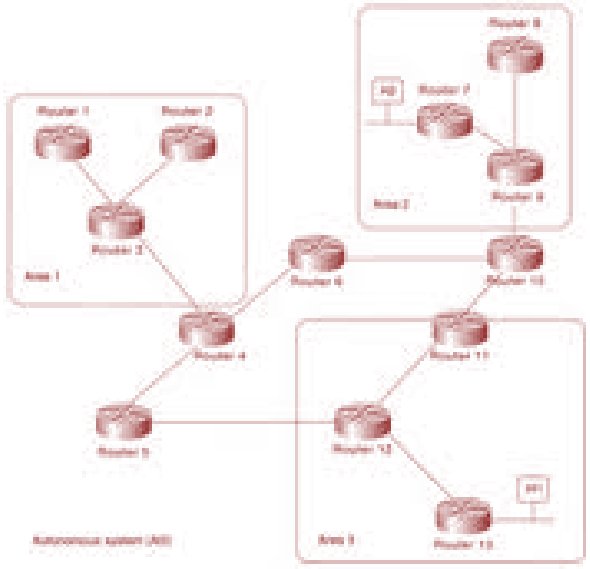
Universidad Nacional Autónoma de México
Facultad de Ingeniería
Algoritmos y estructuras de datos
Tema 4:
ESTRUCTURAS DE DATOS COMPUESTAS:
LISTAS NO LINEALES

4 Estructuras de datos no lineales: listas no lineales

Objetivo: Aplicar las formas de representar y operar en la computadora las principales listas no lineales.

4 Estructuras de datos no lineales: listas no lineales

4.1 Generalidades (Gráficas).
4.2 Árboles.
4.3 Árboles binarios.
4.4 Árboles B.



4.1 Generalidades (Gráficas).

4.1 Gráficas.

Las listas no lineales son estructuras de datos cuyas relaciones son multidimensionales, es decir, a cada nodo de la estructura pueden estar ligados uno o varios nodos.

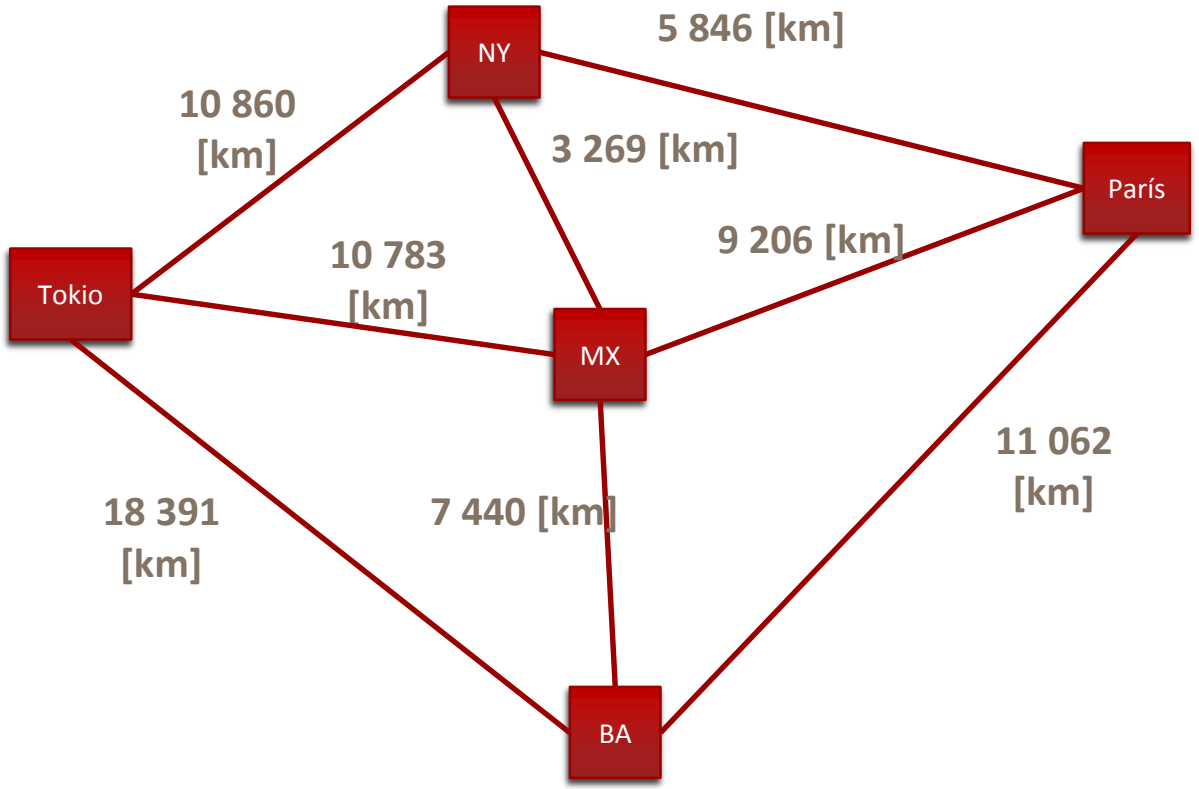
Las listas no lineales más representativas son las gráficas y los árboles.

Gráficas

Las gráficas son estructuras de datos no lineales donde cada elemento puede tener uno o más predecesores y sucesores.

Una gráfica está compuesta por dos entes básicos: nodos (vértices) y arcos (aristas). Las aristas se encargan de conectar los nodos entre sí.

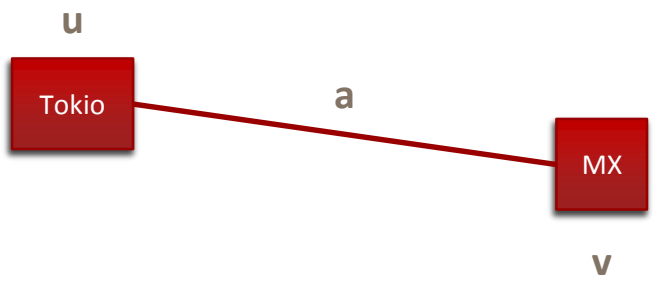
Los vértices almacenan información y las aristas representan la relación que existe entre dicha información.



Una gráfica G, denotada como G = {V, A}, está formada por dos conjuntos: V(G) y A(G). V(G) está formado por los nodos, puntos o vértices de la gráfica G y A(G) son las arcos, aristas o líneas de G que unen a los nodos.

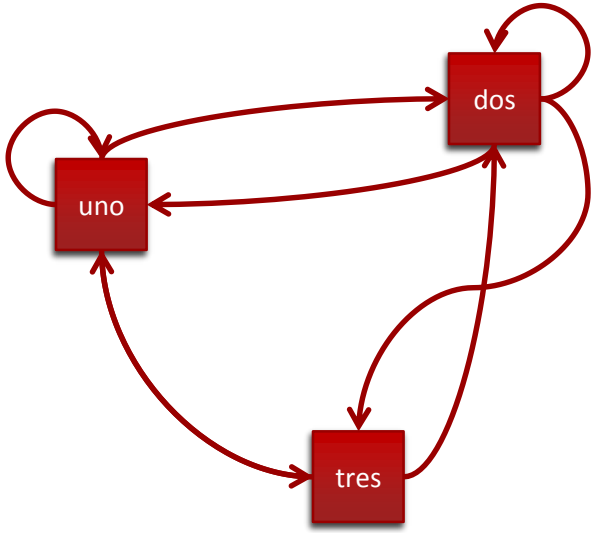
A menos que se especifique lo contrario, se entiende que los conjuntos V(G) y A(G) son finitos.

Cada arista solo puede unir un par de nodos del conjunto V(G). Una arista que enlaza al nodo u con el nodo v se denota como a=(u,v). Por lo tanto, u y v están conectados por a y se dice que a es incidente en u y v.



Si A es un conjunto de elementos {n_1, n_2, n_3, n_4, …}, el producto cartesiano AxA es (<n_1, n_1> <n_1, n_2> <n_1, n_3> ... <n_2, n_1> <n_2, n_2> <n_2, n_3> ... <n_3, n_1> <n_3, n_2> <n_3, n_3> ...), entonces una gráfica G es cualquier subconjunto AxA.

Si A = {uno, dos, tres} y R = {<uno,uno> <uno, dos> <uno, tres> <dos,uno> <dos, dos> <dos, tres> <tres, uno> <tres, dos>}, la gráfica G = {A, R} es:



Conceptos básicos

Grado de un vértice: el grado de un vértice *v* se denota como grado(v) y está definido por el número de aristas que contienen a *v*. Si grado (v) = 0, v no tiene aristas y se dice que es un nodo aislado.

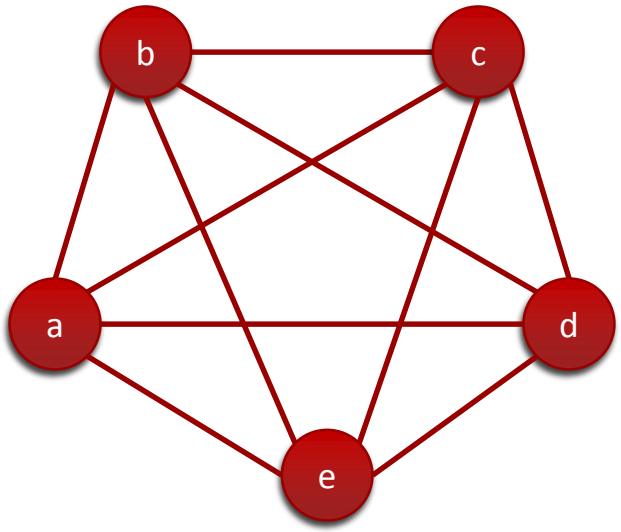
grado(alfa) = 3

grado(cuatro) = 0

Lazo: un lazo es un arista que conecta a un vértice consigo mismo, es decir a = {u, u}.

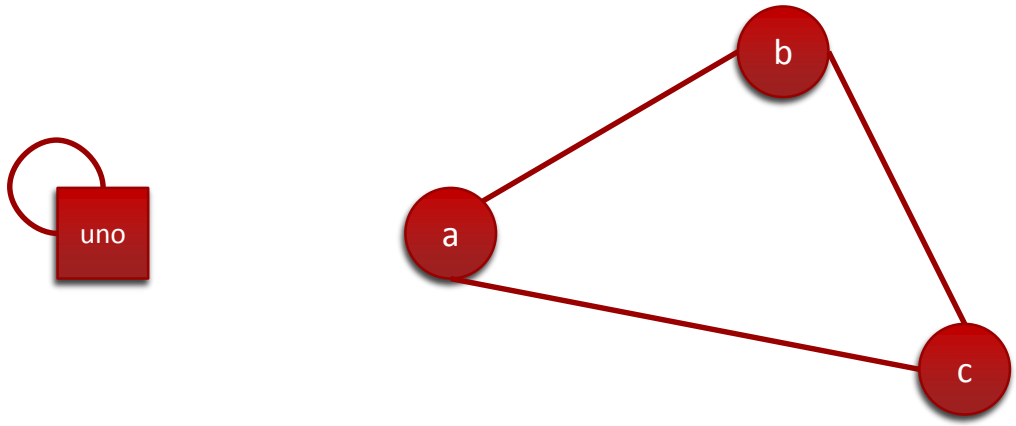
Camino: un camino P de longitud n se define como la secuencia de n vértices que se deben recorrer para llegar del vértice v_1 (origen) al vértice v_n (destino):

$$P = (v_1, \ldots, v_n)$$



¿P(a,d)?
P (a,d) = {a,b,c,d}
P(a,d) = {a, e, d}
P(a,d) = {a, d}

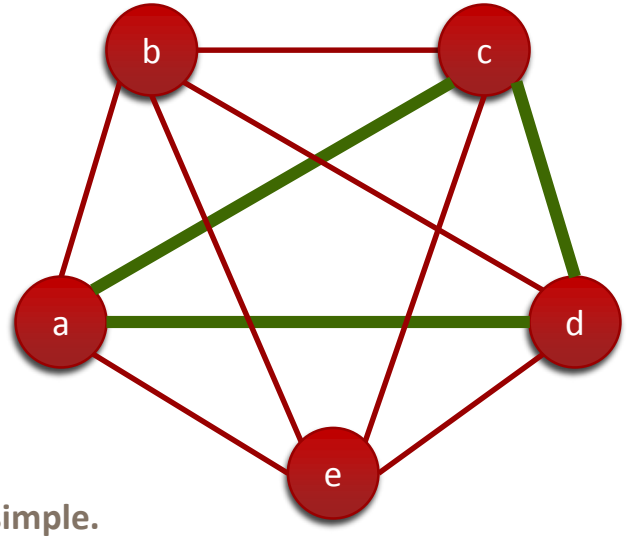
Camino cerrado. P es cerrado si el primero y el último vértices son iguales, es decir, si $v_1 = v_n$.



A-B-C-A es un camino cerrado.

A-B-C no lo es.

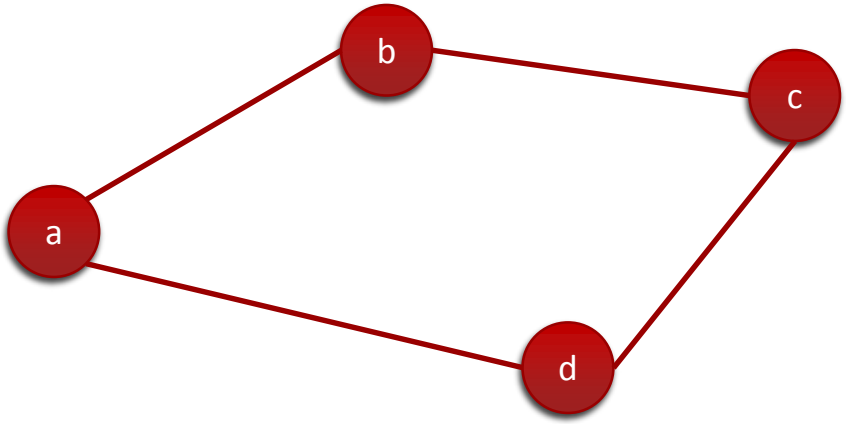
Camino simple. es una ruta P en la que todos sus nodos son distintos. El primero y el último nodo pueden (o no) ser iguales.



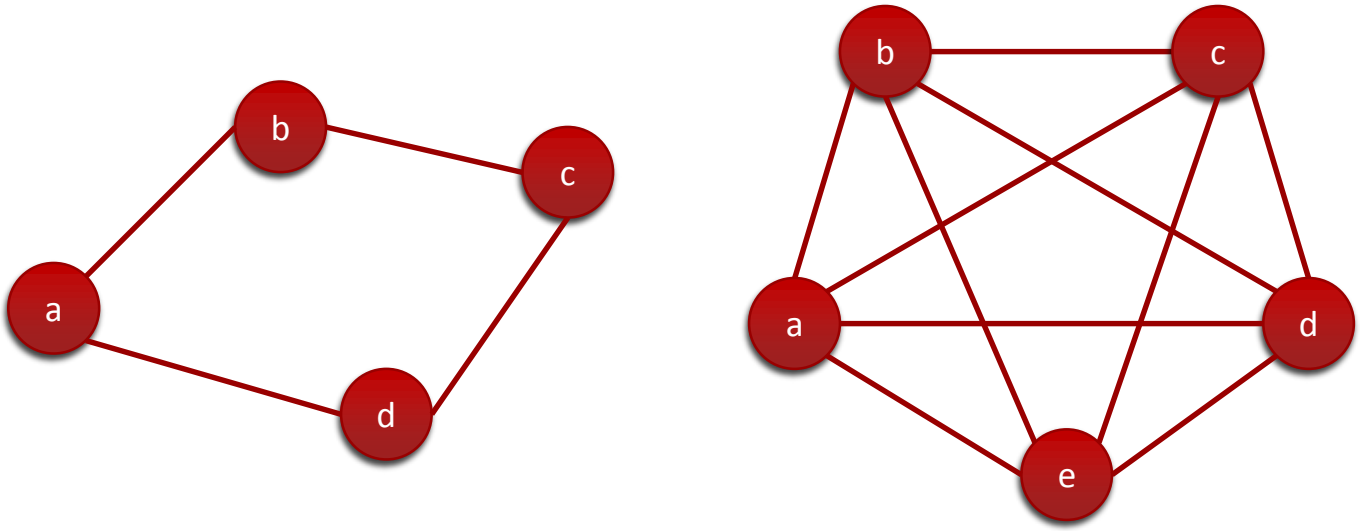
A-C-D-A es un camino simple.

A-C-B-D-C no lo es.

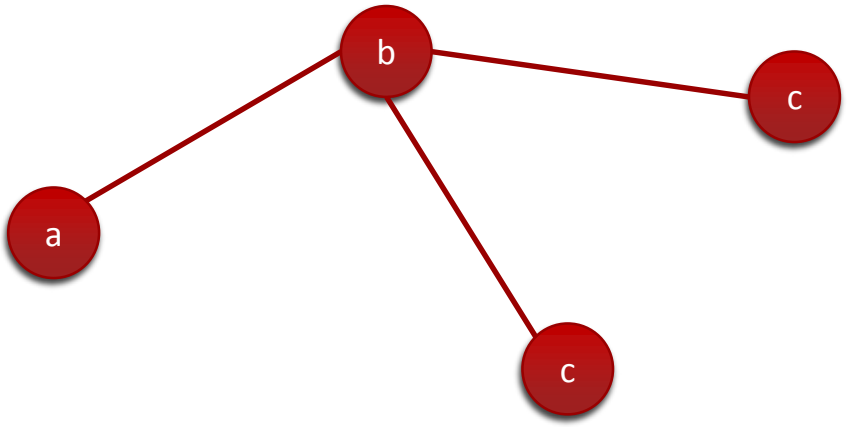
Ciclo: Un ciclo es un camino simple cerrado de longitud 3 o mayor.



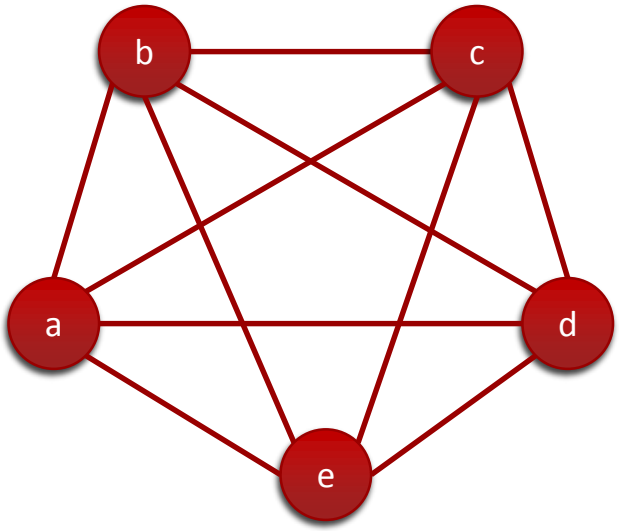
Gráfica conexa: una gráfica es convexa si existe un camino simple entre cualesquiera dos de sus nodos.



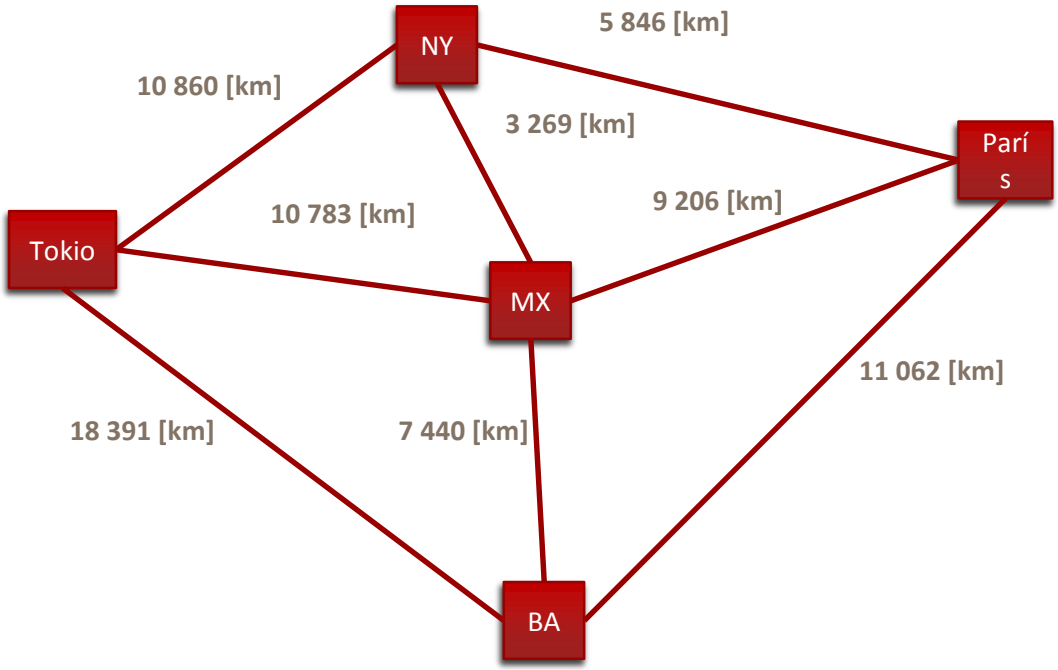
Gráfica árbol: una gráfica es de tipo árbol (o árbol libre) si es una gráfica conexa sin ciclos.



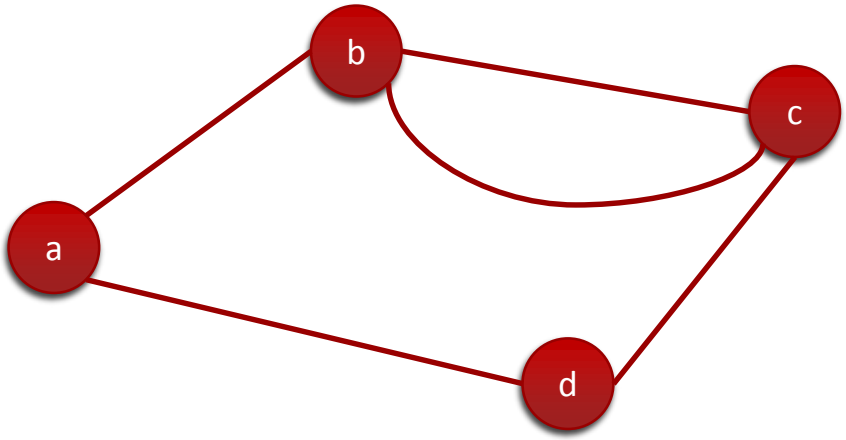
Gráfica completa: una gráfica es completa si cada vértice v es adyacente a todos los demás vértices de G.



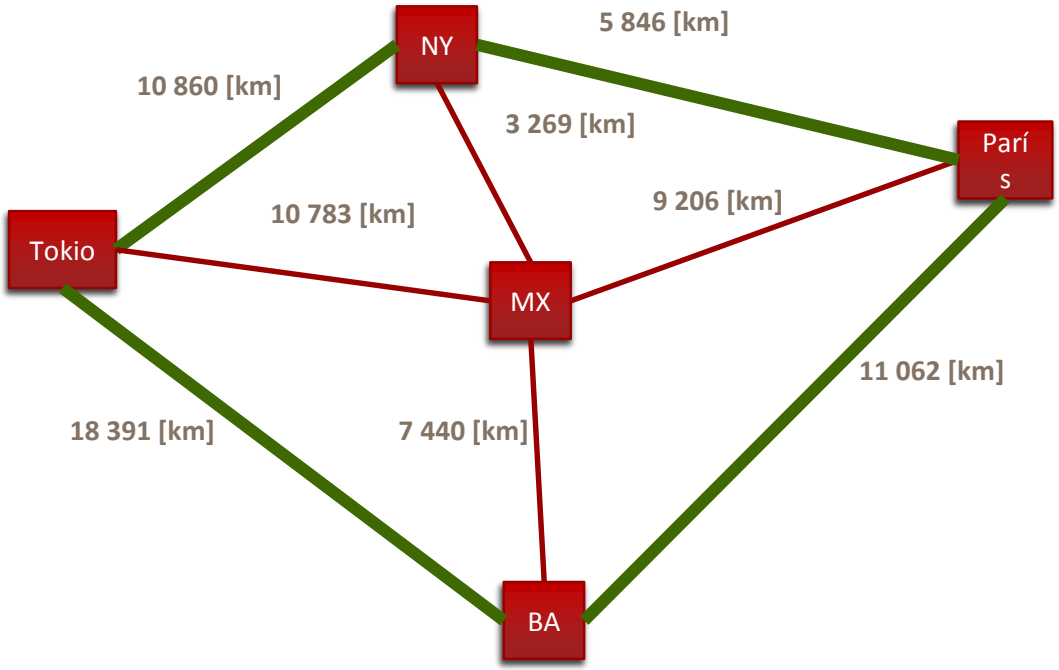
Gráfica etiquetada: una gráfica G está etiquetada si sus aristas a tienen asignado un valor numérico no negativo c(a), llamado costo, peso o longitud.



Multigráfica: una gráfica G se considera multigráfica cuando al menos dos de sus vértices están conectados entre sí por medio de dos aristas (aristas múltiples o paralelas).



Subgráfica: a partir de una gráfica G = {V, A}, G' = {V', A'} es una subgráfica de G si V' ≠ ϕ, V' es un subconjunto de V y A' es un subconjunto de A, donde cada arista A' es incidente con vértices de V'.



Gráficas dirigidas

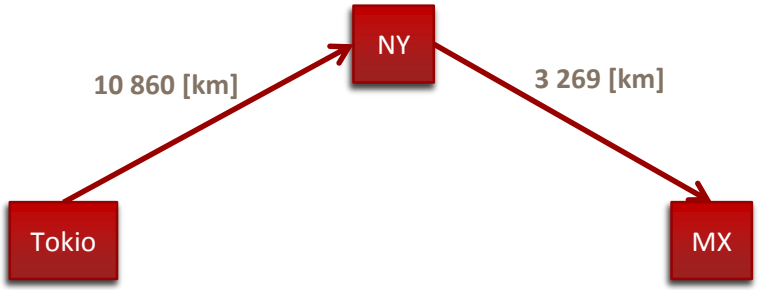
El proceso para solventar un problema consta de dos etapas básicas: el desarrollo del modelo y la implementación del mismo.

La teoría de gráficas proporciona los conceptos necesarios para modelar problemas prácticos.

Una gráfica dirigida se caracteriza porque todas sus aristas tienen asociada una dirección, es decir, está constituida por pares ordenados.

En una gráfica dirigida los nodos representan información y las aristas representan una relación con dirección (o jerarquía) entre los vértices.

Una gráfica dirigida (digráfica) G se caracteriza porque cada arista a tiene una dirección asignada y, por tanto, cada arista está asociada a un par ordenado (u, v) de vértices.



Una arista dirigida a = (u, v) se llama arco y se expresa como

$$u \rightarrow v$$

De la expresión anterior se puede afirmar que:

- a empieza en u y termina en v.
- u es el origen (punto inicial) y v es el destino (o punto final) de a.
- u es predecesor de v y v es sucesor de u.

Dentro de las gráficas dirigidas se tienen dos tipos de grados, los externos y los internos.

El grado externo de un nodo v está dado por el número de líneas que emergen de él. El grado interno de un nodo u está formado por el número de arcos que llegan a él.

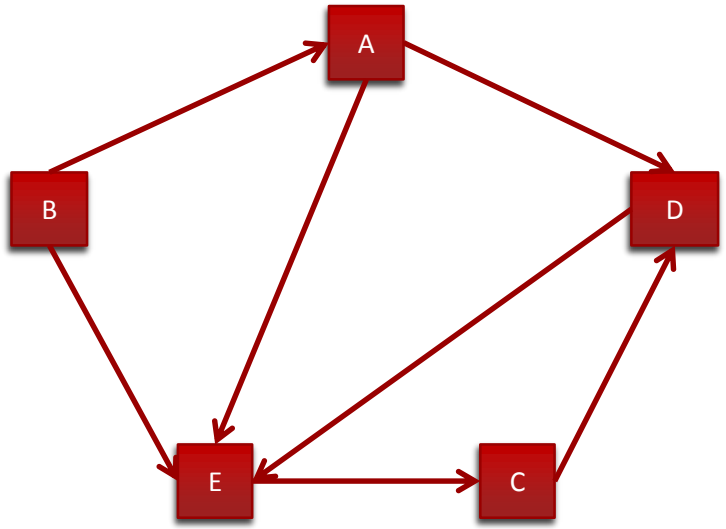
Debido a que las gráficas son estructuras de datos abstractas, su implementación puede ser complicada.

Las representaciones más utilizadas son las matrices de adyacencia y las listas de adyacencia.

Matriz de adyacencia

Una matriz de adyacencia es una matriz booleana de orden n, donde n indica el número de vértices de G. Tanto los renglones como las columnas de la matriz representan los nodos de la gráfica y su contenido representa la existencia (1) o no (0) de arcos entre los nodos *i* y *j*.

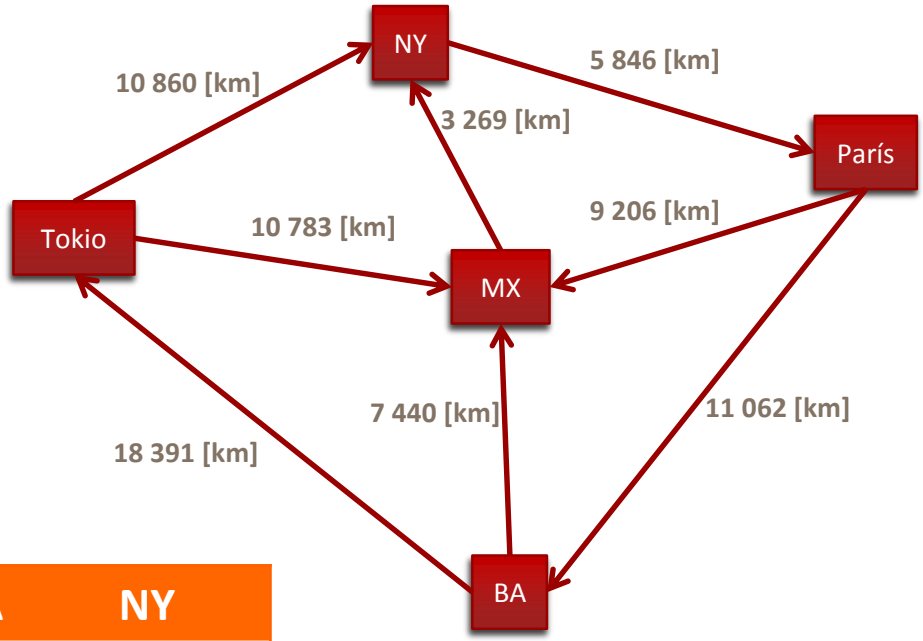
Si G = (V, A) y V = {1, 2, 3, ..., n}, la matriz de adyacencia M que representa a G tiene n x n elementos, donde M [i, j] ($1 \leq j \leq n$ y $1 \leq j \leq n$) es 1 solo si existe un arco que vaya del nodo i al j, y 0 en caso contrario.



	A	B	C	D	E
A	0	0	0	1	1
B	1	0	0	0	1
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	1	0	0

Dentro de una matriz de adyacencia el tiempo de acceso a un elemento es independiente del tamaño de V y A. Por otro lado, el tiempo de búsqueda es del orden O(n).

Una matriz de este tipo requiere un espacio n^2 de memoria aunque el número de arcos de G no llegue a ese número, por tanto, esto representa una desventaja de este arreglo de datos.

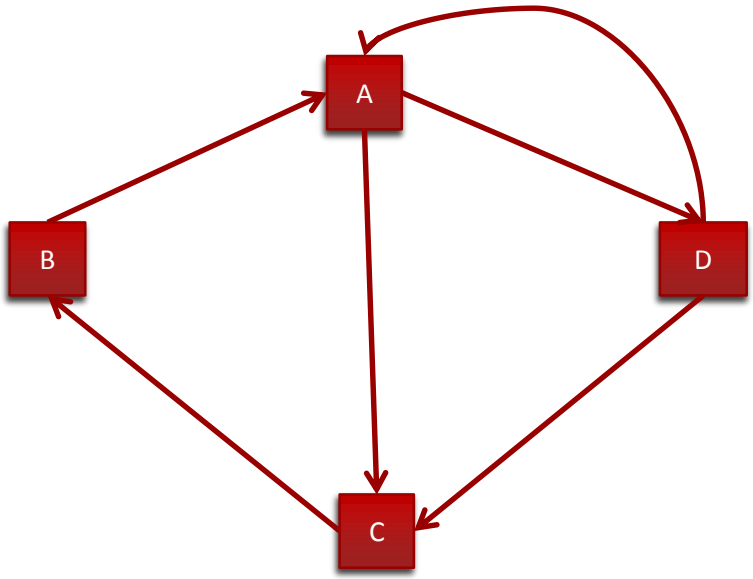


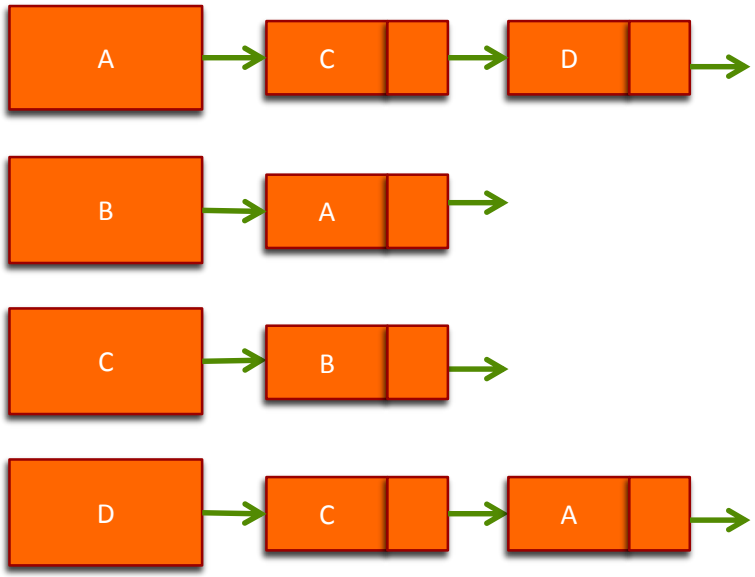
	MX	París	Tokio	BA	NY
MX	0	0	0	0	3 269
París	9 206	0	0	11 062	0
Tokio	10 783	0	0	0	10 860
BA	7 440	0	18 391	0	0
NY	0	5 846	0	0	0

Lista de adyacencia

Una lista de adyacencia para un nodo v es una lista ordenada de todos los vértices adyacentes de n. Por ende, una lista adyacente de una gráfica dirigida está formada por tantas listas como nodos tenga G.

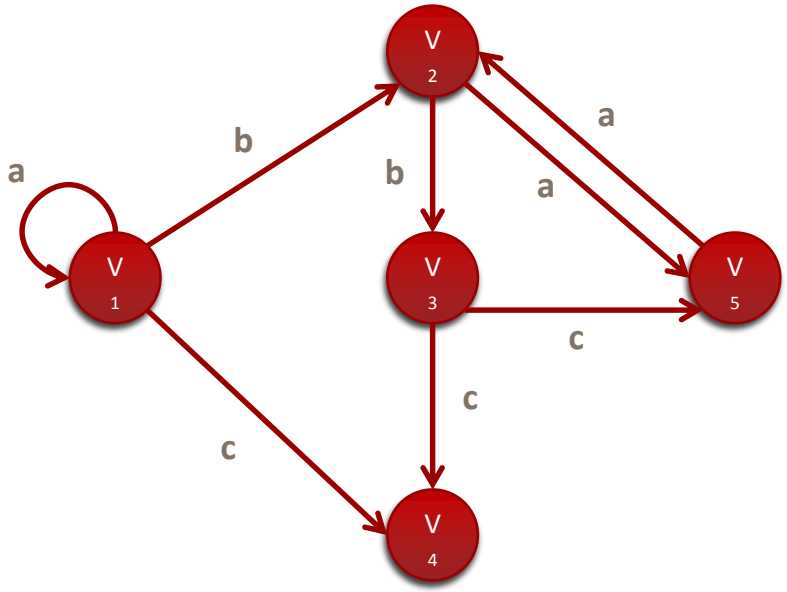
Esta disposición es recomendable cuando el número de aristas es menor a n^2. Sin embargo, el acceso a las aristas de cada nodo puede llegar a ser lento.


A
B
D
C


A
C
D
B
A
C
B
D
C
A

Ejemplo 1

Sea G = {V, A}:



	V_1	V_2	V_3	V_4	V_5
Internos	1	2	1	2	2
Externos	3	2	2	0	1

Ejemplo 1

La matríz de adyacencia está dada por:

$$M[u, v] = \begin{cases} 1 \text{ si existe un arco } (u, v) \\ 0 \text{ en caso contrario} \end{cases}$$

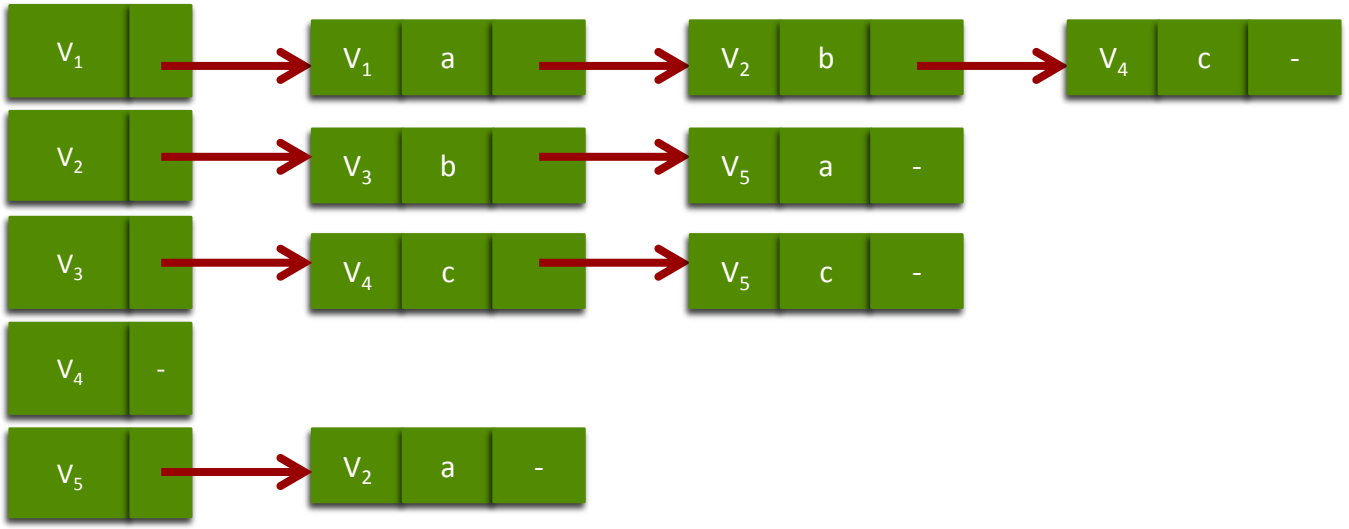
	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	1	0
V_2	0	0	1	0	1
V_3	0	0	0	1	1
V_4	0	0	0	0	0
V_5	0	1	0	0	0

	V_1	V_2	V_3	V_4	V_5
V_1	a	b	-	c	-
V_2	-	-	b	-	a
V_3	-	-	-	c	c
V_4	-	-	-	-	-
V_5	-	a	-	-	-

Ejemplo 1

La lista de adyacencia está definida por:

$$L_v = \{ v \in V \mid (u, v) \in A \}$$

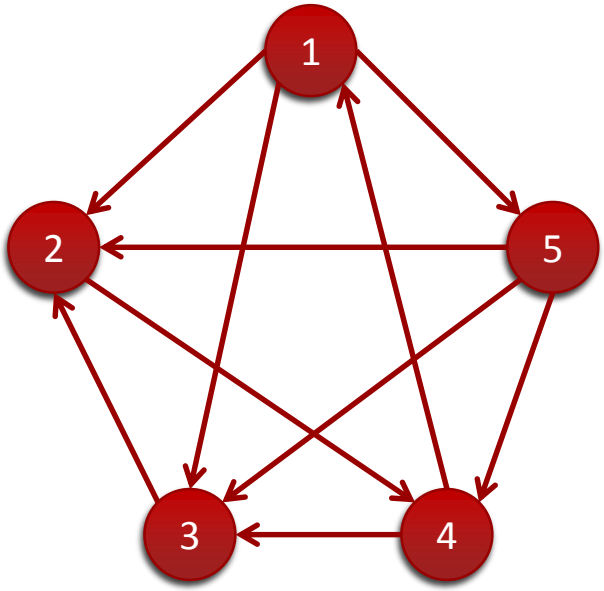


Ejemplo 2

Una aplicación de las digráficas se puede realizar en un torneo cualquiera.

Supóngase que se enfrentan 5 jugadores de tennis en un torneo “Round-Robin”. Los resultados de los torneos (sin tener en cuenta los marcadores) se pueden representar en una digráfica, donde cada arista dirigida (u, v) simboliza que el jugador u venció al jugador v.

Ejemplo 2



M(u, v) =

	1	2	3	4	5	
1	0	1	1	0	1	← 1er lugar
2	0	0	0	1	0	← 3er lugar
3	0	1	0	0	0	← 3er lugar
4	1	0	1	0	0	← 2do lugar
5	0	1	1	1	0	← 1er lugar

Ejemplo 2

De la matriz adyacente anterior se sabe que 1 y 5 quedaron en primer lugar, sin embargo, 1 podría argumentar que venció a 5 y, por tanto, merece ser el ganador.

Así mismo, los jugadores 2 y 3 quedaron en segundo lugar, pero el jugador 3 podría argumentar que debería ser el segundo lugar único dado que venció a 2. Empero, 2 podría argumentar que tiene dos victorias indirectas dado que venció a 4 que, a su vez, derrotó a 3 y a 1 (1er lugar), además de que 3 solo tiene un victoria indirecta y, por tanto, debería quedar en segundo lugar.

Ejemplo 2

Una manera de solucionar la disputa es realizando las sumas de las victorias directas e indirectas, para ello se obtiene $M^2(u,v)$:

$$M^2(u,v) = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 1 & 2 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Ejemplo 2

Si ahora se realiza la suma de las victorias directas con las victorias indirectas se obtiene:

$$\begin{pmatrix} 3 \\ 1 \\ 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 5 \\ 2 \\ 1 \\ 4 \\ 4 \end{pmatrix} = \begin{pmatrix} 8 \\ 3 \\ 2 \\ 6 \\ 7 \end{pmatrix}$$

← 1er lugar
← 4er lugar
← 5er lugar
← 3do lugar
← 5er lugar

En este caso, las victorias indirectas resuelven el empate, aunque no siempre es así. Además, no se tomaron en cuenta los marcadores, lo cual podría ser un criterio de desempate.

Gráficas no dirigidas

En la gráficas no dirigidas (o simplemente gráficas) las aristas son pares no ordenados de vértices, esto es, si existe un costo en el camino de i a j, existe un costo similar en el camino de j a i.

Una gráfica no dirigida G = (V, A) está formado por dos conjuntos finitos, uno de vértices V y otro de aristas A, donde cada arista en A es un par no ordenado de vértices, es decir, si (u, v) es una arista no dirigida, entonces (u, v) = (v, u).

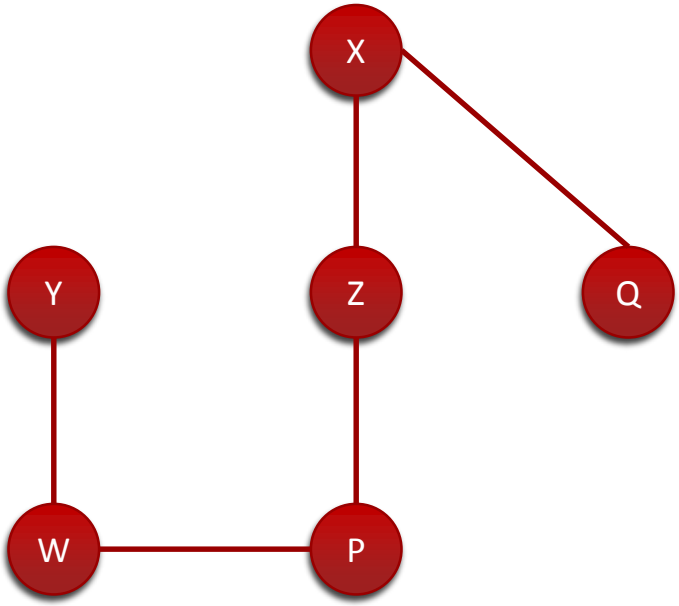
Una gráfica no dirigida se puede representar mediante una matriz de adyacencia o mediante una lista de adyacencia (de manera similar a las gráficas dirigidas).

Debido a la relación entre los nodos, cada arista no dirigida entre u y v se debe cambiar por dos aristas dirigidas (de u→v y de v→u).

Por ende, en este tipo de gráficas la matriz de adyacencia será simétrica y en la lista de adyacencia el vértice u estará en la lista de adyacecia del vértice v y viceversa.

Ejemplo 3

Dada la siguiente gráfica:



Ejemplo 3

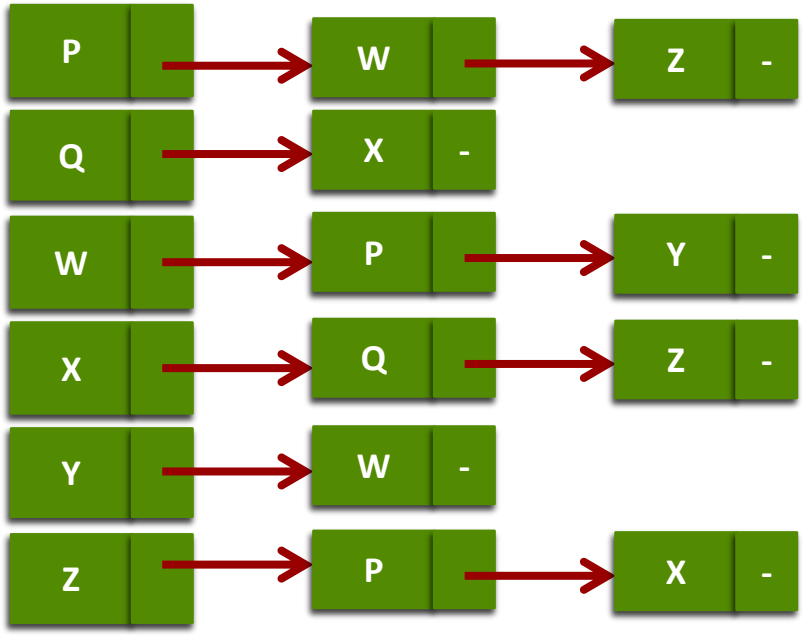
La matriz de adyacencia de la gráfica anterior es:

	P	Q	W	X	Y	Z
P	0	0	1	0	0	1
Q	0	0	0	1	0	0
W	1	0	0	0	1	0
X	0	1	0	0	0	1
Y	0	0	1	0	0	0
Z	1	0	0	1	0	0

Nótese que la gráfica es simétrica, el triángulo superior es igual al triángulo inferior, y la diagonal principal está formada por ceros.

Ejemplo 3

La gráfica también se puede construir con una lista de adyacencia:



Ejemplo 4

En la práctica, las gráficas se utilizan para modelar problemas. Si se considera una gráfica donde los vértices son personas y existe una arista entre dos personas si y solo si son amigos. Este tipo de gráficas se denominan redes sociales y están definidas en cualquier conjunto de personas.

En los últimos años ha surgido una ciencia completa que se encarga del estudio de las redes sociales debido a que las personas y sus comportamientos son mejor entendidos como propiedas del grafo.

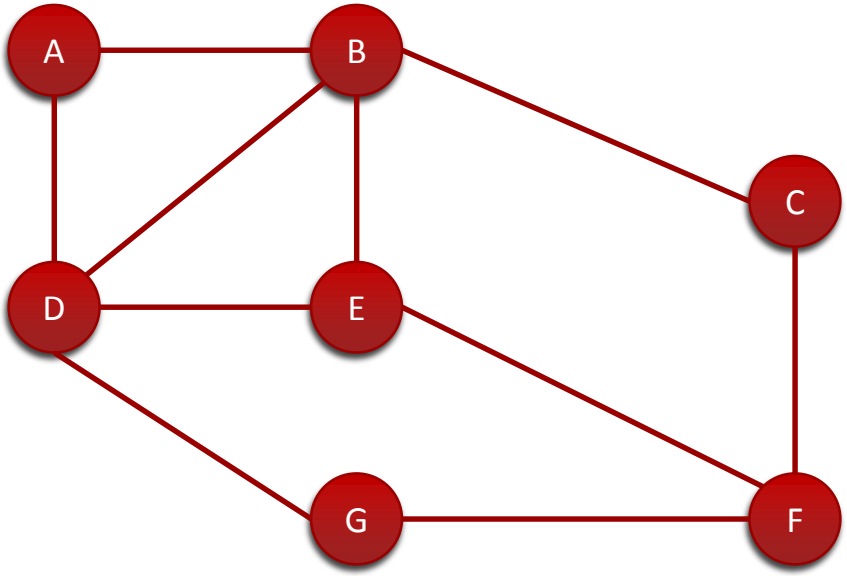
Ejemplo 4

Dentro de una grafica de red social se pueden realizar varias preguntas:

- ¿Si soy amigo de X, significa que X es amigo mio?
- ¿Qué tan cercano es tu amigo?
- ¿Soy amigo mio?
- ¿Quién tiene más amigos?
- ¿Mis amigos viven cerca de mi?
- ¿Eres un individuo o una multitud sin rostro?

Ejemplo 4

Suponiendo que se posee la siguiente red social, se desea visitar todos los nodos sin repetir aristas, es decir, solo se puede pasar una vez por cada arista, ¿es posible realizar el recorrido iniciando en el nodo F?



Tarea 1

Realizar un programa que permita guardar las relaciones de una gráfica. La gráfica puede ser dirigida o no dirigida. Así mismo, la gráfica puede ser etiquetada o no etiquetada.

La implementación del grafo se debe realizar tanto en matriz adyacente como en lista adyacente.

En la práctica existen varios tipos de gráficas: anillo, cadena, estrella, hipercubos, etc. Empero, la estructura más representativa de los grafos son los árboles.

4.2 Árboles.

4.2 Árboles.

Los árboles son estructuras de datos no lineales y dinámicas. Son un tipo de estructuras muy utilizadas en la rama de las tecnologías de la información.

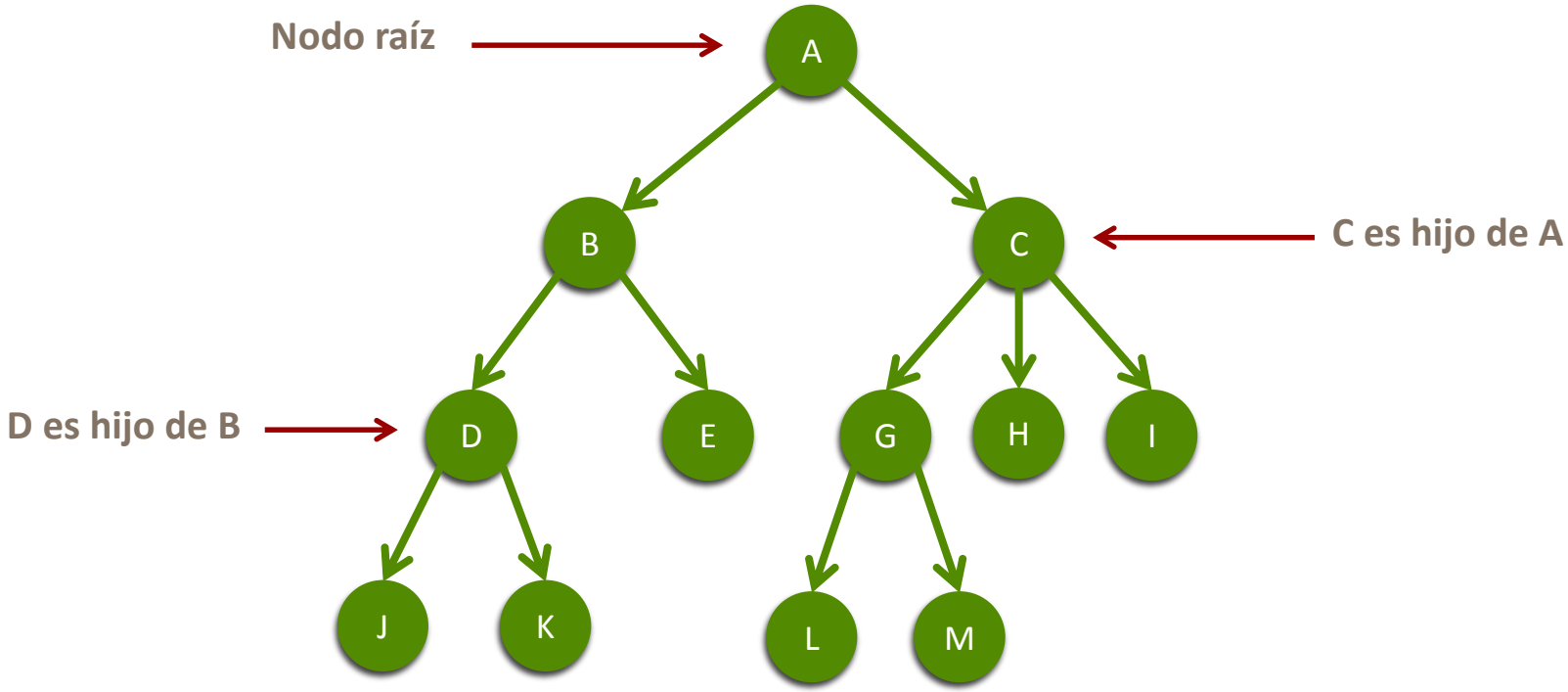
Un árbol es una estructura jerárquica aplicada sobre una colección de elementos u objetos llamados nodos. El primer nodo de un árbol se conoce como raíz y a partir de él se dibuja la jerarquía.

Algunas carácterísticas y propiedades que poseen las estructuras árbol son:

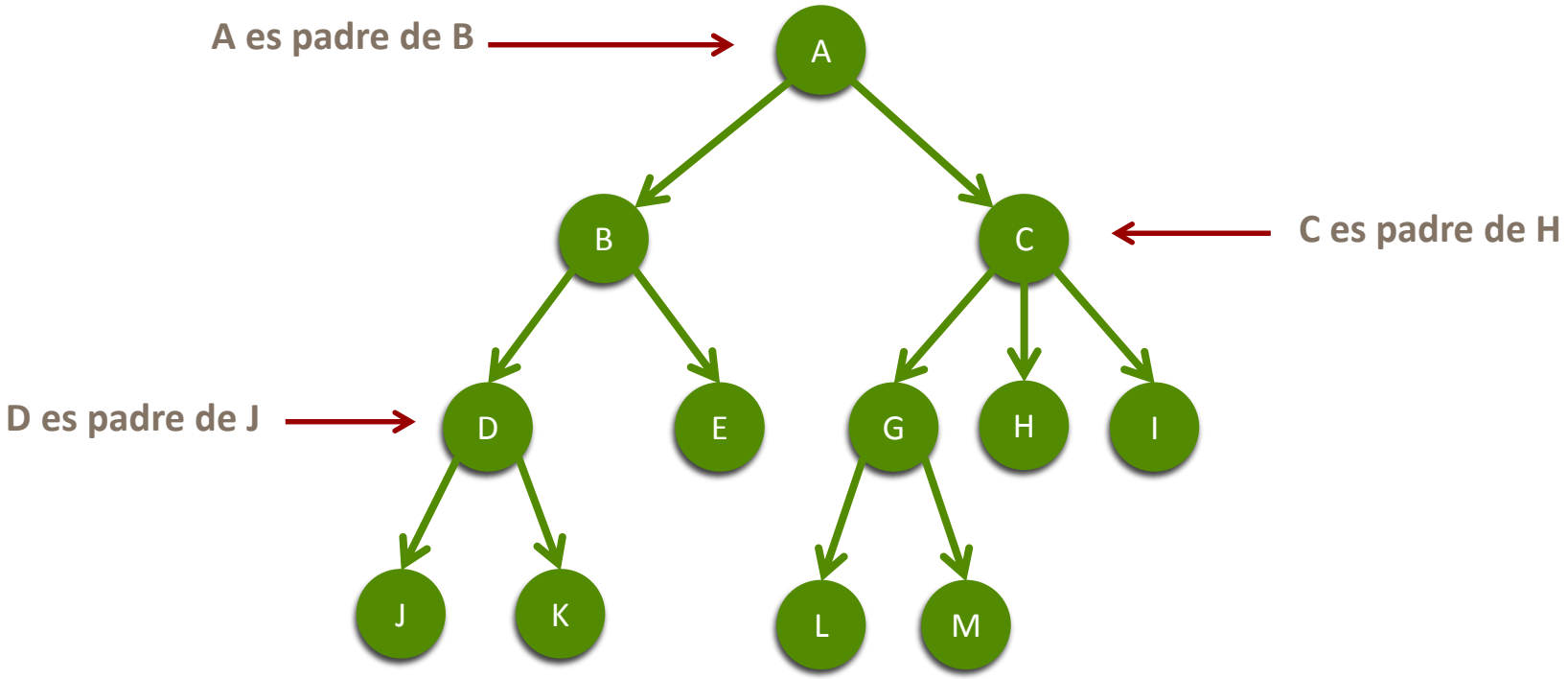
- Todo árbol que no es vacío posee un único nodo raíz.
- Un nodo X es descendente directo de un nodo Y si y solo si el nodo X es apuntado por el nodo Y (X es hijo de Y).
- Un nodo Y es antecesor directo de un nodo X si y solo si el nodo Y apunta al nodo X (Y es padre de X).
- Si varios nodos descienden de manera directa de un mismo nodo, entonces se dice que éstos son hermanos.
- Si un nodo no posee ramificaciones (o hijos) se denomina nodo terminal o nodo hoja.

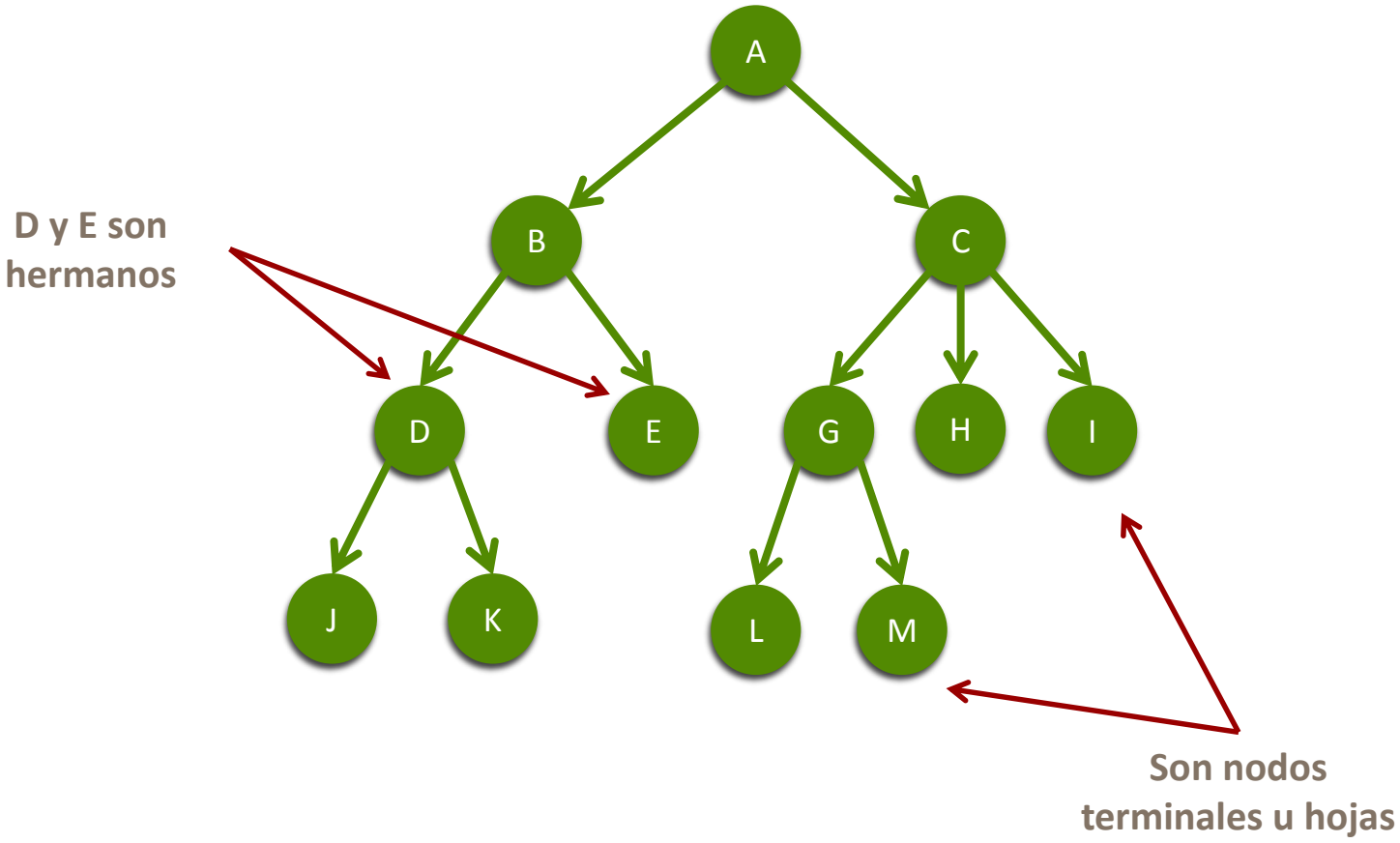
- Cualquier nodo diferente del nodo raíz o del nodo hoja se le conoce como nodo interior.
- El grado de un nodo está determinado por el número de descendentes directos.
- El grado del árbol está determinado por el máximo grado de todos los nodos.
- El nivel de un nodo está determinado por el número de arcos que deben ser recorridos para llegar a dicho nodo.
- La altura de un árbol está determinada por el número máximo de niveles de todos los nodos del árbol.

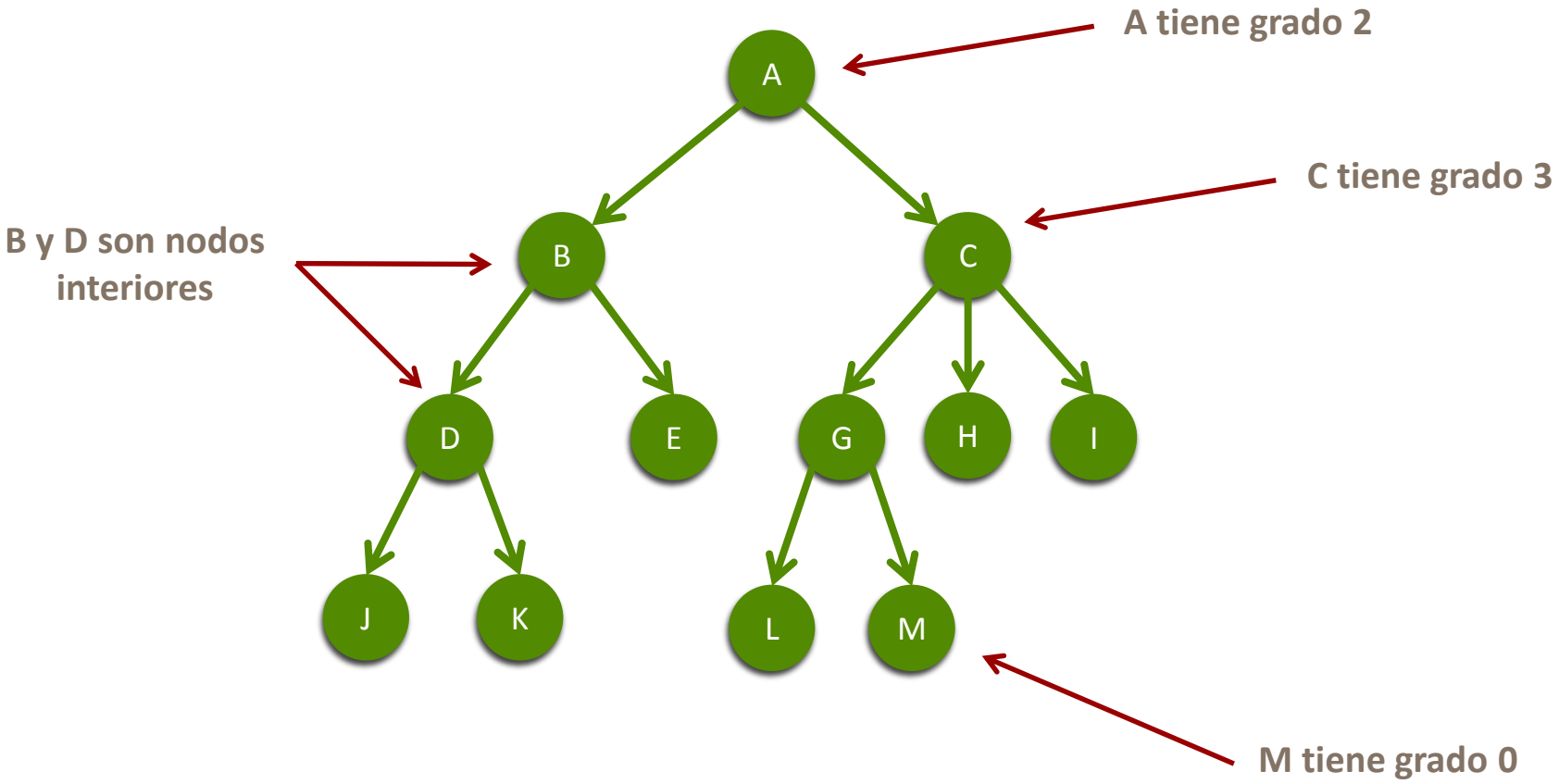
Una estructura árbol se puede representar mediante un grafo de la siguiente manera:



árbol ≠ ⊘

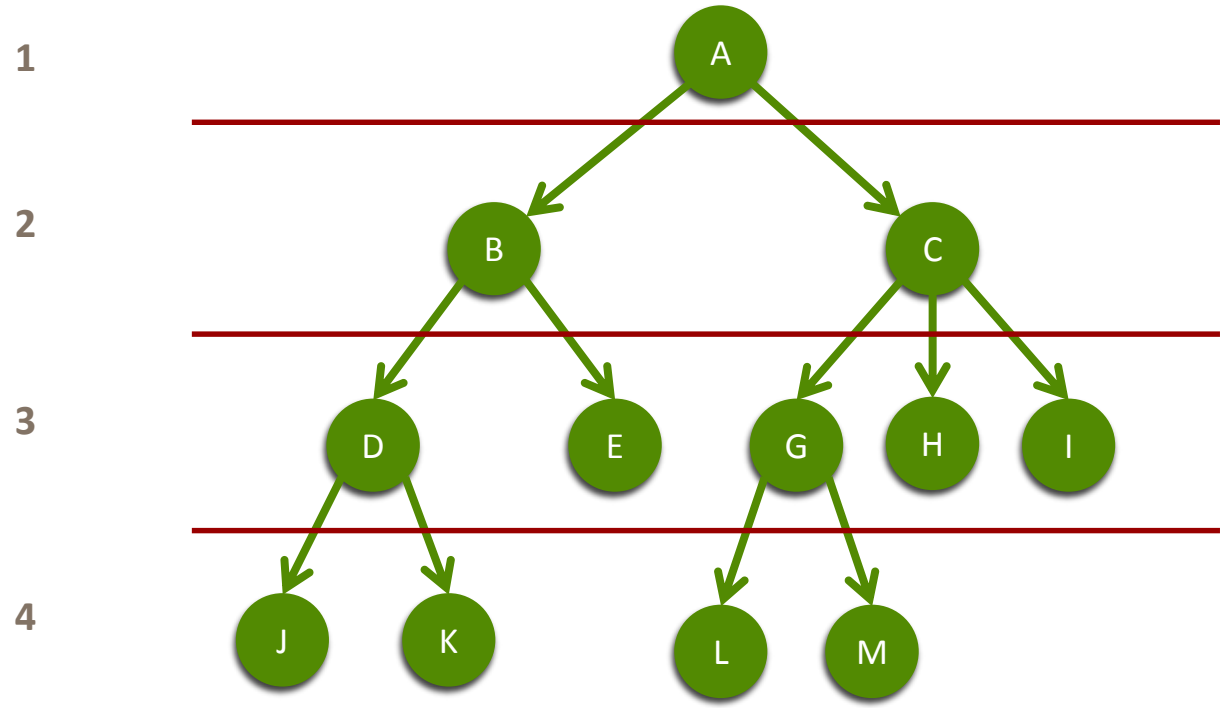

A es padre de B
A
B
C
C es padre de H
D es padre de J
D
E
G
H
I
J
K
L
M


D y E son hermanos
A
B
C
D
E
G
H
I
J
K
L
M
Son nodos terminales u hojas


A tiene grado 2
C tiene grado 3
B y D son nodos interiores
A
B
C
D
E
G
H
I
J
K
L
M
M tiene grado 0

Nivel

El grado del árbol es de 3.



La altura del árbol es de 4.

Por tanto, un árbol es una gráfica G={V, A} en donde el número de nodos es igual al número de aristas más uno:

$$V = A+1$$

Un árbol es una estructura de datos recursiva, es decir, es un conjunto de uno o más nodos en el que hay un nodo especial (el nodo raíz) y los otros nodos son subconjuntos disjuntos T_1, T_2, T_3, ..., T_n ($n \geq 0$), cada uno de los cuales es un árbol y, por ende, cada T_i es un subárbol de la raíz.

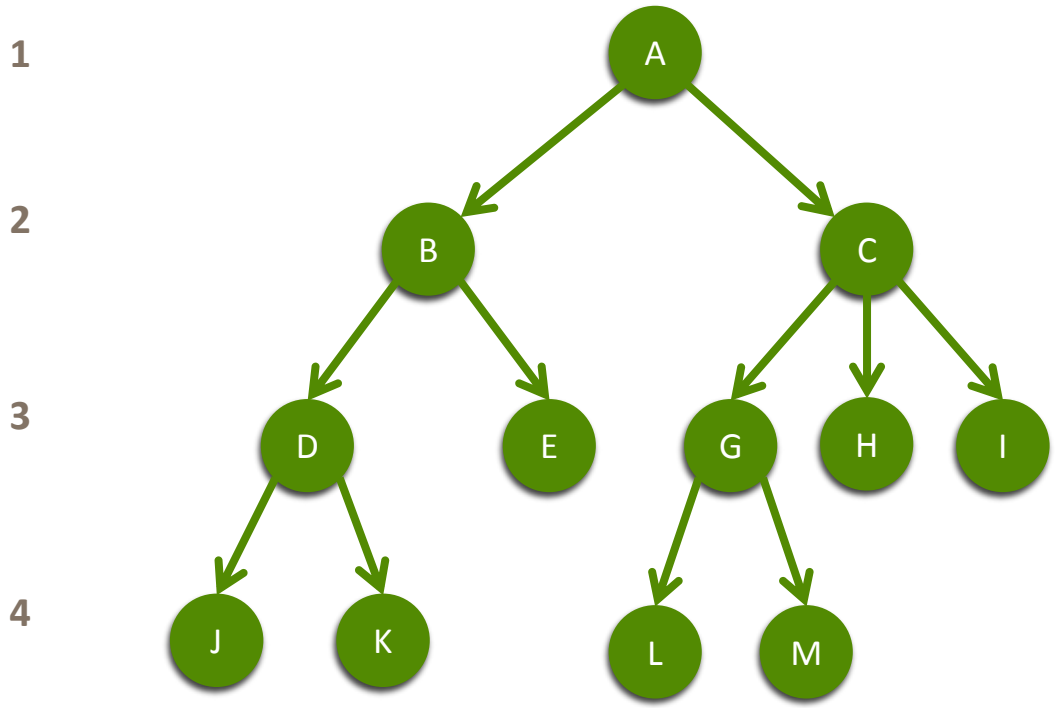
¿Grafo o árbol?

Para verificar si una gráfica es un árbol se toman en cuenta los siguientes aspectos:

- El número de nodos es igual al número de arcos más uno (n = a+1).
- Cualquier nodo diferente de raíz es de grado interno 1.
- No existen ciclos.
- Las trayectorias son simples.
- Entre dos nodos cualquiera solo hay una trayectoria.
- Si se retira cualquier arco de la gráfica, se desconecta una parte de ella.

Longitud de camino

La longitud de camino de un nodo x está definida como el número de arcos que se deben recorrer para llegar desde la raíz hasta dicho nodo.



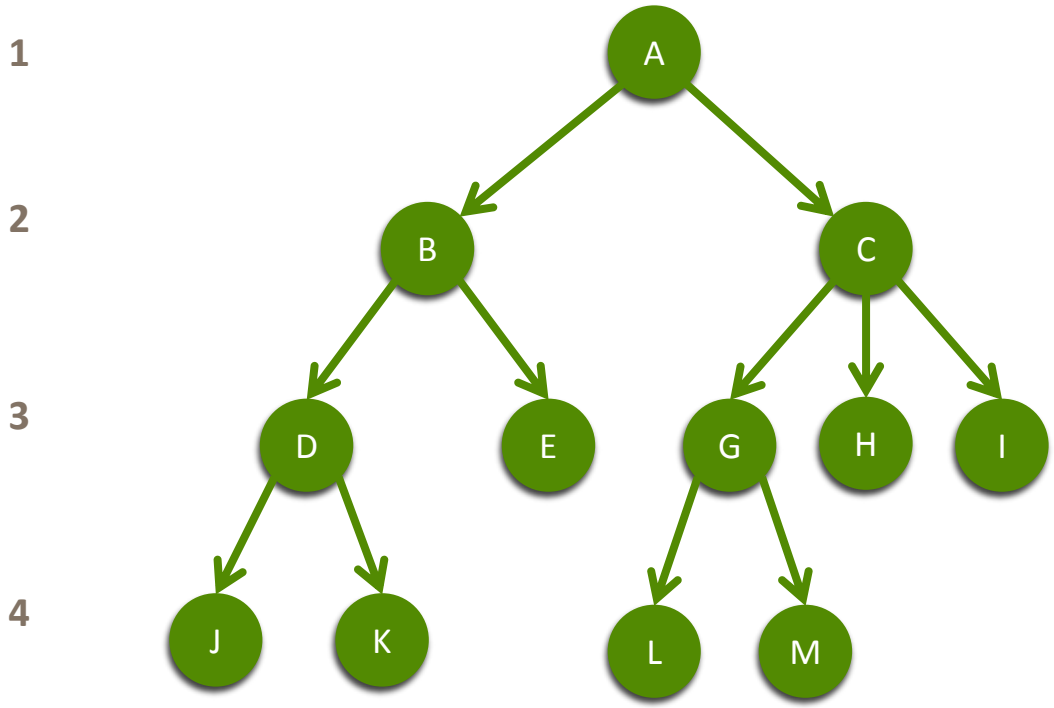
Longitud de camino interno

La longitud de camino interno (LCI) está dado por la suma de las longitudes de camino de todos los nodos del árbol. Se calcula de la siguiente manera:

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

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

Ejemplo 5



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

Si a la longitud de camino interno se le divide entre el número de nodos del árbol, se obtiene la media de la longitud de camino interno. Esta medida permite conocer el promedio del número de máximo de decisiones que se deben tomar para llegar a un determinado nodo partiendo desde la raíz:

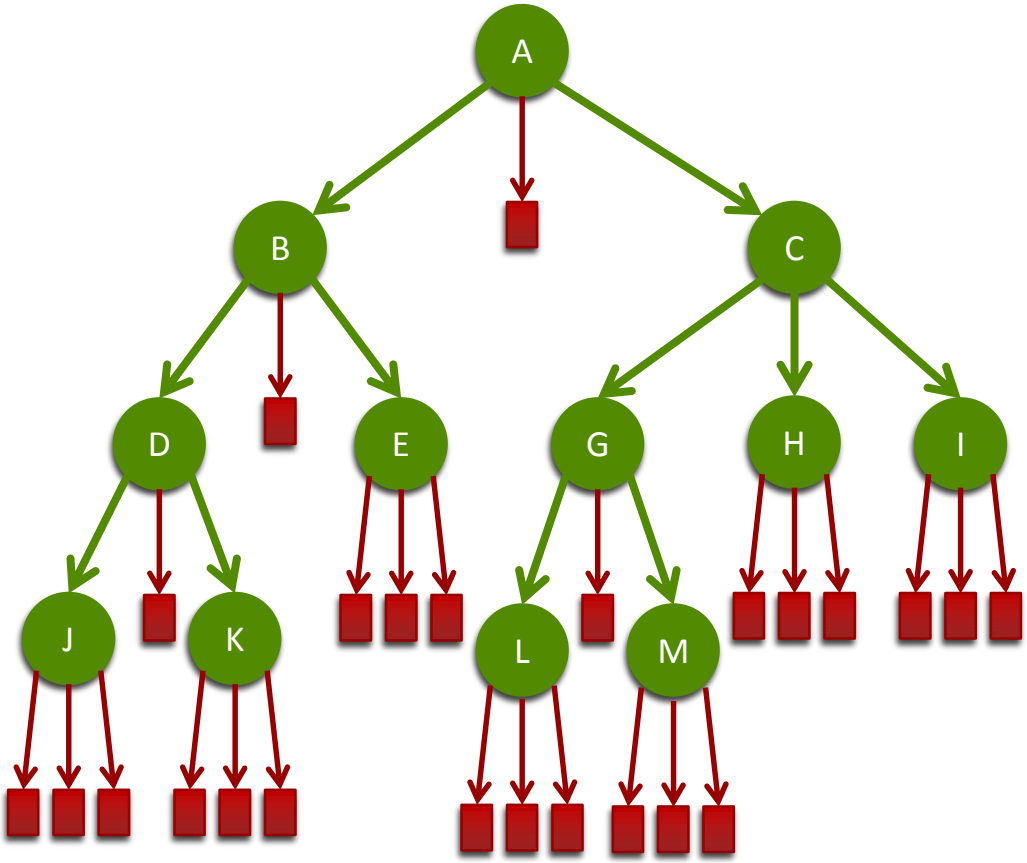
$$LCIM = LCI / n$$
$$LCIM = 36 / 12 = 3$$

Longitud de camino externo

Un árbol donde el número de hijos de cada nodo es igual al grado del árbol se conoce como árbol extendido. Si el árbol no cumple con la condición anterior, se pueden incorporar el número de nodos especiales necesarios para hacerla cumplir.

Los nodos especiales completan la rama vacía (o nula), no pueden tener descendencia y, normalmente, se representan con un cuadrado.

Ejemplo 6



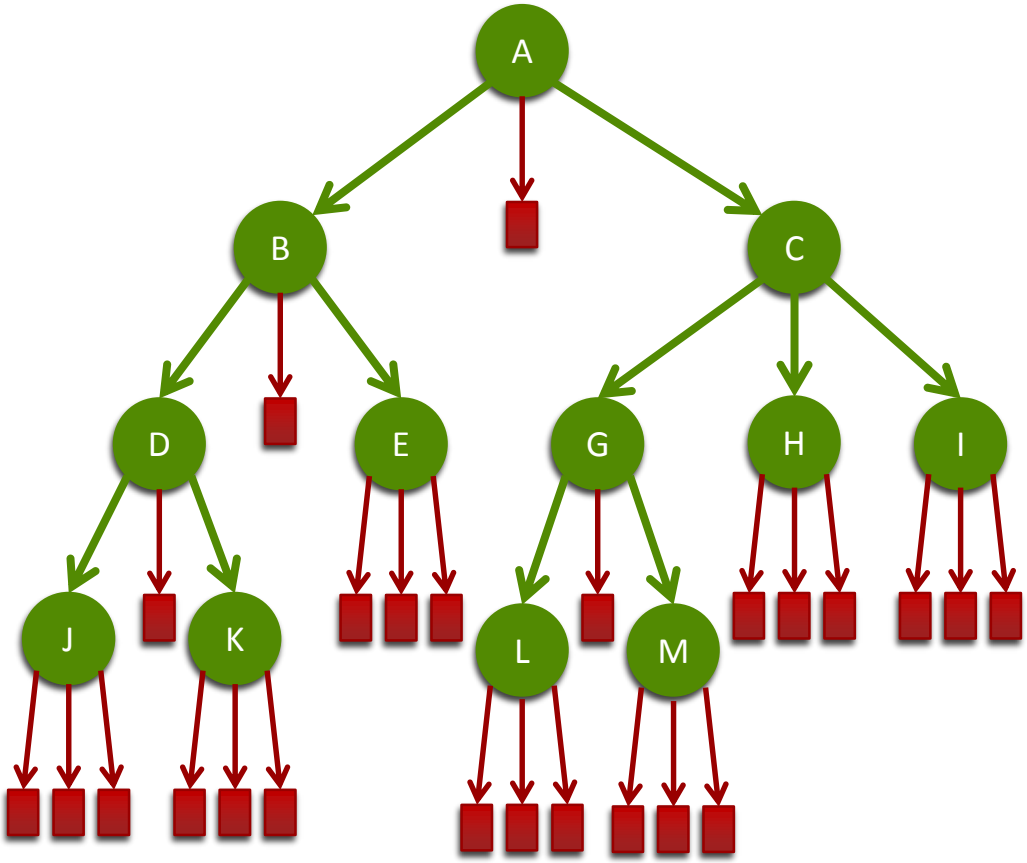
El número de nodos especiales del árbol es de 25

La longitud de camino externo (LCE) de un árbol está dada por las longitudes de camino de todos los nodos especiales del árbol, es decir:

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

donde i representa el nivel del árbol, h su altura y ne_i el número de nodos especial en el nivel i. La fórmula inicia en dos porque a la altura del nodo raíz no puede haber nodos especiales.

Ejemplo 7

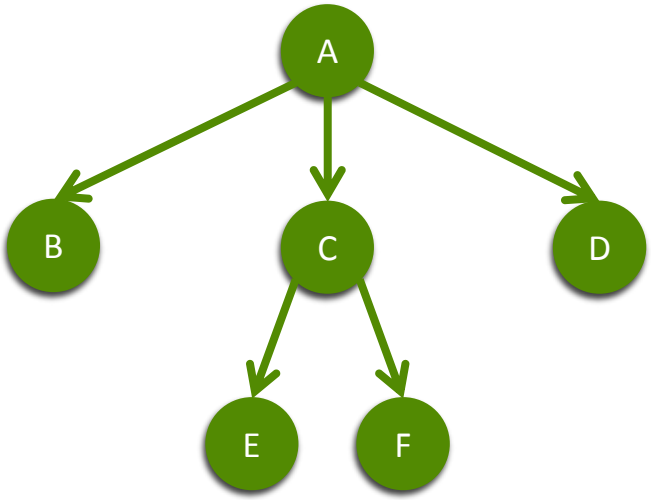


LCE = 1*2 + 1*3 + 11*4 + 12*5 = 109

Si a la longitud de camino externo se le divide entre el número de nodos especiales del árbol, se obtiene la media de la longitud de camino externo. Esta medida permite conocer el promedio del número de arcos que se deben recorrer para llegar a un nodo especial partiendo desde la raíz:

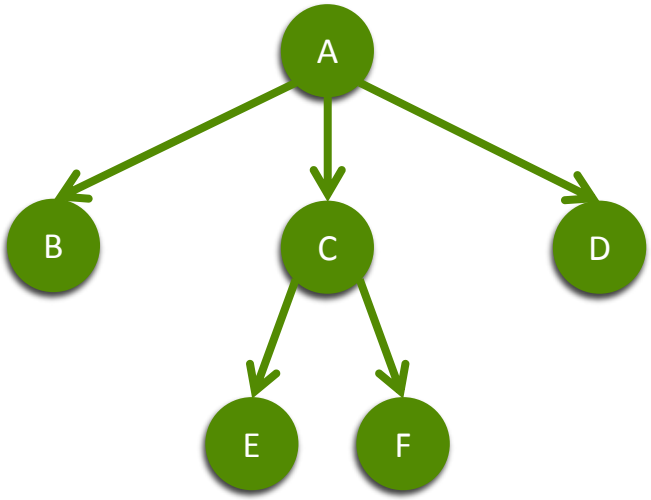
$$LCEM = LCE / n$$
$$LCEM = 110 / 25 = 4.4$$

Debido a que un árbol es un tipo especial de gráfica, se puede representar mediante una matriz de adyacencia de forma idéntica a una gráfica dirigida. Para identificar al nodo raíz se pueden utilizar los elementos de la diagonal principal.



	A	B	C	D	E	F
A	*	1	1	1	0	0
B	0	0	0	0	0	0
C	0	0	0	0	1	1
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Otra manera de representar un árbol es mediante un arreglo bidimensional, donde cada elemento representa un nodo y el contenido representa a su antecesor directo.



A	B	C	D	E	F
*	A	A	A	C	C

Tarea 2

Realizar un programa que permita guardar una estructura de datos tipo árbol en un arreglo unidimensional.

4.3 Árboles binarios.

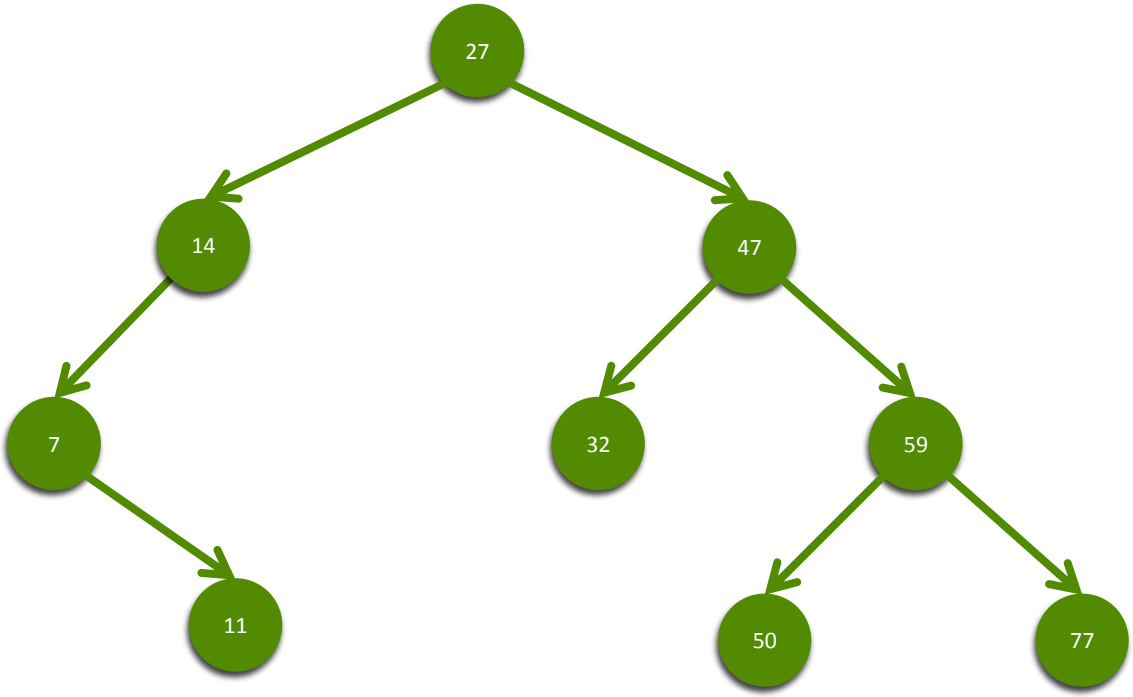
4.3 Árboles binarios.

Un tipo de árbol de gran importancia dentro de las ciencias de la computación es el árbol binario.

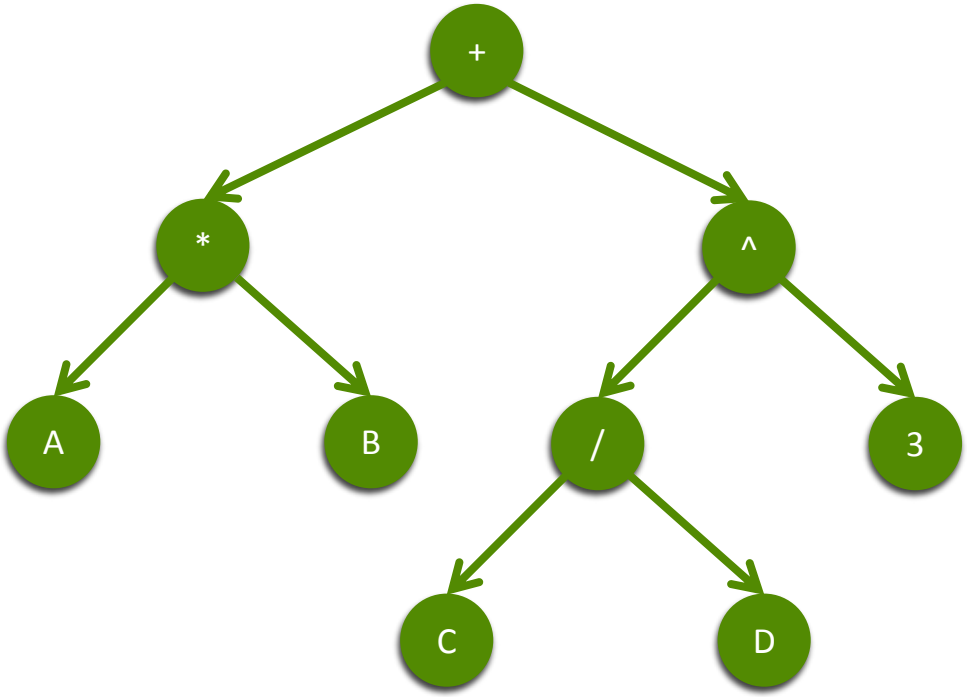
Un árbol ordenado es aquel en el cual la distribución de sus ramas sigue una lógica. Los árboles ordenados de grado 2 permiten representar la información relacionada con la solución de muchos problemas. A este tipo de grafos se les conoce como árboles binarios.

En un árbol binario cada nodo puede tener como máximo dos subárboles (el izquierdo y el derecho).

Por tanto, un árbol binario T es una estructura de datos homogenea, resultado de la unión de un elemento tipo T (llamado raíz) con dos árboles binarios disjuntos (subárbol izquierdo y subárbol derecho).

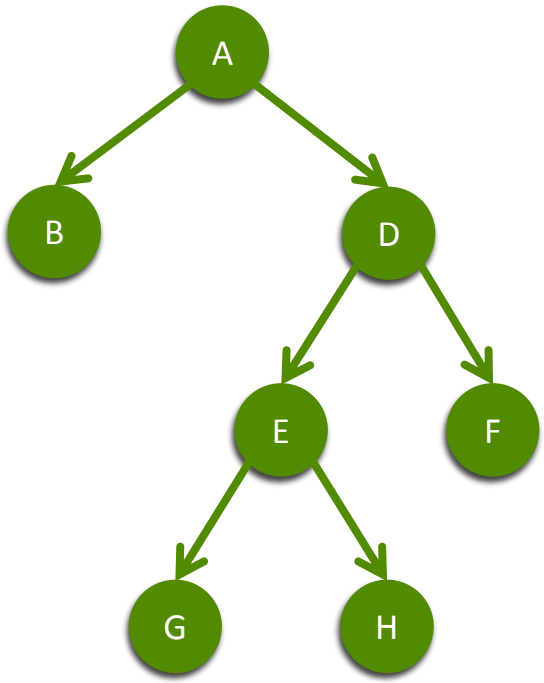


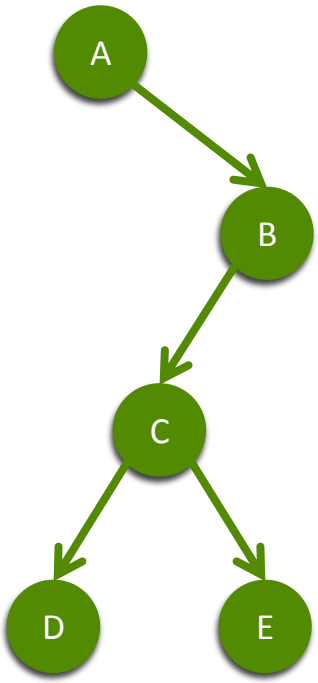
Árbol binario de búsqueda



Expresión algebráica en forma de árbol binario

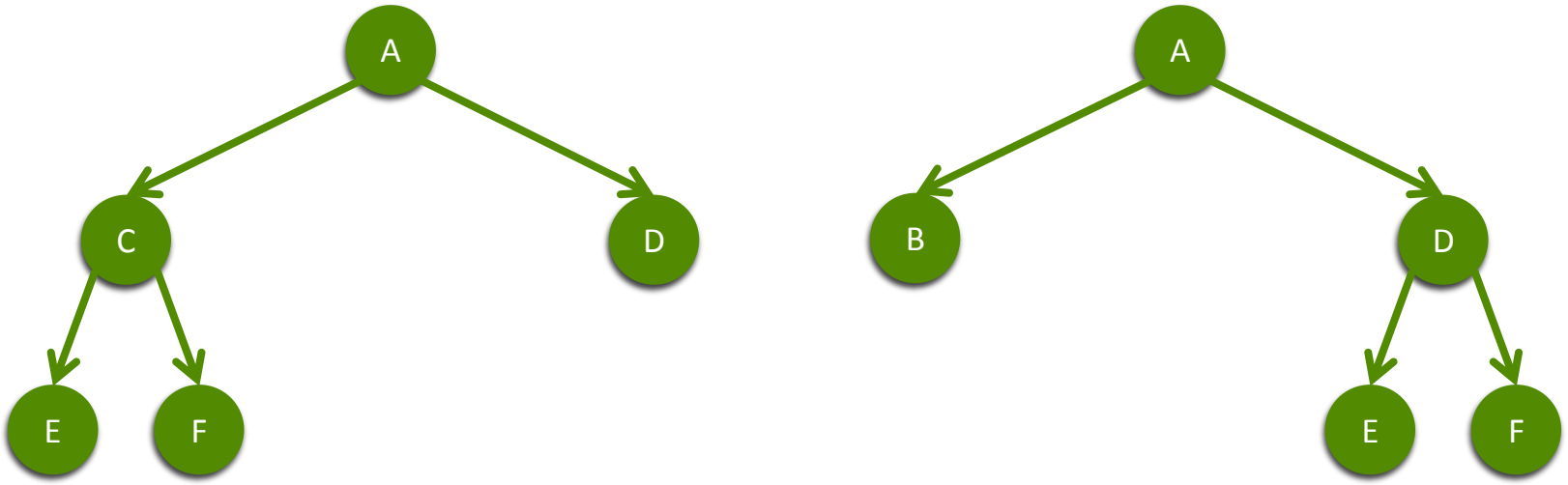
Cuando un árbol tiene cero o dos subárboles el árbol se dice que es un árbol estríctamente binario. Cuando un árbol tiene un subárbol se dice que es un árbol de Knuth.



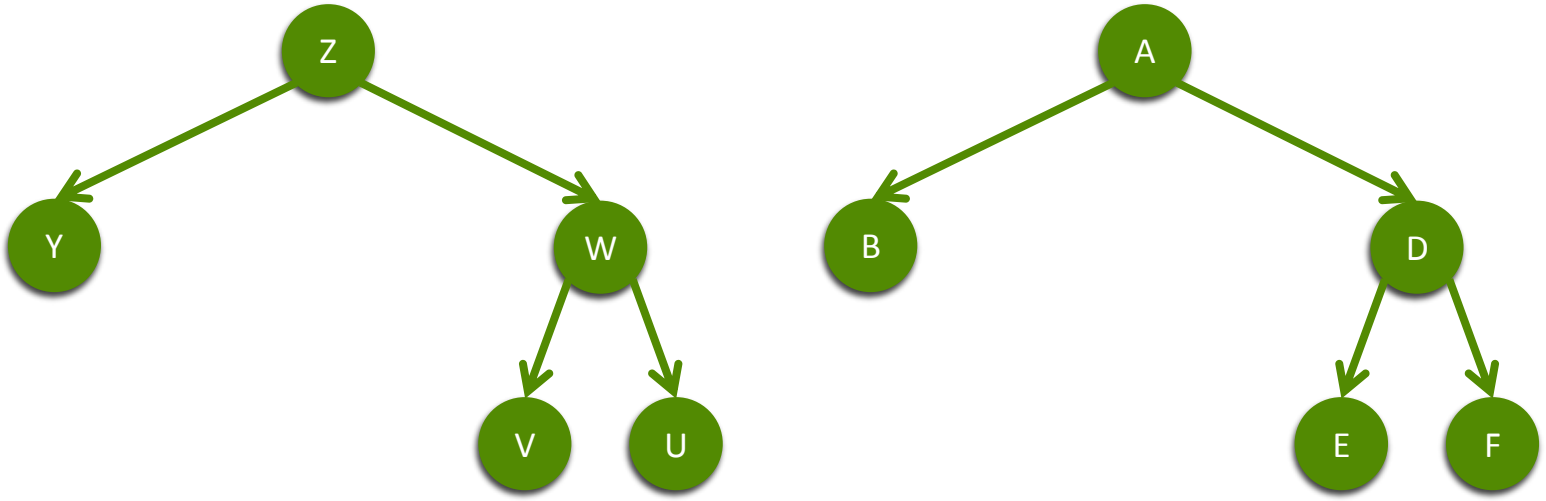


Un árbol estrictamente binario está balanceado cuando el número de nodos terminales es igual a 2^m y la longitud de cualquier trayectoria a la raíz de cualquier nodo terminal es m, donde m es el grado del árbol.

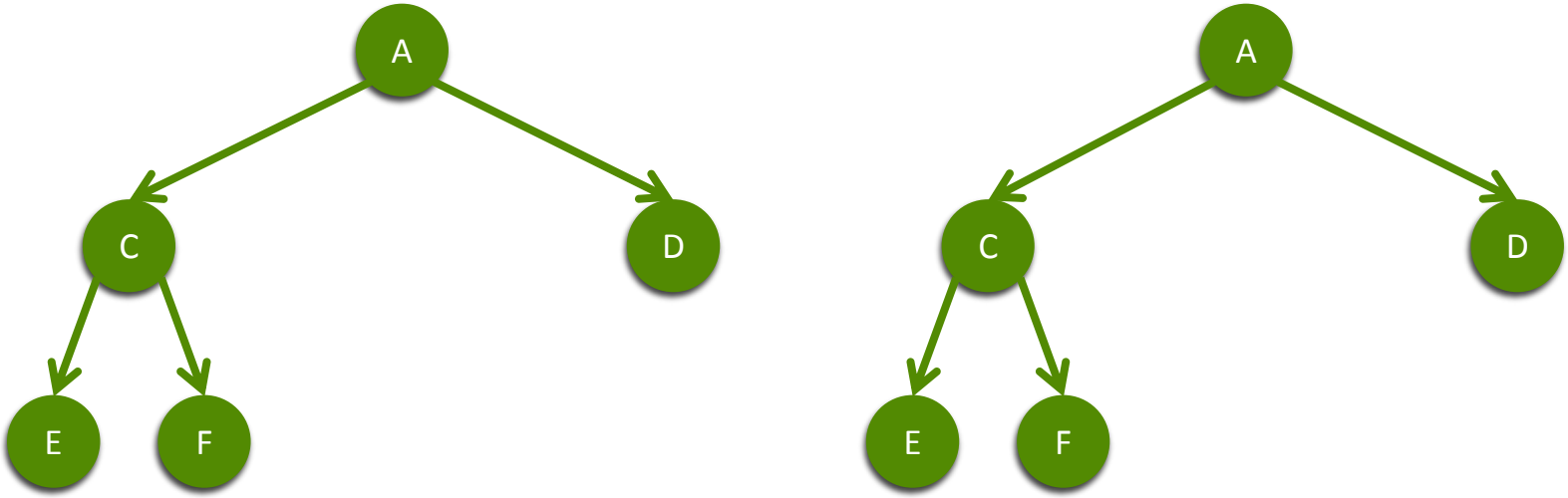
Árboles binarios distintos: Dos árboles binarios son distintos cuando sus estructuras (nodos o arcos) son diferentes.



Árboles binarios similares: Dos árboles binarios son similares cuando sus estructuras (nodos o arcos) son idénticas.

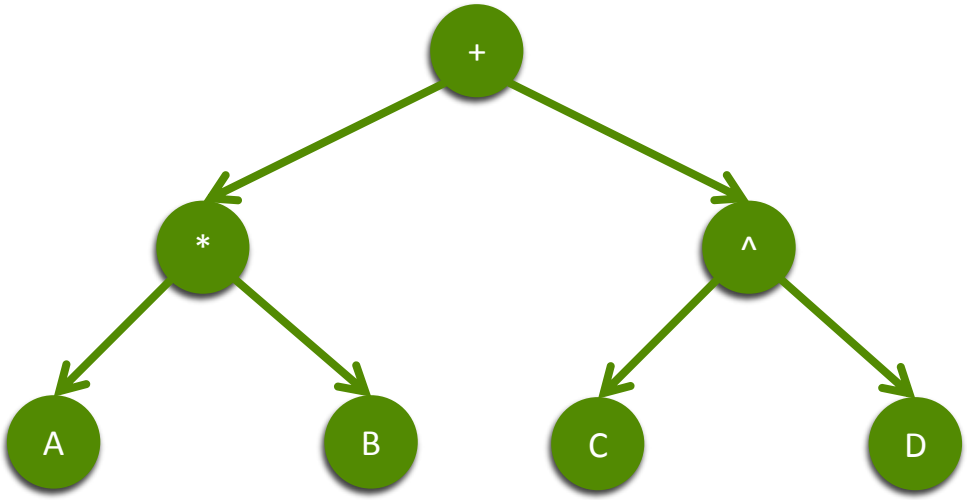


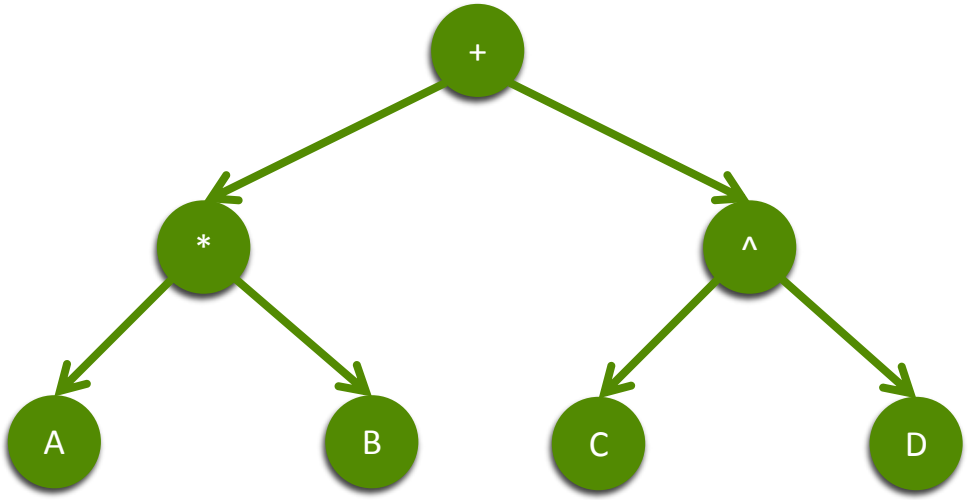
Árboles binarios equivalentes: Dos árboles binarios si son similares y además sus nodos contienen la misma información.



Árbol binario completo

Un árbol binario completo (ABC) es aquel en el que todos los nodos (excepto los del último nivel) tiene dos hijos, es decir, el subárbol derecho y el subárbol izquierdo son similares.





El número de nodos de un árbol binario completo de altura h se pude calcular de la siguiente manera:

$$\#\ \text{nodos (ABC)} = 2^h - 1$$

$$\#\ \text{nodos (ABC)} = 2^3 - 1 = 7$$

Transformación de árboles a árboles binarios

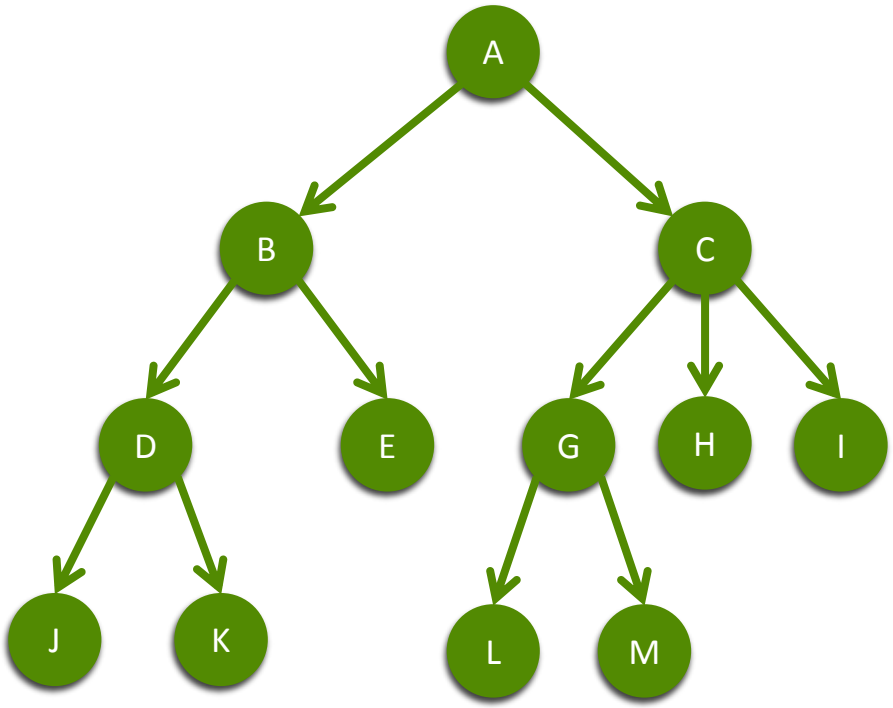
Como ya se mencionó, los árboles binarios son muy útiles en los sistemas computacionales. Por tanto, es muy útil convertir un árbol general en un árbol binario.

Las operaciones que se deben aplicar para convertir un árbol en un árbol binario son:

- Unir los hijos de cada nodo en forma horizontal (hermanos).
- Relacionar verticalmente el nodo padre con el hijo que se encuentra más a la izquierda. Además de eliminar el vínculo de ese padre con el resto de sus hijos.
- Acomodar (girar los nodos) el diagrama resultante, obteniendo así el árbol binario correspondiente.

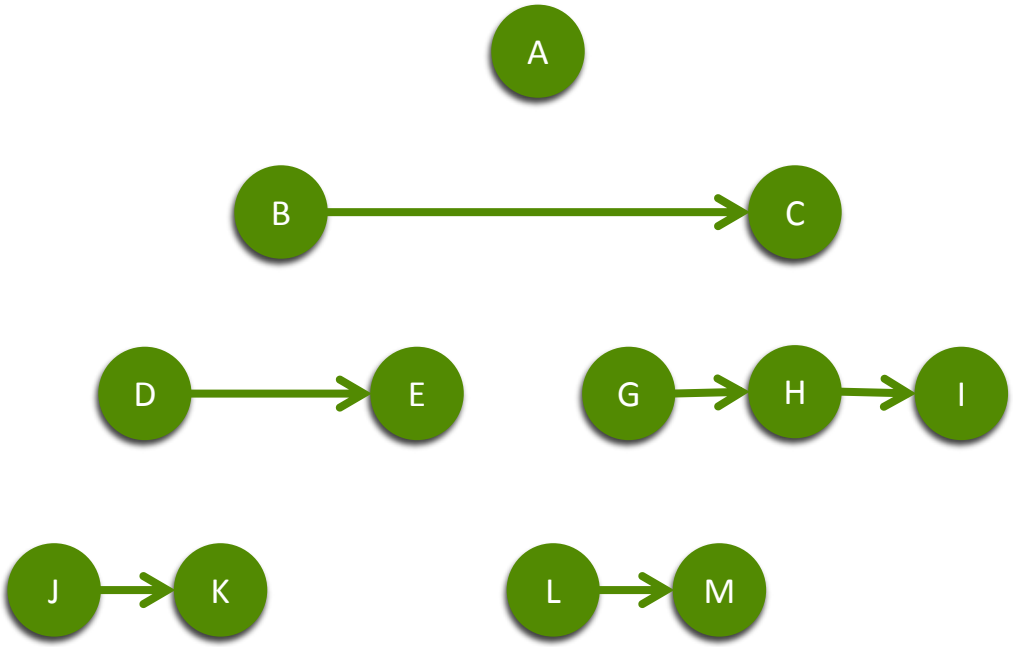
Ejemplo 8

Convertir la siguiente estructura de árbol en un árbol binario.



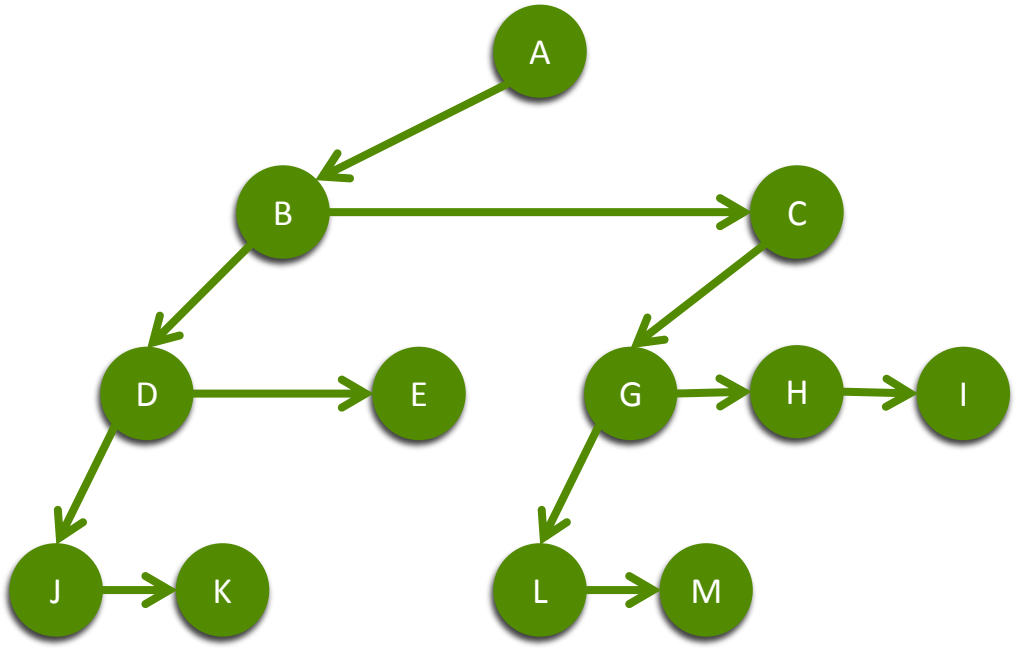
Ejemplo 8

Lo primero que hay que hacer es unir los hijos de cada nodo en forma horizontal (hermanos).



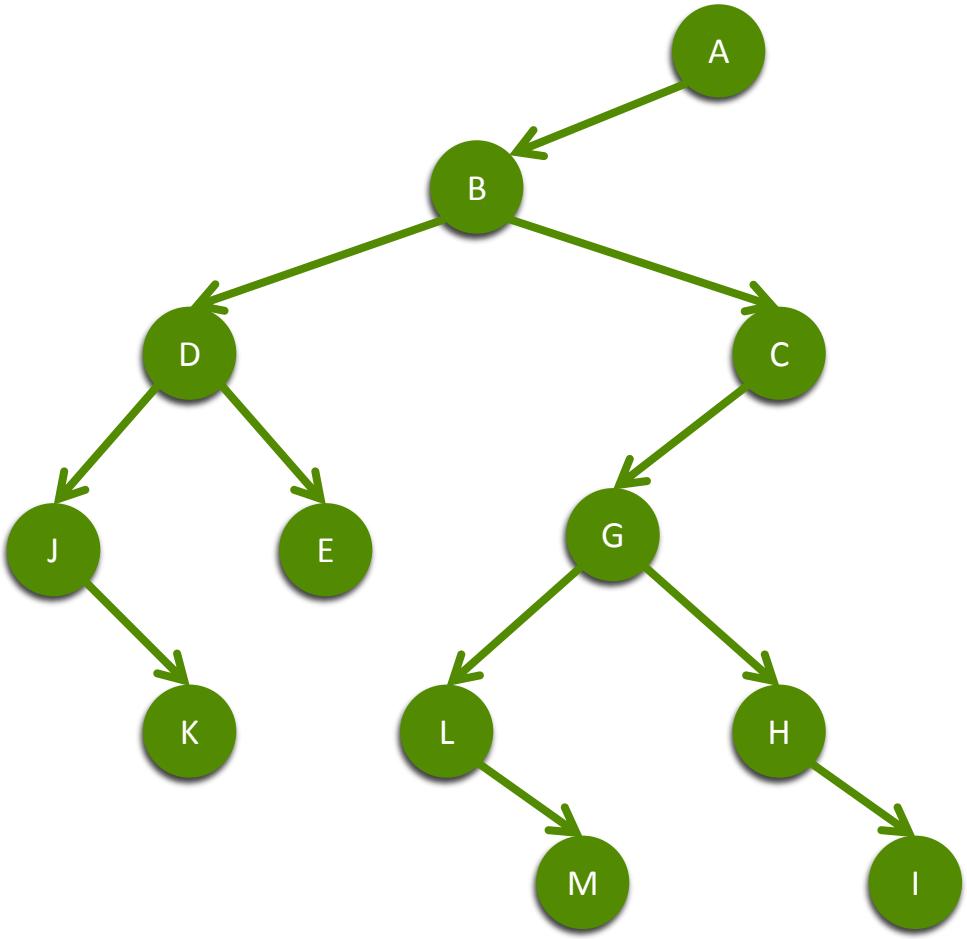
Ejemplo 8

Ahora hay que relacionar verticalmente el nodo padre con el hijo que se encuentra más a la izquierda. Además de eliminar el vínculo de ese padre con el resto de sus hijos.



Ejemplo 8

Acomodar (girar los nodos) el diagrama resultante, obteniendo así el árbol binario correspondiente.



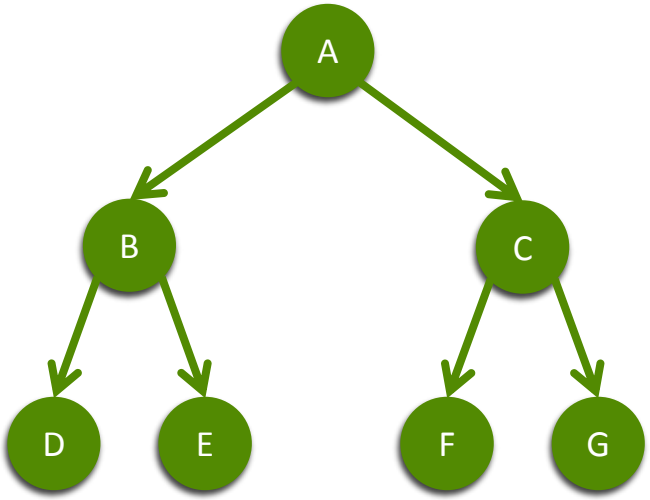
Representación de árboles binarios en la computadora

De la misma manera que un grafo o que un árbol, un árbol binario se puede representar mediante una matriz de adyacecia o mediante una lista de adyacencia.

También se puede representar utilizando un arreglo donde los hijos de cualquier nodo k se encuentran en el elemento 2*k (nodo izquierdo) y 2*k + 1 (nodo derecho). La longitud del arreglo es de 2^{n-1}, donde n es el número de niveles del árbol.

Ejemplo 9

Representación de un árbol binario en un arreglo:

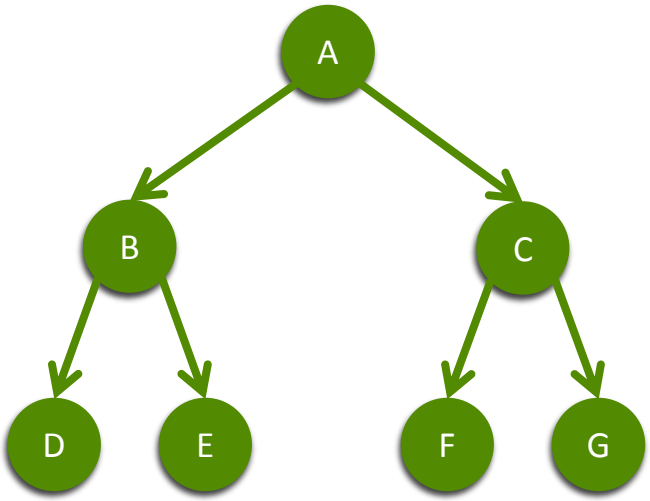


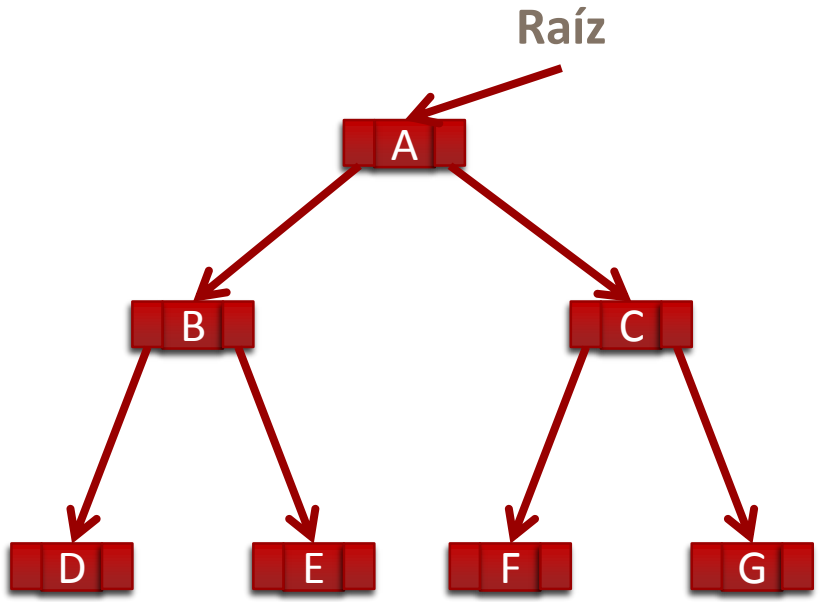
1	2	3	4	5	6	7
A	B	C	D	E	F	G

De igual forma, es posible representar un árbol binario utilizando listas ligadas. En este caso, cada nodo posee tres campos: uno para el dato, otro para la liga izquierda y otro para la liga derecha.

Ejemplo 10

Representación de un árbol binario con listas ligadas:





Tarea 3

Realizar un programa que permita realizar una transformación de árbol a árbol binario.

Operaciones en árboles binarios

Dentro de un árbol se pueden realizar distintos tipos de recorridos. Recorrer un árbol implica visitar los nodos del árbol de manera ordenada y secuencial, de tal manera que todos los nodos sean visitados una sola vez.

Existen 3 formas de realizar los recorridos de los nodos, todas de naturaleza recursiva: preorden, inorden y posorden.

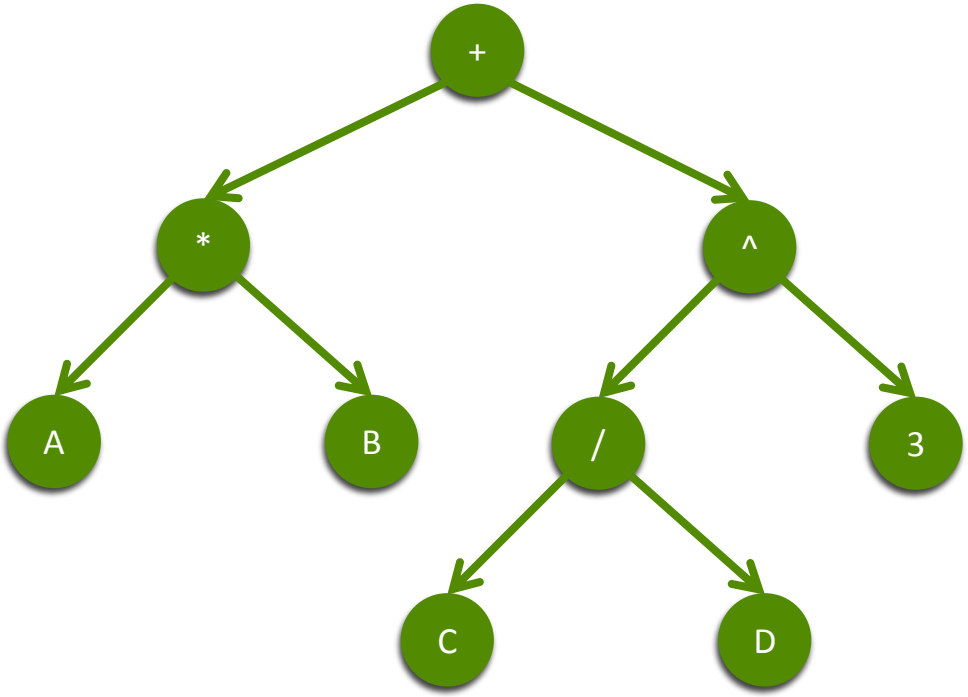
- Preorden
  - ✓ Visitar la raíz
  - ✓ Recorrer el subárbol izquierdo (arriba-abajo-abajo)
  - ✓ Recorrer el subárbol derecho (arriba-abajo-abajo)

- Inorden
  - ✓ Recorrer el subárbol izquierdo (abajo-arriba-abajo)
  - ✓ Visitar la raíz
  - ✓ Recorrer el subárbol derecho (abajo-arriba-abajo)

- Posorden
  - ✓ Recorrer el subárbol izquierdo (abajo-abajo-arriba)
  - ✓ Recorrer el subárbol derecho (abajo-abajo-arriba)
  - ✓ Visitar la raíz

Ejemplo 11

Realizar los recorridos en preorden, inorden y posorden del siguiente árbol.



PRE: + *AB ^/CD3
IN: A*B + C/D^3
POS: AB* CD/3^ +

En el ejemplo anterior, cuando se usa un recorrido PREORDEN se produce lo que se conoce como notación polaca prefija: + *AB ^/CD3.

Cuando el recorrido es INORDEN se genera una notación algebráica convencional (notación polaca infija): A*B + C/D ^ 3.

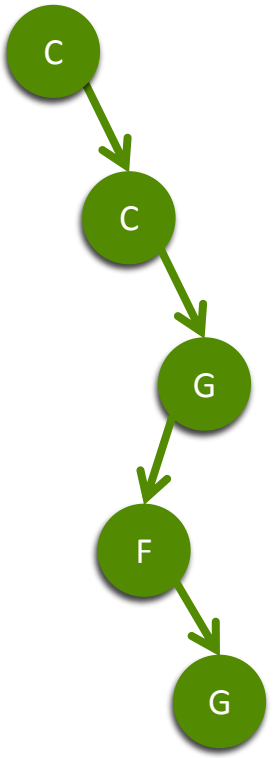
Cuando se realiza el recorrido POSORDEN se obtiene lo que se conoce como notación polaca postfija: AB* CD/ 3^ +.

Tarea 4

Realizar los algoritmos que permitan recorrer un árbol en PREORDEN, INORDEN y POSTORDEN.

Árbol balanceado

Si en un árbol binario se insertan un conjunto de claves ordenadas (de forma ascendente o descendente), se produce un árbol que crece en una sola dirección, lo que aumenta el tiempo de acceso a los nodos.

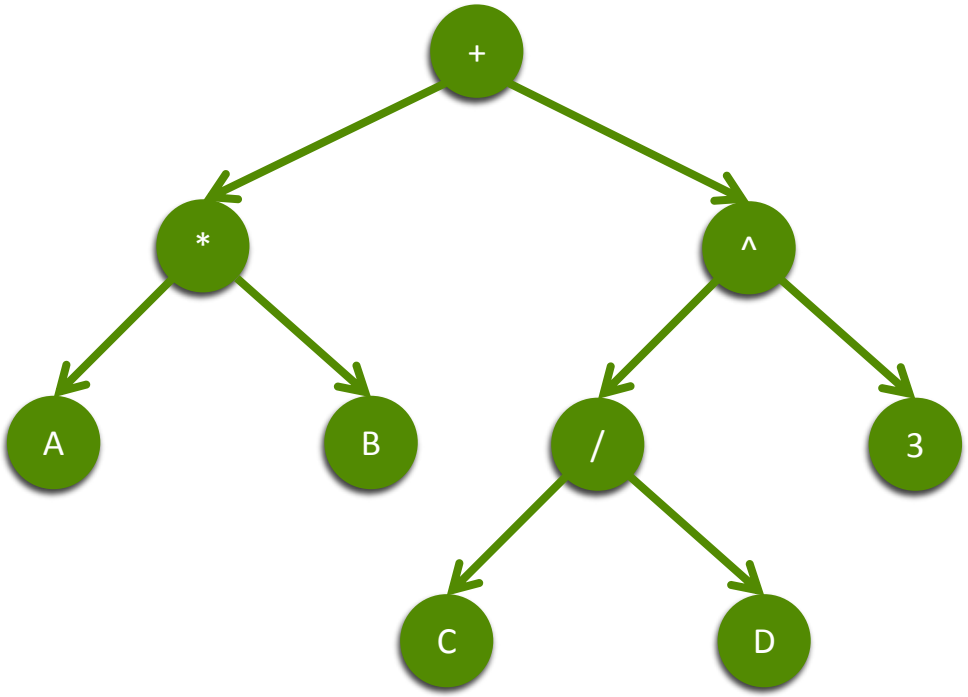


Para conservar la eficiencia en la operación de búsqueda se diseñaron los árboles balanceados. Los árboles balanceados realizan un reacomodo o balanceo de los nodos después de insertar o eliminar un elemento.

Formalmente, un árbol balanceado se define como un árbol binario de búsqueda en el que se debe cumplir que para todo nodo T del árbol, la altura de los subárboles izquerdo y derecho no deben diferir en más de un elemento.

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

donde H_{RI} es la altura del subárbol (rama) izquierdo y H_{RD} es la altura del subárbol (rama) derecho.

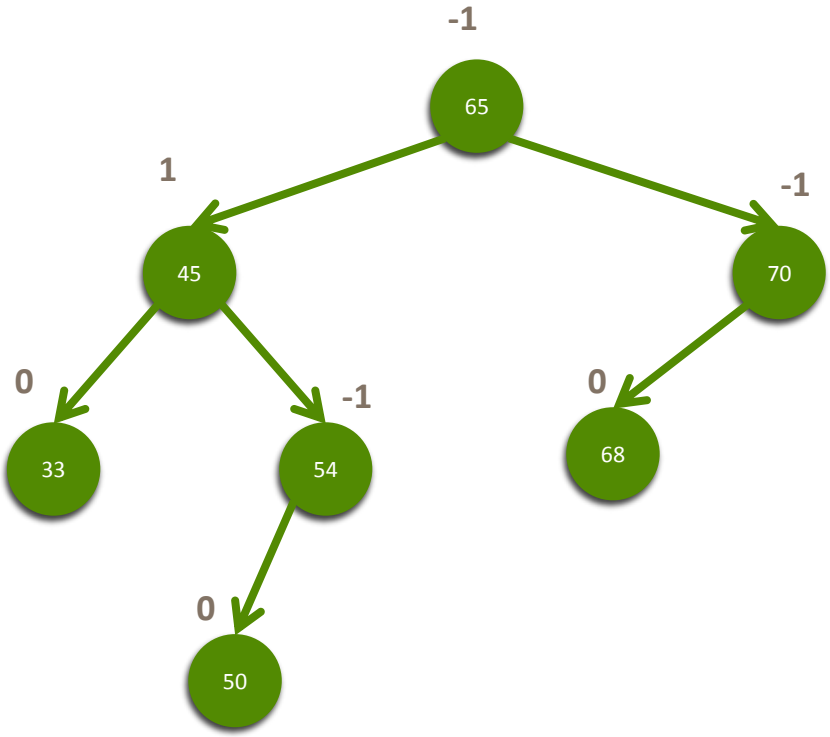


Árbol balanceado

Para determinar si un árbol está balanceado se debe calcular el factor de equilibrio (FE) del nodo. El factor de equilibrio se define como la diferencia entre la altura del subárbol derecho con la altura del subárbol izquierdo:

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

El factor de equilibrio puede ser -1, 0 ó 1. Si FE fuese -2 ó 2 el árbol debe ser restructurado.



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

Inserción en un árbol balanceado

Para mantener un árbol balanceado es necesario comprobar el factor de equilibrio una vez realizada la inserción.

El proceso de inserción en un árbol balanceado es sencillo (sigue los pasos de los otros árboles), empero, para mantener el equilibrio, requiere operaciones auxiliares, lo que complica el algoritmo.

Para insertar un nodo se debe seguir el camino de búsqueda del árbol. Una vez insertado se calcula el FE (que será 0 para dicho nodo) y se regresa por el camino de búsqueda calculando el FE de los nodos visitados. Si en algún nodo se rompe el criterio de equilibrio debe reestructurar el árbol.

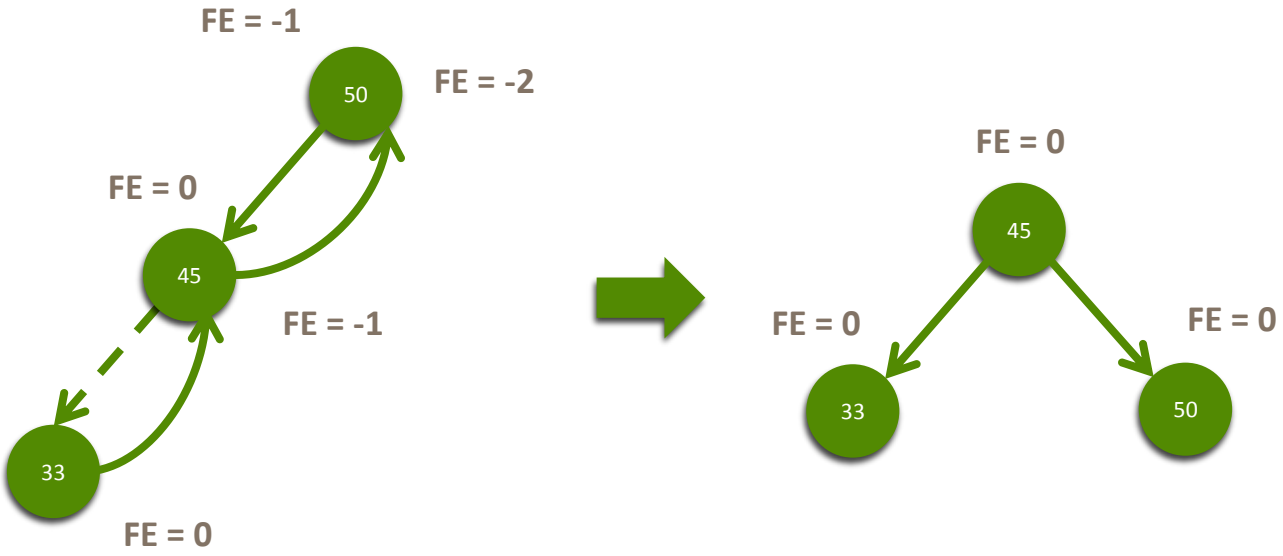
El ciclo anterior se rompe cuando se llega a la raíz del árbol o cuando se realiza el reacomodo en un nodo.

El reacomodo del árbol implica rotar los nodos para equilibrar la estructura. Existen 2 tipos de rotaciones: simple (dos nodos) y compuesta (tres nodos).

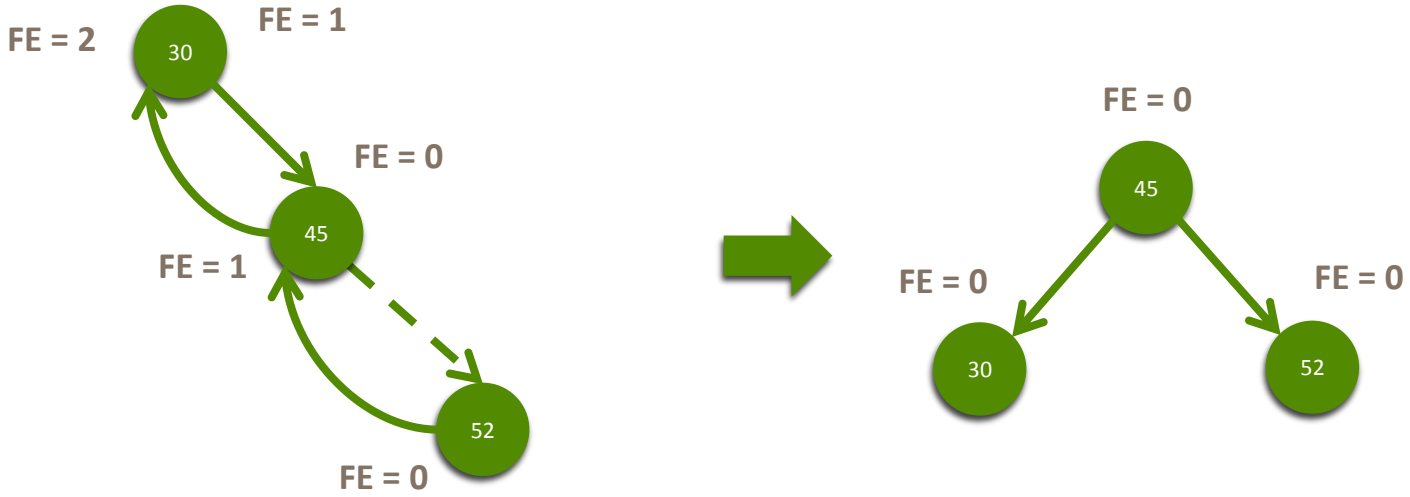
Una rotación simple se puede realizar por la rama derecha (DD) o izquierdas (II) de los nodos involucrados.

Una rotación compuesta se puede realizar por la rama derecha e izquierda (DI) o izquierda y derecha (ID) de los nodos involucrados.

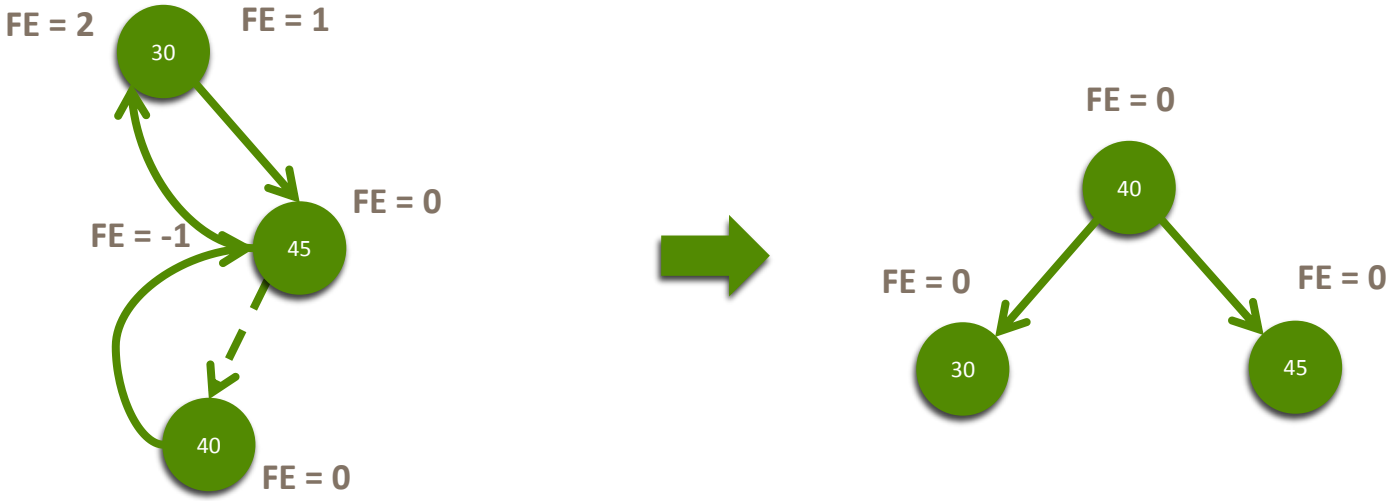
Rotación simple (II)



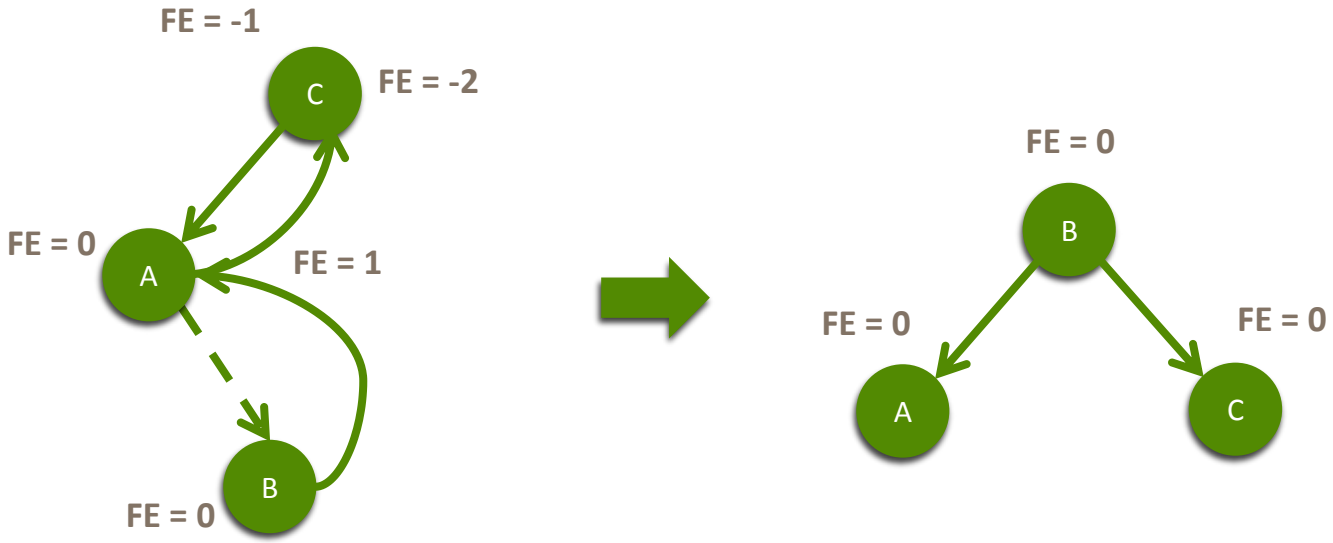
Rotación simple (DD)



Rotación compuesta (DI)



Rotación compuesta (ID)



Ejemplo 13

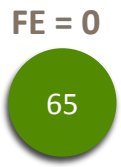
Se desean insertar las siguientes claves en un árbol balanceado vacío:

65 – 50 – 23 – 70 – 82 – 68 – 39

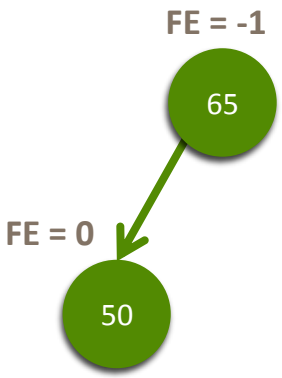
Realizar las inserciones y rotaciones paso a paso.

Ejemplo 13

1) Se inserta la clave 65 en el árbol vacío.

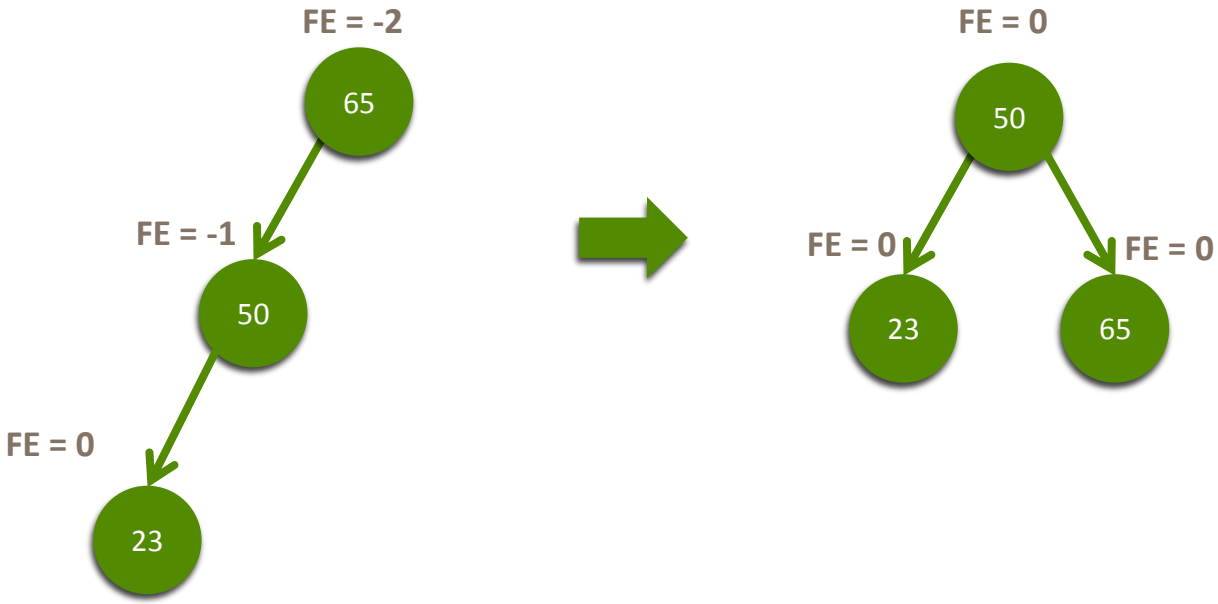


2) Se inserta la clave 50.



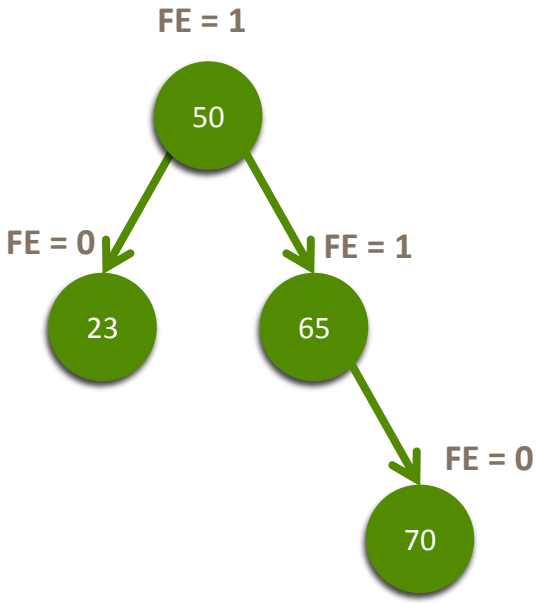
Ejemplo 13

3) Se inserta la clave 23.



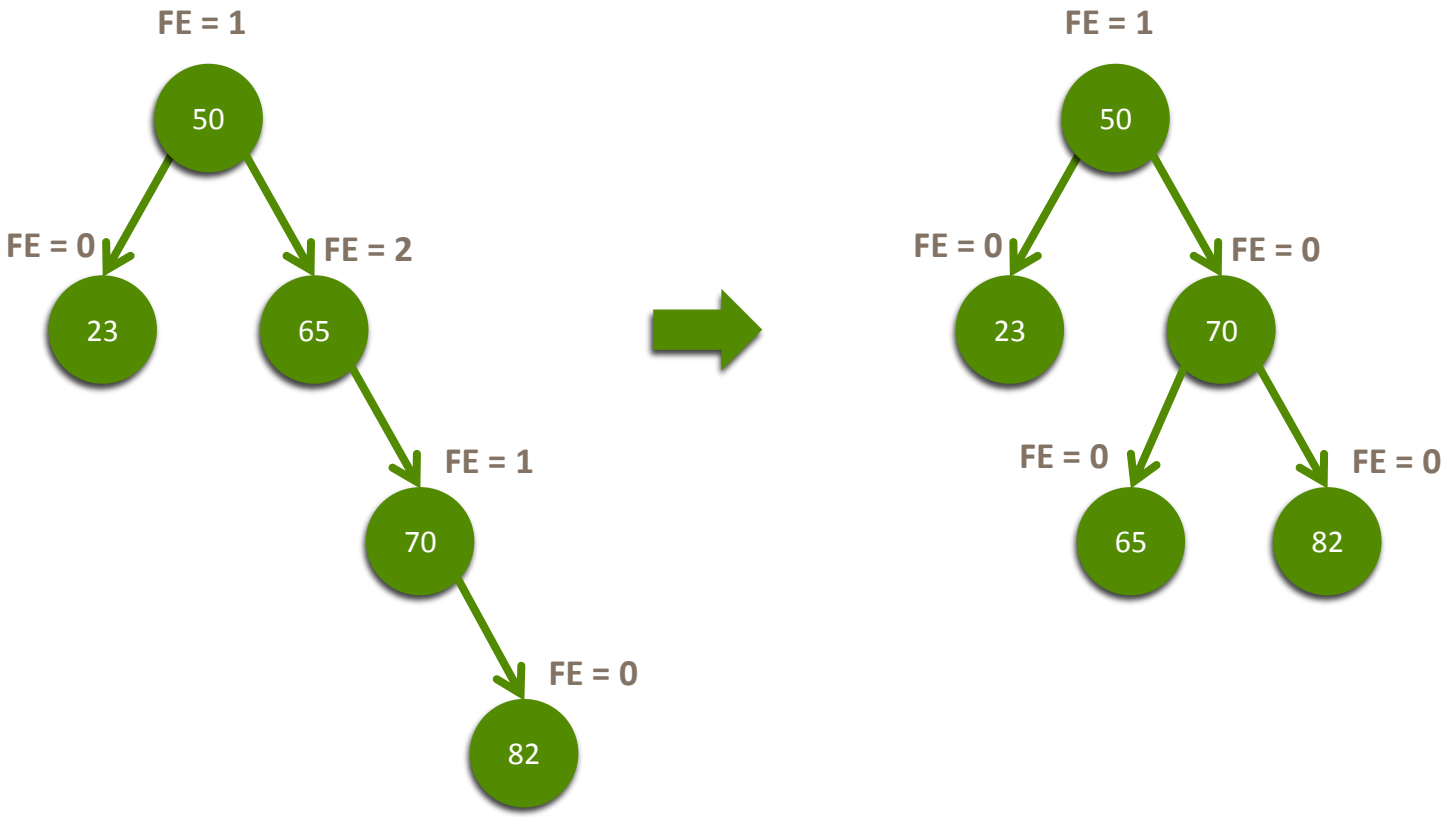
Ejemplo 13

4) Se inserta la clave 70.



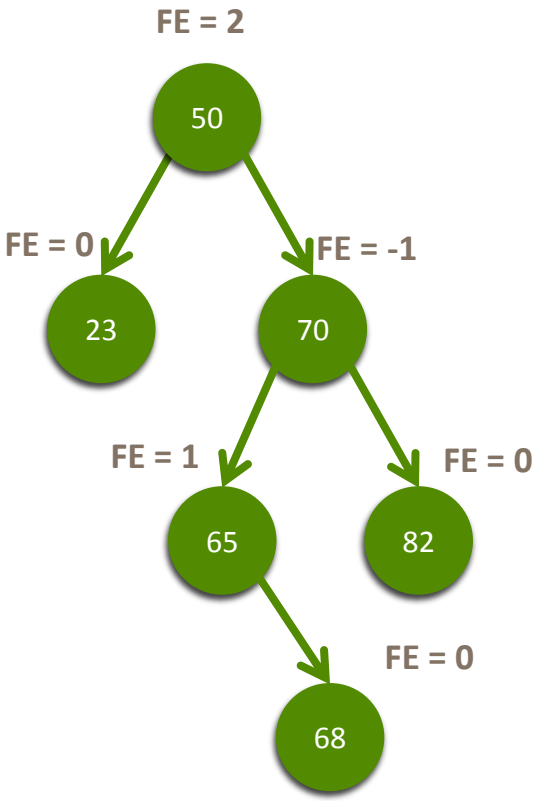
Ejemplo 13

5) Se inserta la clave 82.



Ejemplo 13

6) Se inserta la clave 68.

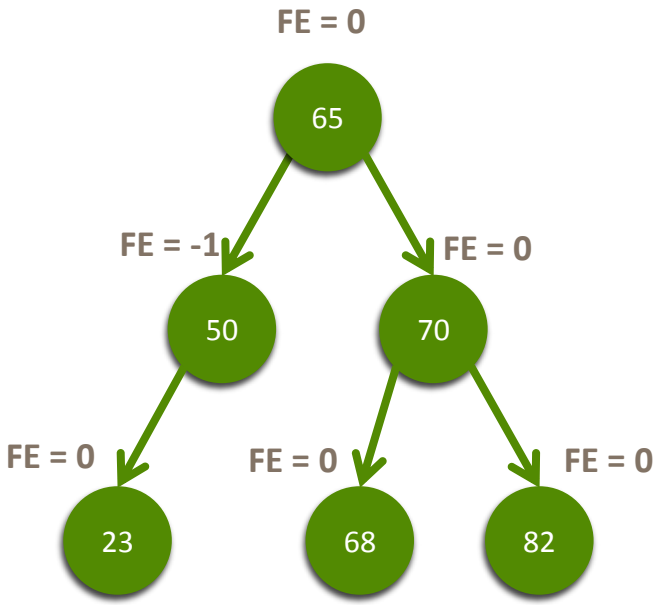


Los nodos que se deben mover son 50, 70 y 65 que forman una estructura compleja DI. Supóngase que n_1 apunta a 50, n_2 a 70 y n_3 a 65. Las rotaciones que se deben realizar son:

n_2.izq → n_3.der
n_3.der → n_2
n_1.der → n_2.izq
n_3.izq → n_1

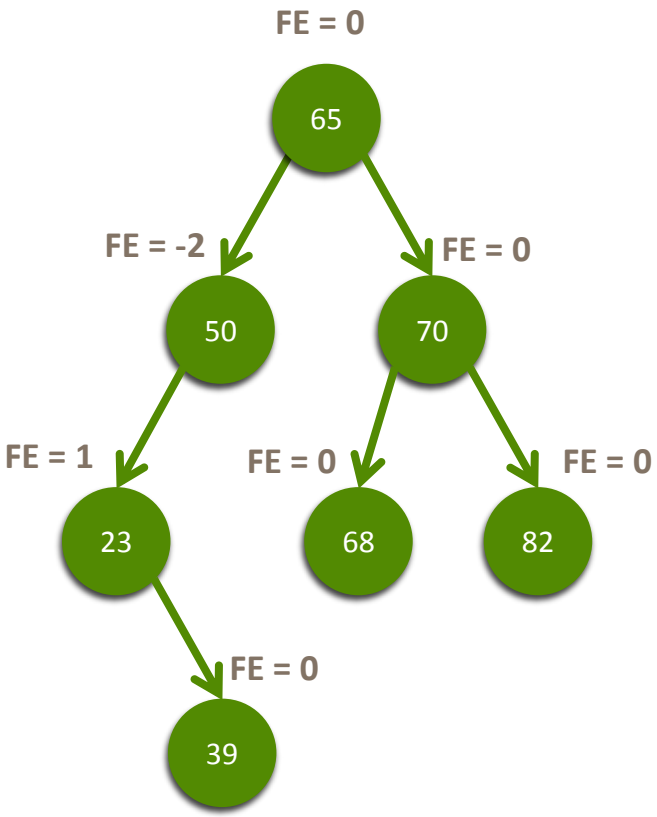
Ejemplo 13

Al final de la rotación el árbol queda de la siguiente manera:



Ejemplo 13

7) Se inserta la clave 39.

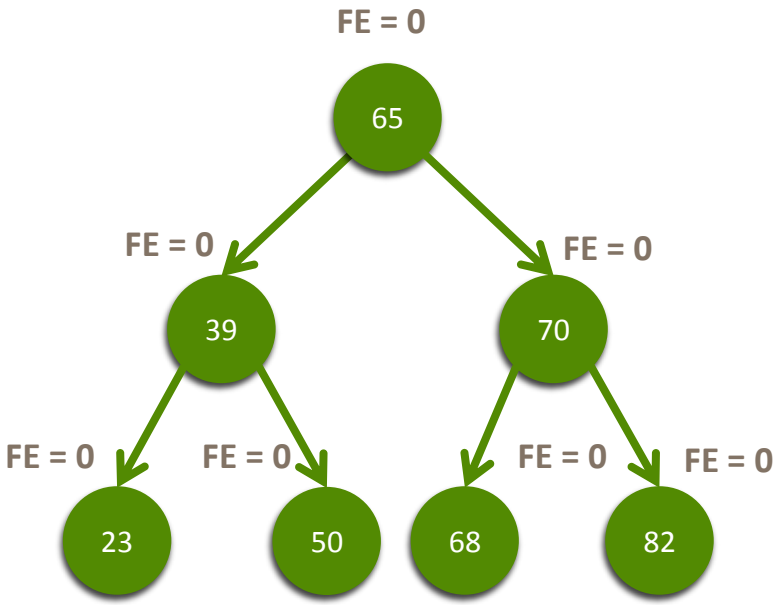


Los nodos que se deben mover son 50, 23 y 39 que forman una estructura compleja ID. Supóngase que n_1 apunta a 50, n_2 a 23 y n_3 a 39. Las rotaciones que se deben realizar son:

n_2.der → n_3.izq
n_3.izq → n_2
n_1.izq → n_3.der
n_3.der → n_1

Ejemplo 13

Al final de la rotación el árbol queda de la siguiente manera:



Eliminación en un árbol balanceado

Para eliminar elementos en un árbol balanceado una vez que se remueve el dato deseado se debe verificar que no se violen los principios del mismo, es decir, que se cumpla el factor de equilibrio.

Para eliminar datos de un árbol balanceado se debe se utiliza un algorimto de eliminación para árboles binarios y las operaciones de reacomodo del algoritmo de inserción de árboles balanceados.

El proceso de eliminación de un nodo debe tomar en cuenta los siguientes casos:

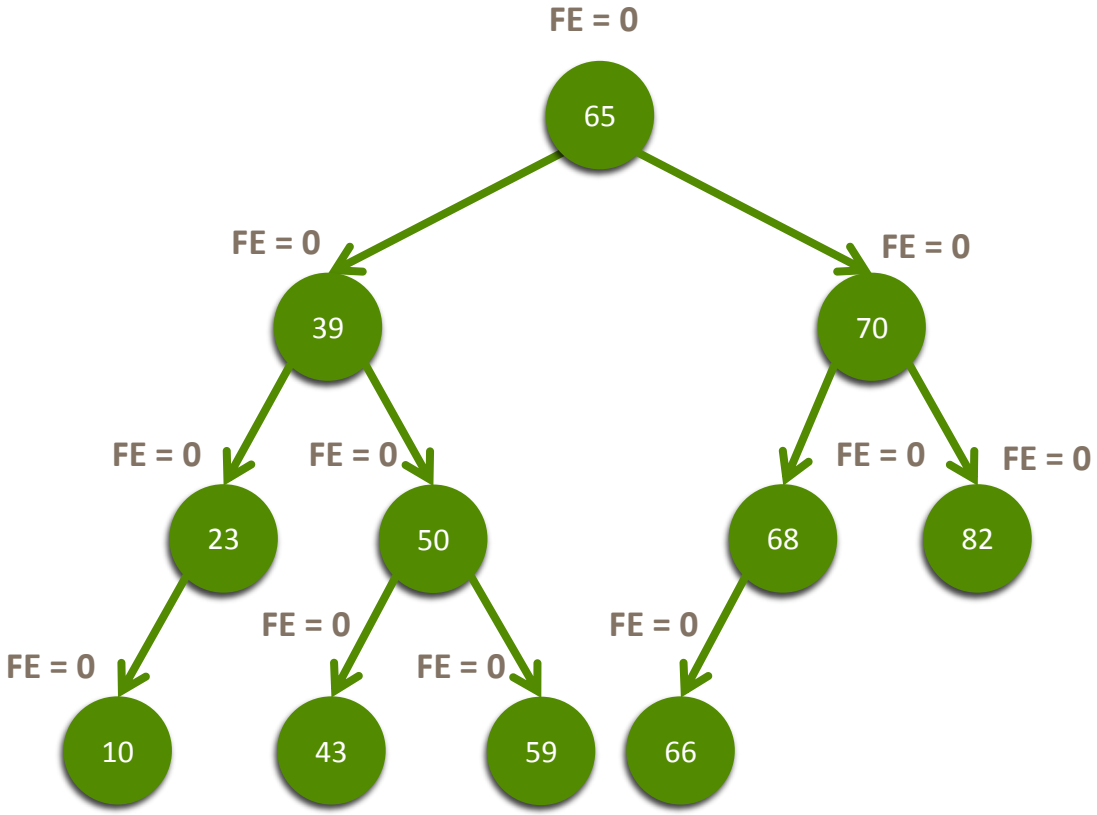
- Si el elemento a eliminar es una hoja, simplemente se suprime.
- Si el elemento a eliminar posee un solo hijo, entonces se tiene que sustituir por ese nodo hijo.
- Si el elemento a eliminar tiene los dos hijos, entonces se sustituye por el nodo que se encuentra más a la izquierda en el subárbol derecho o más a la derecha en el subárbol izquierdo.

Una vez que se haya realizado el proceso de eliminación se debe regresar por el camino de búsqueda calculando el factor de equilibrio de los nodos visitados. Si en algún nodo se rompe el criterio de equilibrio debe reestructurar el árbol.

A diferencia del algoritmo de inserción, el ciclo anterior se rompe cuando se llega a la raíz del árbol, es decir, se pueden efectuar más de una rotación en el camino hacia atrás.

Ejemplo 14

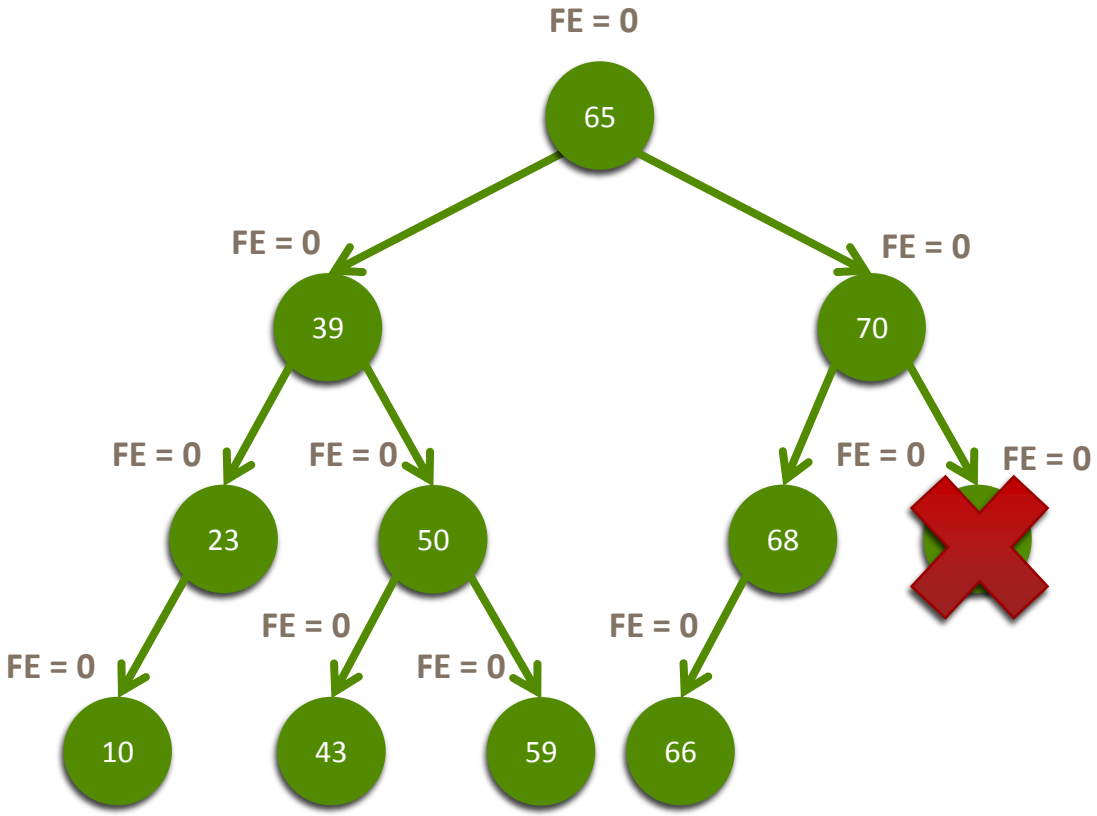
Dado el siguiente árbol



Eliminar los elementos: 82, 10, 39, 65, 70, 23, 50 y 39.

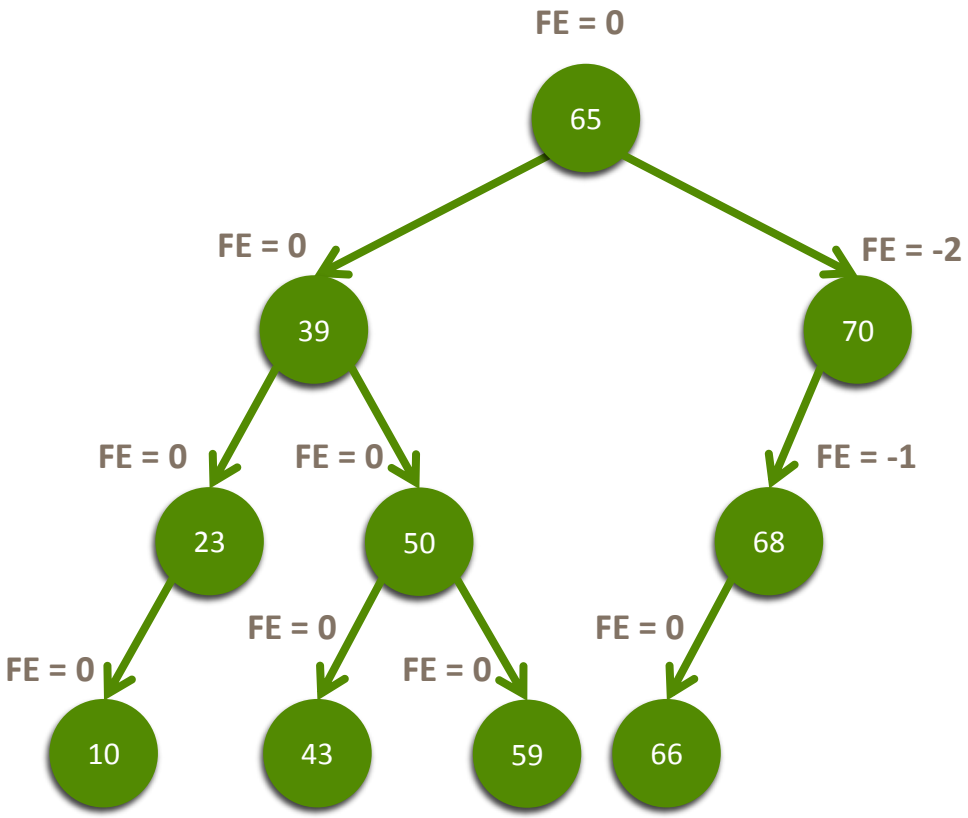
Ejemplo 14

1) Se elimina la clave 82.



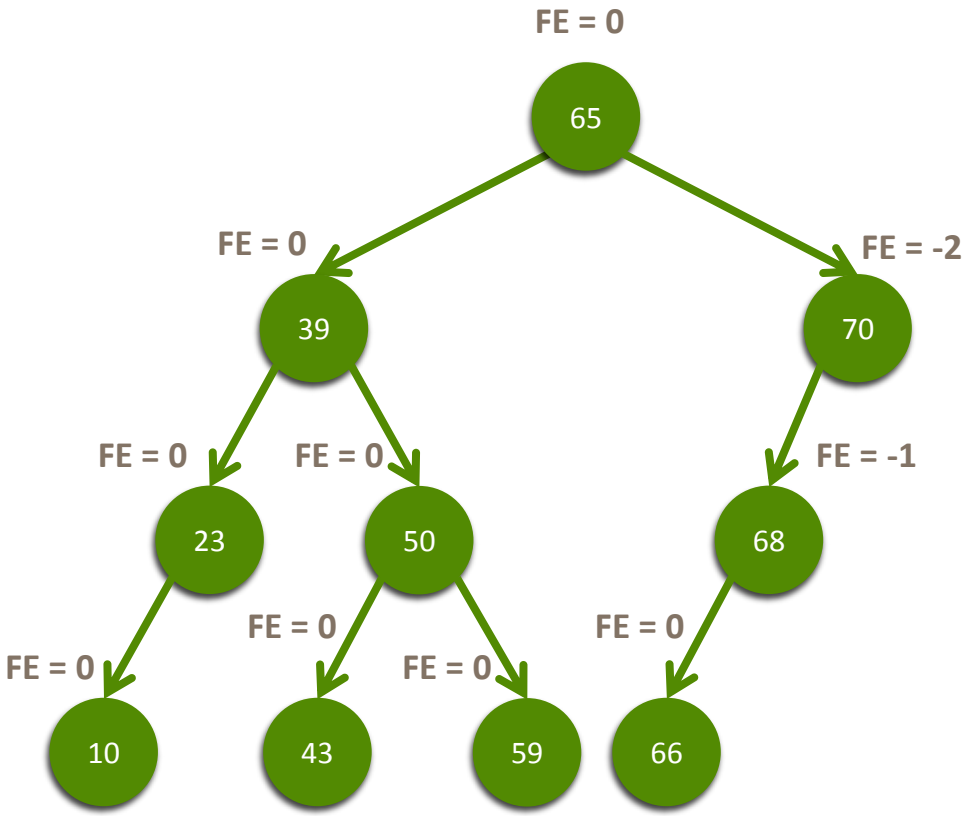
Ejemplo 14

Una vez eliminado el elemento 82, en la clave 70 se rompe el criterio de equilibrio.



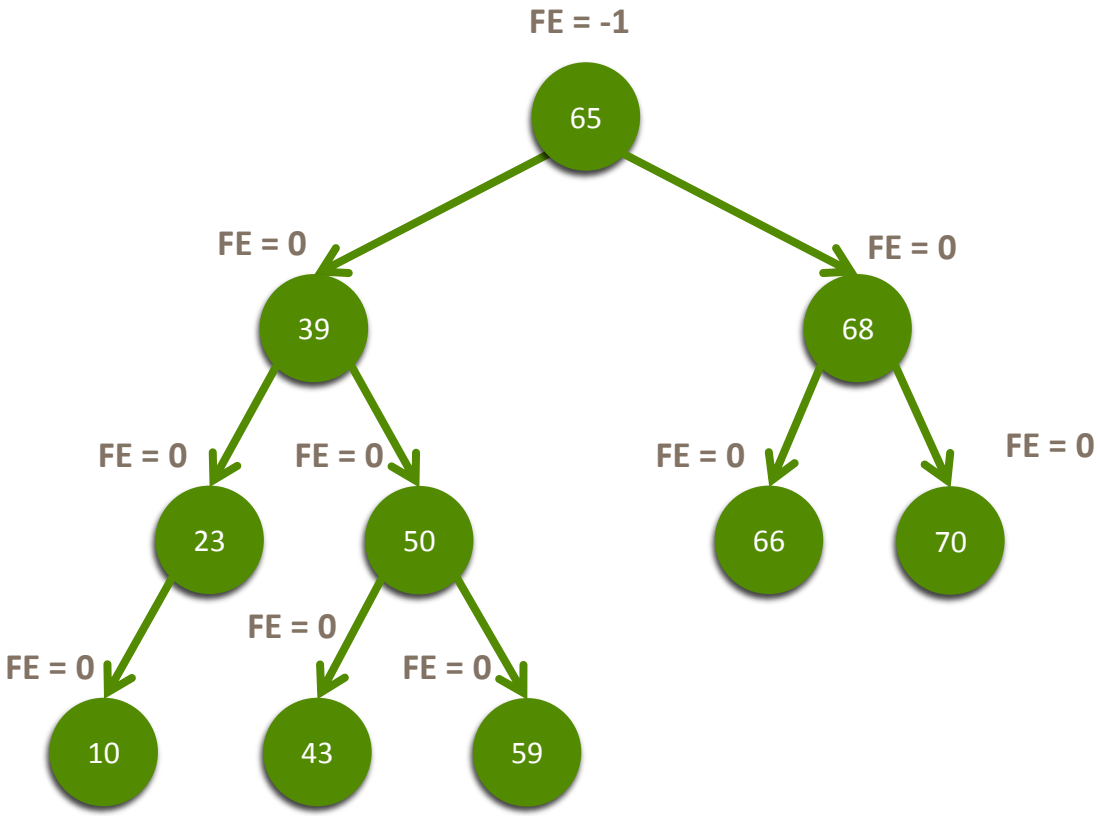
Ejemplo 14

Por tanto, se tiene que hacer una rotación. n_1 apunta a 70, n_2 apunta a la rama izquierda de n_1. Como $fe(n_2)=-1$, se realiza una rotación II y, por tanto, n_3 a 66.



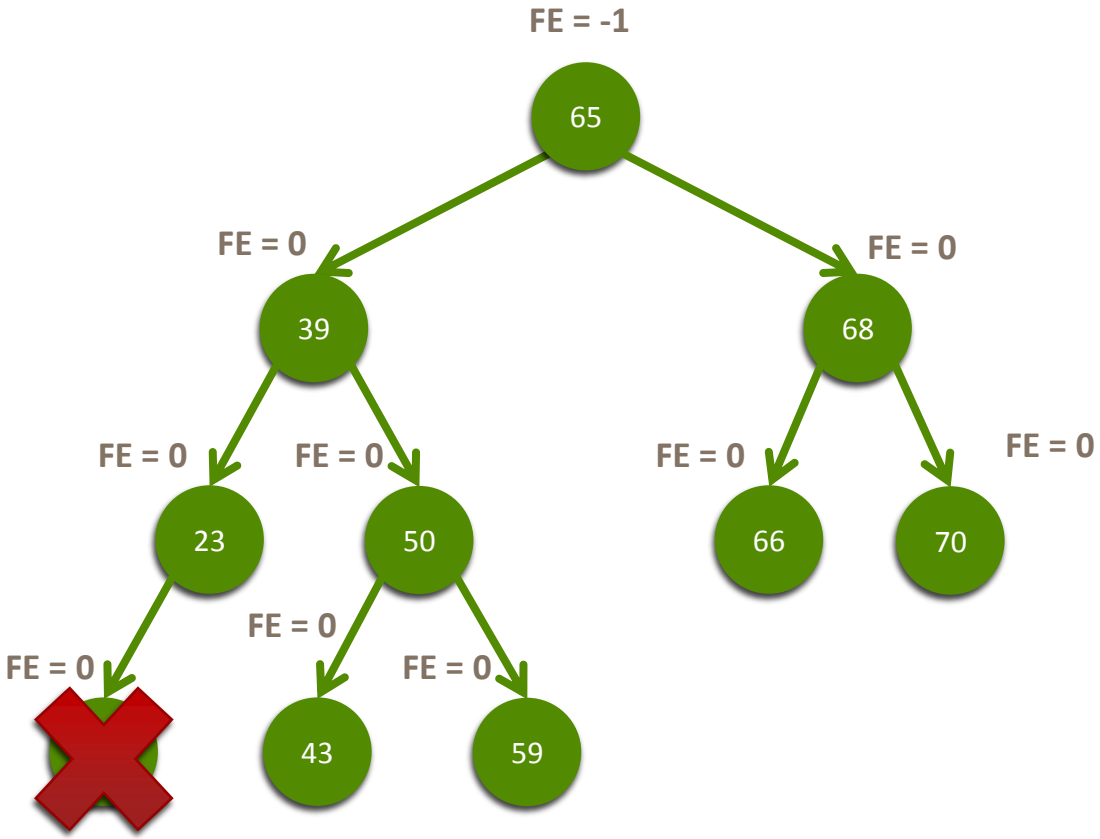
Ejemplo 14

Por tanto, se tiene que hacer una rotación. n_1 apunta a 70, n_2 apunta a la rama izquierda de n_1. Como $fe(n_2)=-1$, se realiza una rotación II y, por tanto, n_3 a 66.



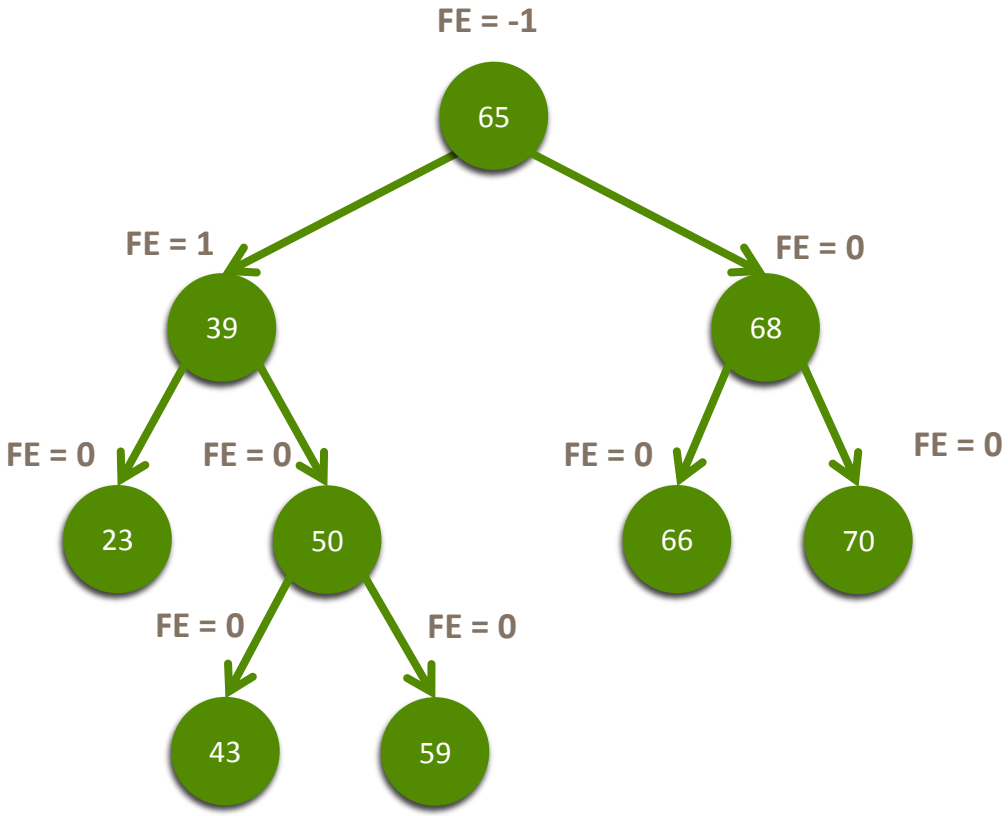
Ejemplo 14

2) Se elimina la clave 10.



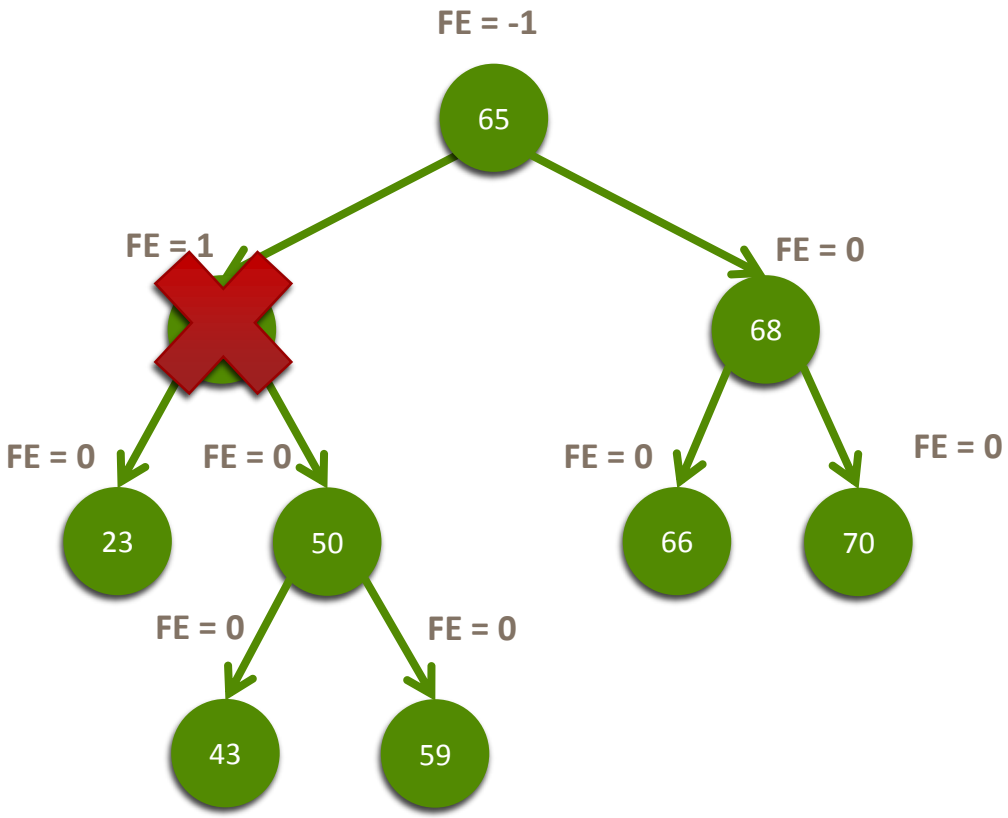
Ejemplo 14

La eliminación de la clave 10 no genera mayor desequilibrio del que ya poseía el árbol.



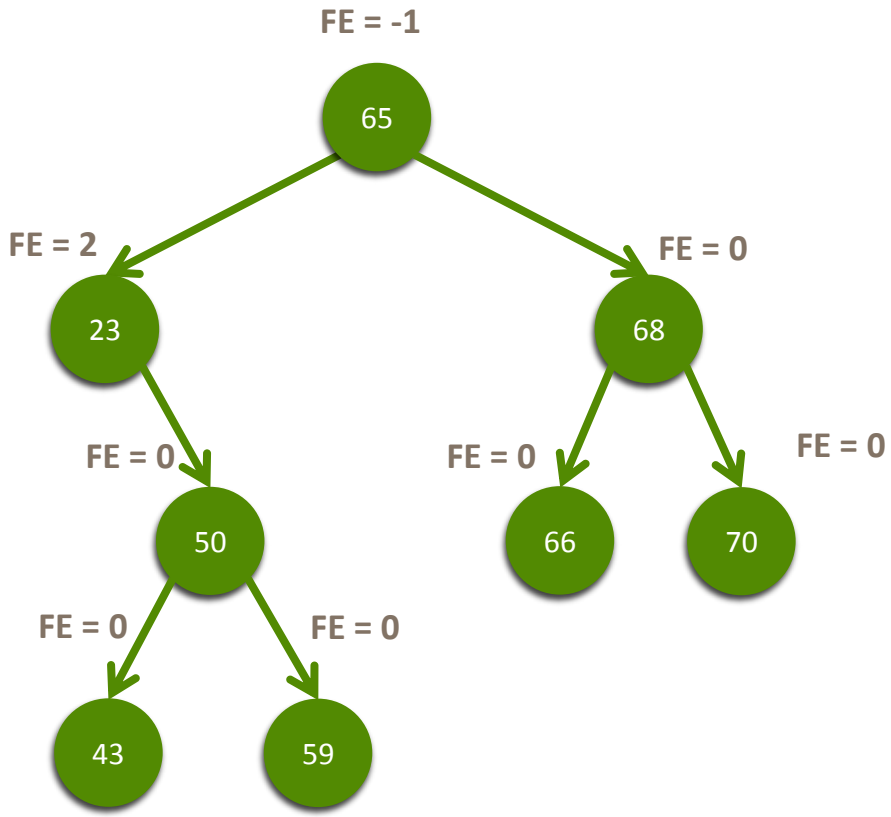
Ejemplo 14

3) Se elimina la clave 39.



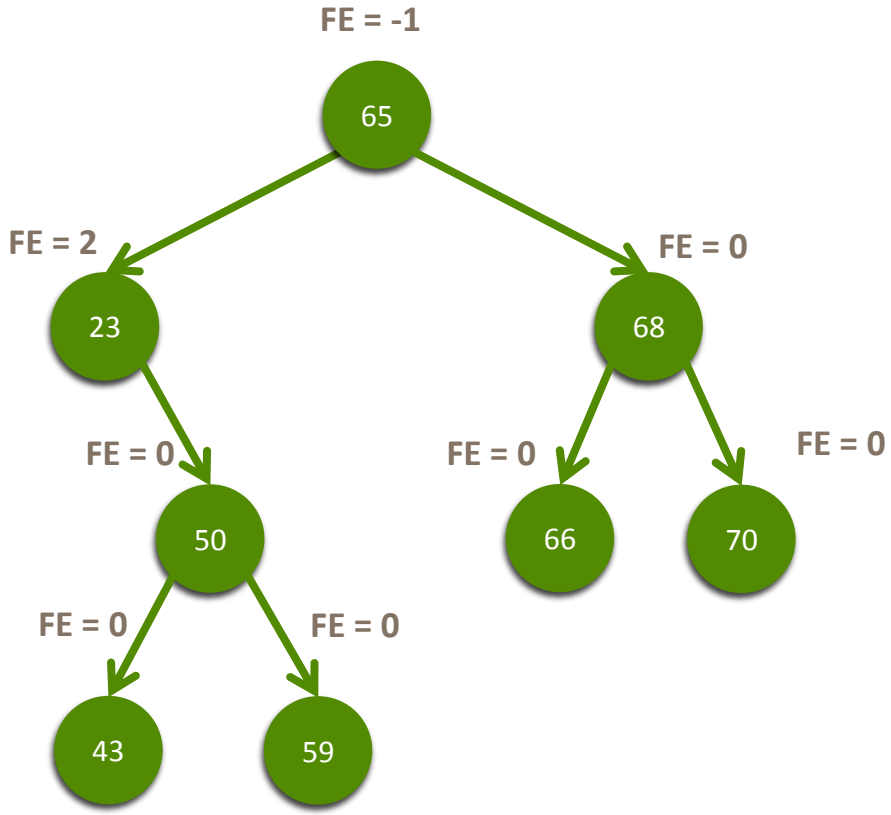
Ejemplo 14

La clave 39 tiene dos hijos, entonces se realiza la sustitución por el nodo que se encuentra más a la derecha en el subárbol izquierdo.



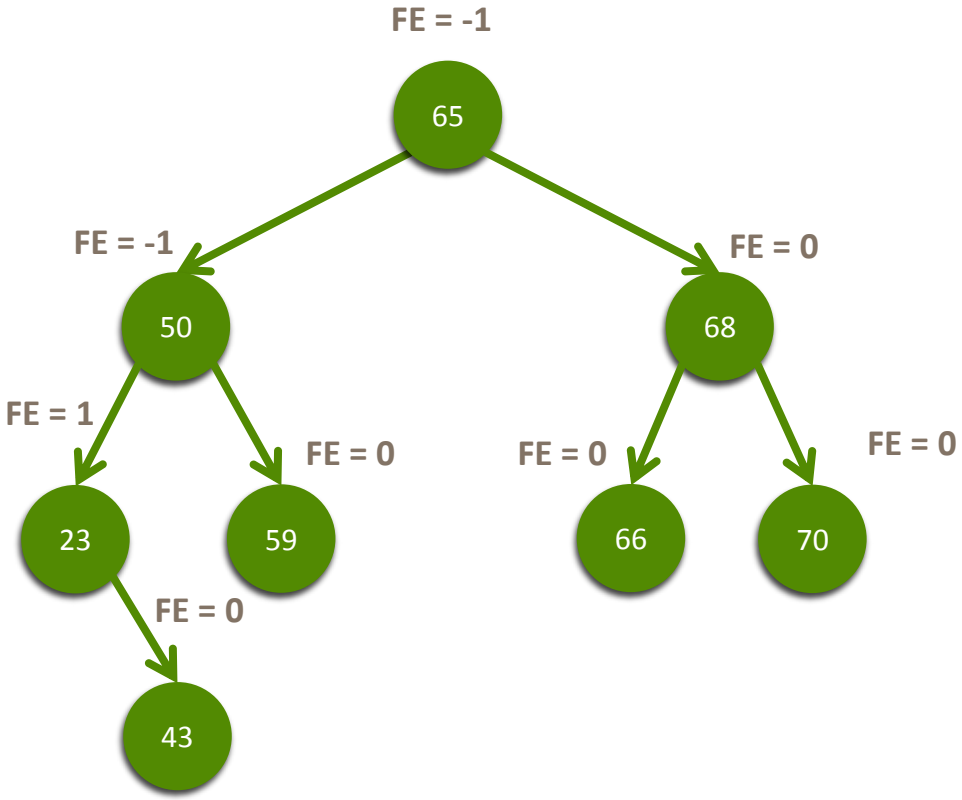
Ejemplo 14

Después de la sustitución, se altera el FE del árbol en el nodo con clave 23 y, por tanto, se debe realizar una rotación. n_1 = 23, n_2 apunta a la rama derecha de n_1. Debido a que FE(n_2)=0, se realiza una rotación DD y n_3 apunta a la rama derecha de n_2.



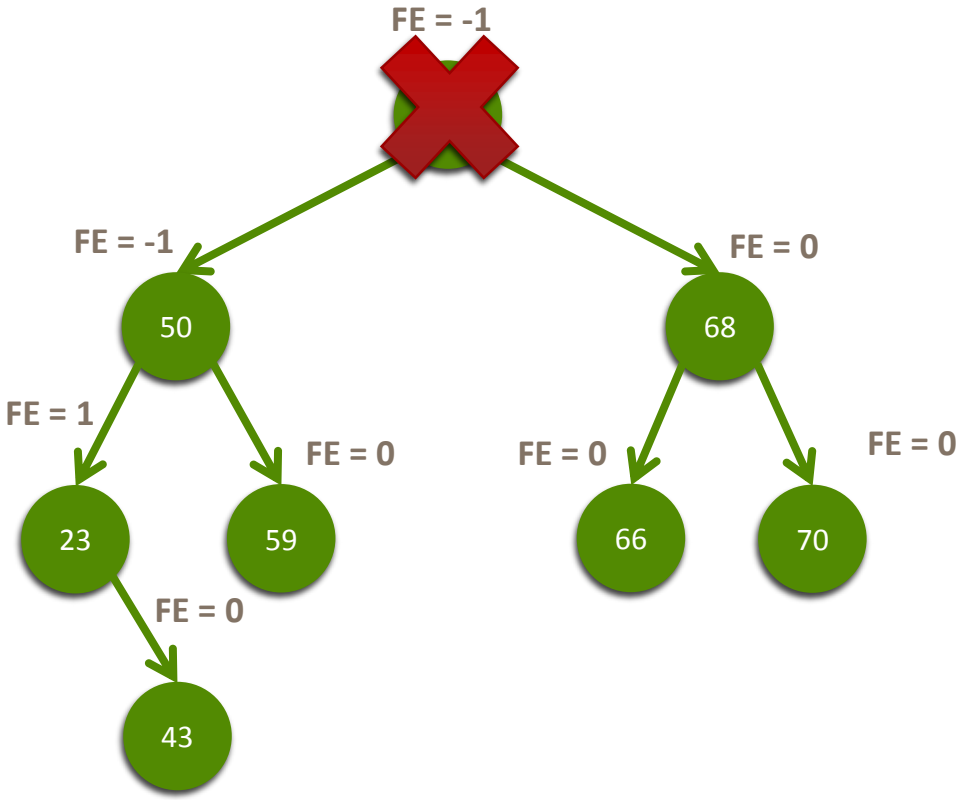
Ejemplo 14

Después de la sustitución, se altera el FE del árbol en el nodo con clave 23 y, por tanto, se debe realizar una rotación. n_1 = 23, n_2 apunta a la rama derecha de n_1. Debido a que FE(n_2)=0, se realiza una rotación DD y n_3 apunta a la rama derecha de n_2.



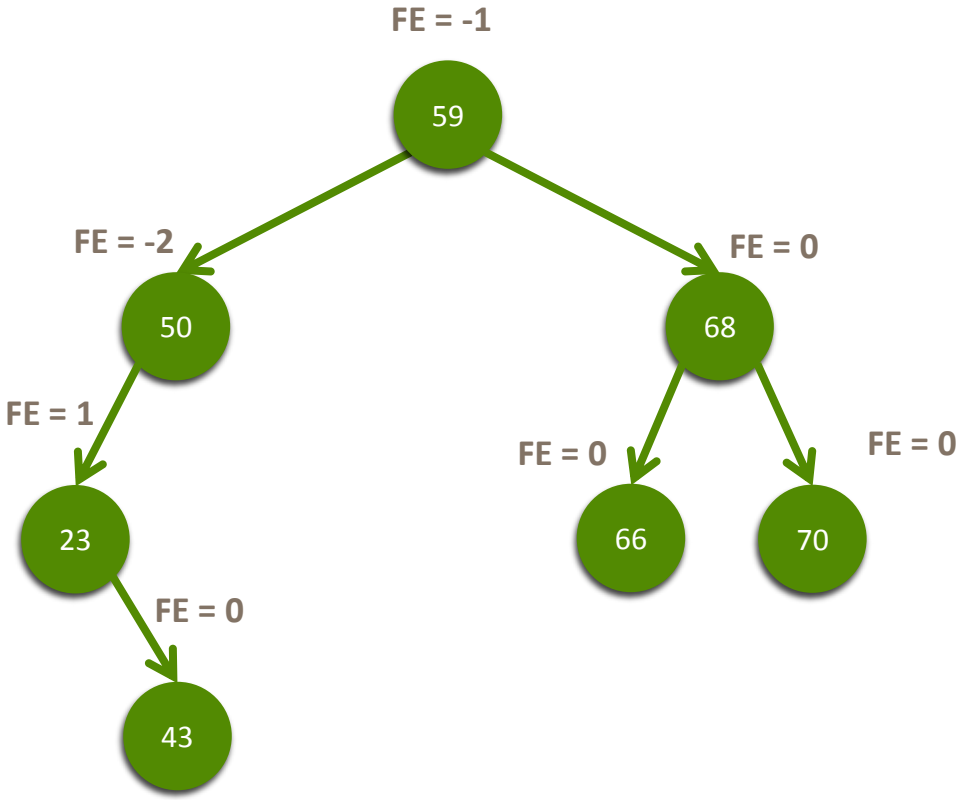
Ejemplo 14

4) Se elimina la clave 65.



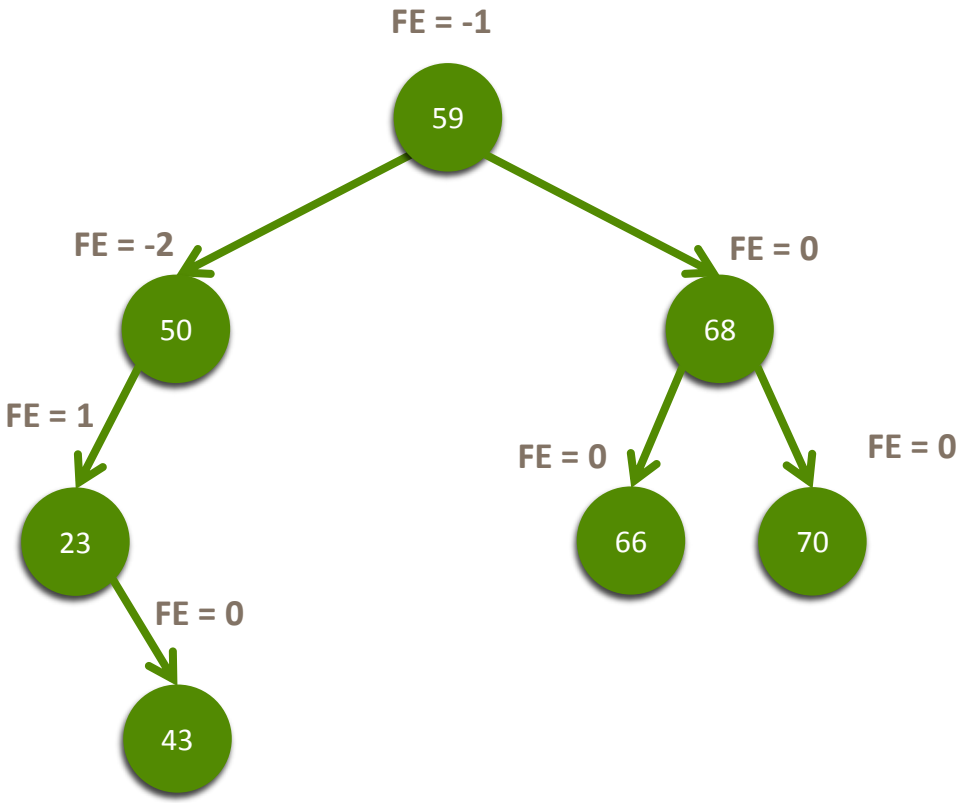
Ejemplo 14

La clave 65 tiene dos hijos, se decide sustituir con el nodo que se encuentra más a la derecha del subárbol izquierdo.



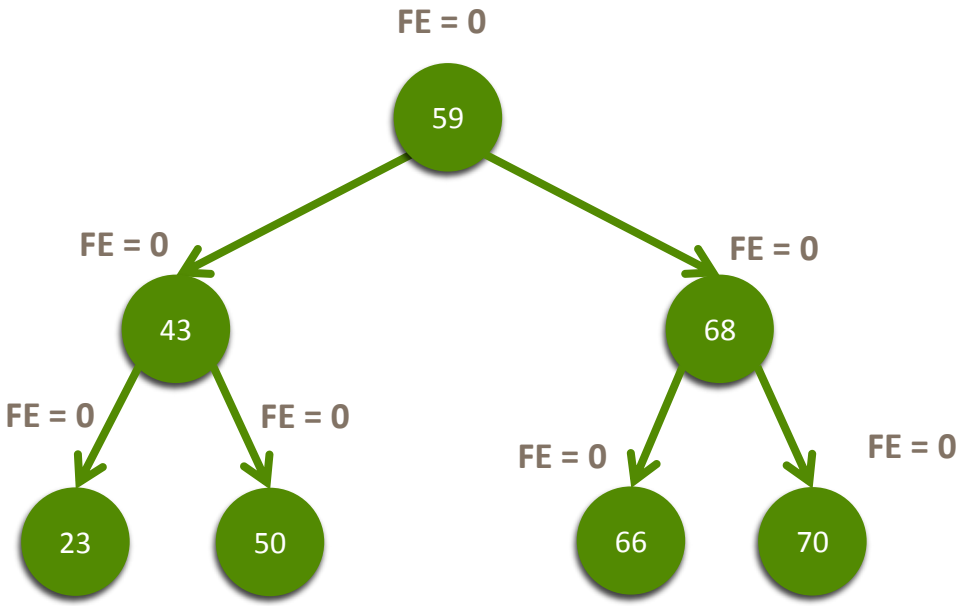
Ejemplo 14

Después de la sustitución se altera el FE del árbol en el nodo con clave 50 y, por tanto, se debe realizar una rotación. n_1 = 50, n_2 apunta a la rama izquierda de n_1. Debido a que FE(n_2)=1, se realiza una rotación ID y n_3 apunta a la rama derecha de n_2.



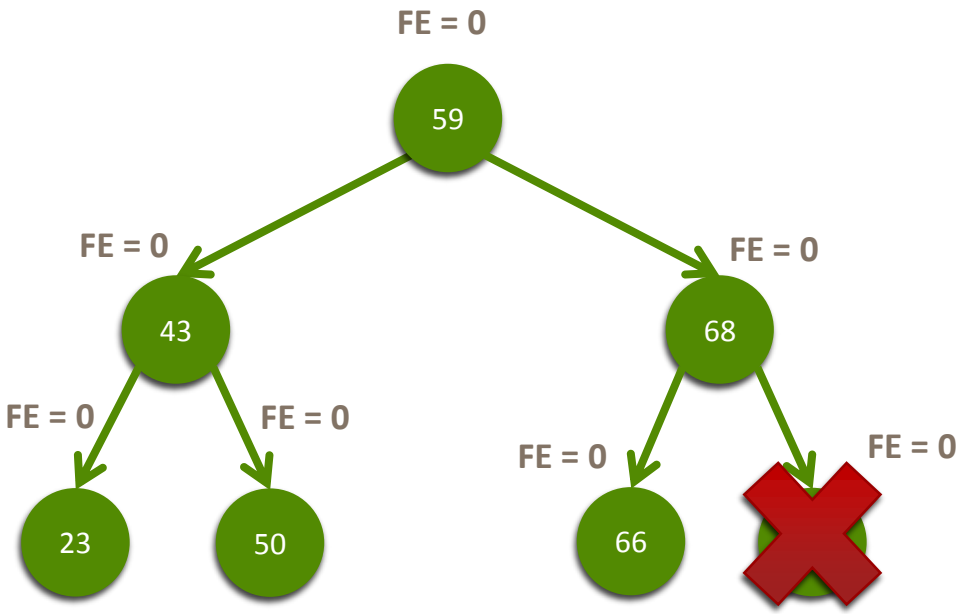
Ejemplo 14

Después de la sustitución se altera el FE del árbol en el nodo con clave 50 y, por tanto, se debe realizar una rotación. n_1 = 50, n_2 apunta a la rama izquierda de n_1. Debido a que $FE(n_2)=1$, se realiza una rotación ID y n_3 apunta a la rama derecha de n_2.



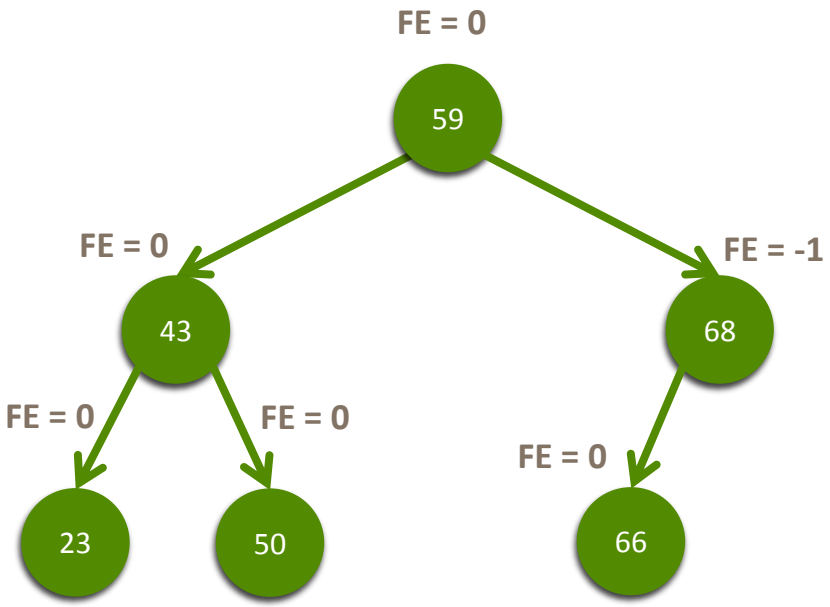
Ejemplo 14

5) Se elimina la clave 70



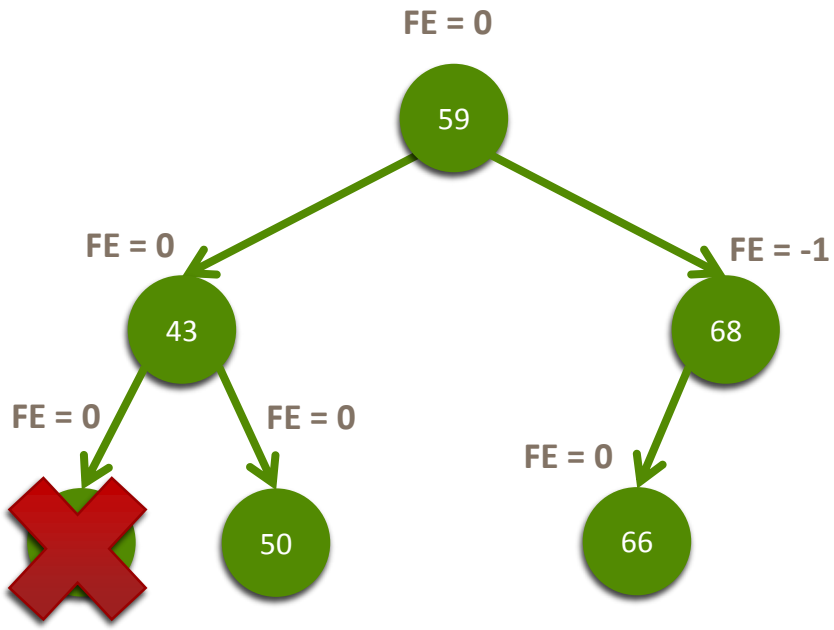
Ejemplo 14

Después de eliminar el nodo 70 el árbol y su FE queda de la siguiente manera:



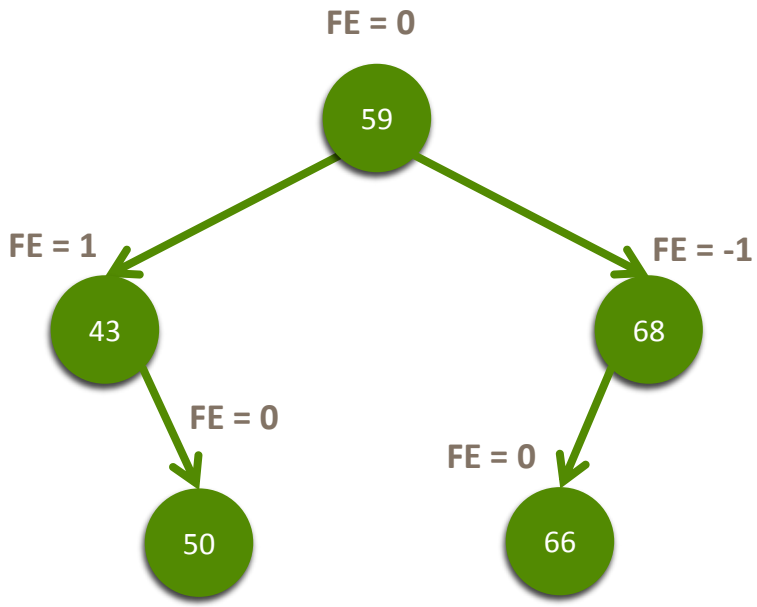
Ejemplo 14

6) Se elimina la clave 23



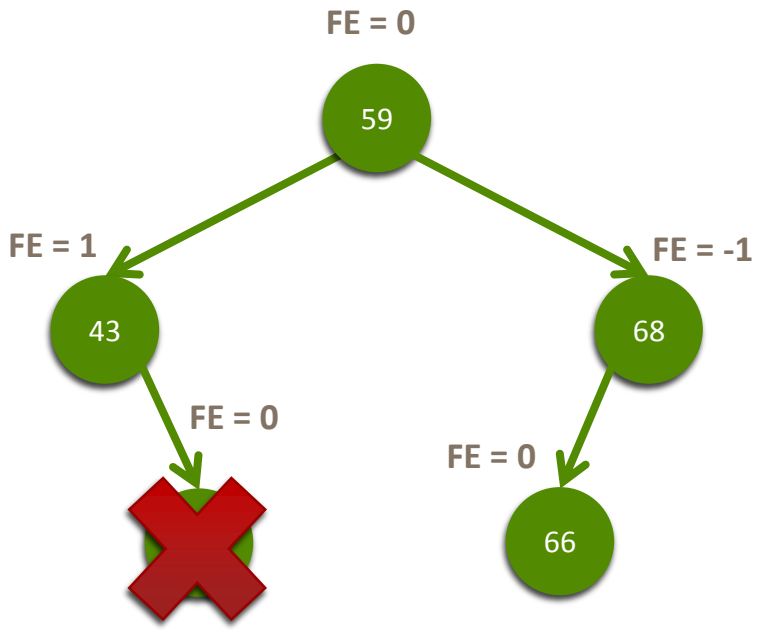
Ejemplo 14

Después de eliminar el nodo 70 el árbol y su FE queda de la siguiente manera:



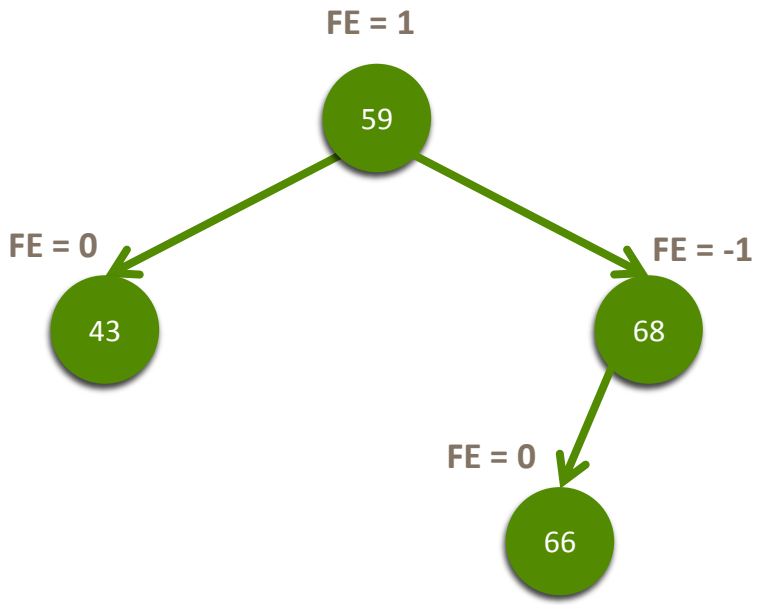
Ejemplo 14

7) Se elimina la clave 50



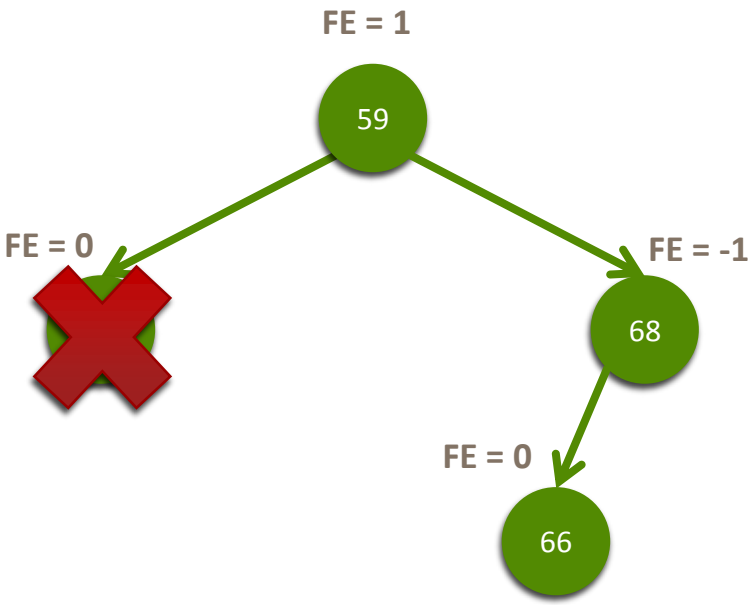
Ejemplo 14

Después de eliminar el nodo 70 el árbol y su FE queda de la siguiente manera:



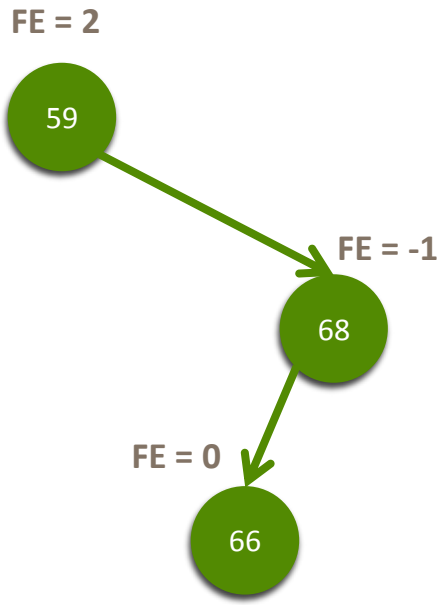
Ejemplo 14

8) Se elimina la clave 43



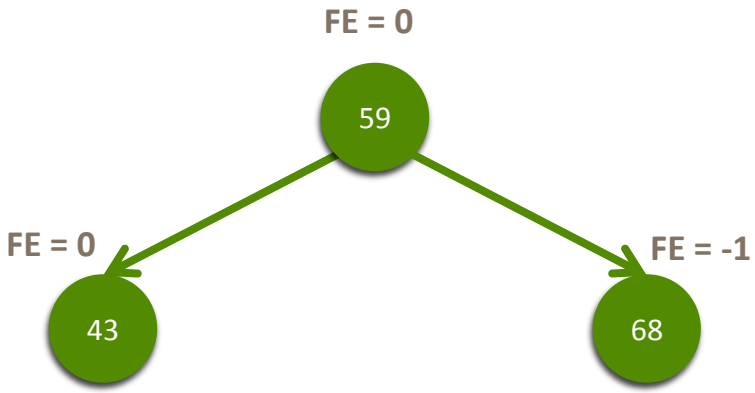
Ejemplo 14

Una vez eliminada la clave 43, se rompe el equilibrio del árbol en el nodo 59. n_1 apunta a 59, n_2 apunta al árbol derecho de n_1 y, como el FE(n_2)=-1, n_3 apunta a la rama izquierda de n_2, se realiza la rotación DI.



Ejemplo 14

Una vez eliminada la clave 43, se rompe el equilibrio del árbol en el nodo 59. n_1 apunta a 59, n_2 apunta al árbol derecho de n_1 y, como el FE(n_2)=-1, n_3 apunta a la rama izquierda de n_2, se realiza la rotación DI.



En general, las rotaciones son más frecuentes en las operaciones de inserción que en las de eliminación. Aproximadamente, por cada dos inserciones se produce una rotación y por cada cinco eliminaciones se produce una rotación.

4.4 Árboles B.

4.4 Árboles B.

En general, los árboles que se han visto hasta el momento trabajan en la memoria principal de la computadora. Empero, existen aplicaciones en las que el volumen de información es tan grande que los datos no caben en la memoria principal y es necesario almacenarlos en dispositivos de almacenamiento secundario (organizados en archivos).

Normalmente, el tiempo que se necesita para localizar un registro en la memoria principal está dado en microsegundos, mientras que el tiempo que se lleva localizar una página (con varios registros) en memoria secundaria se mide en milisegundos.

Por tanto, el tiempo de acceso es miles de veces más rápido en la memoria principal que en la memoria secundaria.

Si se tiene un árbol binario almacenado en disco, el tiempo promedio para localizar un nodo es log n accesos a disco, con n igual al número de nodos del árbol.

Si el árbol contiene 1,000,000 de elementos, se necesitaría, en promedio, 20 accesos al disco para encontrar un elemento. Si el árbol se organizara en páginas, de tal forma que cada página contenga 100 elementos, el tiempo de acceso está dado por $\log_{100}(1000000)$, por lo que se necesitaría solo 3 accesos a disco.

Árboles B

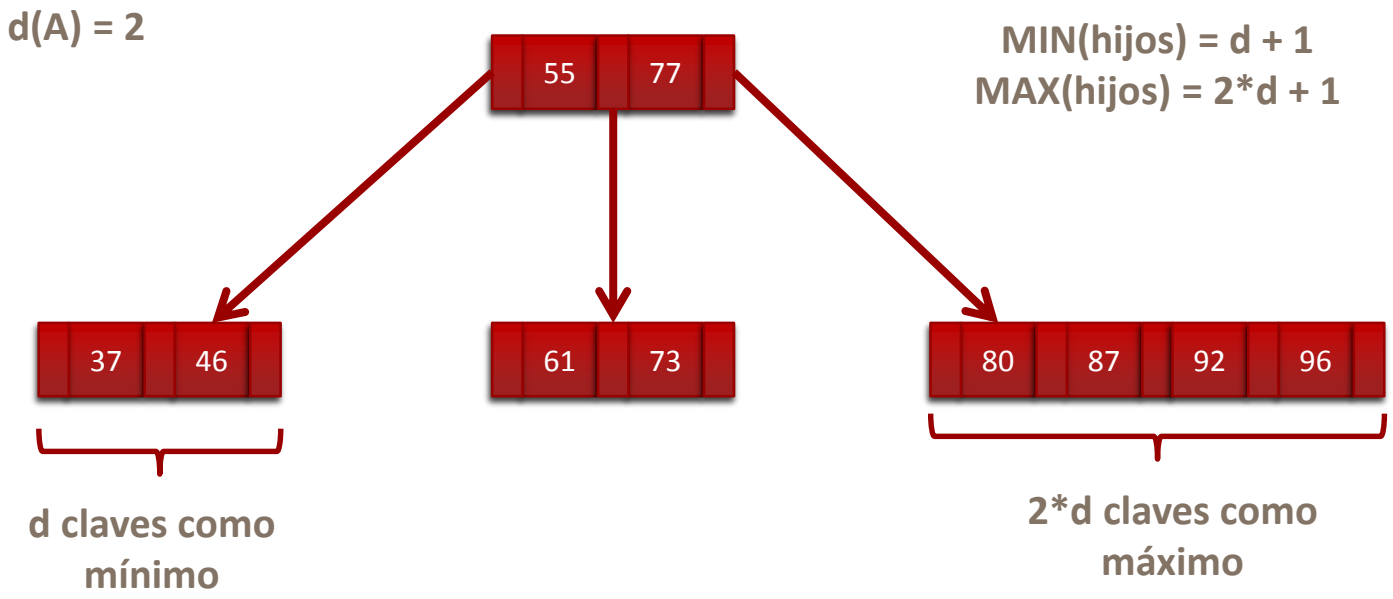
Los árboles B son un tipo de árboles balanceados que permiten organizar archivos con índices (almacenamiento y recuperación de información) en medio externos.

En los árboles B se tienen un grupo de nodos se conocen como páginas. Cada página posee a su vez información de un gruipo de nodos que poseen una clave o llave.

Cada página de un árbol B de orden d contiene 2d claves como máximo y d claves como mínimo. La última condición garantiza que cada página esté llena, como mínimo, hasta la mitad.

Además, cada página en un árbol B de orden d contiene como máximo 2d+1 hijos y como mínimo d+1 hijos, excepto la página raíz que puede contener solo un dato y, por ende, solo dos hijos mínimo.

A continuación se presenta un diagrama de un árbol B de orden 2.



Con el diagrama anterior podemos confirmar las siguientes afirmaciones:

- Cada página, con excepción de la raíz, contiene de d a 2d elementos, donde d es el grado del árbol.
- La raíz puede almacenar de 1 a 2*d elementos.
- Cada página (con excepción de la raíz y las hojas) posee de d+1 a 2*d+1 descendientes.
- La página raíz tiene al menos dos descendientes.
- Las páginas hoja están todas al mismo nivel.

Inserción en árboles B

Debido a que todas las hojas se encuentran al mismo nivel, cualquier camino desde la raíz hasta agluna hoja tiene la misma longitud.

Los árboles B crecen de abajo hacia arriba, es decir, desde las hojas hasta la raíz.

Para insertar un nodo en un árbol B se siguen los siguientes pasos:

1. Localizar la página donde se va a insertar la clave.
2. Si el número de elementos de la página (m) es menor o igual a 2*d entonces
   Se inserta la clave en el lugar que le corresponde
   De lo contrario (si m es mayor a 2*d)
   La página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.

Ejemplo 14

Realizar las inserciones de las siguientes claves

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

dentro de un árbol B de orden 2.

Ejemplo 14

1) Se inserta la clave 10

2) Se inserta la clave 27

Ejemplo 14

3) Se inserta la clave 29

4) Se inserta la clave 17

Ejemplo 14

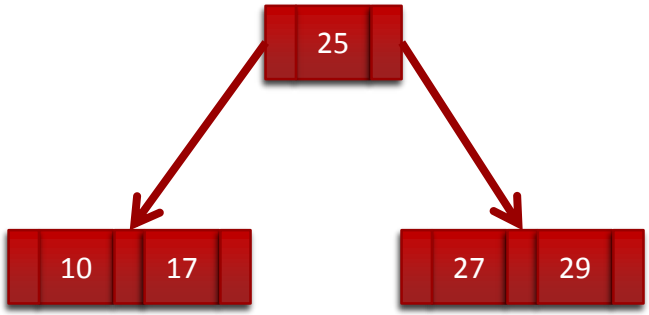
5) Se inserta la clave 25

Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.

Ejemplo 14

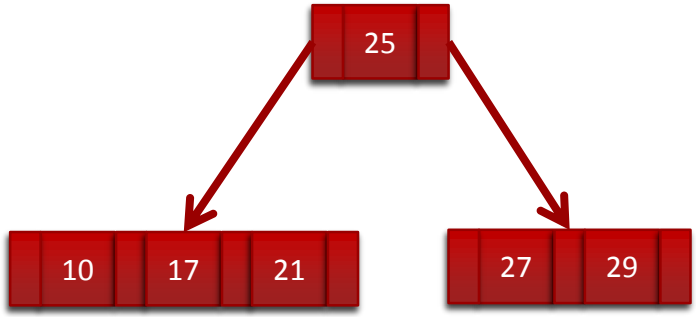
5) Se inserta la clave 25

Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.

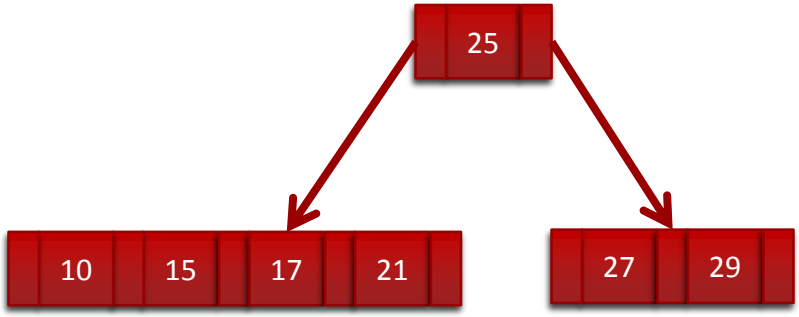


Ejemplo 14

6) Se inserta la clave 21

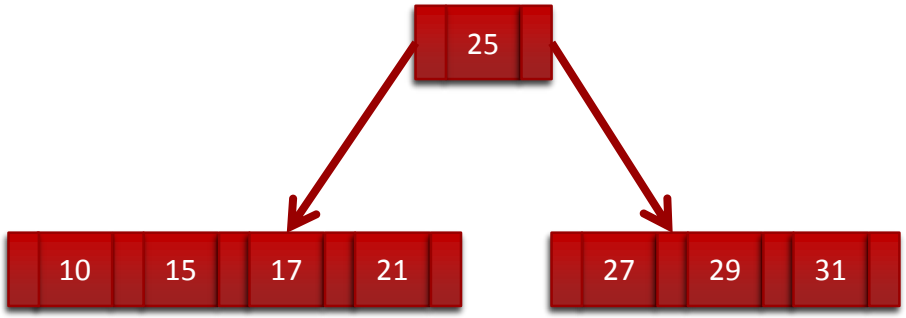


7) Se inserta la clave 15



Ejemplo 14

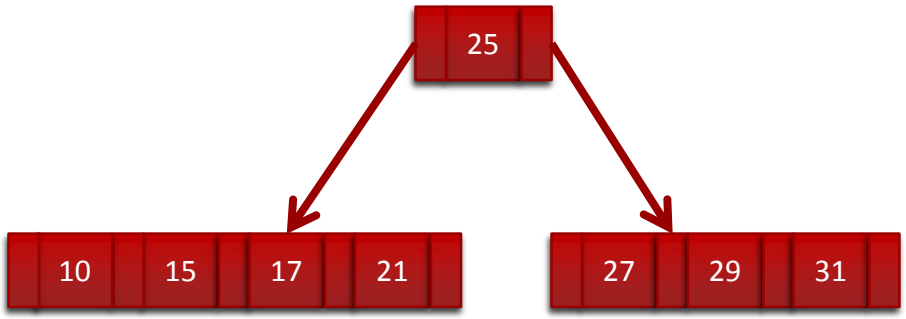
8) Se inserta la clave 31



Ejemplo 14

9) Se inserta la clave 13

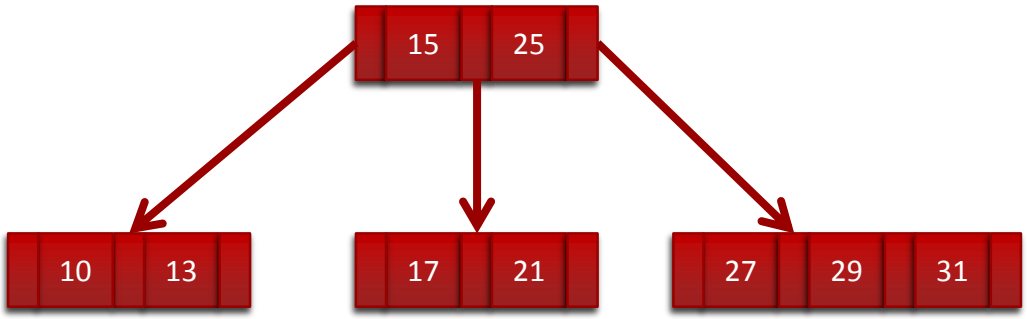
Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



Ejemplo 14

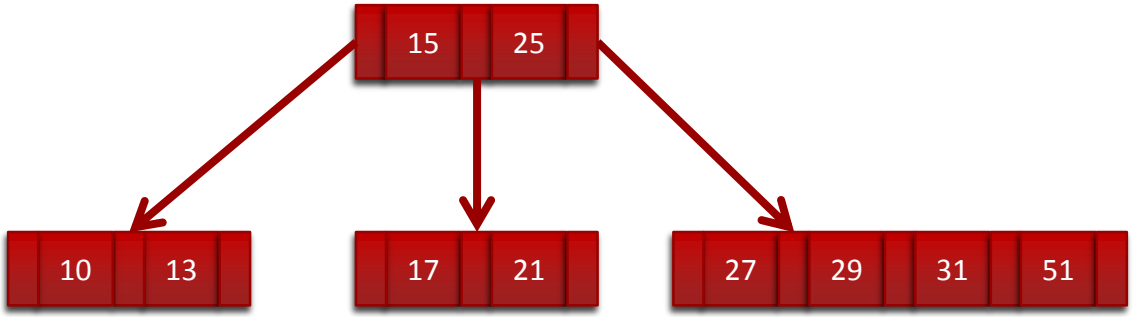
9) Se inserta la clave 13

Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.

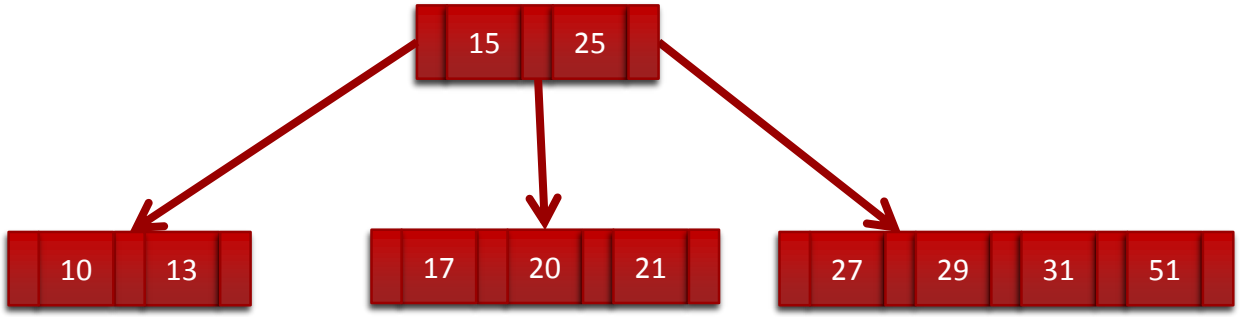


Ejemplo 14

10) Se inserta la clave 51

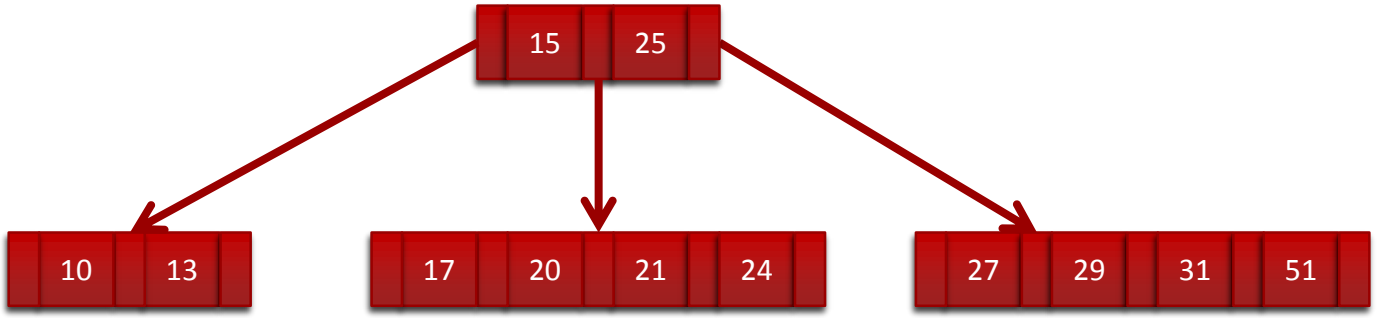


11) Se inserta la clave 20



Ejemplo 14

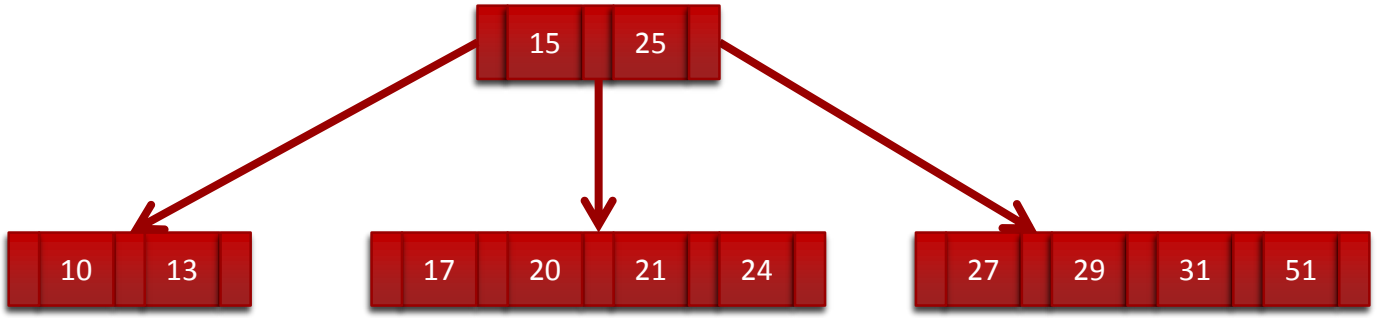
12) Se inserta la clave 24



Ejemplo 14

13) Se inserta la clave 48

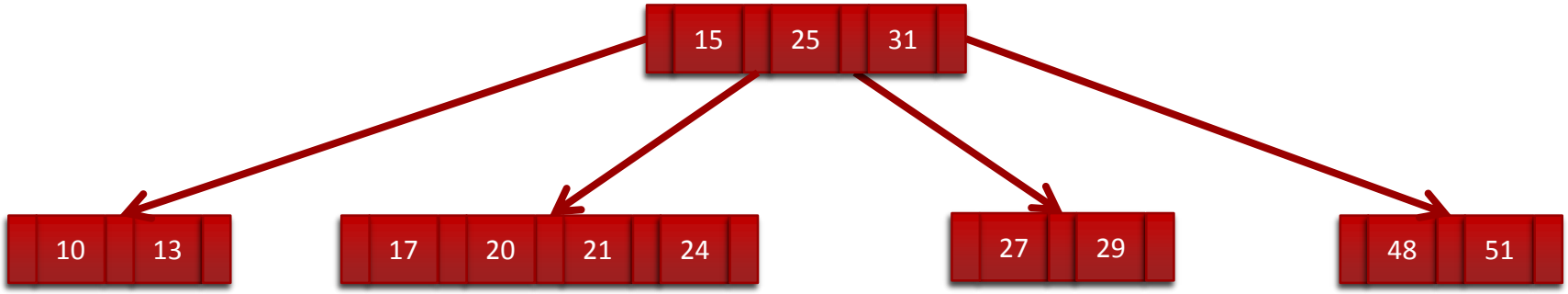
Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



Ejemplo 14

13) Se inserta la clave 48

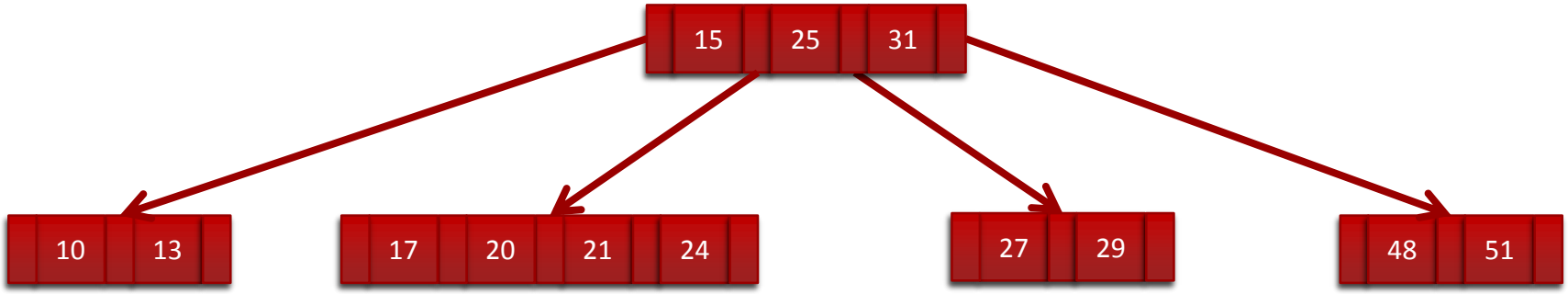
Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



Ejemplo 14

14) Se inserta la clave 19

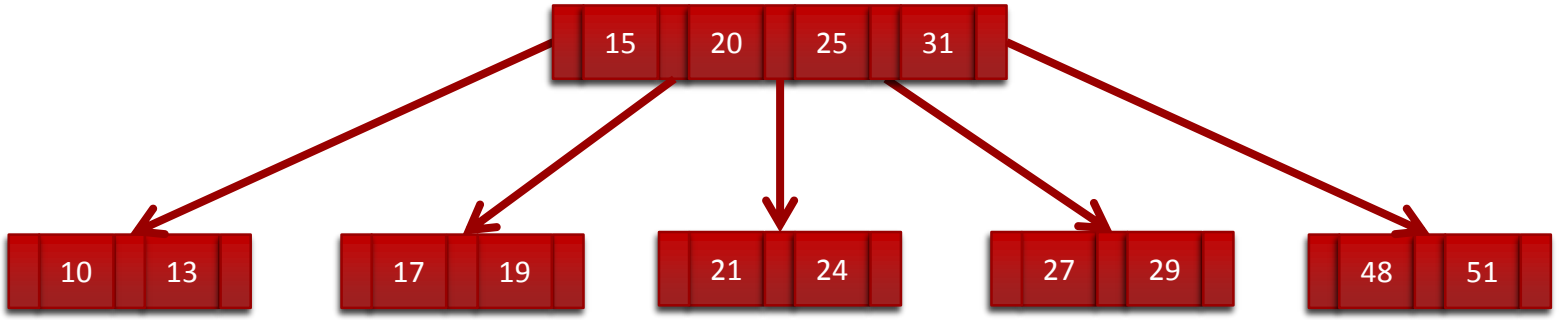
Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



Ejemplo 14

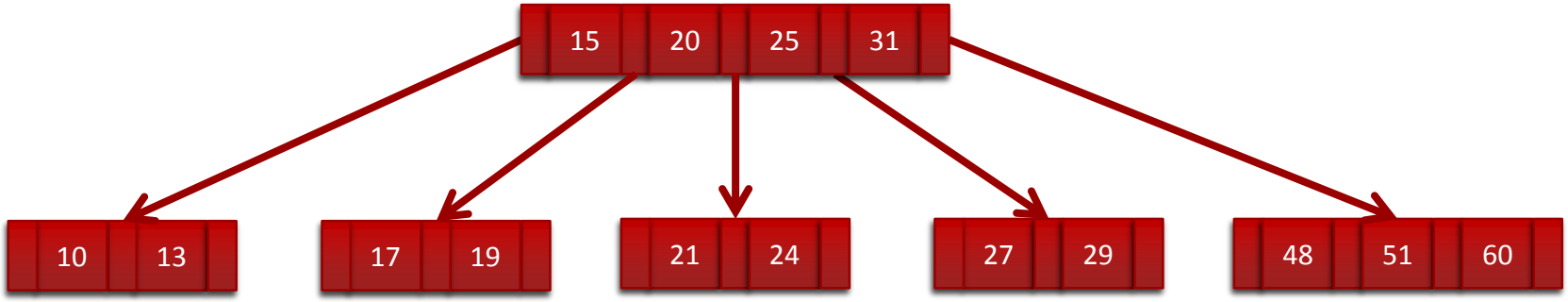
14) Se inserta la clave 19

Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



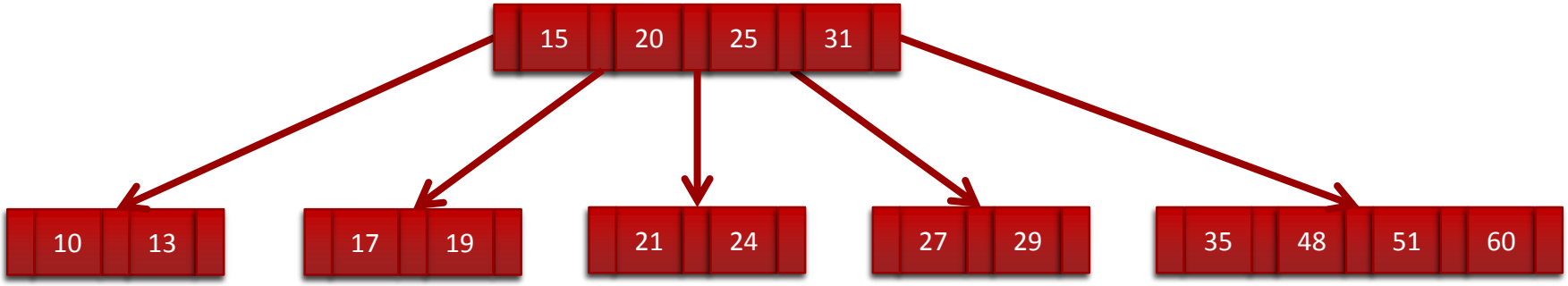
Ejemplo 14

15) Se inserta la clave 60



Ejemplo 14

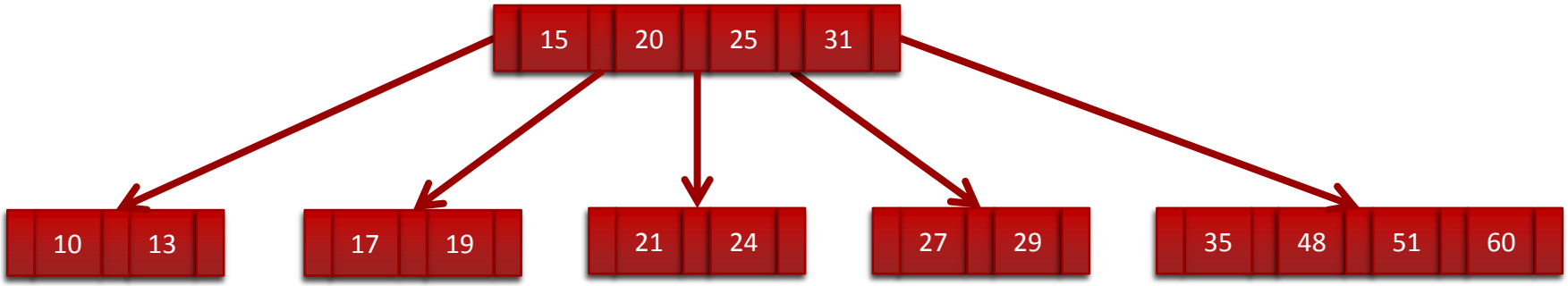
16) Se inserta la clave 35



Ejemplo 14

17) Se inserta la clave 66

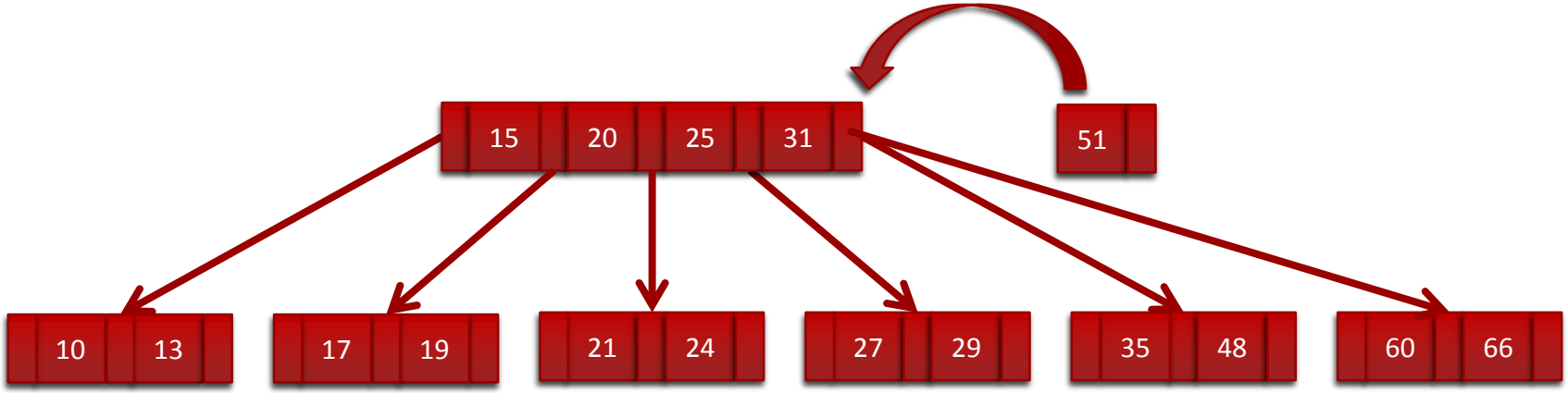
Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



Ejemplo 14

17) Se inserta la clave 66

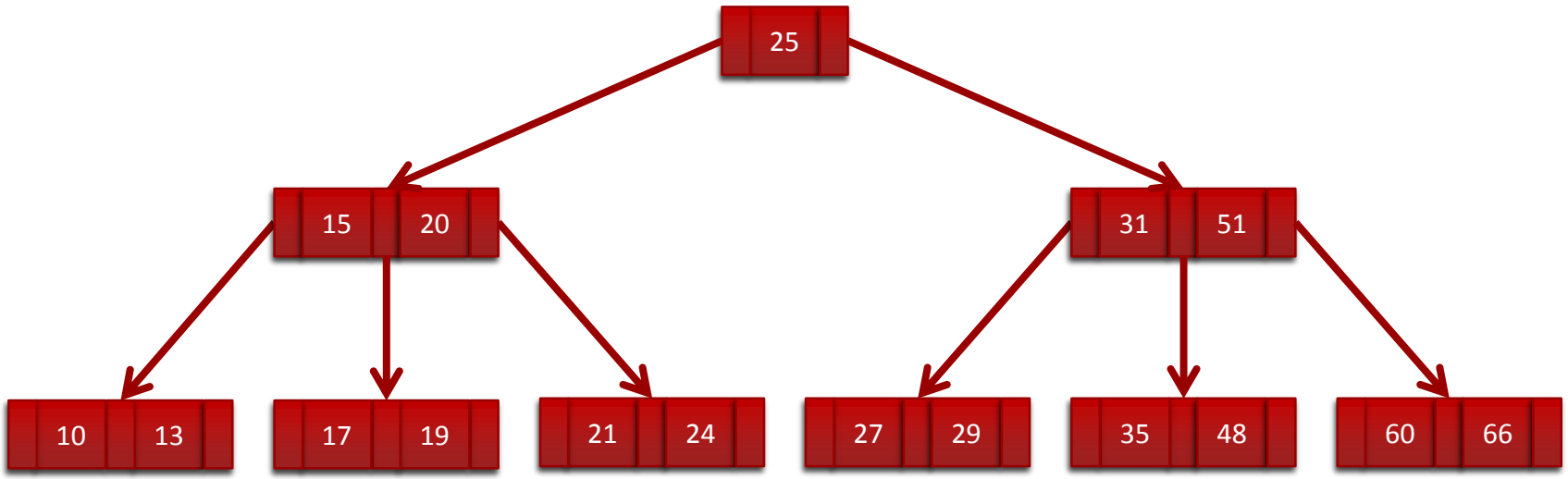
Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



Ejemplo 14

17) Se inserta la clave 66

Como m es igual a 2*d, la página afectada se divide en dos y se distribuyen las m+1 claves de manera equitativa. La clave de en medio sube a la página antecesora.



Eliminación en árboles B

Cuando se elimina un elemento de un árbol B, se debe tener cuidado de no violar la condición del número de elementos del árbol, es decir, por página debe haber entre d y 2*d elementos (con excepción de la raíz). Por tanto, es una operación más complicada aún que la inserción.

Para eliminar un elemento en los árboles B se siguen los siguientes pasos:

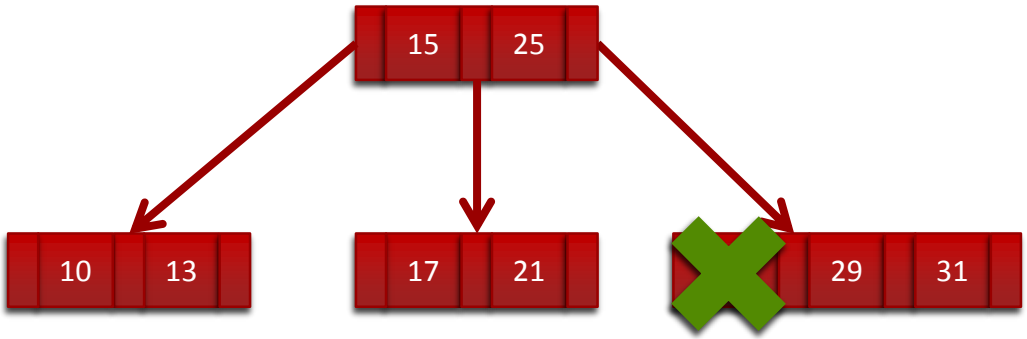
1. Si el elemento a eliminar se encuentra en una página hoja:
    Si el número de elementos de la página (m) es mayor a d
        Se suprime el elemento
    De lo contrario
        Se baja el elemento adyacente de la página antecesora y se sustituye por la clave que se encuentre más a la derecha en el subárbol izquierdo o más a la izquierda en el subárbol derecho.
        Si lo anterior no se puede hacer, se deben unir las páginas que son descendentes directos del elemento que se baja.

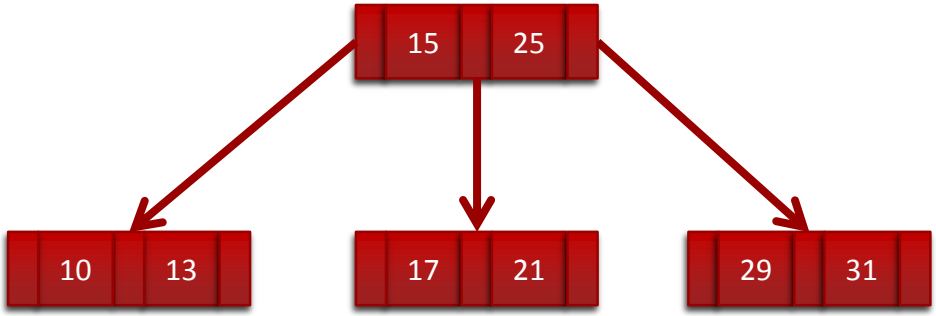
2. Si el elemento a eliminar no se encuentra en una página hoja

Se sustituye por el elemento que se encuentra más a la izquierda del subárbol derecho o por el elemento que se encuentra más a la derecha del subárbol izquierdo.

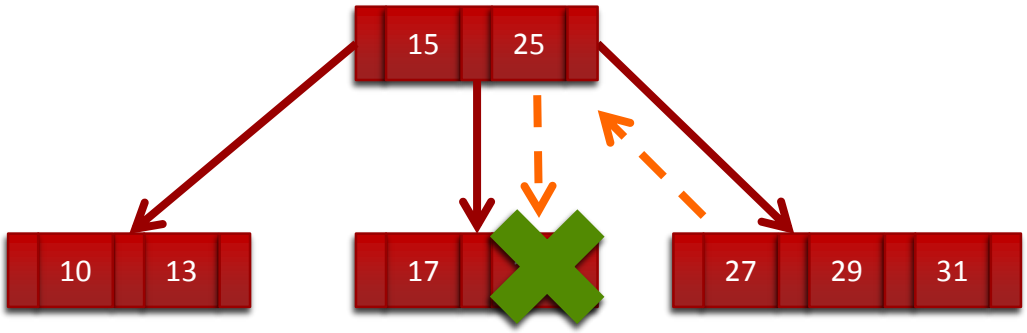
(Se debe mantener $d \leq m \leq 2*d$)

Eiminar la clave 27: Si el nodo es una hoja y m mayor a d simplemente se elimina.

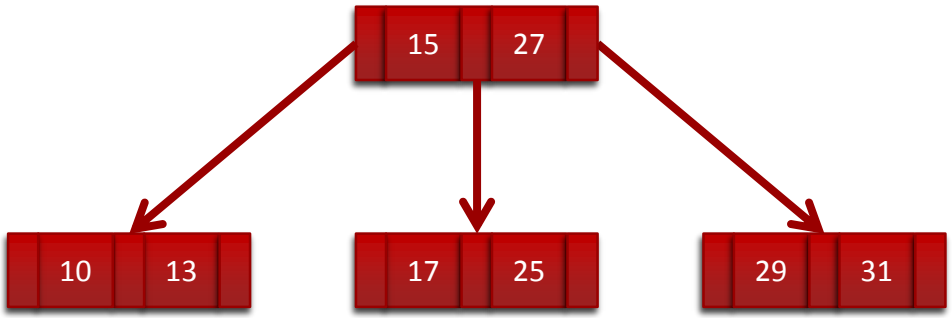




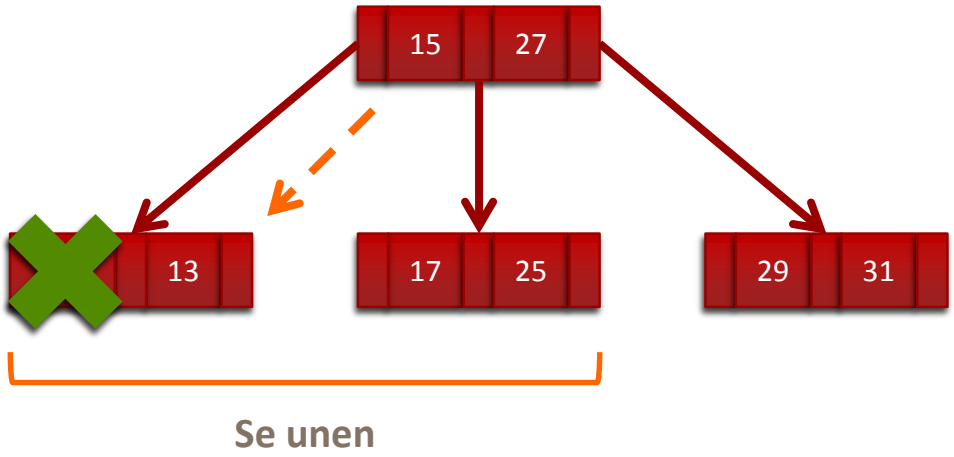
Eiminar la clave 21: Si el nodo es una hoja simplemente pero m es menor o igual a d, se baja el elemento adyacente de la página antecesora y se sustituye por la clave que se encuentre más a la derecha en el subárbol izquierdo o más a la izquierda en el subárbol derecho.



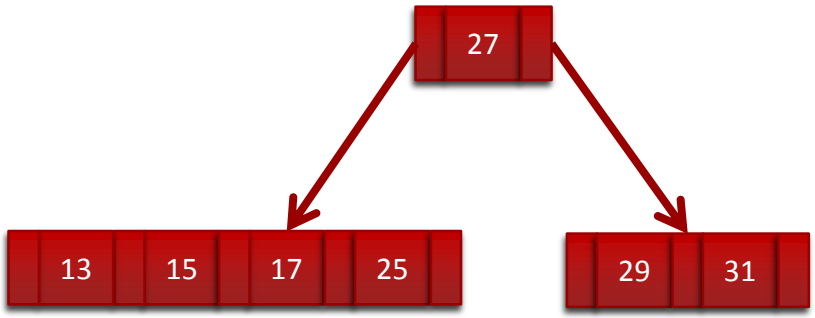
Eiminar la clave 21: Si el nodo es una hoja simplemente pero m es menor o igual a d, se baja el elemento adyacente de la página antecesora y se sustituye por la clave que se encuentre más a la derecha en el subárbol izquierdo o más a la izquierda en el subárbol derecho.



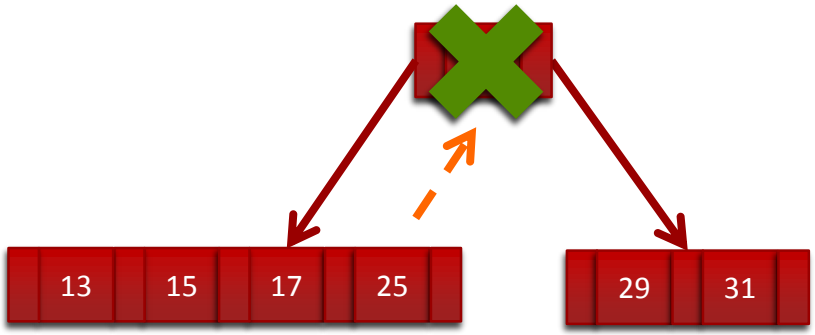
Eiminar la clave 10: Si el nodo es una hoja simplemente pero m es menor o igual a d, como lo anterior no se puede hacer, se deben unir las páginas que son descendentes directos del elemento que se baja.



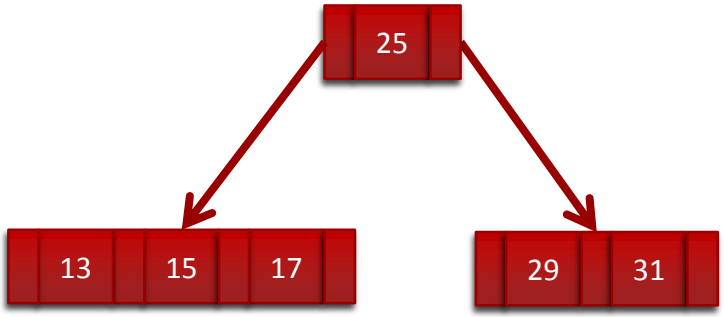
Eiminar la clave 10: Si el nodo es una hoja simplemente pero m es menor o igual a d, como lo anterior no se puede hacer, se deben unir las páginas que son descendentes directos del elemento que se baja.



Eiminar la clave 27: Si el elemento a eliminar no se encuentra en una página hoja, se sustituye por el elemento que se encuentra más a la izquierda del subárbol derecho o por el elemento que se encuentra más a la derecha del subárbol izquierdo.



Eiminar la clave 27: Si el elemento a eliminar no se encuentra en una página hoja, se sustituye por el elemento que se encuentra más a la izquierda del subárbol derecho o por el elemento que se encuentra más a la derecha del subárbol izquierdo.



Árboles B+

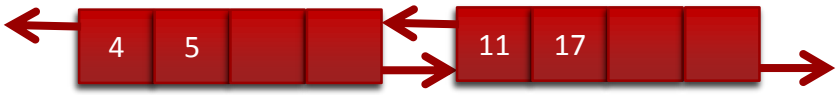
Los arboles B+ son una variante de los arboles B, con la diferencia de que en los arboles B+ toda la informacion se encuentra almacenada en las hojas. En la raiz y en las paginas internas se almacenan indices o claves para llegar a los datos en las hojas.

El árbol B+ combina la estructura de un árbol B con un conjunto secuencial, permitiendo el acceso a la información tanto por referencia como de manera secuencial.

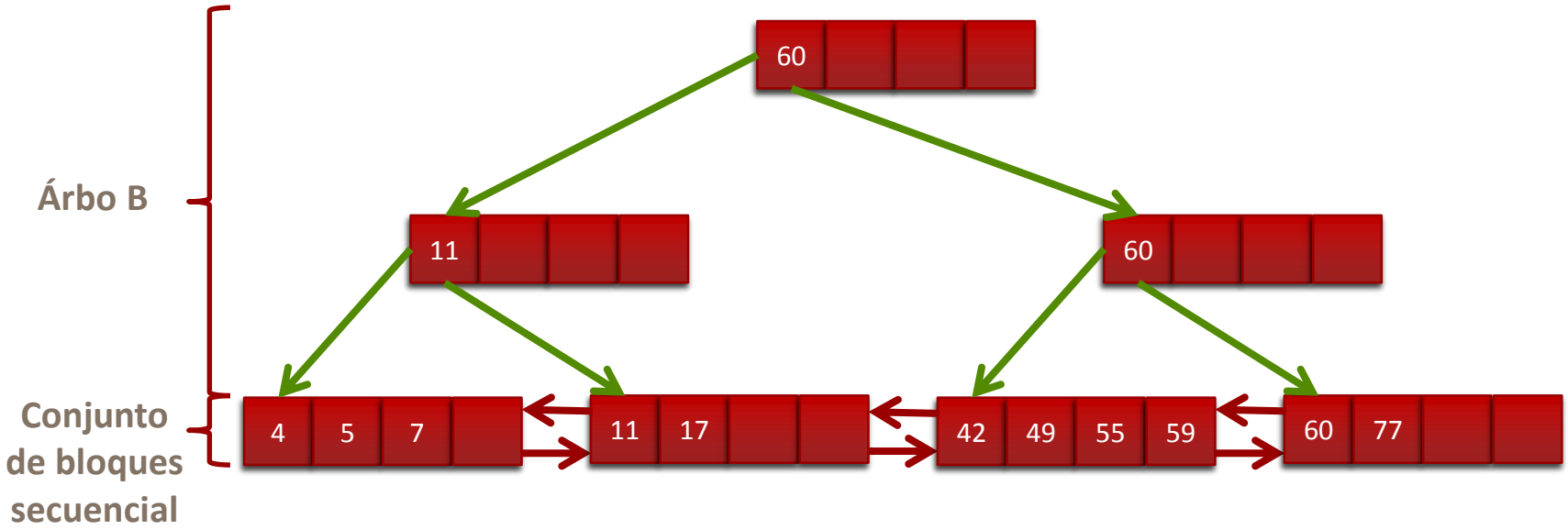
Un conjunto secuencial es un bloque de tamaño fijo que contiene un número máximo de elementos ordenados con base en una clave. Además, cada bloque cuenta con una referencia al elemento siguiente y al anterior.

Los bloques se trabajan de manera idéntica a como se realiza en los árboles B de tal manera que simpre se debe cumplir $d \leq m \leq 2*d$.

Esta estructura permite recorrer secuencialmente todas las claves por bloque y, cuando se alcanza el final de éste, pasar al siguiente de manera inmediata.

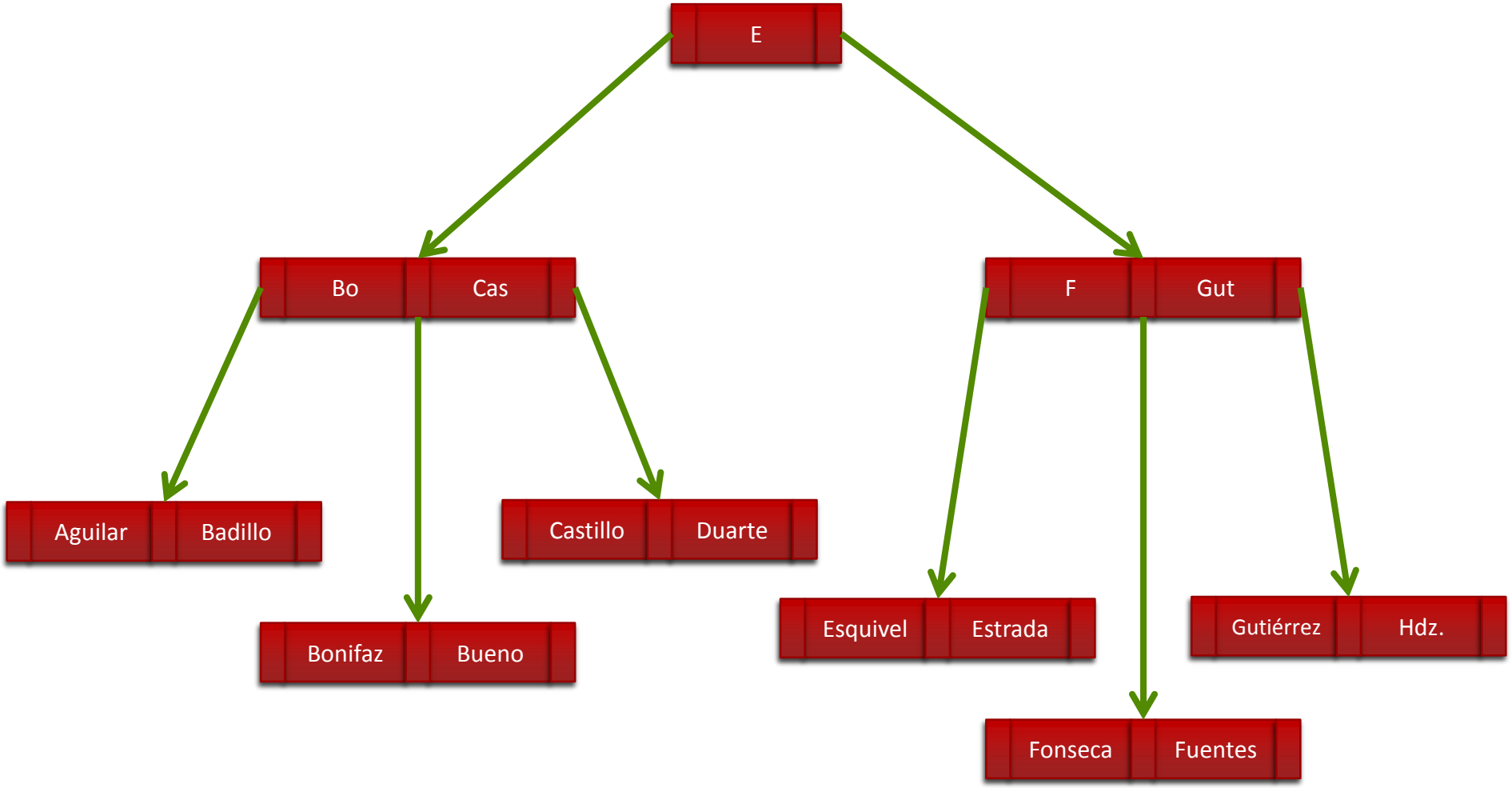


Sobre el conjunto de bloques secuencial se construye un conjunto indexado (árbol B), donde cada nodo interno o raíz del árbol puede contener un separador que permite saber qué ruta seguir para encontrar el elemento deseado.



Prefijos simples

Los separadores pueden ser una clave o un prefijo que permita hacer la separación. A los árboles B+ que guardan prefijos para separar en vez de claves completas se les llama árboles B+ de prefijo simple y utilizan un algoritmo para determinar el prefijo más corto que pueda usarse como separador en cada caso.


E
Bo
Cas
F
Gut
Aguilar
Badillo
Castillo
Duarte
Bonifaz
Bueno
Esquivel
Estrada
Gutiérrez
Hdz.
Fonseca
Fuentes

Inserción

Para realizar una inserción se debe identificar el bloque donde se desea ingresar el dato, para ello se busca el bloque a través del árbol B.

Si el bloque donde encontrado no está lleno, se inserta la clave y se verifica que el separador siga siendo válido, si no es así se actualiza (cuando se inserta la clave al principio de un bloque).

Si el bloque está lleno se debe realizar una separación en la cual no se promueven claves sino que todas se distribuyen en los dos bloques resultantes. Una vez distribuidas las claves se selecciona un separador y éste es el que se promueve como índice.

Este proceso se repite hasta que el separador no genere un sobrecupo en el bloque interno, es decir, mientras el nodo superior no esté lleno (2*d).

Eliminación

Las eliminación se realiza sobre un bloque secuencial. Si al hacer una baja el bloque es menor a d, se intenta redistribuir las claves con los bloques vecinos y, si se puede redistribuir, se actualiza el separador de bloques.

Si no puede redistribuirse porque los bloques contiguos tienen la cantidad mínima de claves se hace una concatenación y se elimina del árbol el separador actual de los bloques, este proceso se sigue hacía arriba hasta que todos los nodos cumplan con $d \leq m \leq 2*d$.

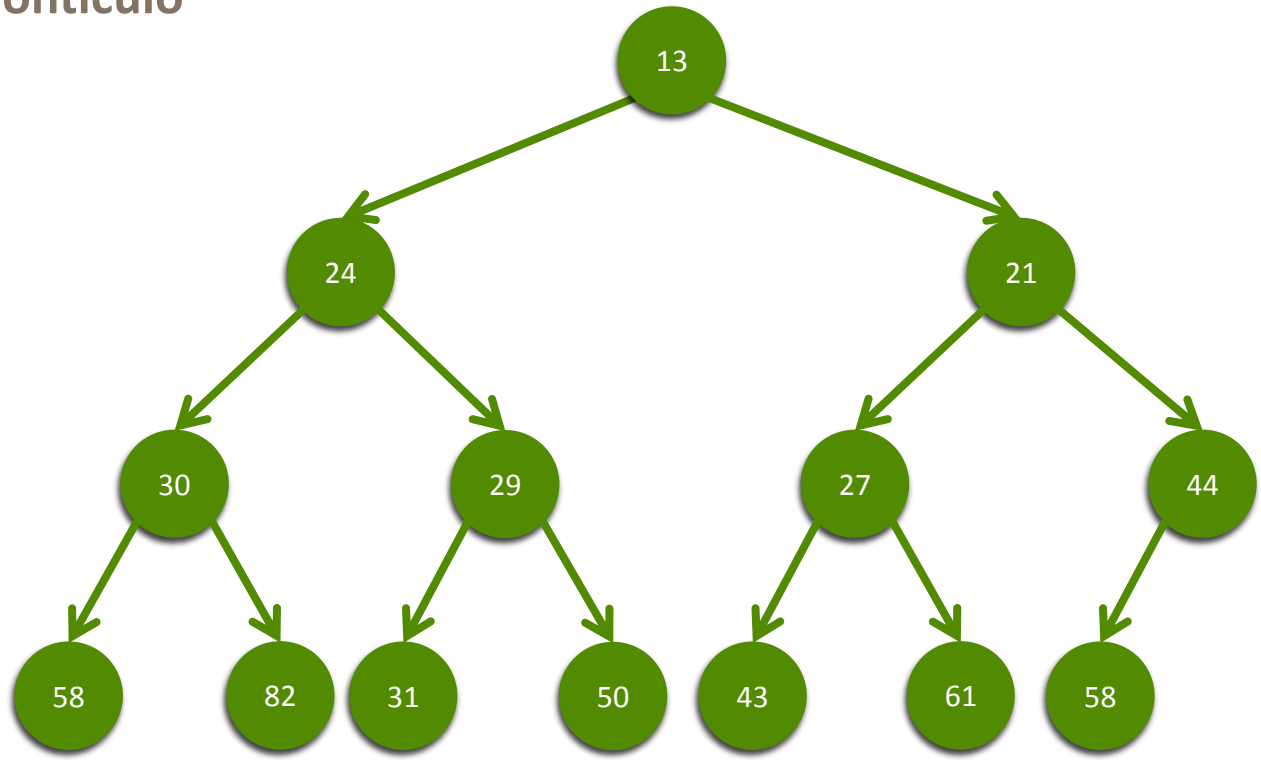
Si al eliminar un elemento, su clave se encuentra, no es necesario eliminarla, ya que el índice solo actúa como separador de elementos.

Montículo (Heap)

Un montículo es una estructura de datos ordenadas en forma de gráfo, más específicamente, en forma de un árbol binario balanceado.

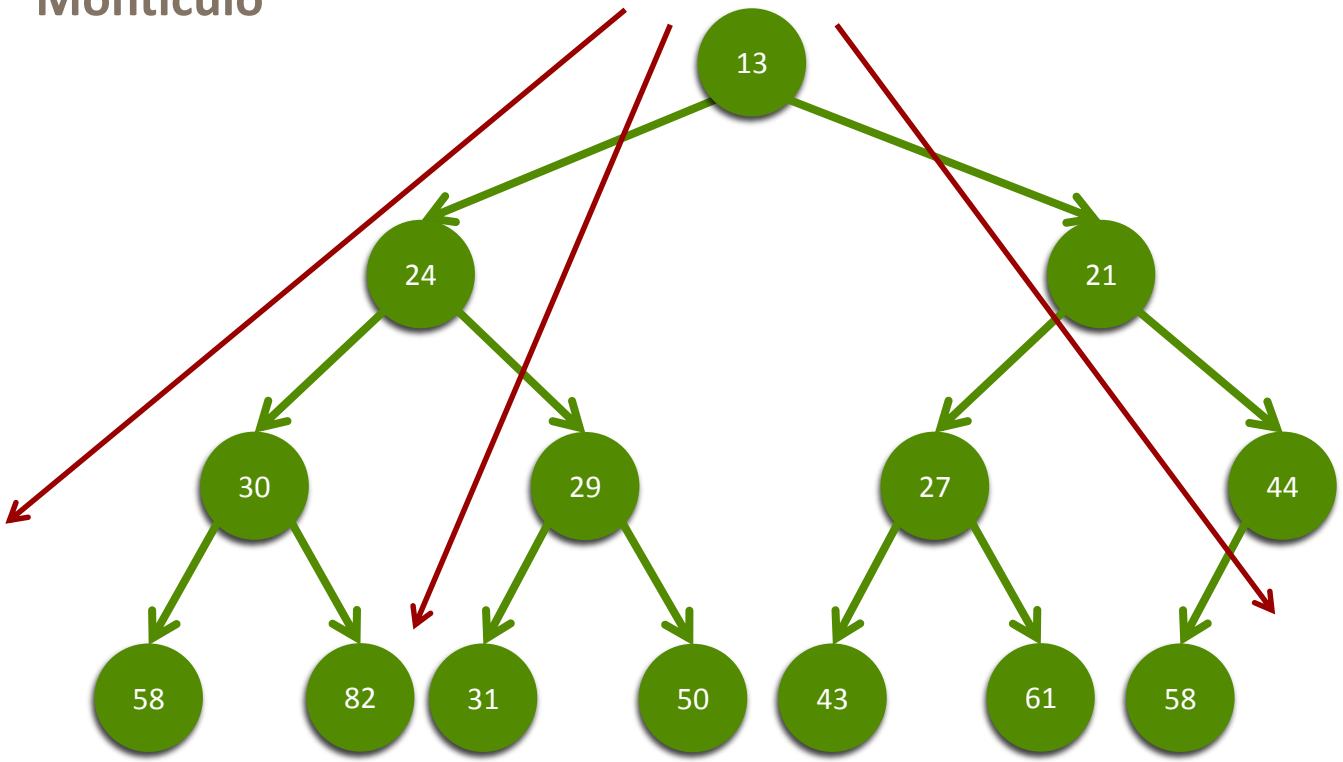
Cada nodo del montículo posee un valor que cumple con la propiedad de que cada nodo en el árbol no debe ser mayor que los valores de los hijos.

Montículo



Como se puede observar, un montículo no está ordenado, la única propiedad que se cumple es que los hijos de cualquier nodo son menores en todo momento. Esto se pude comprobar leyendo el árbol de arriba hacia abajo.

Montículo



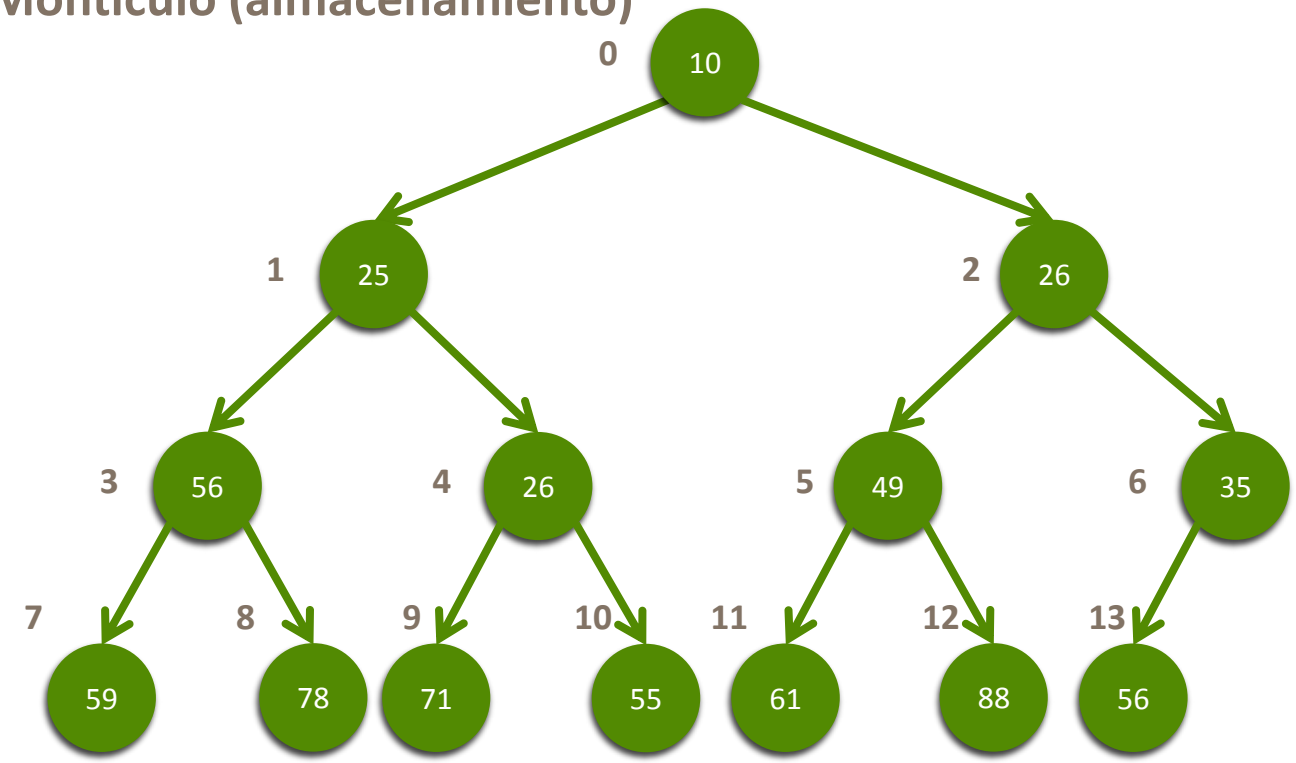
El almacenamiento dentro de un montículo se lleva a cabo de izquierda a derecha de arriba hacia abajo.

Por tanto, la altura de la raíz a cualquier hoja es de orden logarítmico (O (log n)).

Otra consecuencia de la propiedad de los montículos es que el valor de menor tamaño se encuentra en la raíz y, por tanto, es fácil de encontrar (O(1)).

Debido a que un montículo se acomoda de la misma manera que un árbol binario, es posible almacenar esta estructura dentro de una estructura lineal.

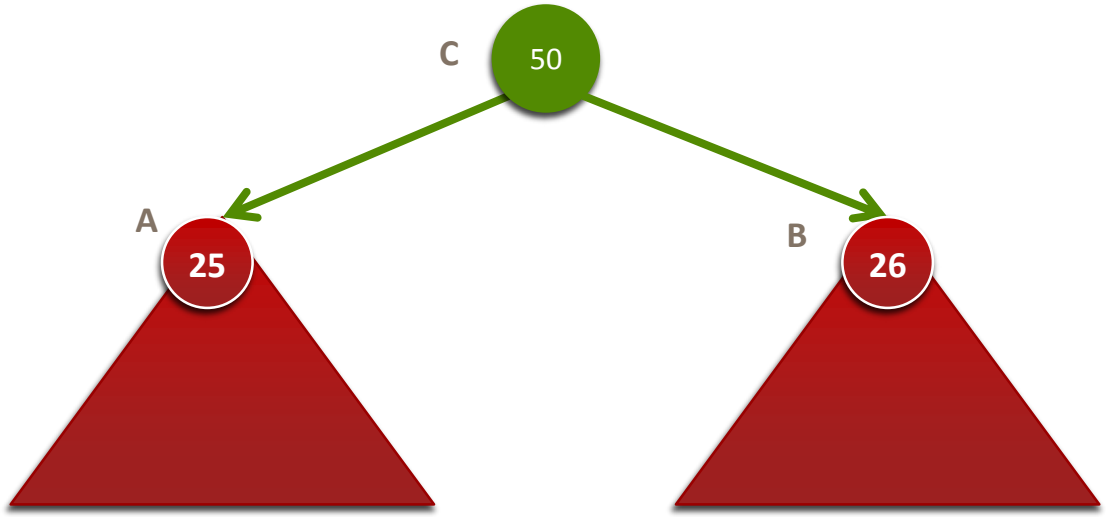
Montículo (almacenamiento)



La inserción de un dato genera un conjunto de movimientos (parecidos a los que se requieren en un árbol binario), pero en sentido inverso, es decir, se parte desde la raíz y se va recorriendo el montículo de arriba hacia abajo tratando de encontrar un lugar donde el nuevo dato no rompa la propiedad de la estructura.

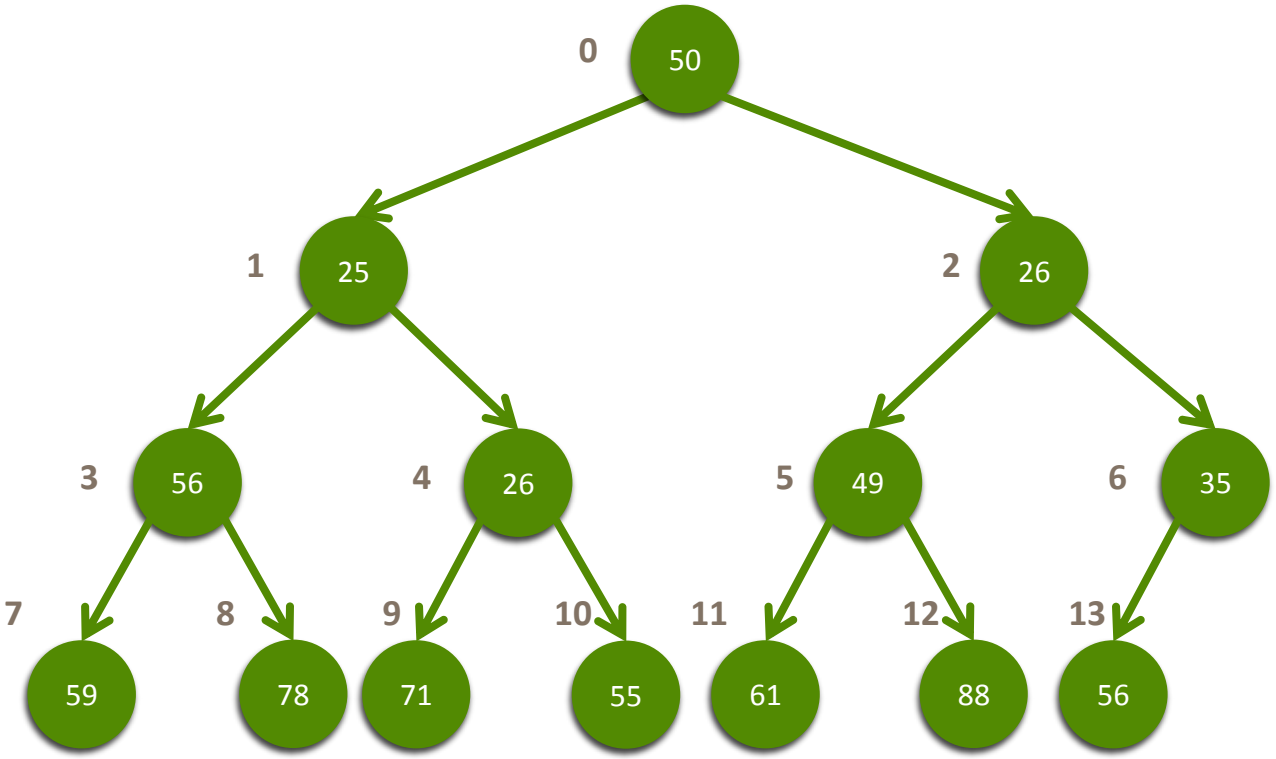
Ejemplo

Se tienen dos montículos A y B, y se quieren unir a un montículo C que consta de un solo elemento.

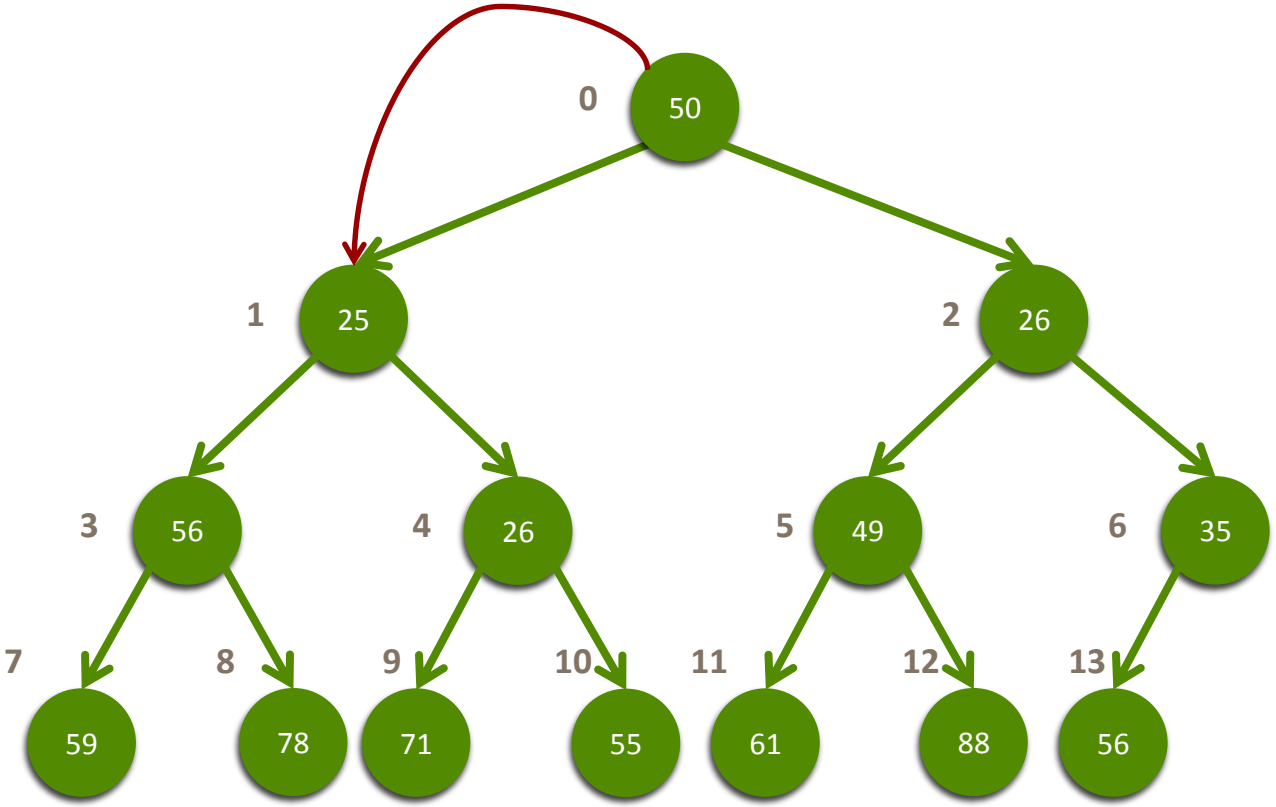


Ejemplo

El valor que conviene cambiar para realizar el reacomodo es el de menor tamaño, es decir, en este caso cambiar el nodo con valor 50 por el nodo con valor 25.

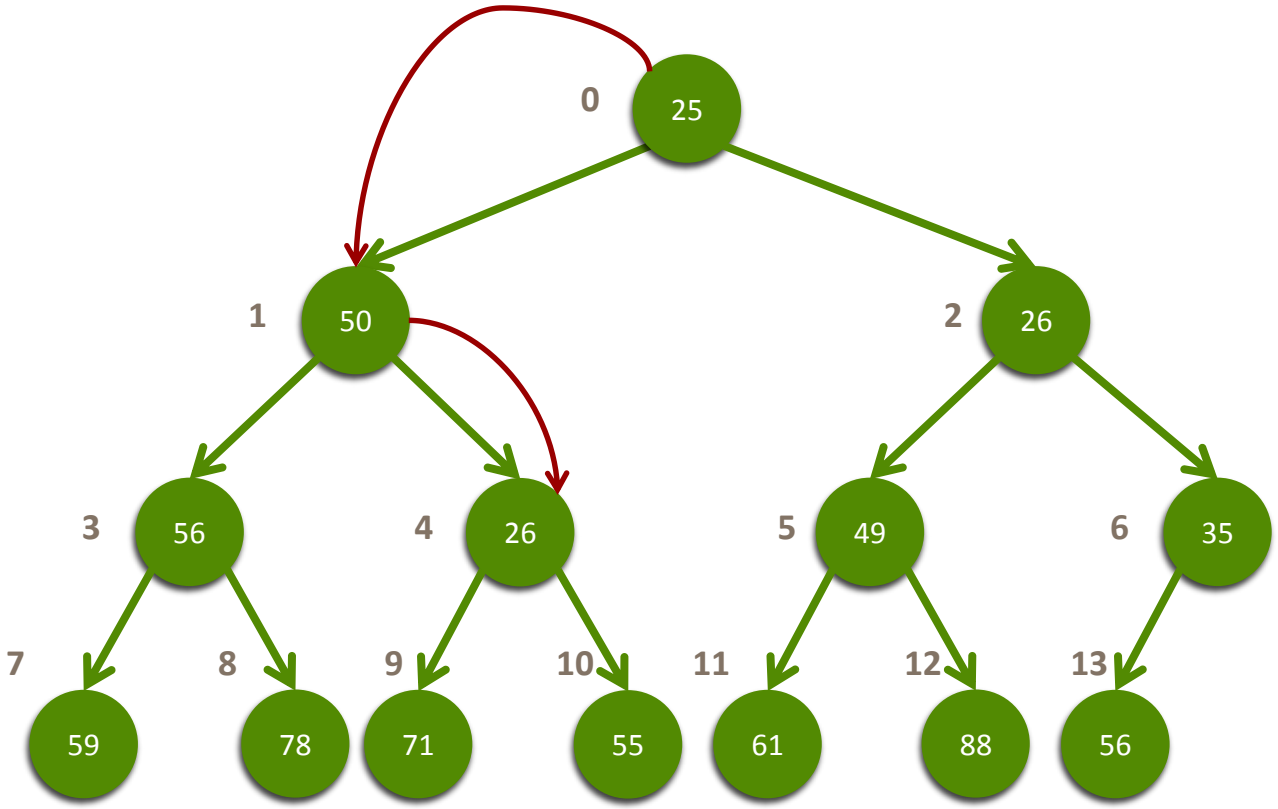


Ejemplo

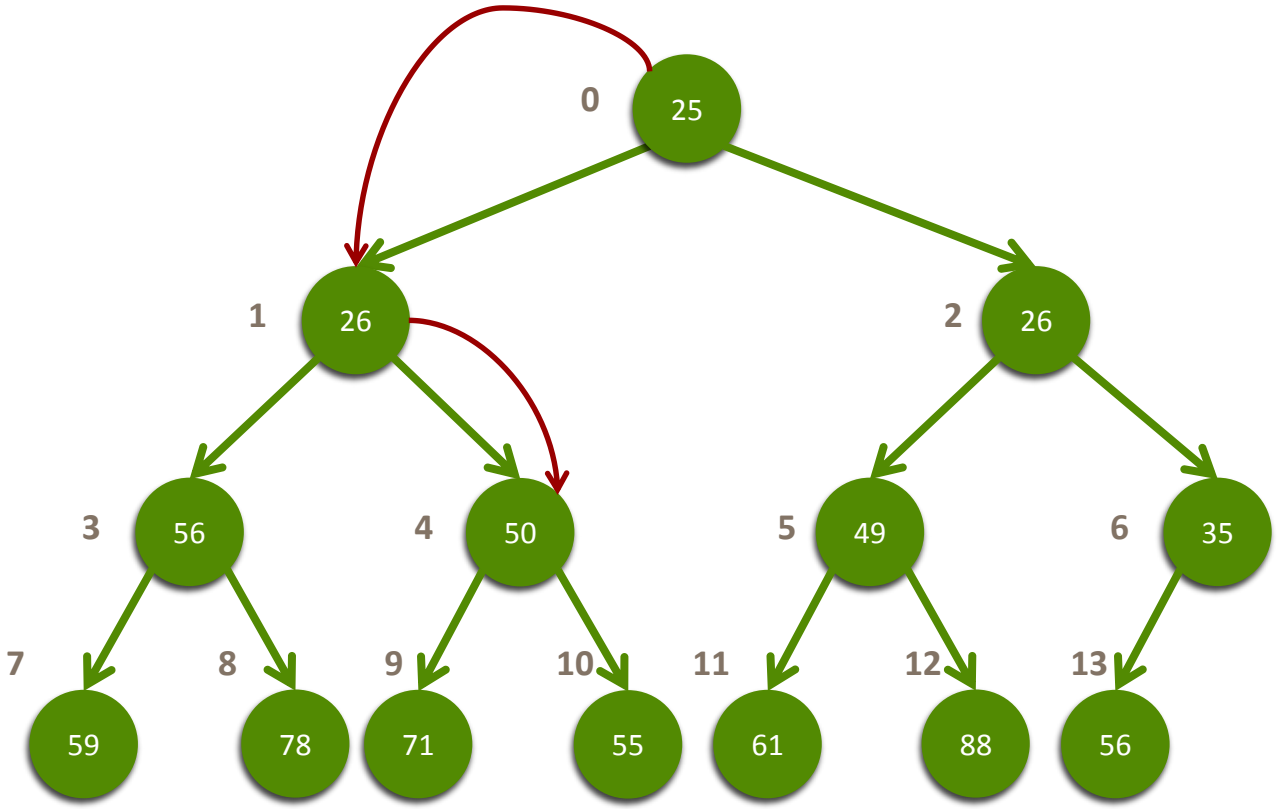


Este proceso se debe repetir (recursivamente) hasta que ninguno de los hijos sea menor al valor.

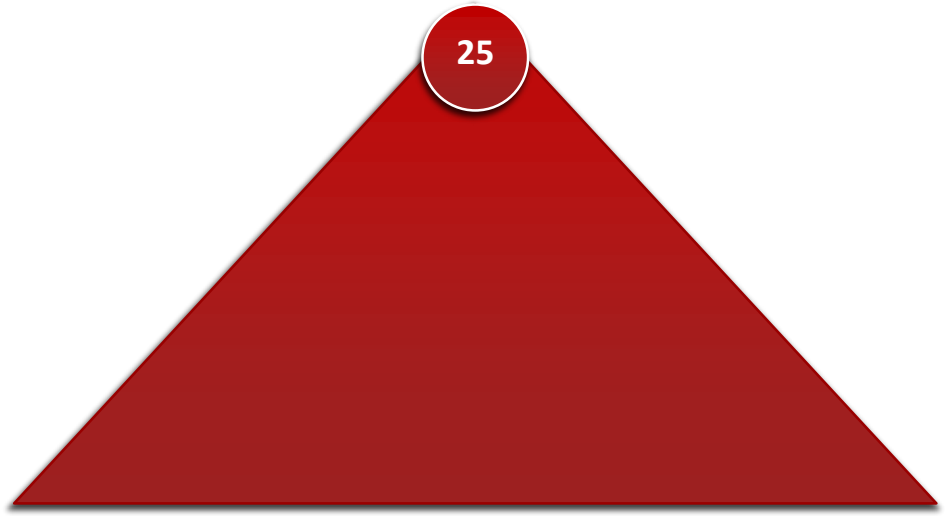
Ejemplo



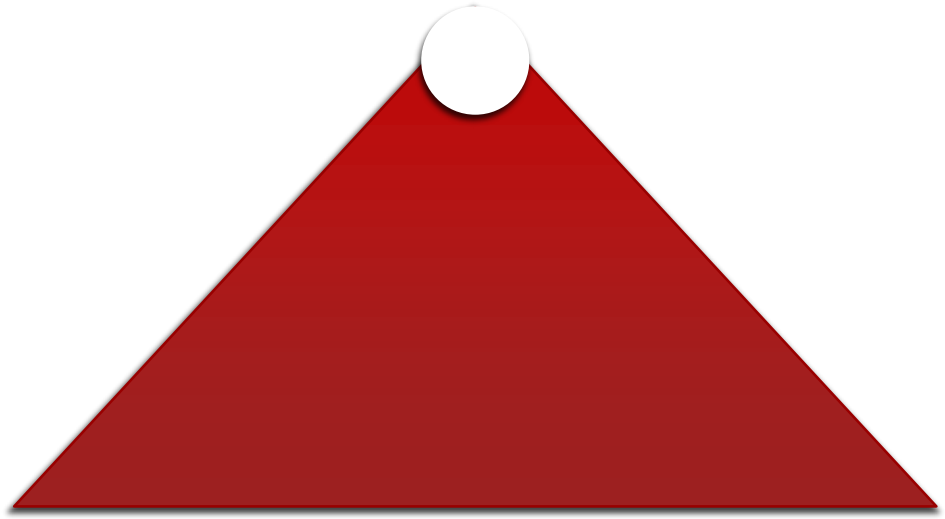
Ejemplo



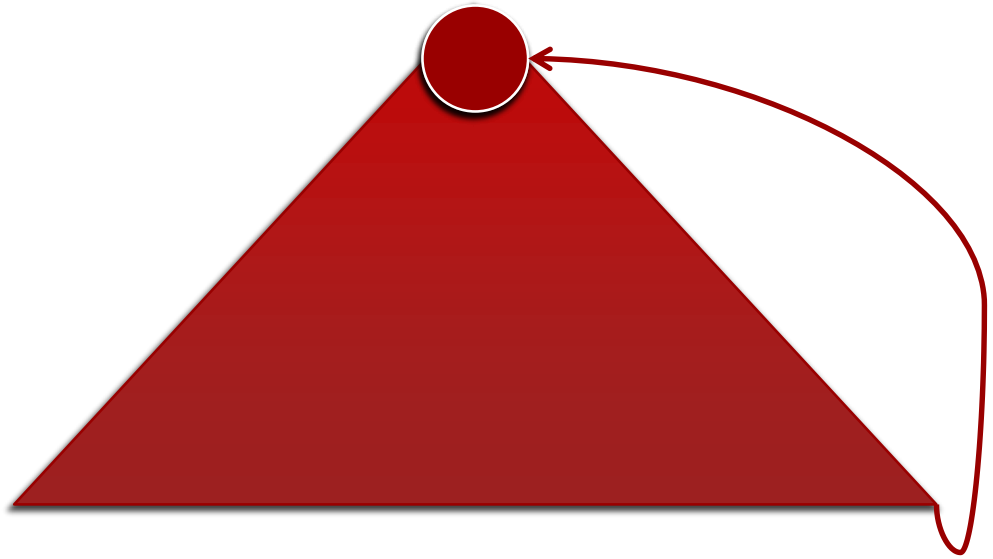
Por lo visto anteriormente, eliminar u obtener el valor de menor tamaño del montículo es rápido (O(1)).



Sin embargo, una vez eliminado el elemento superior el montículo queda sin el elemento raíz.



Para conservar la características de montículo, se debe obtener el último elemento de la estructura, insertarlo en la raíz e ir reacomodando el nodo según corresponda.



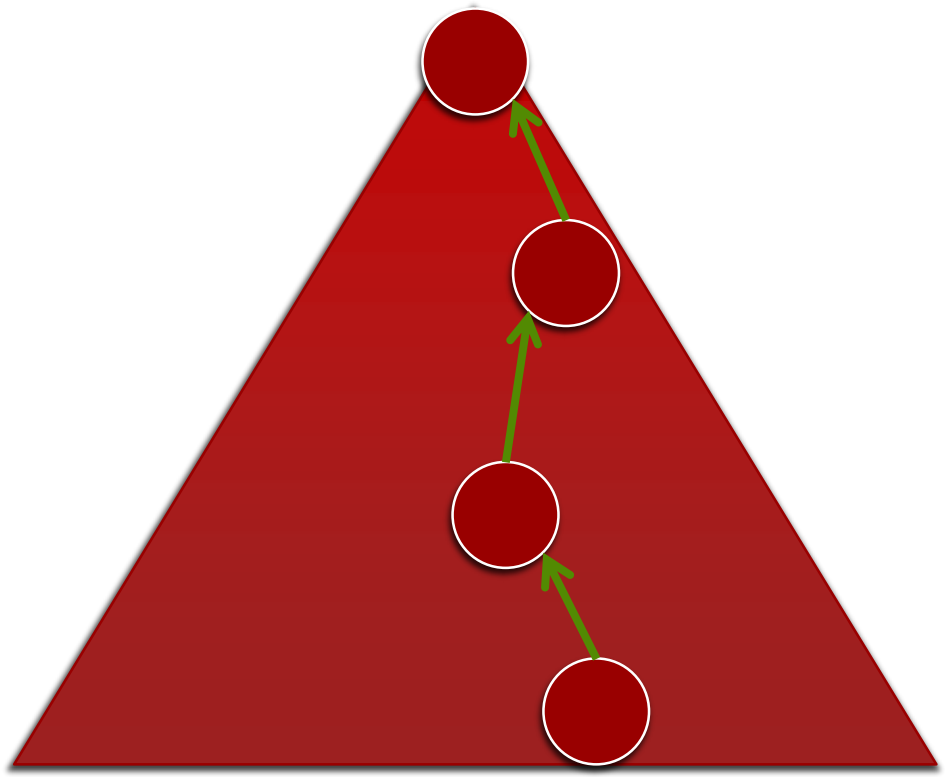
Para terminar se verifica la posición que le corresponde al nodo como se vió en el ejemplo anterior.

Por lo tanto, los 3 pasos para realizar una eliminación son:

- Eliminar el elemento heap[0] (O(1)).
- Copiar el elemento heap[n-1] a heap[0] (O(1)).
- Reacomodar el montículo (O(log n)).

Como se puede observar, el tiempo que ocupa un montículo para eliminar un elemento es O(log n).

Para insertar un elemento en el montículo se debe realizar en la última posición e ir moviéndolo hasta alcanzar la posición adecuada.



Estructuras de datos principales

Estructura	Ventajas	Desventajas
Arreglo	Rápida inserción. Acceso rápido a la información si se conoce el índice.	Búsqueda lenta. Eliminación lenta. Tamaño fijo.
Arreglo ordenado	Búsqueda más rápida que el arreglo.	Inserción lenta. Eliminación lenta. Tamaño fijo.
Pila	Provee acceso directo al último elemento insertado.	Acceso lento a otros elementos.

Estructura	Ventajas	Desventajas
Cola	Provee acceso directo al primer elemento insertado.	Acceso lento a otros elementos.
Lista ligada	Rápida inserción. Rápida eliminación.	Búsqueda lenta.
Árbol binario	Búsqueda rápida. Inserción rápida. Eliminación rápida.	El algoritmo de eliminación es lento.
Árbol rojo-negro	Búsqueda rápida. Inserción rápida. Eliminación rápida.	Complejo.

Estructura	Ventajas	Desventajas
Árbol 2-3-4	Búsqueda rápida. Inserción rápida. Eliminación rápida. Árbol siempre balanceado.	Complejo.
Tabla Hash	Acceso rápido si se conoce la clave. Rápida inserción.	Eliminación lenta. Acceso lento si no se conoce la clave. Uso ineficiente de memoria.
Montículo (heap)	Inserción rápida. Eliminación rápida.	Acceso lento a otros elementos.
Gráfo o gráfica	Modela situaciones del mundo real.	Algunos algoritmos son lentos y complejos.

4 Estructuras de datos no lineales: listas no lineales

Objetivo: Aplicar las formas de representar y operar en la computadora las principales listas no lineales.

4.1 Generalidades (Grafos).
4.2 Árboles.
4.3 Árboles binarios.
4.4 Árboles B.