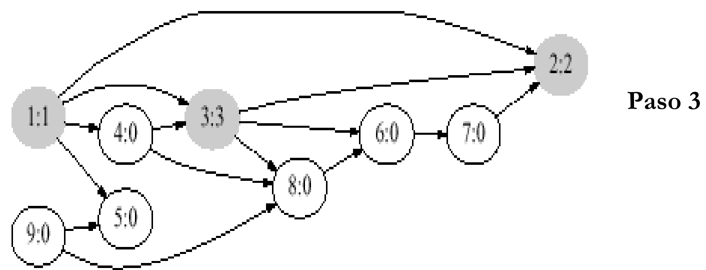
Orden de Visita de Nodos: 1, 2, 3, 6, 7, 8, 4, 5, 9



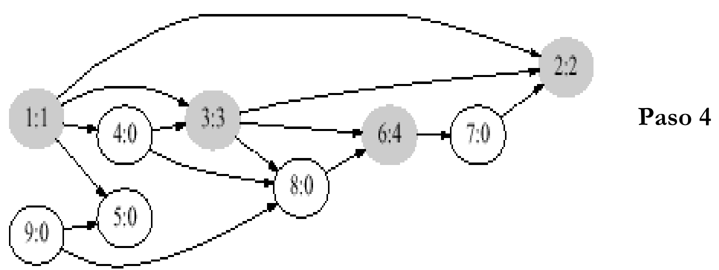
Paso 1



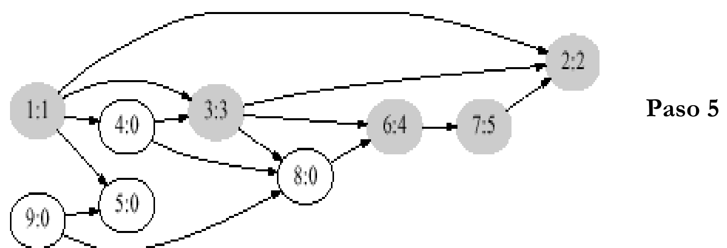
Paso 2



Paso 3



Paso 4



Paso 5

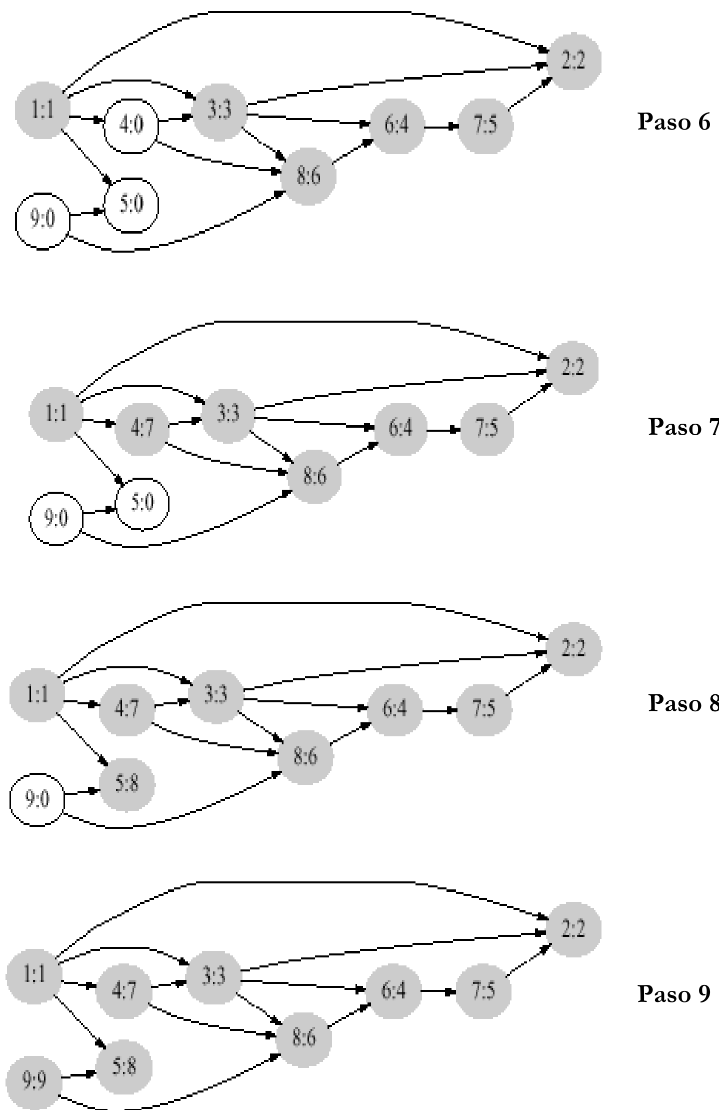


Figura 4.16 Traza Recorrido en Profundidad

Recorrido Primero en Amplitud(Breadth-First)

Otro método de recorrido es el primero en amplitud, visita todos los sucesores de un nodo visitado antes de visitar cualquier sucesor de cualquier otro sucesor. Esto en contraposición con el recorrido primero en profundidad que visita los sucesores de un nodo visitado antes de cualquiera de sus hermanos . mientras que el recorrido primero en profundidad tiende a crear árboles muy amplio y cortos.

En la implantación del recorrido primero por profundidad, cada nodo visitado se coloca en una pila (ya sea en forma implica vía recursión), refleja el hecho de que el ultimo nodo visitado es el primer nodo cuyos sucesores serán visitado. El recorrido primero en amplitud se implanta usando una cola , que representa el hecho de que el primer nodo visitado es el primer nodo cuyo sucesores son visitado. A

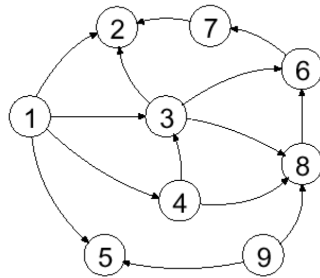
continuación se presentara un algoritmo *BreadthFirst(s)* para recorrer un grafo usando recorrido primero en amplitud y comenzando en el nodo s:

```
Breadthfirst(n)
D,A: puntero
Inicio
   NdCola= NULL
   Mientras  n!=NULL Hacer
       Visit(n)
       Insertcola(ndCola,n)
       Mientras ndCola<>NULL Hacer
             X=removercola(ndCola)
              yptr=X^.Lady
              nd=yptr^.ptrV
              Mientras nd<>NULL Hacer
                   Si NotVisited(nd) Entonces
                         Visit(nd)
                          Insertcola(ndcola,nd)
                    Fin si
                    Yptr=yptr^.Sgte
                    Nd=yptr^.ptrV
              Fin Minetras
       Fin Minetras
   Fin Minetras
Fin
```

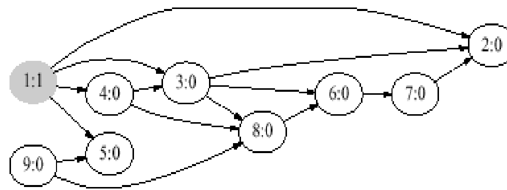
Este recorrido puede usarse para alguna de las mismas aplicaciones que el recorrido primero en profundidad. En particular el recorrido primero en amplitud puede ser usado para determinar si un grafo no dirigido esta conectado y para identificar los componentes conexos del grafo. El recorrido primero en amplitud también puede ser usado para determinar si un grafo es cíclico. Para un grafo dirigido, esto se detecta cuando se encuentra una arista trasera; para un grafo no dirigido, se detecta cuando se encuentra una arista cruzada dentro del mismo árbol.

Para los grafos sin peso, el recorrido primero en amplitud también puede ser usado para encontrar el camino mas corto (con menos arcos) de un nodo a otro. Comenzando el recorrido en el primer nodo y parta cuando el nodo blanco haya sido alcanzado. El camino de la raíz del nodo objeto en el árbol extensivo primero por amplitud es el camino mas corto entre los dos nodos.

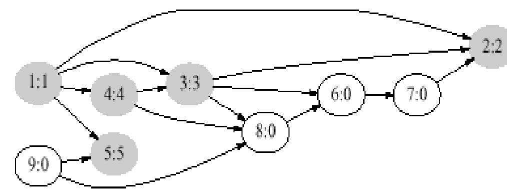
Recorrido en Anchura: Ejemplo



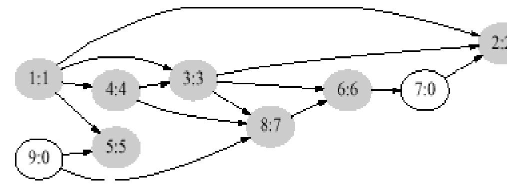
v	u	w	1	2	3	4	5	6	7	8	9	
1			1	0	0	0	0	0	0	0	0	$< 1 >$
	1		-	-	-	-	-	-	-	-	-	$<>$
		2	-	2	-	-	-	-	-	-	-	$< 2 >$
		3	-	-	3	-	-	-	-	-	-	$< 2, 3 >$
		4	-	-	-	4	-	-	-	-	-	$< 2, 3, 4 >$
		5	-	-	-	-	5	-	-	-	-	$< 2, 3, 4, 5 >$
	2		-	-	-	-	-	-	-	-	-	$< 3, 4, 5 >$
	3		-	-	-	-	-	-	-	-	-	$< 4, 5 >$
		2	-	-	-	-	-	-	-	-	-	—
		6	-	-	-	-	-	6	-	-	-	$< 4, 5, 6 >$
		8	-	-	-	-	-	-	-	7	-	$< 4, 5, 6, 8 >$
	4		-	-	-	-	-	-	-	-	-	$< 5, 6, 8 >$
		3	-	-	-	-	-	-	-	-	-	—
		8	-	-	-	-	-	-	-	-	-	—
	5		-	-	-	-	-	-	-	-	-	$< 6, 8 >$
	6		-	-	-	-	-	-	-	-	-	$< 8 >$
		7	-	-	-	-	-	-	8	-	-	$< 8, 7 >$
	8		-	-	-	-	-	-	-	-	-	$< 7 >$
		6	-	-	-	-	-	-	-	-	-	—
	7		-	-	-	-	-	-	-	-	-	$<>$
9			-	-	-	-	-	-	-	-	9	$< 9 >$
	9		-	-	-	-	-	-	-	-	-	$<>$
	9	5	-	-	-	-	-	-	-	-	-	—
	9	8	-	-	-	-	-	-	-	-	-	—
			1	2	3	4	5	6	8	7	9	



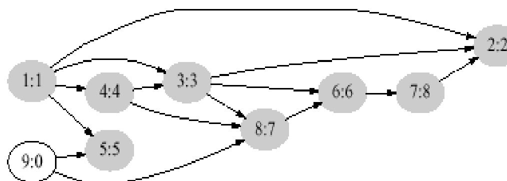
Paso 1



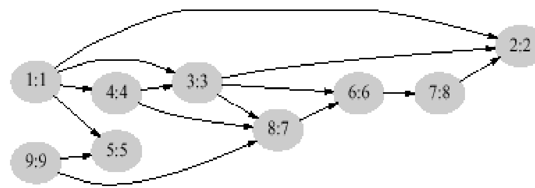
Paso 2



Paso 3



Paso 4



Paso 5

Figura 4.17 Traza Recorrido en Amplitud

Búsquedas

Búsqueda en profundidad

Se implementa de forma recursiva, aunque también puede realizarse con una pila. Se utiliza un array `val` para almacenar el orden en que fueron explorados los vértices. Para ello se incrementa una variable global `id` (inicializada a 0) cada vez que se visita un nuevo vértice y se almacena `id` en la entrada del array `val` correspondiente al vértice que se está explorando.

La siguiente función realiza un máximo de V (el número total de vértices) llamadas a la función `visitar`, que implementamos aquí en sus dos variantes: representación por matriz de adyacencia y por listas de adyacencia.

```
Buscar_profundidad(n,Infor)
D,A: puntero
Inicio
  Insertacola(n)
  A= n^.lady
  D=A^.ptrV
  Mientras ( A!=NULL) Y (infor<>D^.Vertice) Hacer
     Si Not Verificarnodo(n) Entonces
          Buscar profundidad (D,Infor,Band )
     Sino
          A=A^.Sgte
          D=A^.PtrV
     Fin Si
   Fin Mientras
Fin
```

Búsqueda en amplitud o anchura

La diferencia fundamental respecto a la búsqueda en profundidad estriba en el cambio de estructura de datos: una cola en lugar de una pila. En esta implementación, la función del array `val` y la variable `id` es la misma que en el método anterior.

```
Buscar_Amplitud(n,Infor)
t : puntero
Inicio
  Insertacola(&cola,k);
  Mientras Not vacia(cola) Hacer
    eliminacola(cola,k)
    Id = id + 1
    val[k] = id
    t=ady[k]
    Mientras t<>z Hacer
      Si val[t^.vert]=0 Entonces
        encolar(&cola,t^.vert)
        val[t^.vert]=-1
      Fin Si
      t=t^.Sgte
    Fin Mientras
  Fin Mientras
Fin
```

Matriz de Camino, Cierre Transitivo

Conceptos importantes a considerar en la construcción de grafos son: camino entre par de vértices, la matriz de camino y el cierre transitivo.

Camino entre par de vértices

Se G un grafo de n vértices y A su matriz de adyacencia de tipo lógico. Obsérvese la expresión logica A[i,k] and A[k,j]. Esta expresión será cierta cierta si y solo si los valores de ambos operadores lo son, lo cual implica que hay un arco desde

el vértice i al vértice k y otro desde k a j. También podemos decir que la expresión sea cierta implica la existencia de un camino de longitud 2 desde el vértice i al j.

Ahora consideremos la expresión:

(A[i,1] and A[1,j]) or (A[i,2] and A[2,j]) or …. Or (A[1, Numverts] and A[Numverts,j])

Si esta expresión es cierta implica que hay al menos un camino de longitud 2 desde el vértice I al vértice j que pase a través del vértice 1, o a través del vértice Numverts.

Recordando el producto matricial A x A observamos que la expresión anterior si cambiamos and por producto y or por suma representa el elemento A_{ij} de la matriz A^2 . según esto los elementos (A_{ij}) de la matriz A^2 son verdadero si existe un camino de longitud 2 desde el vértice i al vértice j $\forall$ i,j = 1..n.

De igual forma el producto matricial $A^2 \times A = A^3$ nos permite determinar si existe un camino de longitud 3 entre cualquier par de vértices del grafo. En general para determinar la existencia de camino de longitud m entre cualquier par de vértices se forma el producto bolean de los caminos de la matriz A^{m-1} con la matriz adyacente.

En la figura siguiente tenemos un grafo dirigido.

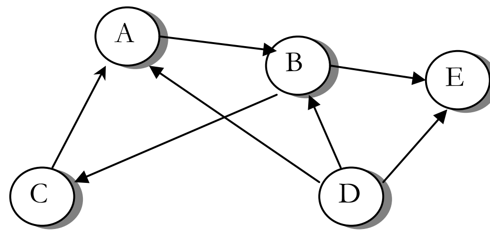


Figura 4 . 18 grafo dirigido

La matriz de adyacencia

$$\begin{pmatrix} F & T & F & F & F \\ F & F & T & F & T \\ T & F & F & F & F \\ T & T & F & F & T \\ F & F & F & F & F \end{pmatrix}$$

El producto bolean A x A

$$\begin{pmatrix} F & F & T & F & T \\ T & F & F & F & F \\ F & T & F & F & F \\ F & T & T & F & T \\ F & F & F & F & F \end{pmatrix}$$

Así, A_{15} es verdadero (True) ya que hay un camino de longitud 2 desde A-E(A→B→E).

El producto bolean A^2 x A

$$\begin{pmatrix} F & T & F & F & F \\ F & F & T & F & T \\ T & F & F & F & F \\ T & T & F & F & T \\ F & F & F & F & F \end{pmatrix}$$

Así, A_{41} es verdadero ya que hay un camino de longitud 3 desde D – A (D→B→C→A)

Con el producto bolean se ha obtenido si hay camino de longitud m entre un par de vértices. En caso de que la matriz de adyacencia esta representada mediante 0,1 podemos obtener no solo si hay caminos de longitud *m* sino además el numero de caminos.

Sea G=(V,A) un grafo representado por la matriz de adyacencia. A tal que el elemento A_1(i,j) es 1 si hay un arco desde el vértice i al j, también nos dan el numero de caminos del longitud 1 desde i a j. Haciendo un razonamiento similar, la expresión :

*(A[i,1] * A[1,j]) + (A[i,2] * A[2,j]) + …. + (A[1, Numverts] * A[Numverts,j])*

nos da todos los posibles caminos de longitud 2 desde el vértice i al j, además recordamos el producto matricial A x A, representa el elemento A_{ij} de la matriz A^2 . podemos generalizar, la forma de obtener el numero de caminos de longitud k entre cualquier par de vértices del grafo es obtener el producto matricial A^2, A^3, ... A^k, entonces el elemento A_k(i,j)nos da el numero de caminos de longitud k desde el vértice i hasta el vértice j.

Utilizando el grafo de la figura 4. cuya matriz de adyacencia es

El producto A x A

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{A^2} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

El producto A^2 x A

$$\mathbf{A^3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Existe un camino de longitud 3 desde el vértice C al vértice D.

```
Producto_A(A,K,Nvs,AK)
I : Entero
Inicio
   AK =A
   Si  K>=2 Entonces
      Prod(A,A,Ak)
      Para I=3 hasta K Hacer
         Prod(Ak,A,Ak)
      Fin de Para
   Fin Si
Fin
```

```
Prod(A,B,N)
F,c,k,S : Entero
Inicio
   Para f=1 hasta Nvs Hacer
      Para c=1 hasta Nvs Hacer
         S=0
         Para k=1 hasta Nvs Hacer
            S=S+A[f,k]*B[c,k]
         Fin de Para
         N[f,c]=S
      Fin de Para
   Fin de Para
Fin
```

Matriz de caminos

Se a un grafo con n vértices. La matriz de caminos de G es la matriz de $n \times n$ P definidas:

$$P = \begin{cases} 1 \text{ si hay un camino desde } V_i \text{ a } V_j \\ 0 \text{ si no hay un camino desde } V_i \text{ a } V_j \end{cases}$$

el camino de V_i a V_j se ra siempre si $V_i \neq V_j$, o bien un ciclo cuando los extremos sean el mismo vértice $V_i = V_j$. Al ser el grafo G de n vértices, un camino simple ha de tener longitud de n-1 o menos, y un ciclo a de tener longitud n o menor. Según esto podemos encontrar la siguiente relación entre la matriz de caminos P, la matriz de adyacencia A y la sucesiva potencia de A.

Dada la matriz $B_n = A + A_2 + A_3 + ... + A_n$ la matriz de camino $P=(P_{ij})$ es tal que un elemento $P_{ij}=1$ sis y solo si $B_n(i,j) \geq 1$ y en otro caso $P_{ij}=0$.

Una vez que tenemos la matriz de camino de un grafo G dirigido podemos determinar de manera mas fácil si el grafo es fuertemente conexo. Recordar que para que G sea fuertemente conexo se ha de cumplir para todo par de vértices V_i ,V_j ha de existir un camino de V_i a V_j y un camino de V_j a V_i. Por tanto, para que G sea fuertemente conexo la matriz de caminos P ha de tener todos los elementos a 1.

Cierre Transitivo

El cierre o contorno transitivo de un grafo G es otro grafo G' que consta de los mismos vértices que G y que tiene como matriz de adyacencia la matriz de caminos P del grafo G. Según esta definición, un grafo es fuertemente conexo si y solo si su cierre transitivo es un grafo completo.

· **A** , matriz cierre transitivo que queremos calcular
· **Ady**k, matriz que representa la existencia de caminos de longitud k
· **Prod,** matriz auxiliar que representa el producto booleano de dos matrices

```
CierreTransitivo_MM(Ady , A )
i,j,k: Integer;
Adyk, P: Matriz;
Inicio
   A = Ady;
   P = Ady ;
   Para k=2 Hasta n Hacer
        prod(Adyk, P, Ady)
   Fin de Para
   Para i=1 Hasta n Hacer
      Para  j=1 Hasta n Hacer
         A + [i,j]:= A + [i,j] or Adyk [i,j]
         P:= Adyk
      Fin de Para
   Fin de Para
End
```

Ejemplo de un cierre transitivo.

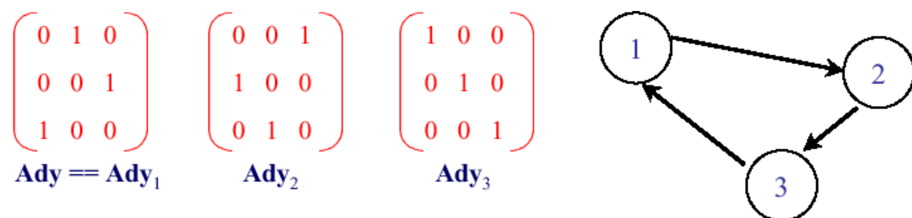


Figura 4.19. Grafo de Cierre transitivo

Árbol de cobertura Mínimo(MST)

Dado una red G(V,A) un árbol de cobertura mínimo es el árbol que conecta todos los vértices de la red con n-1 arcos (n siendo el numero de vértices y m el numero de arcos) pero con el menor costo total.

Los dos algoritmos más importantes para determinar cobertura mínima son:

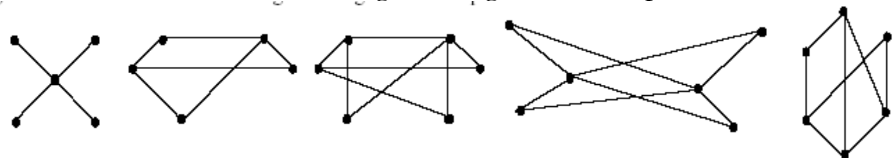
Algoritmo de Kruskal (1956). El cual primero ordena todos los arcos en orden no decrecientes, después los examina de menor a mayor costo de la red y si no forman ciclos con los ya seleccionados, se incorporan al árbol de cobertura mínima. La construcción se detiene cuando se han seleccionados, los n-1 arcos o se han examinado los m arcos de la red.

Algoritmo de Prim's(1957). Construye un árbol de cobertura partiendo con un vértice arbitrario s el cual se une con los vecinos más cercanos examinando los arcos incidentes a s que posee el menor costo hasta revisar todos los arcos de la red.

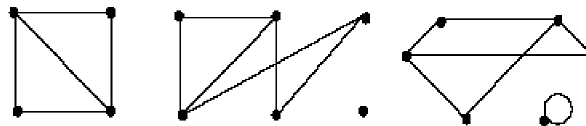
Para grafos pequeños (hasta 100 nodos) Prim's funciona mejor, pero para grafos mayores y densos Kruskal presenta un mejor desempeño.

Ejercicios

4.1 Construir un grafo de orden 5 cuyos vértices tengan grados 1, 2, 2, 3, 4. ¿Cuál es el tamaño del grafo?
4.2 A una fiesta asisten 20 invitados. Cada uno de ellos conoce a un nº diferente de invitados. ¿Es esto posible? Probar que todo grafo sin bucles de orden n³2 tiene al menos dos vértices con el mismo grado.
4.3 En un mapa de carreteras de una región aparecen 25 tramos de carretera. Sabiendo que los cruces de carretera se producen siempre en una población y que de cada lugar parten al menos cuatro caminos, ¿cuántos lugares aparecen en el mapa?
4.4 Si G es un grafo simple con 52 aristas, ¿cuál es el menor número de vértices que puede tener G?
4.5 Dibujar los grafos simples no isomorfos de orden 4.
4.6 Disponemos de 6 ordenadores y 9 cables de conexión. Queremos que cada ordenador se conecte con otros 3 ordenadores. ¿Existe alguna forma de conectarlos? ¿Es única?
4.7 Determinar cuáles de los siguientes grafos son bipartidos:

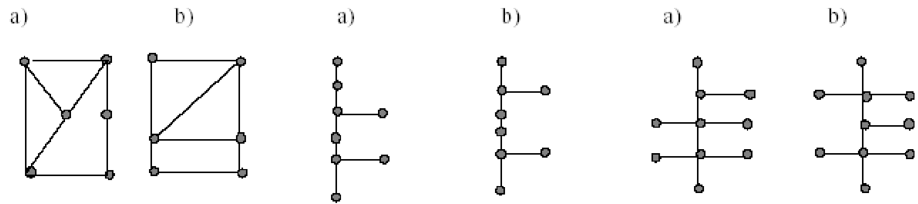


4.8 ¿Cuántas aristas tiene un grafo simple si sus vértices tienen los siguientes grados 4,3,3,2,2? Dibuja tal grafo.
4.9 Estudia si las siguientes sucesiones son gráficas:
3,3,3,3,2 1,2,3,4,4 3,4,3,4,3 5,5,4,4,3,3,3,1,0,0
3,3,2,2,2,2,1,1 6,4,4,4,3,3,3,3 5,3,3,3,2,2,1,1
4.10 Estudia la regularidad de los grafos, Kn , Cn , Wn y Qn para los diferentes valores de n
4.11 Describe todos los grafos no isomorfos cuya sucesión de grados sea 3,3,2,2,2.
4.12 Sea G un grafo simple con n vértices y q aristas, cuyos vértices tienen grado k ó k+1. Demostrar que si G tiene nk vértices de grado k entonces nk=(k+1)n-2q
4.13 Obtener la lista de adyacencia y las matrices de adyacencia e incidencia para los siguientes grafos:



4.14 Representa la matriz de adyacencia de cada uno de los siguientes grafos:
a) K4 b) K1,4 c) K2,3
d) C4 e) W4 f) Q3

4.15 Determinar cuáles de los siguientes pares de grafos son isomorfos:



Problemas

4.16 Realizar un procedimiento que imprima un grafo

4.17 Construir un procedimiento que determine el número de componentes conexas que posee un grafo.

4.18 Construir procedimientos que devuelvan el grado de entrada y grado de salida de un vértice para un grafo dirigido y grado de un vértice para uno no dirigido.

4.19 Un grafo no dirigido se dice de Euler si existe un camino Euleriano que incluye a todas sus aristas. Construir una función que dado un grafo no dirigido determine si es de Euler.

4.20 Realizar un procedimiento que dado un grafo no dirigido determine cual es su grafo dual.

4.21 ¿Puede recuperarse un grafo no dirigido a partir de sus recorridos en anchura y profundidad?

4.22 Dado un grafo no dirigido G=(V,E),con v>1 vértices, demostrar que las 3 siguientes afirmaciones son equivalentes:

e. G es conexo y no tiene ciclos simples.
f. G es conexo y tiene v-1 aristas.
g. Cada par de vértices de G están conectados por exactamente un camino.

Bibliografía

Algoritmo y Estructura de Datos
Wirth, Niklaus

Diseño de Algoritmos y Estructuras de datos.
Joyanes Aguilar Luis

Estructura de Datos en Pascal
Tenenbaum, Aron M.

Estructura de Datos y Algoritmo
Aho, Alfred V.

Diseño de Algoritmos y Estructuras de datos Orientadas a Objetos
Heileman, Gregory L.

Fundamentos de Programación, Algoritmos y Estructuras de datos Orientadas a Objetos
Joyanes Aguilar Luis

Estructura de datos en C y C++
Langsam, Yedidyah

Estructuras de datos
Cairo, Osbaldo

Fundamentals of Computer Algoritmits.
Ellis Horowitz.

Estructura de datos en C
Tenenbaum, Aron M.