ALGORITMOS Y ESTRUCTURAS DE DATOS

Unidad Nro. 1: Archivos.

Contenidos: Definición y características generales. Organizaciones secuencial, secuencial indexada, directa e indexada. Acceso a archivos. Diseño de archivos. Aplicaciones. Análisis de costo y beneficio. Normalización.

Unidad Nro. 2: Modelos matemáticos y representaciones computacionales.

Contenidos: Definición de estructura de datos. Modelos matemáticos. Representaciones. Teoría de grafos. Grafos dirigidos. Clasificación. Paso. Camino. Hipergrafos. Aplicaciones.

Unidad Nro. 3: Estructuras lineales.

Contenidos: relaciones lineales y listas, array, stack, cola, doble cola, listas, multilistas. Representaciones linkeadas, algoritmos. Representaciones alternativas de estructuras lineales. Costos, probabilidad y error.

Unidad Nro. 4: Árboles.

Contenidos: Definición, propiedades, formalización de conceptos. Árboles binarios. Barrido de árboles binarios. Árboles n-arios, representación. Transformaciones. Árboles balanceados. Árboles AVL. Multimodales. Árboles B.

Unidad Nro. 5: Grafos irrestrictos y Redes.

Contenidos: Estructuras irrestrictas. Diseño de celdas. Búsquedas. Sort topológico. Algoritmo de Warshall.

Unidad Nro. 6: Introducción a la Algoritmia.

Contenidos: Definición. Eficiencia de los algoritmos. Problemas. Notación asintótica. Análisis de algoritmos. Estructuras de control. Caso medio. Resolución de recurrencias.

Unidad Nro. 7: Algoritmos.

Contenidos: Algoritmos voraces. Algoritmos binarios. Búsqueda en grafos. Algoritmos probabilistas. Algoritmos paralelos. Complejidad computacional.

UNIDAD NRO. 1: ARCHIVOS

DEFINICIÓN:

Un *archivo* o *fichero* es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas *registros* o *artículos* (relacionados entre sí) que son de igual tipo y que constan, a su vez, de diferentes entidades de nivel más bajo denominadas *campos*.

OTROS CONCEPTOS SOBRE ARCHIVOS:

Campo: un campo es la unidad mínima de información de un registro o también se dice que es un *ítem* o *elemento de datos elementales*, tal como un nombre, número de empleado, ciudad, DNI, etc.

Un campo está caracterizado por su tamaño o longitud y su tipo de dato (cadena de caracteres, entero, lógico, etc.). Incluso los campos pueden variar en su longitud.

Los datos contenidos en un campo se pueden dividir en *subcampos*, por ejemplo, el campo fecha se divide en los subcampos día, mes, año.

Registro: un registro es una colección de información, normalmente relativa a una entidad particular. También podría decirse que es una colección de campos lógicamente relacionados que pueden ser tratados como una unidad por algún programa.

Un ejemplo de un registro puede ser la información de un determinado empleado, que contiene los campos de nombre, dirección, fecha de nacimiento, estudios, salario, etc.

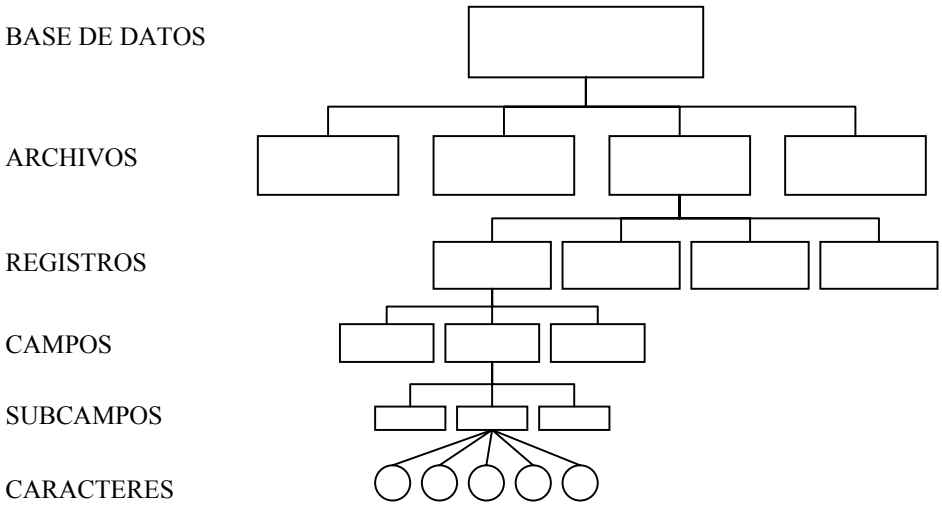
Los registros pueden ser de *longitud fija* (cuando los registros contienen el mismo número de campos, cada uno de la misma longitud) o de *longitud variable*.

Los registros organizados en campos se denominan *registros lógicos*.

Clave: una *clave* (*key*) o *indicativo* es un campo de datos que identifica al registro y lo diferencia de otros registros. Esta clave se utiliza para facilitar la extracción de información de un registro. Se espera identificar en forma unívoca cada registro almacenado en el archivo. Claves típicas son nombres o números de identificación.

Base de datos: una colección de archivos a los que puede accederse por un conjunto de programas y que contienen todos ellos datos relacionados, constituye una base de datos. Así, una base de datos de una universidad puede contener archivos de estudiantes, archivos de nóminas, inventarios de equipos, etc.

Estructura jerárquica:



ORGANIZACIÓN DE ARCHIVOS:

El *acceso a los archivos* es el procedimiento necesario para situarse en un registro determinado con el objeto de realizar una operación. El *costo* es la cantidad de accesos para encontrar un dato.

Según las características del soporte empleado y el modo en que se han organizado los registros, se consideran dos tipos de acceso a los registros de un archivo: *acceso secuencial* y *acceso directo*.

La *organización* de un archivo define la forma en la que los registros se disponen sobre el soporte de almacenamiento, o también se define la organización como la forma en que se estructuran los datos en un archivo.

1- Organización Secuencial:

Es la forma más simple de almacenar y recuperar registros en un archivo. En un archivo secuencial se almacenan los registros uno tras otro ocupando espacios de memoria contiguos. No existen posiciones sin uso.

El acceso secuencial permite el acceso a los registros de un archivo del primero al último y de *uno* en *uno*. Una operación de lectura o escritura, lee o escribe y avanza el puntero al siguiente registro. Para acceder a un registro *n* dado es necesario pasar por todos los *n-1* registros que le preceden (costo=n+1; costo promedio=(n+1)/2).

Estos ficheros contienen un registro particular (el último) que contiene una marca de fin de archivo (EOF).

Altas: El nuevo registro se agrega a continuación del último dado de alta.

Bajas: No hay forma física de dar de baja un registro intermedio y realizar el corrimiento, en estos casos se debe hacer una baja lógica. La única forma de realizar la baja física es generar un archivo nuevo sin el registro marcado.

Ventajas:

- Es el más óptimo en cuanto a aprovechamiento de memoria, ya que se ocupa espacio a medida que se necesita, no requiere espacio inicializado previamente.

Desventajas:

- No es fácil la actualización puntual de un archivo secuencial, para esto se puede almacenar otro archivo temporal con todas las modificaciones y luego lanzar un proceso de batch o apareo para actualizar el archivo maestro.
- El método de acceso lo realiza ineficiente si se requiere hallar un registro particular en un archivo muy grande.
- Las bajas sólo se pueden realizar lógicamente.

Uso:

- Cuando se va a manejar grandes volúmenes de datos por lotes.
- Si es necesario acceder a todos los registros en el archivo para una aplicación particular, o aproximadamente alrededor de la mitad de los registros van a ser accedidos cada vez. En general, si se necesitan menos del 10 % de los registros en un archivo durante una ejecución común del programa, el archivo no debe ser secuencial. Si se desea acceder a más del 40 % de los registros, se debería usar el secuencial. Entre el 10 % y el 40 % dependería del tamaño y la frecuencia de uso.

2- Organización Directa:

Esta organización permite procesar o acceder en forma rápida a los registros haciendo referencia directamente a su posición relativa en el soporte de almacenamiento sin pasar por la información anterior.

Cada registro tiene asociado un número relativo, que no es la dirección física que tiene realmente en el disco, es el sistema operativo quien se encarga de relacionar el número relativo de registro con la dirección física.

Generalmente es necesario definir el espacio de memoria que va a ocupar el archivo (dimensionarlo). Para realizar esto nos basamos en dos datos: la longitud del registro (dato conocido) y la cantidad de éstos (lo que implica una gran desventaja). Actualmente los lenguajes de programación lo redimensionan a medida que se quiere realizar una alta.

Acceso: Para acceder, ante la petición de un registro determinado, el método calcula la dirección del bloque físico que lo contiene y accede a él directamente (Hashing). También es posible acceder al registro mediante su campo clave, si este coincidiera con el número relativo de registro, lo cual no es muy usual. Si no se pudiera realizar de las maneras anteriores, se puede simular una lectura secuencial, comparando hasta encontrar el registro buscado.

Altas: para dar de alta a un registro es necesario posicionarse según el número relativo de registro y luego grabar la información. Como se accede directamente a los *n* registros, entonces es posible dejar espacios entre uno y otro. Si se accede por simulación de acceso secuencial, se da de alta uno después que otro (como en archivos secuenciales).

Bajas: Las bajas son lógicas, pero es posible crear un archivo de menor dimensión copiando los registros que no están marcados como dados de baja, o utilizar el campo marcado lógicamente como dado de baja para determinar cuando es posible sobrescribir los datos de un registro.

Ventajas:
- Cuando la información de los registros no es de gran variabilidad, tanto en el número como en la cantidad de transacciones. Ej.: tablas con índices sobre provincias o países.
- Cuando se puede hacer correspondencia de claves con la posición de los registros en el archivo.

Desventajas:
- Tener que definir el número máximo de registros.

3- Organización Indexada:

Representa una especie de balance entre la organización secuencial y la directa ya que el método usa un examen secuencial del índice, seguido del acceso directo a la dirección física apropiada en el área de datos. Un archivo con organización indexada consta de un área de índice y un área de datos.

Para hallar un registro cuando se desconoce la dirección de almacenamiento, es necesario examinar todos los registros, sin embargo, la búsqueda será más rápida si se busca en un índice en vez de en un archivo completo de datos.

Para localizar un registro específico se requiere: leer el índice, ubicar la clave, tomar la dirección y buscar el dato en el área de datos.

Características Generales:
- Los índices ocupan espacio adicional pero proporcionan un método rápido de localizar los registros.

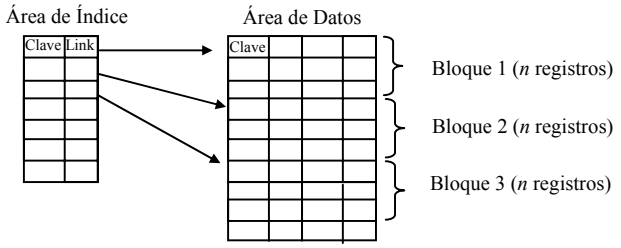
Es más rápido que el archivo secuencial, pero más lento que el direccionamiento directo.

3-a) Organización Secuencial Indexada:

El área de datos consta de dos segmentos: el área primaria y el área de saturación o derrama (overflow). El área primaria está organizada por bloques, y dentro de cada bloque una cantidad fija de registros. Los registros contenidos en los bloques están dispuestos en forma consecutiva (secuencial).

El área de derrama se utiliza para almacenar aquellos registros que al intentar ser dados de alta no han tenido espacio en el área primaria. Esta área también posee organización secuencial.

El área de índice solo almacena las direcciones correspondientes al comienzo de cada bloque y su clave asociada.



Acceso: el área de índice debe estar siempre ordenada y debe contener información activa (que no haya sido dada de baja). Éste índice trabaja en memoria principal y se crean imágenes en memorias auxiliares.

Altas: se busca en qué bloque debe ir la clave a dar de alta y luego (si hay espacio en el bloque) se inserta este registro. Si el bloque no estaba creado, se debe crear e indicar en el índice esta nueva clave y la dirección de este bloque.

Bajas: se puede marcar lógicamente el registro dentro del bloque, o bien producir el corrimiento. En el caso que el registro borrado sea el mismo del índice (clave) deberemos modificar el área de índice.

Modificaciones: se realiza accediendo por el índice y recorriendo el área de datos en forma secuencial hasta encontrar la clave buscada. Finalmente se realiza la modificación. Si se cambia la clave, se debe reorganizar el área de índice.

Desventajas:
- Se asigna espacio de almacenamiento que puede no ser utilizado dentro de un bloque.
- El área de derrama puede crecer excesivamente, lo que supone una disminución en la rapidez en su tratamiento y una mayor dificultad en su procesamiento.

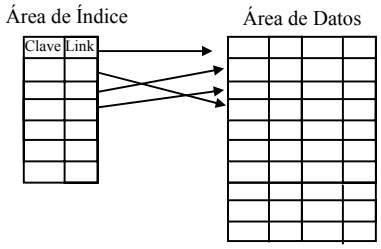
3-b) Organización Indexada no secuencial:

Consiste de un área de índice y un área de datos, estableciéndose una relación uno a uno entre ellas. Los registros en el área de datos no se ubican en forma contigua necesariamente (ni ordenada). El área de índice almacena una clave y una dirección asociada por cada registro en el área de datos. Puede haber varios índices si hay más de un atributo significativo.

Altas: se coloca el registro nuevo en cualquier área desocupada y después todos lo índices que se refieren a los atributos existentes de este registro tienen que actualizarse.

Modificaciones: se puede regrabar directamente sobre el registro. Si se cambia la clave se debe reorganizar el área de índice.

Bajas: la información del índice debe ser siempre activa (que no haya sido dada de baja), así que en esta área las bajas son físicas. En el área de datos las bajas pueden darse lógicamente. No es necesario reorganizar el archivo de datos, salvo que deseemos limpiarlo de las posibles marcas de eliminaciones; el que debe estar siempre ordenado es el índice.



RECONSTRUCCIÓN DE UN ARCHIVO INDEXADO

Es factible la pérdida de información por cortes de energía si se está trabajando en Memoria Principal o por rupturas de pistas si la información está alojada en disco.

Si se pierde el área de índice completa no existe forma de acceder, por esto todo administrador de archivos prevé una regeneración en el área de índice.

Un regenerador es una herramienta que permite ir directamente al área donde están los datos. Si es contigua y la clave es parte del registro, arma el área de índice con la clave y la dirección, si en el registro no está la clave nunca podría generarse el área de índice. Si el espacio no es contiguo utiliza el nexo y procede de la misma forma.

Es fundamental tener la dirección del primer registro por ello todo administrador de archivos lo hace (en DOS se guarda en la FAT, en FOX –> REINDEX, en COBOL –> RECOVERY).

Además es importante guardar la clave en el registro de dato para poder regenerar el índice.

Nota: si se pierde la dirección del primer registro se podría recuperar manualmente trabajando con las direcciones reales (DEBUG).

NORMALIZACIÓN:

Se utiliza para simplificar el contenido del almacenamiento de datos y para ayudar a que estos diseños sean más eficientes, eliminando los grupos repetitivos.

- *Primera forma normal (1FN)*: cualquier relación está en 1FN cuando no incluye ningún grupo repetitivo. Las relaciones en la 1FN pueden tener dos tipos de complejidad:
    - Si la clave principal es concatenada, alguno de los elementos no-clave pueden depender de una sola parte de la clave, y no de la clave completa.
    - Alguno de los elementos no-clave pueden estar interrelacionados.
- *Segunda forma normal (2FN)*: una relación normalizada está en la 2FN si *todos* los elementos no-clave son funciones completamente dependientes de la clave principal (la clave completa y no una parte de esta si es concatenada).
- *Tercera forma normal (3FN)*: para todo campo de una relación que esté en 3FN se cumple que depende sólo de la clave y de ningún otro campo de la relación.

UNIDAD NRO. 2: MODELOS MATEMÁTICOS Y REPRESENTACIONES COMPUTACIONALES

Modelos matemáticos. Representaciones. Teoría de grafos. Grafos dirigidos. Clasificación. Paso. Camino. Hipergrafos. Aplicaciones.

DEFINICIÓN DE ESTRUCTURA DE DATOS:

Colección de datos organizados de un modo particular. Es una forma de representar un modelo de la realidad. Puede considerarse como aquella que es capaz de soportar a los datos y sus relaciones. Básicamente se compone de una dupla que soporta el conjunto de datos y las relaciones que se dan entre ellos.

MODELOS MATEMÁTICOS. REPRESENTACIONES:

1. *Representación matemática o formal*:
   a. Por comprensión: se representa en forma global.
   b. Por extensión: detalla o enumera todos los puntos y todas las relaciones. Se debe tener en cuenta el modelo en su totalidad.
2. *Representación matricial*: es una forma directa y sencilla de guardar información, permite buscar y consultar fácilmente y a muy bajo costo. Utiliza matrices de adyacencia; una matriz de adyacencia para un grafo **G** es una matriz **A** de "*n x n"*, de ceros y unos, donde A [i,j] es uno si y sólo sí existe un arco que vaya del vértice *i* al *j*. En estas matrices los unos indican que existe relación y los ceros que no existe relación. Sólo se pueden representar algunos modelos de la realidad o parte de ellos. La desventaja de este modelo es que se debe dimensionar la matriz, es decir, se debe conocer la cantidad de elementos que tiene el grafo.
   a. El conjunto izquierdo se determina identificando los *unos* en las columnas.
   b. El conjunto derecho se determina identificando los *unos* en los renglones.
   c. El conjunto minimal se determina identificando las columnas que están en *cero* en su totalidad.
   d. El conjunto maximal se determina identificando los renglones que están en *cero* en su totalidad.
   e. El mínimo, cuando hay una sola columna con ceros.
   f. El máximo, cuando hay solo un renglón con ceros.

$$a_{i,j} = 1 \Rightarrow (i,j) \in R \qquad a_{i,j} = 0 \Rightarrow (i,j) \notin R$$

3. *Representación encadenada*: las celdas, posiciones o bloques de memoria, se consideran unidades de almacenamiento de información. Están compuestas como mínimo por dos partes o campos, una donde se almacena la información y otra de enlace que vincula la celda con otra que componga la misma estructura. El área de información se puede dividir en *n* cantidades de partes o funciones de asignación. Además, para encadenar las celdas es necesario tener tantos campos de enlace como el mayor grado de salida que exista entre los nodos.

TEORÍA DE GRAFOS:

Concepto de grafo: conjunto de puntos y conjunto de líneas (relaciones), cada una de las cuales une un punto a otro. Los puntos se llaman *nodos* o *verticales* del grafo y las líneas se llaman *aristas* o *arcos* (*edges*).

En símbolos: $G = \{P, R, \underbrace{f_1, f_2, ...f_3}, \underbrace{g_1, g_2, ...g_n}\}$

Funciones asignadas a nodos — Funciones asignadas a arcos

Orden: es el número de elementos de un grafo.

Paso:

Secuencia de nodos de un determinado grafo entre un nodo origen y un nodo destino. Puede existir más de un paso entre dos nodos.

En símbolos: $\rho(x,z); x \wedge z \in P =< y_0, y_1, ...y_n > \Leftrightarrow$

1- $x=y_0$ (nodo origen); $z=y_n$ (nodo destino)
2- $y_{i-1} \neq y_i$.
3-$(y_{i-1},y_i) \in R$; $1 \leq i \leq n$

Loop: Es el paso entre un nodo y sí mismo donde la longitud de paso es cero (por convención, ya que no cumple la definición de paso). En símbolos: $|\rho(x,x)| = 0$.

Ciclo: es la existencia de un paso entre un nodo y sí mismo, tal que la longitud de ese paso sea mayor o igual que dos. En símbolos: $|\rho(x,x)| \geq 2$.

Camino: ente dos puntos *x* e *y* es la secuencia de puntos en los cuales n_0=x es el origen y n_k=y es el destino. No interesa el sentido de los arcos. En símbolos:

1- $x=n_0$ (nodo origen); $z=n_k$ (nodo destino)
2- $y_{i-1} \neq y_i$.
3-$(y_{i-1},y_i) \in R$ v $(y_i,y_{i-1}) \in R$

Circuito: es el equivalente al ciclo, teniendo en cuenta la relación de camino. No interesa el sentido de los arcos. En símbolos: $|W(x,x)| \geq 2$

Longitud de Paso:

Es la cantidad de arcos existentes entre el nodo de origen y el nodo de destino. En símbolos: $|\rho(x,z)|$

Conjunto Izquierdo de un nodo dado:

Conjunto de nodos desde los cuales se llega directamente al nodo considerado (paso de longitud 1). El conjunto izquierdo puede ser cero.

En símbolos: $L(x) = \{y / y \in P \wedge (y,x) \in R\}$

Conjunto Derecho de un nodo dado:

Conjunto de nodos a los cuales llego directamente (paso de longitud 1).

En símbolos: $R(x) = \{z / z \in P \wedge (x,z) \in R\}$

Grado de Entrada: es la cardinalidad del conjunto izquierdo. En símbolos: $|L(y)|$.

Grado de Salida: es la cardinalidad del conjunto derecho. En símbolos: $|R(y)|$

Conjunto Minimal: es el conjunto de nodos a los cuales no les llega ningún arco.

Conjunto Maximal: es el conjunto de formado por los nodos desde los cuales no parte ningún arco.

Mínimo: cuando el conjunto minimal tiene un solo elemento, a éste se lo llama mínimo. Es el único punto del grafo al cual no llegan grafos.

Máximo: cuando el conjunto maximal tiene un solo elemento, a éste se lo llama máximo. Es el único grafo desde el cual no salen arcos.

Vecindad Right: elementos a los cuales llegan arcos del nodo examinado con paso de longitud mayor igual que dos.

Vecindad Left: aquel conjunto de elementos de los cuales parten arcos hacia el nodo tratado con una longitud de paso mayor igual que dos.

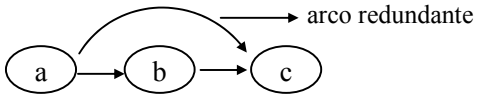
Ideal Principal Izquierdo: es el conjunto de puntos desde los cuales puedo partir para llegar al nodo dado (no interesa la longitud de paso). En símbolos: $\overline{L}(x) = \{y / y \in P \wedge \exists(y,x)\}$

Ideal Principal Derecho: es el conjunto de puntos a los cuales puedo llegar partiendo del nodo dado (no interesa la longitud de paso). En símbolos: $\overline{R}(x) = \{z / z \in P \wedge \exists(x,z)\}$

Grafo Básico:

G es grafo básico si:

1- Está libre de loops.
2- No existen arcos redundantes, es decir, no debe haber dos formas distintas de llegar desde un nodo a otro. Para encontrar un grafo básico hay que ir eliminando pasos que estén de más, sin destruir la estructura. En símbolos: $\forall x, y \in P, \left|\rho(x,y)\right| \geq 2 \Rightarrow (x,y) \notin R$.



Grafo cíclico: contiene por lo menos un ciclo. Puede existir más de una representación básica de él.

$$\forall x \in P, \exists \mid x, x \mid \geq 2$$

Grafo acíclico: grafo que no tiene ciclos. Existe una única representación básica de él.

$$\forall x \in P, NO\exists \mid x, x \mid \geq 2$$

Grafo conectado o conexo: grafo que no tiene punto aislados – no tiene elementos sin relacionar. Grafo en el que existe camino entre 2 vértices cualesquiera. Un grafo es conexo si todos sus pares de vértices están conectados.

Grafo fuertemente conectado: un grafo G es fuertemente conectado si la relación de paso inducida es una relación de equivalencia. Los grafos fuertemente conectados se caracterizan porque dado cualquier par de puntos x y z existe un paso $\exists\rho(x,z)$ que los une.

Igualdad de grafos: dados dos grafos $G_1(P_1,R_1)$ y $G_2(P_2,R_2)$ se puede hablar de igualdad si uno es copia del otro. En símbolos: $G_1 = G_2 \Leftrightarrow P_1 = P_2 \wedge R_1 = R_2$

Isomorfismo de grafos:

Dos grafos son isomorfos cuando tienen la misma estructura implementada, cambiando solo la información de los nodos.

$$G_1 \cong G_2 \Leftrightarrow x, y \in P_1, \exists (x,y) \in R_1 \Rightarrow \varphi : P_1 \rightarrow P_2 \wedge (\varphi(x), \varphi(y)) \in R_2 \wedge \varphi(x), \varphi(y) \in P_2$$

Grafo completo: es aquel en que cada vértice está conectado con todos y cada uno de los restantes nodos.

Grafo lineal: G es lineal si se verifica que:

1- $\exists x \in P / L(x) = 0$
2- $\forall y \in P : \left|L(y)\right| \leq 1; \left|R(y)\right| \leq 1$
3- G tiene que ser conectado

Hipergrafo: grafo que tiene un conjunto de puntos y más de una relación definida en dicho conjunto. Sobre el mismo modelo es necesario analizar las distintas relaciones. Todo hipergrafo puede llevarse a 2 ó más grafos dependiendo del número de relaciones que contenga el hipergrafo.

$H=(P,R,R',f_1,f_2,..f_n,g_1,g_2,...g_n,h_1,h_2,...h_n)$
$G_1=(P,R)$, $G_2=(P,R')$.

Clausura transitiva: grafo que se obtiene como resultado de inducir o generar la relación de paso (la relación transitiva) sobre la relación original de un grafo cualquiera.

Subgrafo: S es un subgrafo de G cuando: G=(P,R)
S=(P',R') siendo $P' \subseteq P$
y R'=R|P (restringida por una clase de equivalencia, es decir, la parte que elegimos debe ser representativa).

Grafo finito: la cantidad máxima de flechas que puede haber es en un grafo G finito, de *n* nodos es: n^2-n (Se le resta *n* para excluir los loops). El paso máximo entre dos nodos de un grafo finito no se puede contar (es infinito), debido a que se puede entrar en ciclos.

Grafo ponderado o con peso: es aquel en el que cada arista tiene un valor.

UNIDAD NRO. 3: ESTRUCTURAS LINEALES

INTRODUCCIÓN:

Las estructuras dinámicas de datos son estructuras que "crecen a medida que se ejecuta un programa". Una *estructura dinámica de datos* es una colección de elementos (llamados nodos) que son normalmente registros. Al contrario que un array que contiene espacio para almacenar un número fijo de elementos, una estructura dinámica de datos se amplia y se contrae durante la ejecución del programa basada en los registros de almacenamiento de datos del programa.

DEFINICIÓN:

G es una estructura lineal si:
$$\begin{cases} 1\text{-}\ \exists x \in P / L(x) = 0 \\ 2\text{-}\ \forall y \in P : |L(y)| \le 1; |R(y)| \le 1 \\ 3\text{-}\ \text{G tiene que ser conectado} \end{cases}$$

LISTAS:

Una *lista lineal* es una serie de $n \geq 0$ nodos x[1], x[2], ..., x[n] cuyas propiedades estructurales incluyen esencialmente, sólo las posiciones relativas lineales (una dimensión) de los nodos.

Esta es una definición muy general que incluye los ficheros y vectores. Los elementos de una lista lineal se almacenan normalmente contiguos en posiciones consecutivas de memoria.

Las listas así definidas se denominan *contiguas*. Las operaciones que se pueden realizar con estas listas son: insertar, eliminar o localizar un elemento; determinar el tamaño (nro. de elementos) de la lista; recorrer la lista para localizar un determinado elemento; clasificar los elementos de la lista en orden ascendente o descendente; unir dos o más listas en una sola; dividir una lista en varias sublistas; copiar la lista; borrar la lista.

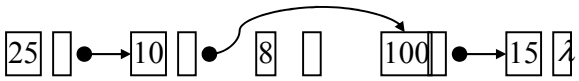
LISTAS ENLAZADAS:

Los inconvenientes de las listas contiguas se eliminan con las listas enlazadas. Se pueden almacenar los elementos de una lista lineal en posiciones de memoria que no sean contiguas o adyacentes.

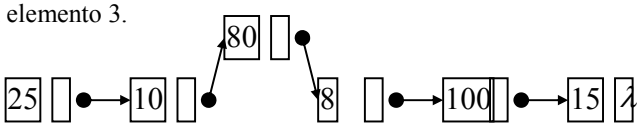
Una *lista enlazada* o *encadenada* es un conjunto de elementos en los que cada elemento contiene la posición (o dirección) del siguiente elemento de la lista. Cada elemento de una lista enlazada debe tener al menos dos campos: un campo con el valor del elemento y un campo que contiene la posición del siguiente elemento (link o enlace).

Un *puntero* es una variable cuyo valor es la dirección o posición de otra variable.

Para eliminar el 45º elemento de una lista lineal con 2500 elementos sólo es necesario cambiar el puntero en el elemento anterior (44º), que apunte ahora al elemento 46º.



Para insertar un elemento en la posición 3 es necesario cambiar el puntero del elemento 2 al nuevo elemento y hacer que el nuevo elemento apunte al elemento 3.

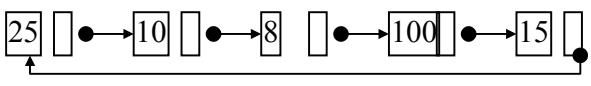


Procesamiento de listas enlazadas:

Para procesar una lista enlazada se necesitan las siguientes informaciones: primer nodo (cabecera de la lista), el tipo de sus elementos y el tamaño de la lista.

LISTAS CIRCULARES:

Las listas simplemente enlazadas no permiten a partir de un elemento acceder directamente a cualquiera de los elementos que le preceden. En lugar de almacenar un puntero nulo en el campo *siguiente* del último elemento de la lista, se hace que el último elemento apunte al primero o principio de la lista. Este tipo de estructura se llama *lista circular*.



Las listas circulares presentan las siguientes ventajas respecto de las listas enlazadas simples:

- Cada nodo es accesible desde cualquier otro nodo de la lista.
- Las operaciones de concatenación y división de listas son más eficaces con listas circulares.

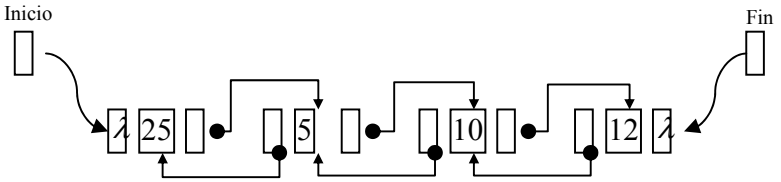
Las desventajas, por el contrario, son:

- Se pueden producir lazos o bucles infinitos. Una forma de evitarlos es disponer de un nodo especial que se encuentre permanentemente asociado a la existencia de la lista circular. Este nodo se denomina *cabecera*. Este nodo *cabecera* puede tener un valor especial en su campo o una bandera que lo señale, es decir que la información del nodo cabecera no se utiliza.

LISTAS DOBLEMENTE ENLAZADAS:

En las listas lineales anteriores el recorrido de ellas sólo podía hacerse en un único sentido: de izquierda a derecha (principio a final). En muchas ocasiones se necesita recorrer las listas en ambas direcciones.

Las listas que pueden recorrerse en ambas direcciones se denominan *listas doblemente enlazadas*. En estas listas cada nodo consta del campo de datos y dos campos de enlace: *anterior* y *siguiente* que apuntan hacia delante y hacia atrás. Su desventaja es que una lista doblemente enlazada ocupa más espacio en memoria que una lista simplemente enlazada para la misma cantidad de información.



La variable *inicio* y el puntero al nodo siguiente permiten recorrer la lista en sentido normal y la variable *fin* y el puntero al noto anterior permiten recorrerla en sentido inverso.

PILAS (STACK):

Una *pila* (*stack*) es un tipo especial de lista lineal en la que la inserción y borrado de nuevos elementos se realiza sólo por un extremo que se denomina *cima* o *tope* (*top*). Dado que las operaciones de insertar y eliminar se realizan por un solo extremo (el superior), los elementos sólo pueden eliminarse en orden inverso al que se insertan en la pila. El último elemento que se pone en la pila es el primero que se puede sacar; por ello, a estas estructuras se las conoce por el nombre *LIFO* (*Last In, First Out* o primero entrado, primero salido).

Las operaciones más usuales asociadas a las pilas son:

- *Push (meter o poner)*: operación de insertar un elemento en la pila.
- *Pop (sacar o quitar)*: operación de eliminar un elemento de la pila.

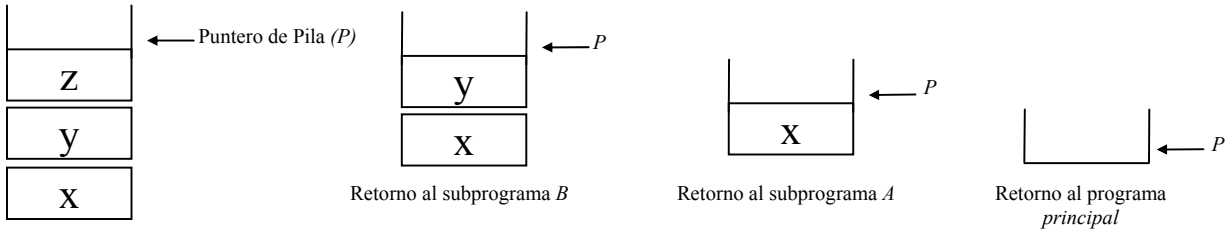
Para representar una pila (ya sea estática o dinámicamente) básicamente necesitamos un puntero (índice en el caso de un array) indicándonos la cima o tope de la pila.

Aplicaciones de las pilas: *Llamadas a subprogramas*

Cuando dentro de un programa se realizan llamadas a subprogramas, el programa principal debe recordar el lugar donde se hizo la llamada, de modo que pueda retornar allí cuando el subprograma se haya terminado de ejecutar.

Supongamos que tenemos tres subprogramas *A*, *B* y *C*, y supongamos también que *A* invoca a *B* y *B* invoca a *C*. Entonces *B* no terminará su trabajo hasta que *C* haya terminado y devuelto su control a *B*. De modo similar, *A* es el primero que arranca su ejecución pero es el último que la termina, tras la terminación y retorno de *B*.

Cuando un subprograma termina, debe retornar a la dirección siguiente a la instrucción que lo llamó. Cada vez que se invoca a un subprograma, la dirección de la siguiente línea (*x*,*y* o *z*) se introduce en la pila. El vaciado de la pila se realizara por los sucesivos retornos, decrementándose el puntero de pula que queda siempre apuntando a la siguiente dirección de retorno.



COLAS:

Las colas son otro tipo de estructura lineal de datos similar a las pilas diferenciándose de ellas en el modo de insertar/eliminar elementos. Una *cola* (*queue*) es una estructura lineal de datos en la que las eliminaciones se realizan por el principio de la lista (*frente* o *front*). En las colas el elemento que entró primero sale también primero, por ello se conocen como listas *FIFO* (First In, First Out). Así pues, la diferencia con las pilas reside en el modo de entrada/salida de datos; en las colas las inserciones de datos se realizan al final de la lista. Por ello, las colas se usan para almacenar datos que necesitan ser procesados según el orden de llegada.

Para representar una cola ya sea mediante una lista enlazada o con un array se necesitan dos punteros *inicio* y *fin*.

Aplicaciones de las colas:

Un ejemplo de cola es el caso de las tareas asignadas a un microprocesador. Se utiliza una cola para almacenar los programas o las peticiones de los diferentes periféricos que esperan su turno de ejecución. El procesador atiende (normalmente) por orden de llamada, por lo tanto, todas las llamadas se almacenan en una cola. En la realizad, existe una llamada *cola de prioridades*, en ella el micro tiene una lista de las tareas que debe ejecutar primero y sólo dentro de las peticiones de igual prioridad se producirá una cola.

DOBLE COLA:

Existe una variante de la cola simple mencionada anteriormente que es la *doble cola*. La doble cola o bicola es una cola dibimensional en la que las inserciones y eliminaciones se pueden realizar en cualquiera de los dos extremos de la lista.

Existen también dos variantes de la doble cola: doble cola de entrada restringida (acepta inserciones sólo al final) y doble cola de salida restringida (acepta eliminaciones sólo al frente).

COLA CIRCULAR:

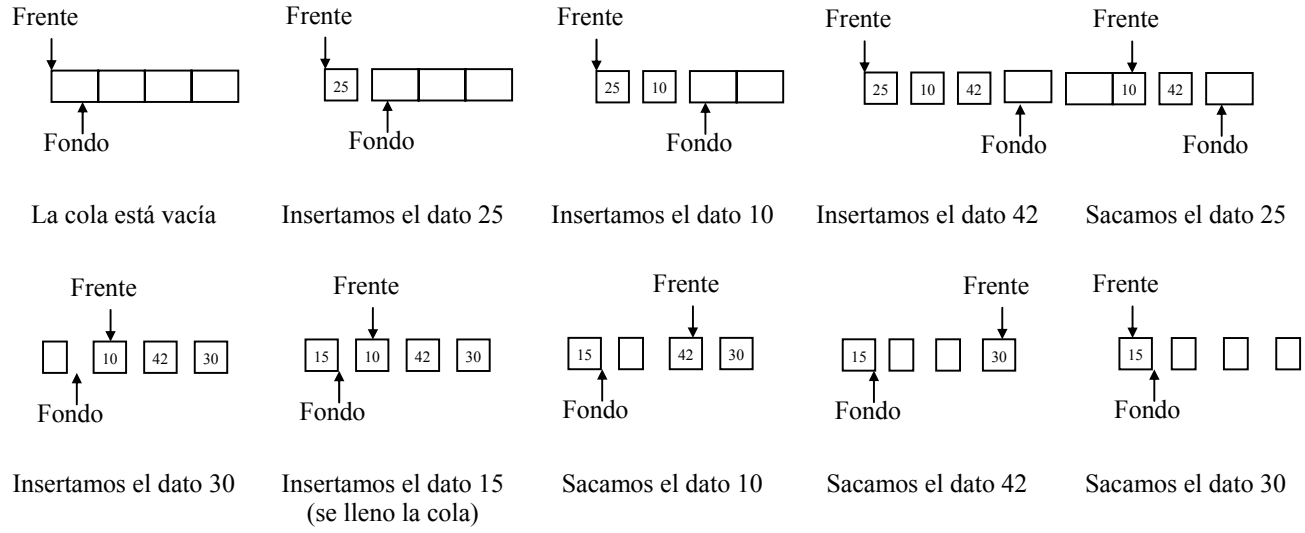
Se puede implementar ***únicamente*** en un arreglo estático.

Se la maneja con dos datos, *frente* y *fondo*.

- *Altas*: En un principio estas variables apuntan al mismo nodo. Cuando se realiza la primer alta se inserta el elemento en la posición de fondo, incrementándose este puntero en uno. Para las posteriores altas se continua con el mismo procedimiento. Cuando el *fondo* llega a la última posición del vector y se da otra alta (suponiendo que hay lugar para hacerlo), el *fondo* se corre al primer elemento de la lista. Finalmente, cuando *fondo* alcanza la posición de *frente* nos indica que la cola se ha llenado.
- *Bajas*: Lo que se hace es sacar el elemento apuntado por *frente* e incrementar en uno esta variable. Cuando la variable *frente* alcanza o llega a la variable *fondo* nos indica que se ha vaciado la cola.

NOTA: tanto la pila como la cola son estructuras destructivas, las operaciones de consulta no están permitidas, la lectura destruye la información contenida en éstas.

Ejemplo de altas y bajas en una cola circular:



UNIDAD NRO. 4: ÁRBOLES

DEFINICIÓN Y PROPIEDADES:

Un árbol es un conjunto finito *T* de uno o más nodos donde:

1. Hay un nodo especial llamado *raíz* del árbol, raíz (*T*);
2. Los nodos restantes están agrupados en $m \geq 0$ conjuntos distintos $T_1, T_2, \ldots, T_m$ y cada uno de estos conjuntos es, a su vez, un árbol. Los árboles $T_1, T_2, \ldots, T_m$ se llaman *subárboles* de la raíz.

La definición de árbol implica una estructura recursiva. De esto se desprende que cada nodo de un árbol es la raíz de algún subárbol contenido en la totalidad del mismo. (*T*=(P,R))

Propiedades:

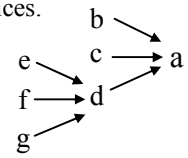
1. El grafo *T* verifica la existencia de un nodo *t* el cual es mínimo (no le llega ningún arco), al cual llamamos raíz o: $\exists\, x\ único / L(x) = 0$
2. Ese grafo es libre de circuitos y de loops.
3. La cantidad de nodos es la cantidad de arcos más uno: $|P| = |R| + 1$.

Otras definiciones:

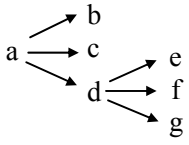
- *Raíz del árbol*: todos los árboles que no están vacíos tienen un único nodo raíz. Todos los demás elementos o nodos se derivan o descienden de él. El nodo raíz no tiene *padre*.
- *Nodo*: son los vértices o elementos del árbol.
- *Nodo terminal u hoja*: es aquel nodo que no tiene ningún subárbol.
- *Grado*: es el número de subárboles que tiene un árbol. Un árbol de grado cero es una hoja.
- *Bosque*: es un conjunto (normalmente ordenado) de cero o más árboles disjuntos. Si eliminamos la raíz de un árbol tendríamos un bosque.
- A cada nodo que no es hoja se le asocia uno o varios subárboles llamados *descendientes* o *hijos*. De igual forma, cada nodo tiene asociado un antecesor o *ascendiente* llamado *padre*.
- Los nodos de un mismo padre se llaman *hermanos*.
- *Camino*: es el enlace entre dos nodos consecutivos.
- *Rama*: camino que termina en una hoja.
- *Nivel*: cada nodo tiene asociado un número de nivel que se determina por la longitud del camino desde la raíz hasta el nodo específico.
- *Altura o profundidad*: es el número máximo de nodos de una rama. Equivale el nivel más alto de los nodos más uno (es el paso de mayor longitud entre la raíz y las hojas).
- *Peso*: es el número de nodos terminales.
- *Árbol lleno*: es aquel árbol cuyo grado de salida es el mismo para todos los nodos excepto las hojas.
- *Árbol completo*: aquél árbol lleno que tiene todas sus hojas en el mismo nivel.

CLASIFICACIÓN:

- *Árbol principal izquierdo (API)*: es una estructura jerárquica de tipo ascendente. De cada uno de sus nodos puede salir como máximo un arco y existe al menos un elemento que no tiene conjunto derecho (no parte ningún arco). El API es poco aplicable puesto que posee muchas raíces.
  - ∃ un $\boldsymbol{x}$ máximo.
  - $|R(y)| = 1,\ \forall \boldsymbol{y} \neq \boldsymbol{x}\ ;\ \boldsymbol{y} \in$ P.



- *Árbol principal derecho*: se caracteriza porque para todo punto se verifica que como máximo puede recibir un arco y existe al menos un punto al cual no le llega ningún arco (raíz). Todo árbol con elemento mínimo es principal derecho, y consecuentemente todo elemento del árbol es alcanzable desde este punto mínimo.
  - ∃ un $\boldsymbol{x}$ mínimo (raíz).
  - $|L(y)| = 1\ ,\ \forall \boldsymbol{y} \neq \boldsymbol{x}\ ;\ \boldsymbol{y} \in$ P.



ÁRBOL BINARIO:

Un árbol binario es un conjunto finito de cero o más nodos tales que:

- Existe un nodo denominado raíz del árbol.
- Cada nodo puede tener 0, 1 ó 2 subárboles conocidos como *subárbol izquierdo* y *subárbol derecho*.

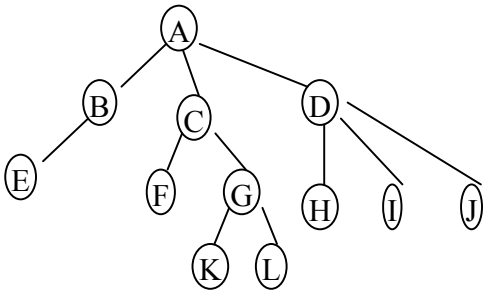
Conceptos sobre árboles binarios:

- *Árboles Similares*: se dice que dos árboles binarios son similares si tienen la misma estructura.
- Árboles E*quivalentes*: dos árboles son equivalentes si son similares y contienen la misma información.
- *Árbol Equilibrado*: un árbol binario está equilibrado si las alturas de los dos subárboles de cada nodo del árbol se diferencian en una unidad como máximo (altura[subárbol izq.]-altura[subárbol der.] ≤ 1).
- *Árbol Completo*: un árbol binario se llama *completo* si todos sus nodos tienen exactamente dos subárboles, excepto los nodos de los niveles más bajos de dos.
- *Árbol lleno*: árbol binario completo, tal que todos los niveles están llenos.
- *Árbol degenerado*: es un árbol en el que todos sus nodos tienen solamente un subárbol, excepto el último.

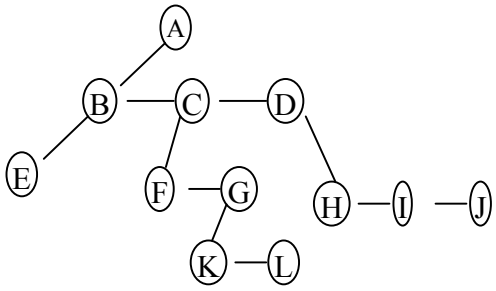
Conversión de un árbol general en un árbol binario:

Dado que los árboles binarios son una estructura fundamental en la teoría de árboles, será preciso disponer de algún mecanismo que permita la conversión de un árbol general (*A*) en un árbol binario (*B*). Aparte, los árboles binarios son más fáciles de programar que los árboles generales. El algoritmo de conversión cuenta con los siguientes pasos:

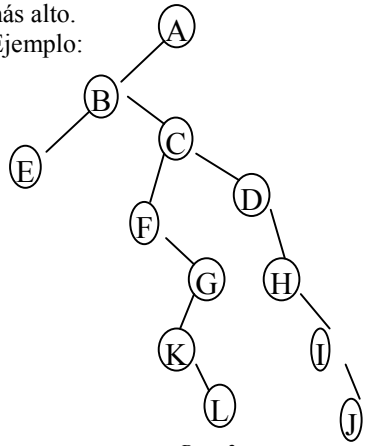
1) La raíz de *B* es la raíz de *A*.
2) *a)* Enlazar el nodo raíz con el camino que conecta el nodo más a la izquierda (su hijo).
   b) Enlazar este nodo con los restantes descendientes del nodo raíz en un camino con lo que se forma el nivel 1.
   c) A continuación repetir los pasos *a)* y *b)* con los nodos del nivel 2, enlazando siempre en un mismo camino todos los hermanos (descendientes del mismo nodo). Repetir estos pasos hasta llegar al nivel más alto.
3) Girar el diagrama resultante 45º para diferenciar entre los subárboles izquierdo y derecho. Ejemplo:



Árbol a convertir



Paso 1



Paso 2

RECORRIDO (BARRIDO) DE UN ÁRBOL BINARIO:

Se denomina *recorrido de un árbol* al proceso que permite acceder una sola vez a cada uno de los nodos del árbol. Cuando un árbol se recorre, el conjunto completo de nodos se examina. Los algoritmos de recorrido de un árbol binario presentan tres tipos de actividades comunes: visitar el nodo raíz, recorrer el subárbol izquierdo, recorrer el subárbol derecho.

De esta manea , se presentan los siguientes barridos:

1) *Barrido pre-orden*:
   a) Visitar la raíz.
   b) Recorrer el subárbol izquierdo.
   c) Recorrer el subárbol derecho.
2) *Barrido in-orden (simétrico)*:
   a) Recorrer el subárbol izquierdo
   b) Visitar la raíz.
   c) Recorrer el subárbol derecho.
3) *Barrido post-orden:*
   a) Recorrer el subárbol izquierdo.
   b) Recorrer el subárbol derecho.
   c) Visitar la raíz.
4) *Recorrido por niveles*.

ÁRBOL BINARIO DE BÚSQUEDA:

El árbol binario de búsqueda se construye teniendo en cuenta las siguientes premisas:

1. El primer elemento se utiliza para crear el nodo raíz.

2. Los valores del árboles deben ser tales que pueda existir un orden (entero, real, carácter, etc.).
3. En cualquier nodo todos los valores del subárbol izquierdo son menor o igual al valor de nodo. De modo similar todos los valores del subárbol derecho deben ser mayores que los valores del nodo.

Con estas características el barrido in-orden (simétrico) informa los valores ordenados en forma creciente.

Inserción de un elemento:

Para insertar un elemento se comprueba primero que este valor no se encuentre en el árbol y luego, si no existe, se realiza la inserción en un nodo que tengo al menos uno de los dos punteros (Izq. o Der.) con valor *nil*. Para realizar esto se desciende en el árbol a partir del nodo raíz dirigiéndose de izquierda a derecha de un nodo según que el valor a insertar sea inferior o superior al valor del campo clave de ese nodo. Cuando se alcanza un nodo del árbol en que no se puede continuar, el nuevo elemento se engancha a la izquierda o derecha de este nodo en función de que su valor sea inferior o superior.

Eliminación de un elemento:

La eliminación de un elemento debe conservar el orden de los elementos del árbol. Se consideran diferentes casos según la posición del elemento o nodo en el árbol:

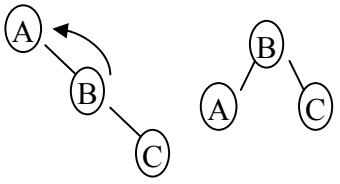
- Si el elemento es una hoja, simplemente se suprime.
- Si el elemento no tiene más que un descendiente, se sustituye entonces por ese descendiente.
- Si el elemento tiene dos descendientes se sustituye por el elemento más a la derecha o a la izquierda de modo que permita conservar la ordenación.

ÁRBOLES BALANCEADOS:

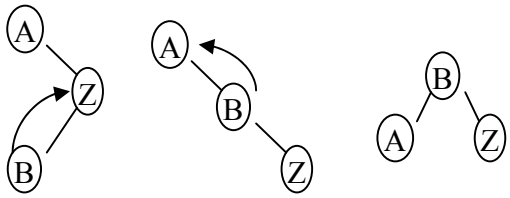
AVL: un árbol AVL es un árbol binario de búsqueda en el cual las alturas de los subárboles izquierdo y derecho de la raíz difieren a lo sumo en uno y además sus subárboles son AVL.

Transformaciones o rotaciones para solucionar el desbalance:

1. Rotación Simple: se realiza cuando el desbalance se produjo en el mismo sentido de las ramas del árbol.
2. Rotación Doble: se realiza cuando el desbalance se produce en sentidos contrarios. Consiste en dos rotaciones simples (o también elevando dos niveles el nodo que produjo el desbalance).



Rotación Simple



Rotación Doble

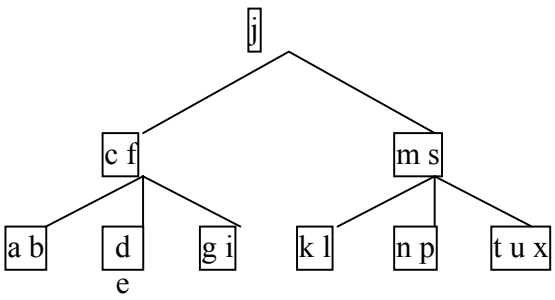
Árboles Multimodales:

En estos árboles, para algún entero *m* llamado o*rden* del árbol, cada nodo tiene como máximo *m* hijos. La cantidad de claves de cada nodo está dada por *k* ($k \leq m-1$) y las claves dividen a los hijos a manera de árbol de búsqueda.

Árboles B:

Es un árbol multimodal de orden *m* que cumple las siguientes condiciones:

1. Todos los nodos terminales están en el mismo nivel.
2. Los nodos internos tienen a lo sumo *m* hijos no vacíos y como mínimo *m*/2 hijos no vacíos.
3. El número de claves en cada nodo interno es uno menos que el número de sus hijos y estas claves dividen la de los hijos a manera de un árbol binario de búsqueda.
4. La raíz tiene como máximo *m* hijos, como mínimo 2 si no es hoja y ninguno si el árbol consta de la raíz solamente.



Ejemplo de árbol B de orden 5

Nodos internos: 3 hijos

$m/2 \leq 3 \leq m$

$2 \leq 3 \leq 5$