Algoritmos y Estructuras de Datos

Lic. en Sistemas de Información - FCyT – UADER

ALGORITMOS

Es una *secuencia* de ***acciones*** o ***pasos*** que permite resolver un *problema*.

Un mismo problema puede ser resuelto con distintos algoritmos.

Diferencia entre algoritmo y programa

Ambos son un conjunto de instrucciones. Pero un algoritmo es una forma de resolver un problema, mientras que un programa está más ligado a la realización de una o más tareas por una computadora.

Un programa puede implementar uno o varios algoritmos, o puede ser tan simple que no sea un algoritmo. Un programador comienza diseñando algoritmos para resolver los problemas y luego implementarlos e incluirlos en un programa.

Un programa siempre será ejecutado por una computadora mientras que un algoritmo podría ser ejecutado por una persona.

Un programa está escrito en lenguaje máquina o un lenguaje compilado o interpretado por algún tipo de máquina (a veces una máquina virtual).

ESTRUCTURAS DE DATOS

Forma particular de organizar ***datos*** para ser utilizados eficientemente. Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.

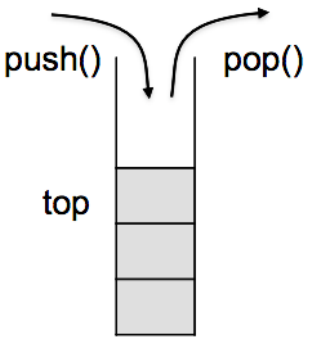
Las ***estructuras de datos*** son un medio para manejar grandes cantidades de datos de manera eficiente para usos tales como grandes **bases de datos** y **servicios de indización** de Internet. Por lo general, las estructuras de datos eficientes son clave para diseñar algoritmos eficientes.

ESTRUCTURAS LINEALES

Una estructura es **lineal** cuando un elemento sólo puede tener un ***antecesor*** y un ***sucesor***

Encontramos: **Pilas - Colas - Listas**

PILAS



Las **pilas** son *estructura de datos lineales* que presentan ***restricciones*** en cuanto a la posición en la cual pueden realizarse las inserciones y las extracciones de elementos.

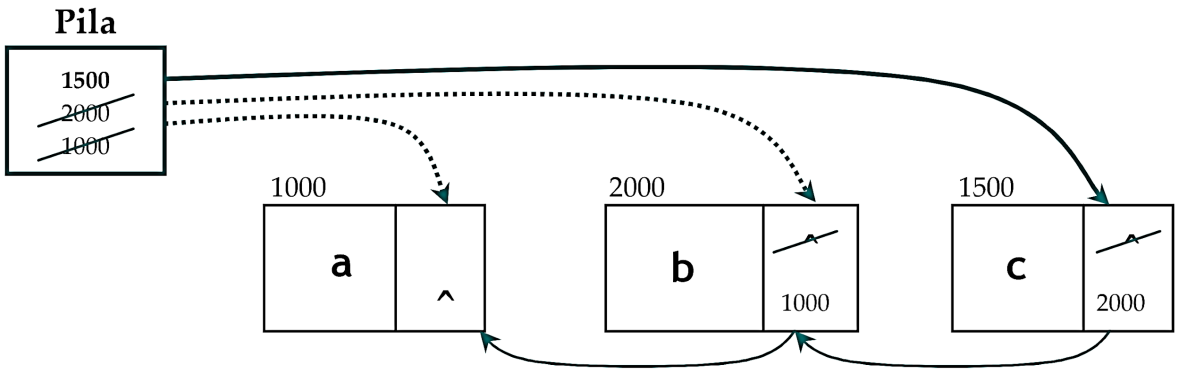
Es un ***tipo especial de lista lineal*** en la que la inserción y borrado de elementos se realiza sólo por uno de los extremos. Como consecuencia, los elementos de una pila serán eliminados en orden inverso al que se insertaron.

Debido al orden en que se insertan y eliminan los elementos en una pila, también se le conoce como estructura de tipo **LIFO** (***Last In***, ***First Out***), es decir: *último en entrar, primero en salir*.

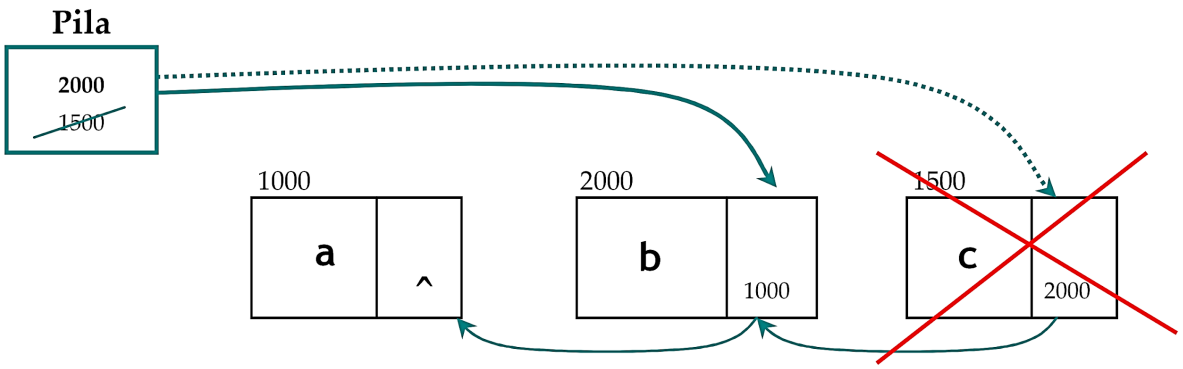
En cuanto a su representación en memoria, las pilas **NO** son estructuras de datos fundamentales porque no están definidas como tales en *lenguajes de programación*

Las ***pilas*** pueden representarse mediante **Arreglos** o **Listas enlazadas**.

ALTAS

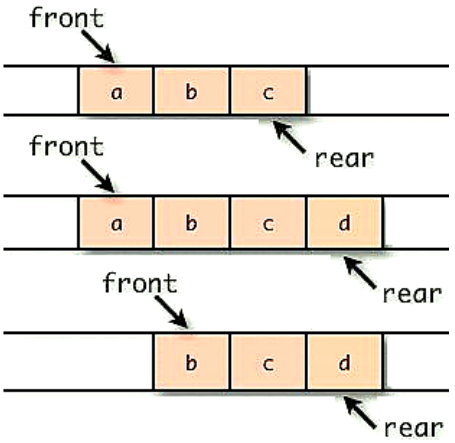


BAJAS



COLAS

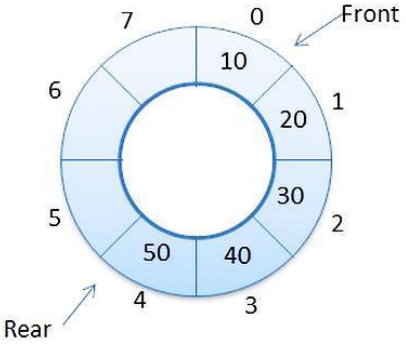
Es una estructura de datos en la cual la adición se realiza por un extremo y la de eliminación por el otro.



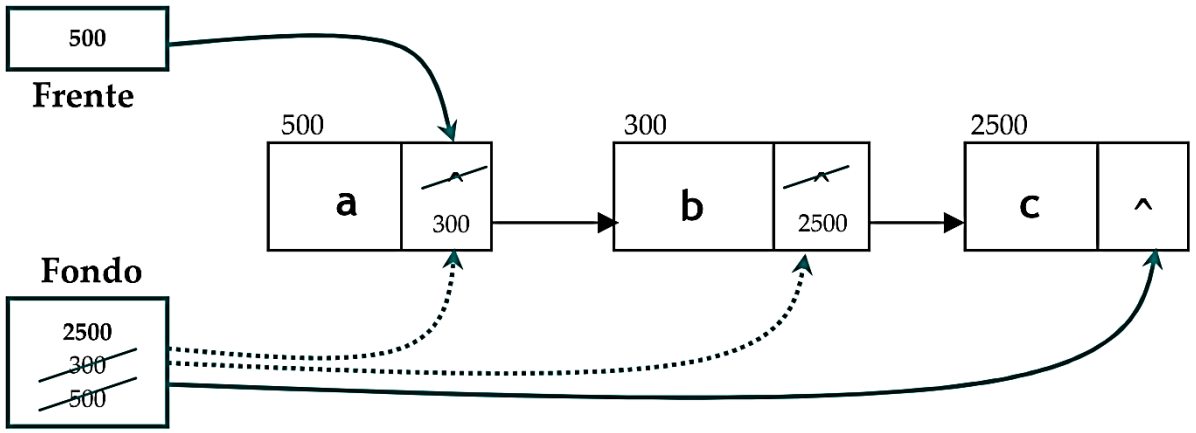
Es una estructura de tipo **FIFO** (***First In***, ***First Out***), es decir: *primero en entrar, primero en salir*.

Al igual que con las **PILAS**, las **COLAS** se pueden representar mediante ***Arreglos*** o ***Listas Simplemente Enlazadas***.

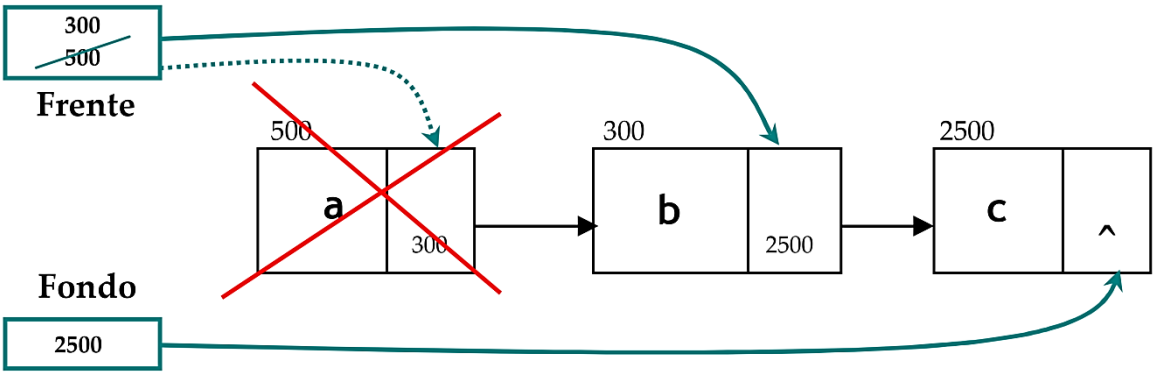
Para la representación con ***Arreglos***, se utilizan las **Colas Circulares**. Son útiles para controlar el ***tamaño de memoria*** (ya que lo define), una vez que llegó al valor debo borrar el primero para seguir agregando más. Mantengo hasta ***n elementos*** y el más viejo lo elimino para poder seguir agregando. Posee **frente** y **fondo**.



ALTAS



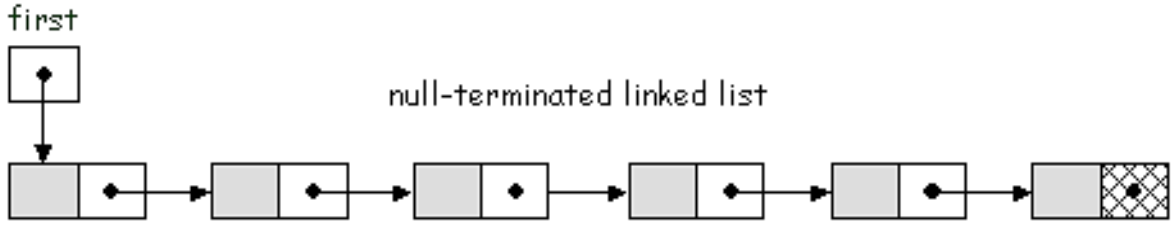
BAJAS



Tanto las **PILAS** como las **COLAS**, son de ***acceso destructivo***. Para poder ver el contenido de algún nodo, debo sacarlo de su estructura.

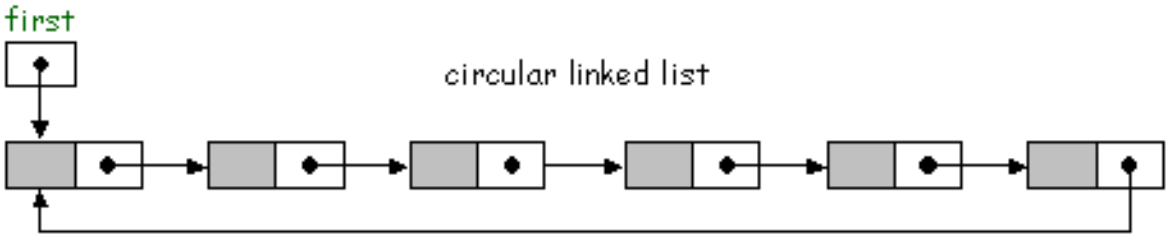
LISTAS

Este es un tipo de *estructura lineal y dinámica*: **lineal** porque a cada elemento le puede seguir solo otro elemento, **dinámica** porque su tamaño va cambiando en tiempo de ejecución.



Posee variaciones:

- ***Lista Simplemente Enlazada***: los elementos sólo apuntan al **siguiente**
- ***Lista Doblemente Enlazada***: los elementos de esta lista apuntan a su **predecesor** y su **sucesor**.
- ***Lista Simplemente Enlazada Circular***: similar a la **LSE** con la diferencia que el ***último elemento*** apunta al **primero**.
- ***Lista Doblemente Enlazada Circular***: similar a la **LDE** con la diferencia que el ***último elemento*** apunta al **primero** y *viceversa*.



Lista Circular Simple

Una lista, puede ser una **MULTILISTA** (***lista de lista***). Es decir una estructura compleja que utiliza 2 tipos de lista (**Lista Principal** y **Lista Auxiliar**), para definir una *estructura multinivel*. Dicho de otra manera es una lista principal, *simplemente enlazada*, donde cada nodo de esta lista apunta además, a una lista auxiliar, también *simplemente enlazada*.

En las estructuras de datos **NO LINEALES**, cada elemento puede tener más de un ***antecesor*** y/o ***sucesor***.

ÁRBOLES

Son Estructuras de Datos **no lineales** y **dinámicas**.

Un **Árbol** es una estructura jerárquica aplicada sobre una colección de elementos conocidos como **nodos**, donde uno de ellos es conocido como **raíz**.

Estás ED son estructuras recursivas, ya que cada ***nodo interno*** forma un *árbol*, y un nodo de ese subárbol forma otro, y así sucesivamente.

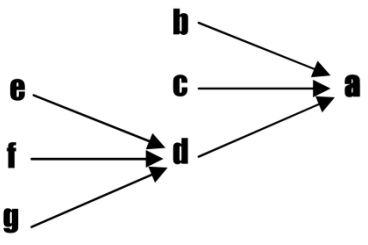
Algunas definiciones que debemos manejar, son:

- **Longitud de camino**: ***cantidad de ramas*** desde la **raíz** hasta el **nodo deseado**
- **Altura de nodo**: ***cantidad de nodos*** desde la **raíz** hasta el **nodo hoja** más alejado
- **Profundidad/Nivel**: ***cantidad de nodos*** desde la **raíz** hasta el **nodo**
- **Nodos hermanos**: aquellos que poseen el ***mismo nodo predecesor*** (o **padre**)
- **Orden**: ***cantidad de sucesores*** que puede tener un mismo nodo
- **Grado de nodo**: ***cantidad de sucesores*** que posee.
- **Subárbol:** es un nodo con todos sus descendientes. Una ***hoja*** es *subárbol*
- **Hoja**: nodos que no disponen de ningún subárbol (*elementos terminales*).

Acá podemos destacar lo siguiente:

- **Árboles Llenos**: todos los *nodos hojas* están al mismo nivel y cada nodo (excepto las *hojas*) posee la ***máxima cantidad de hijos***.
- **Árboles Completos**: todos los nodos (excepto las *hojas*) poseen la ***máxima cantidad de hijos.*** $\forall\, y \in P,\ y \neq \text{maximal} \Rightarrow |R(y)| = n$
- **Arboles Degenerados**: cada nodo (incluyendo la raíz) posee un ***único sucesor***. Su figura se asemeja a la de una ***LSE***

Árbol Principal Izquierdo (API)



Si cumple las siguientes dos condiciones:

1) Existe x perteneciente a P que es máximo

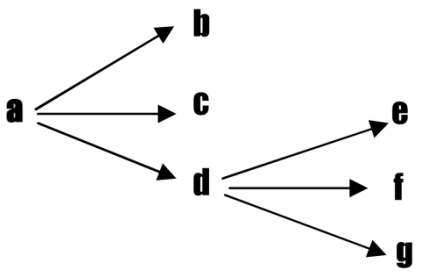
$$\exists\, x \in P / x \text{ es } \textbf{\textit{máximo}}$$

2) El *grado de salida* de un punto cualquiera salvo la *raíz* es **1**.

$$|R(y)| = 1;\ \forall\, y \neq x,\ \forall\, y \in P$$

Es poco aplicable puesto que posee muchas raíces

Árbol Principal Derecho (APD)



Si cumple las siguientes dos condiciones:

1) Existe x perteneciente a P que es mínimo.

$$\exists\, x \in P / x \text{ es } \textbf{\textit{mínimo}} \text{ (raíz)}$$

2) El grado de entrada de un punto cualquiera salvo la raíz es **1**.

$$\left| \mathrm{L}(y) \right| = 1;\ \forall\ y \neq x,\ \forall\ y \in \mathrm{P}$$

De un nodo, se puede llegar a todos los demás. Existe un solo nodo de acceso. Su estructura de datos es muy eficiente. Los algoritmos son más complicados.

Árboles Balanceados

Un árbol está ***equilibrado*** en el sentido en que el número de nodos del **subárbol izquierdo** difiere a lo más en **1** unidad del nro de nodos del **subárbol derecho**

Los árboles balanceados presentan la **mitad** de sus nodos en el subárbol izquierdo y la otra mitad en el subárbol derecho.

La ventaja es que se aprovechan al máximo los punteros de los nodos. Además, decidir si un elemento está en alguno de los subárboles quedará reducido a la mitad el número de elementos a buscar.

Si se conoce el número de nodos a insertar en el árbol, es fácil diseñar un algoritmo para construir este tipo de árboles, pero el esfuerzo para mantener el árbol balanceado puede ser muy alto.

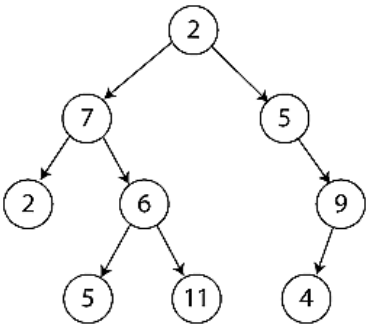
Árboles Multimodales o de Búsqueda Multimodal

Son aquellos árboles que se generalizan de los árboles binarios de búsqueda. Sus características principales son:

- Todo árbol multimodal tiene un ***orden m***.
- Cada nodo tendrá como máximo ***m*** **hijos**. Además, si $k \leq m$, el nodo tendrá $k - 1$ ***claves***, que divide todas las claves en ***k*** **subconjuntos**.
- Si alguno de estos subconjuntos está ***vacío***, los hijos de estos subconjuntos también estarán ***vacíos***. Dependiendo de la cantidad de claves se asocia cuantos hijos tendrá en cada elemento (será uno más).

Árbol Binario de Búsqueda o *Lexicográficos*

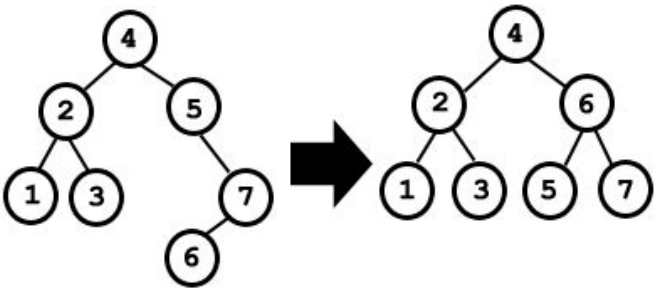
Son árboles de ***orden 2*** donde dado un nodo no hoja, su ***subárbol izquierdo*** es menor que el propio nodo, y el ***subárbol derecho*** es mayor.



A medida que se van ingresando los elementos (altas), se van colocando los menores o iguales a la izquierda y los mayores a la derecha de la raíz.

Árbol AVL

Estos árboles (al igual que los **ABB**) son de búsqueda, y poseen la particularidad de ser ***autobalanceables***. Los árboles AVL están *siempre equilibrados*, de tal modo que la altura del ***subárbol izquierdo*** no difiere en más de una unidad de la altura del ***subárbol derecho***. Para ello utiliza las "**rotaciones AVL**" y un **Factor de Equilibrio** (que es un elemento adicional que se usa en cada nodo para determinar si se mantiene o no la condición de AVL. Posee tres estados: ***balanceado***, ***desbalanceado*** con carga a la **derecha**/**izquierda**)

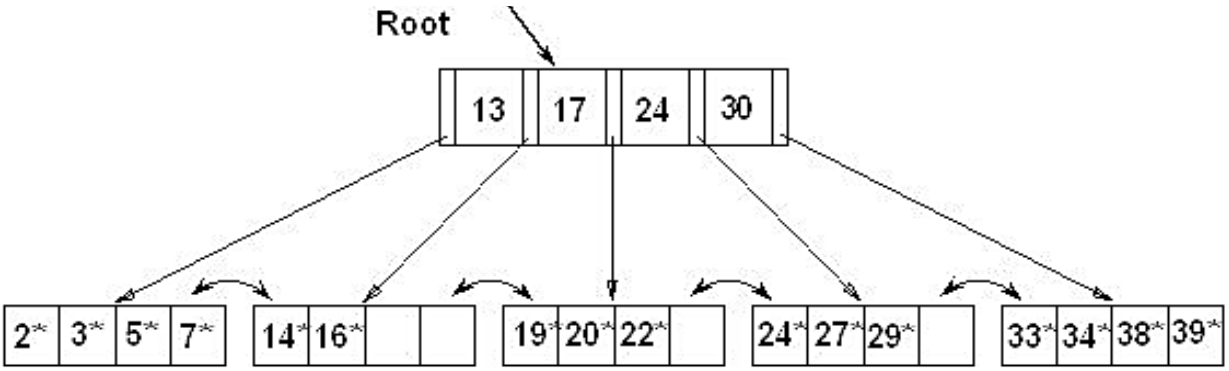


- **Rotación Simple**: el exceso en profundidad se da en un sentido, el crecimiento es a *izquierda* o a *derecha*. La raíz del sub-árbol donde se produjo el desbalance sube un nivel, y a partir de aquí el árbol se reestructura como un árbol binario de búsqueda.
- **Rotación doble**: se da para el caso en el que el desbalance no se produce en un sentido único. La raíz del sub-árbol en el que se produjo el exceso sube dos niveles, se efectúan dos rotaciones.

Árbol B

Es una estructura multimodal de **orden *m***. Cumple con las siguientes condiciones:

- Todas las ***hojas*** están al mismo nivel
- Todos los ***nodos internos*** tienen a lo sumo ***M*** *hijos* y como mínimo ***m/2*** hijos.
- El ***número de claves*** en cada *nodo interno* es ***m-1***



La **búsqueda** es similar a la de los *árboles binarios*. Se empieza en la ***raíz***, y se recorre el árbol hacia abajo, escogiendo el sub-nodo de acuerdo a la posición relativa del valor buscado respecto a los valores de cada nodo. Típicamente se utiliza la búsqueda binaria para determinar esta posición relativa.

Las **inserciones** se hacen en los ***nodos hoja***.

Realizando una búsqueda en el *árbol*, se halla el **nodo hoja** en el cual debería ubicarse el *nuevo elemento*.

Si el **nodo hoja** tiene menos elementos que el *máximo número de elementos permitidos* (***m-1***), entonces hay lugar para uno más. Inserte el nuevo elemento en el nodo, respetando el orden de los elementos.

De otra forma, el nodo debe ser dividido en dos nodos.

La ***división*** se realiza de la siguiente manera:

1) Se escoge el **valor medio** entre los elementos del *nodo* y el *nuevo elemento*.

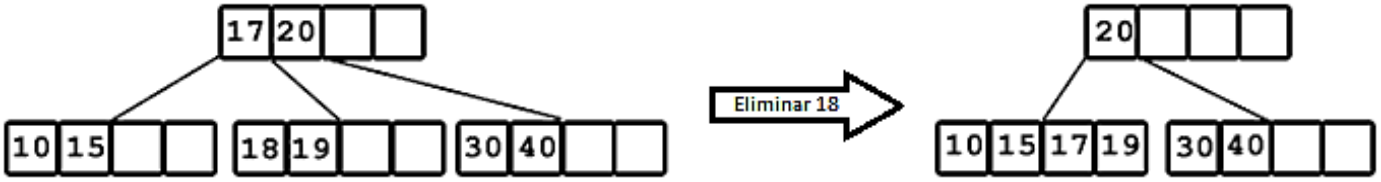
2) Los valores menores que el valor medio se colocan en el **nuevo nodo izquierdo**, y los valores mayores que el valor medio se colocan en el **nuevo nodo derecho**; el valor medio actúa como ***valor separador***.

3) El ***valor separador*** se debe colocar en el **nodo padre**, lo que puede provocar que el padre sea dividido en dos, y así sucesivamente.

Eliminación

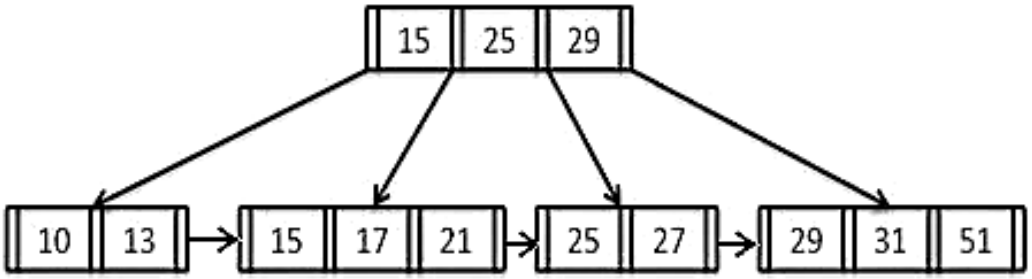
Es una operación más complicada, consiste en eliminar o quitar la clave sin violar los principios del ***árbol B***.

Si la clave a borrar se encuentra en una ***hoja*** simplemente se suprime, si no debe bajarse la ***clave adyacente*** de la *página antecedente* y sustituir esta clave por la que se encuentra más a la derecha del ***sub-árbol izquierdo*** o por la que se encuentra más a la izquierda del ***sub-árbol derecho***.



Árbol B+

Similar al **Árbol B**, pero al dividir un *nodo hoja* se deja una ***copia del padre*** en el *nodo izquierdo*. Además cada *nodo hoja* apunta a su ***hermano*** o ***primo***.



Principales características de los **arboles B+** de ***orden m*** son:

- La **raíz** almacena como mínimo **1** dato y como máximo $m - 1$ datos.
- La **página raíz** tiene como mínimo **2** ***descendientes***.
- Las **páginas intermedias** tienen como ***mínimo*** $(m - 1)/2$ (*Parte entera*) datos
- Las **páginas intermedias** tienen como ***máximo*** $m - 1$ datos.
- Todas las **paginas hojas** tienen la misma ***altura***
- La **información** se encuentra ***ordenada***.
- Toda la **información** se encuentra almacenada en las ***páginas hoja***, por lo que en las páginas internas se puede duplicar las **claves**.

Inserción en un árbol B+

Es similar a la del ***árbol B*** se diferencia en el momento que una *página* deja de cumplir la condición del número de datos almacenados. Para realizar se debe subir una copia de la ***clave mediana*** de los datos del nodo a la ***página padre***, solo se duplica la información cuando la clave que sube es de una página hoja.

Los **pasos** son los siguientes:

1. Se ubica en la ***página raíz***.

2. Se evalúa si es una ***página hoja***

2.1. Si la respuesta es *afirmativa* se evalúa si NO sobrepasa los **límites** de datos

2.1.1. Si la respuesta es *afirmativa*, entonces se procede a insertar el nuevo valor en lugar del correspondiente.

2.1.2. Si la respuesta es *negativa*, se divide la página en **2**, se sube una *copia* de la ***mediana*** a la **página padre**, si la *página padre* se encuentra llena se debe de partir igual y así el mismo proceso hasta donde sea necesario. Si se llega hasta la ***raíz***, la **altura** del árbol aumenta en **1**.

2.2. Si no es **hoja**, se compara el elemento a insertar con cada uno de los valores almacenados para encontrar la ***página descendiente*** donde proseguir la búsqueda. Se regresa al paso **1**.

Eliminación en un árbol B+

Es más simple que en **árboles-B**, porque las claves a eliminar siempre se encuentran en las páginas hojas.

Deben distinguirse los *siguientes casos*:

1) Si al eliminar una **clave**, la *cantidad de llaves* es mayor o igual que $m/2$ entonces termina la operación. Las claves de los ***nodos raíz*** o ***internos*** no se modifican por más que sean una copia de la clave eliminada en las **hojas**.

2) Si al eliminar una **clave**, la *cantidad de claves* es menor que $m/2$ entonces debe realizarse una ***redistribución de claves***, tanto en el **índice** como en las **páginas hojas**

Barridos

Es un algoritmo que devuelve la *secuencia* de ***todos los nodos*** contenidos en un árbol. Todo árbol considerado como estructura es accedido por la raíz.

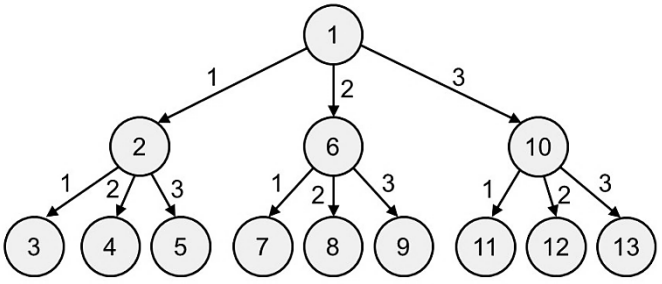
Hay **4** criterios para el barrido de árboles, **3** de los cuales son aplicables sólo en ***árboles binarios*** y el cuarto barrido (**por niveles**) es aplicable a cualquier árbol Los criterios son: **preorden**, **postorden**, **simétrico** (***postorden***) y **por niveles**

Los recorridos de un árbol en los órdenes previo y posterior permiten determinar los antecesores de un nodo.

- **PREORDEN**: primero se procesa la ***raíz***, seguido del ***subárbol izquierdo*** y por último el ***subárbol derecho*** (**R-I-D**)
- **INORDEN**: primero se procesa el ***subárbol izquierdo***, seguido de la ***raíz*** y por último el ***sub-árbol derecho*** (**I-R-D**)
- **POSTORDEN**: primero se procesa el ***subárbol izquierdo***, seguido del ***subárbol derecho*** y por último la ***raíz*** (**I-D-R**)
- **RECORRIDO POR NIVELES**: se muestra la ***raíz***, y se sigue hacia abajo de ***izquierda*** a ***derecha***.

Arboles N-arios

Es una estructura de datos donde cada nodo posee un ***número indeterminado de hijos***. Es una estructura ***recursiva***, y corresponde a la generalización de un árbol binario de cuyos nodos pueden desprenderse múltiples arboles binarios.



Las reglas que aplican a los arboles binarios se pueden transpolar a los árboles n-arios así como los consejos base.

Un **árbol n-ario** posee ***n*** **elementos** asociados a cada uno de sus componentes. Encontramos **3** tipos de recorridos: **INORDEN – PREORDEN - POSTORDEN**

- Posee los mismos conceptos que un *árbol binario*: **Nodo**, **Raíz**, **Hoja**, **Camino**, **Rama**, **Altura**, **Peso**.
- Las 3 operaciones que se pueden hacer a partir de estos árboles son: **Adición**, **Sustracción**, **Búsqueda**.

Respuestas

	Binario	Balanceados	*AVL*	Multimodales	*B*
CARACTERISTI.	-Es binario. -Es organizado. -Todas las claves del subárbol izq. son menores que el padre y del subárbol der. son mayores.	-Puede ser binario ó n-ario. -Todas las ramas izq. y der. tienen la misma altura. -Todas las hojas están al mismo nivel. -Es un árbol lleno.	-Es binario. -La altura de los subárboles izq. y der. difieren a lo sumo en 1. -Todos los subárboles deben cumplir con ello.	-Es de orden *m*. -*m* es la cant. máx. de hijos. -*k* es la cant. de claves por nodos. -Cada nodo contiene *k*-1 clave. -Si tiene 5 hijos tiene 4 claves.	-Es Multimodal. -Los nodos terminan en igual nivel. -El nº de clave es uno menos que el de hijos. -La raíz tiene como máx. *m*. Hijos y mín. *m*/2 y puede tener 0 si solo es raíz.
ALTAS	-El nº es raíz -El menor a la izq. -El mayor a la der.	-El nº es raíz -El menor a la izq. -El mayor a la der.	-Menor a la izq. -El mayor a la der. -Si la altura de los subárboles izq. y der. difieren a lo sumo en 1 ⇒ rotación.	-Si la clave es nueva y el nodo está lleno se da de alta la hoja. -si está lleno se divide el nodo en 2 en el mismo nivel y la clave sube al padre.	
BAJAS	-Si es hoja se borra. -Si tiene un hijo, sube el hijo. -Si tiene dos hijos, sube el de la der.	-Si es hoja se borra. -Si tiene un hijo, sube el hijo. -Si tiene dos hijos, sube el de la der.	-Si el nodo es raíz sube a la izq. ó der. y se acomoda. -Si no es raíz y tiene dos hijos sube el de la der. -Si tiene un hijo sube ese. -Si no tiene hijos se borra.	-Si no está en una hoja se elimina y se pone el predecesor en su lugar. -Si está en una hoja con más del mín. de claves, se borra. -Si contiene el nº mín. de claves.	
DIFERENCIA			-Cada nodo posee una sola clave. -Se da de alta comprobando con solo la raíz. -Es de orden 2.	-Puede tener hijos vacíos.	-Cada nodo puede poseer *m*-1 claves ordenadas. -Todas las hojas están en igual nivel. -Se da de alta en los hijos. -No puede tener hijos vacíos.

GRAFOS

Son una ED que están formados por un conjunto de ***nodos*** (denominado **P**, y simboliza los **objetos**) y un conjunto de ***arcos*** (denominado **R**, simboliza las ***relaciones*** *entre los objetos*).

Un **grafo** está definido (***por comprensión***) de la siguiente manera: $G = (P, R)$

$$P = \{x/x\ es\ un\ nodo\ del\ modelo\}$$

ej: $P = \{elem1,\ elem2,\ldots,\ elem\ n\}$

$$R = \{(x, y)/(x, y) \in P \wedge \text{“}x\ está\ relacionado\ con\ y\text{”}\}$$

ej: $R = \{(x_1, y_1),\ (x_2, y_2),\ldots,\ (x_n, y_n)\}$

- **Funciones de asignación**: Características o propiedades de los nodos o de las relaciones.
- **Función de asignación a nodo**: Caracterizan al conjunto de nodos
- **Función de asignación a arco**: Caracterizan a las relaciones.

Sabiendo esto, podemos definir lo siguiente:

- **SUBGRAFO**: es un grafo incluido en otro. $G' = (P', R')$ es un subgrafo de $G = (P, R)$ si P' está incluido o es igual a P y si R' es igual a R sobre los valores de P'.

Sea G'= (P', R') y G= (P, R); G' es subgrafo de G si P' ⊆ P y R' = R|P

- **PASO**: $\rho(x, z)$ es la secuencia de $\langle y_0,\ y_1,\ \cdots,\ y_n\rangle$, con $n \geq 0$ que cumple con las siguientes condiciones:
  - $x = y_0$ y $z = y_n$ [todo paso tiene ***origen*** (x) y ***final*** (y)]

    Un número **NO** puede ser igual al anterior o al siguiente
  - $y_{i-1} \neq y_i$ (***Secuencialidad***: no hay *pasos consecutivos* iguales)
  - $(y_{i-1},\ y_i) \in R$ con $1 \leq i \leq n$ (***Paso simple***: los elementos consecutivos deben estar a un arco de diferencia entre uno y otro)

- **LONGITUD DE PASO**: cantidad de **arcos** que hay entre el <u>*nodo origen*</u> (***inicial***) y el <u>*nodo destino*</u> (***final***) de la secuencia de puntos

$\lvert\rho(x, z)\rvert =$ cantidad de ***arcos*** entre x y z

- **CAMINO**: $C(x, z)$ es la secuencia $\langle y_0,\ y_1,\ \cdots,\ y_n\rangle$, con $n \geq 0$ en la que no interesa el sentido de la relación y que cumple las siguientes condiciones:
  - $x = y_0$ y $z = y_n$ [todo paso tiene ***origen*** (x) y ***final*** (y)]
  - $y_{i-1} \neq y_i$ (no hay *pasos consecutivos* iguales)
  - $(y_{i-1},\ y_i) \in R \ \vee\ (y_i,\ y_{i-1}) \in R$ con $1 \leq i \leq n$
- **LONGITUD DE CAMINO**: $\lvert C(x, z)\rvert =$ cantidad de **arcos** en ese ***camino***
- **CICLO**: es un **paso** entre un nodo y sí mismo, cuya ***longitud de paso*** es **>1**

$\lvert\rho(x, x)\rvert \geq 2$ (longitud de paso entre x y x)

- **CIRCUITO**: es una secuencia de nodos entre x y x de longitud mayor o igual a **2**, sin importar el sentido de la relación.

$\lvert C(x, x)\rvert \geq 2$ (cantidad de nodos entre x y x)

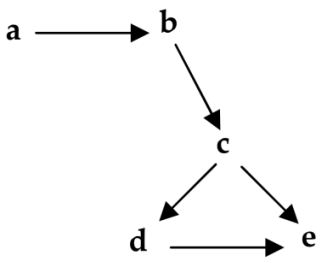
- **LOOP o Bucle**: es un paso entre un nodo y sí mismo de longitud **0**. Paso de longitud cero y cuyo nodo origen es igual al nodo destino.

$$\lvert\rho(x, x)\rvert = 0$$

- **CONJUNTO IZQUIERDO (*Left*)**: conjunto de nodos desde los cuales se puede llegar al nodo considerado con longitud de paso igual a 1 (es decir, directamente)

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

<u>*Ejemplo:*</u>

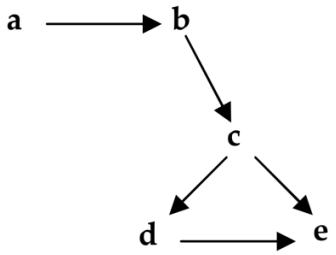


L (e) = {c, d}

- **CONJUNTO DERECHO (*Right*)**: es el conjunto de nodos a los cuales se puede llegar desde el nodo considerado, con longitud de paso igual a 1.

$$R(x) = \{y/y \in P \wedge (x,y) \in R\}$$

Ejemplo:



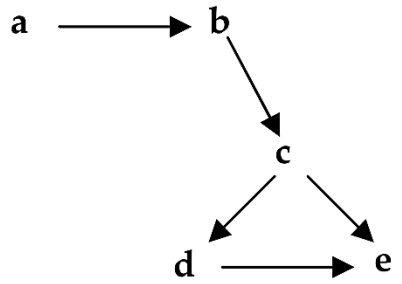
R (c) = {d, e}

- **GRADO DE ENTRADA**: cantidad de **arcos** que llegan a un nodo. $|L(x)|$
- **GRADO DE SALIDA**: cantidad de arcos que salen de un nodo. $|R(x)|$
- **GRADO DE ENTRADA (*SALIDA*) DE UN GRAFO**: grado del nodo que posee el mayor grado de **entrada** (***salida***) del grafo.
- **CONJUNTO IDEAL IZQUIERDO**: es el conjunto de nodos desde los cuales se puede llegar al nodo considerado, directa o indirectamente.

$\overline{L(y)} = \{ x/x \in P, \exists \rho (x, y) \}$

Ejemplo

$\overline{L(d)} = \{a, b, c\}$



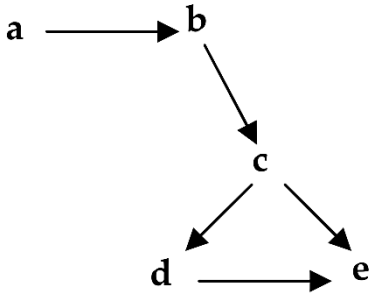
- **CONJUNTO IDEAL DERECHO**: es el conjunto de nodos a los cuales puedo llegar desde el nodo considerado, directa o indirectamente.

$\overline{R(y)} = \{ z/z \in P, \exists \rho (y, z) \}$

Ejemplo

$\overline{R(a)} = \{b, c, d, e\}$

$\overline{R(d)} = \{e\}$



- **CONJUNTO MINIMAL**: conjunto de puntos a los cuales no llega ningún arco. El grado de entrada es **0**
- **CONJUNTO MAXIMAL**: conjunto de puntos desde los cuales no sale ningún arco. El grado de salida es **0**

- **MÍNIMO**: único nodo en el grafo al cual no le llegan arcos. Se da cuando el ***conjunto minimal*** está formado por un único nodo.

$$y \text{ es mínimo} \Leftrightarrow \forall x \in P, \exists \rho(y, x) \wedge L(y) = \varnothing$$

- **MÁXIMO**: único nodo en el grafo desde el que no salen arcos. Se da cuando el ***conjunto maximal*** está formado por un único nodo.

$$y \text{ es máximo} \Leftrightarrow \forall x \in P, \exists \rho(x, y) \wedge R(y) = \varnothing$$

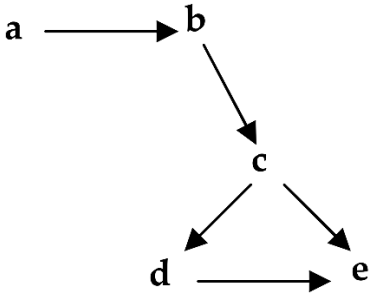
- ***Ideal Principal Derecho***: es el *conjunto de nodos* a los cuales **PUEDO LLEGAR** desde el nodo considerado, directa o indirectamente.

$$\overline{R(y)} = \{ z / z \in P, \exists \rho (y, z) \}$$

Ejemplo:

$\overline{R(a)} = \{b, c, d, e\}$

$\overline{R(d)} = \{e\}$

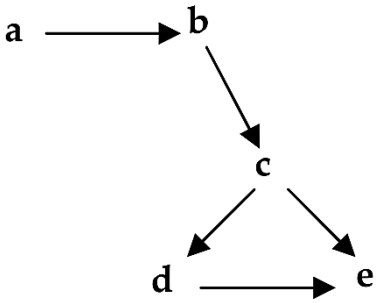


- ***Ideal Principal Izquierdo***: es el *conjunto de nodos* desde los cuales **SE PUEDE LLEGAR** al nodo considerado, directa o indirectamente

$$\overline{L(y)} = \{ x / x \in P, \exists \rho (x, y) \}$$

Ejemplo

$\overline{L(d)} = \{a, b, c\}$



TIPOS DE GRAFOS

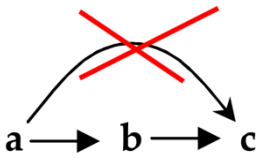
Grafo Básico

dado un grafo ***G***, se dice que es ***básico*** si cumple con las siguientes condiciones:

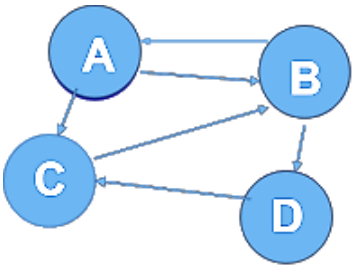
- Está libre de ***bucles*** o loops.
- $\forall x, y \in P, \quad si \exists |\rho(x, y)| \geq 2 \rightarrow (x, y) \notin R$ (no existen *arcos redundantes*)

Para encontrar un grafo básico hay que ir eliminando pasos que estén demás, sin destruir la estructura. Por lo general, se elimina el que tiene menor longitud de paso

Ejemplo: si R= {(a, b); (b, c); (a, c)}



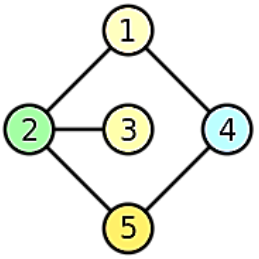
Grafo Cíclico



Grafo que contiene por lo menos **1 ciclo**. Puede existir más de una representación básica de él

$$\forall x \in P, \quad \exists\, |\rho(x,x)| \geq 2$$

Grafo Conexo



- No tiene ***puntos aislados***
- Existe un ***camino*** entre dos nodos cualesquiera
- Todos sus pares de nodos están *conectados*

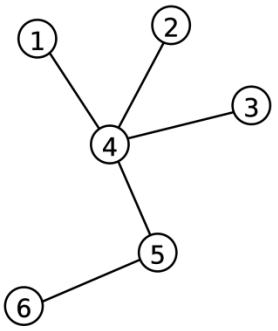
Grafo Lineal: ***G*** es ***lineal*** ⇔ se verifica que

- $\exists\, x \in P / L(x) = 0$ (hay al menos un punto al cual no le llegan arcos –***minimal***–)
- $\forall y \in P, \ |L(y)| \leq 1, \ |R(y)| \leq 1$ (grado de entrada y salida ≤1)
- **G** debe ser **conexo**

Ejemplo:

a ⟶ b ⟶ c ⟶ d es un Grafo Lineal

Grafo Finito



Si **P** es un *conjunto finito*. Cantidad máxima de flechas que puede haber en un grafo G finito = $n^2 - n$. (***n*** es la cantidad de nodos o puntos).

Grafos Isomorfos

Sea $G_1 = (P_1, R_1)$ y $G_2 = (P_2, R_2)$, se dice que son **isomorfos** si:

$$\exists \varphi: P_1 \longrightarrow P_2 / \forall\ x, y \in P_1, (x,y) \in R_1 \Leftrightarrow (\varphi(x), \varphi(y)) \in R_2 \wedge \varphi(x), \varphi(y) \in P_2$$

Grafos que tienen la misma estructura pero sus nodos (componentes) distintos

G_1= a ⟶ b ⟶ c

G_2= 1 ⟶ 2 ⟶ 3

φ(a)=1; φ(b)=2; φ(c)=3;

Hipergrafo

Grafo que tiene un conjunto de puntos y más de una relación definida en dicho conjunto. Sobre el mismo modelo de la realidad es necesario analizar distintas relaciones. Las funciones de asignación a nodos no varían porque el conjunto de puntos es el mismo, pero a las funciones de ***asignación a arcos*** se agregan aquellas características que corresponden a los arcos de las nuevas relaciones.

Esquema resultante: $H = (P, R, R', \ldots, R^{(n)}, f_1, f_2, \ldots, f_k, g_1, g_2, \ldots, g_m, h_1, h_2, \ldots, h_p, \ldots)$

Grafos Irrestrictos

Son grafos que definen su estructura a partir del modelo de la realidad que se desea implementar. Esto hace que *no se conozca a priori la cantidad de nodos ni la cantidad de arcos necesarios, ni las relaciones*, etc.

Los grafos pueden ser representados por un *arreglo bidimensional* (**matriz de adyacencia**), o bien una representación dinámica mediante una *estructura multi-enlazada* (**lista de adyacencia**) y ***Lista de Adyacencia Completa***.

La elección depende del tipo de grafo y de las operaciones que se vayan a realizar sobre los nodos y los arcos.

- **Matriz de Adyacencia**: el grafo se representa mediante una ***matriz*** $\mathbf{A_{NxN}}$ (siendo **N** el número de nodos), tal que todo elemento a_{ij} puede tomar los valores:

  0 si **NO** hay **arco** (v_i, v_j)

  1 si hay un **arco** (v_i, v_j)

Estableciendo las relaciones se obtiene el número de arcos.

Siempre se lee de **Columna** a **Fila**.

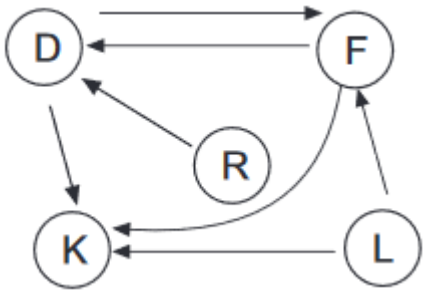
Representa un ***listado de nodos***. Indica mediante alguna condición de la matriz, cuales son los nodos adyacentes.

Características

- N debe conocerse de antemano, es decir se usan Grafos poco volátiles.

- Permite ejecutar consultas rápidas.
- Estas matrices no se almacenan, se alojan en memoria.
- Es rígida tengo que saber con anticipación la cantidad de datos, de lo contrario podría definir una mega matriz, solo para tener o generar muchos espacios subocupados.
- Tiene mucha dispersión de datos
- Ocupa muchísimos espacios para guardar alguna información.
- Cuando la cantidad de arcos es significativamente menor que la cantidad de nodos, resulta contraproducente Ej: mil nodos, 5 arcos. Es un gran trabajo para obtener pocos resultados.

$$A = \begin{vmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix}$$



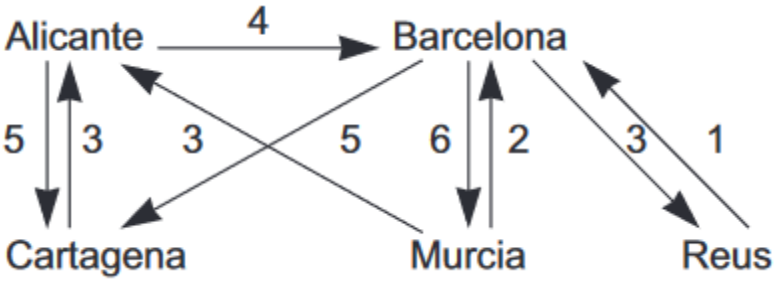
Donde

$$P = \{D,\ F,\ K,\ L,\ R\}$$

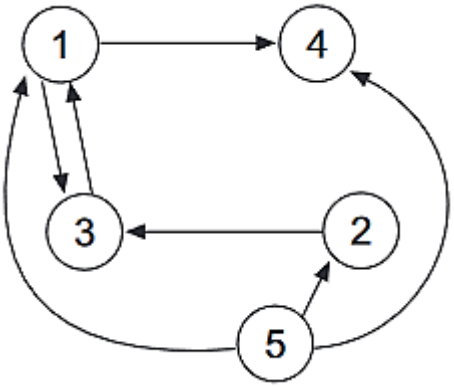
$$R = \{(D, F);\ (D, K);\ (F, D),\ (F, K);\ (L, F);\ (L, K);\ (R, D)\}$$

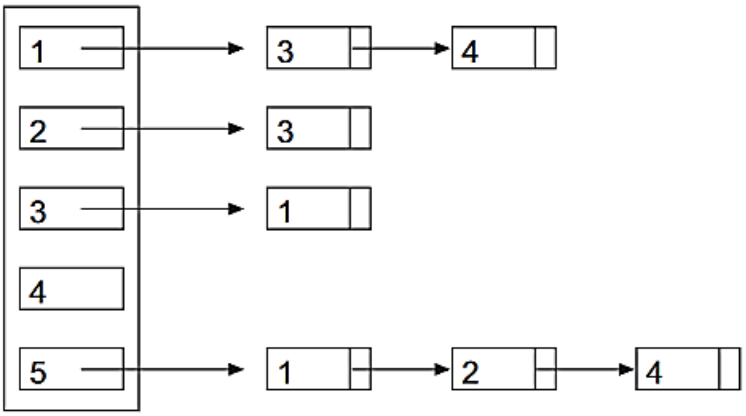
En el caso de que se trate de arcos con un valor se hace uso de una matriz valorada

$$P = \begin{vmatrix} 0 & 4 & 5 & 0 & 0 \\ 0 & 0 & 3 & 6 & 3 \\ 3 & 0 & 0 & 0 & 0 \\ 5 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{vmatrix}$$



- ***Lista de Adyacencia***: es una estructura multi-enlazada formada por una "*tabla directorio*" donde cada nodo representa un elemento (**vértice**) del grafo, del que emerge una lista enlazada con todos sus vértices adyacentes. Las listas de adyacentes no son ordenadas.


1
4
3
2
5


1
2
3
4
5
3
4
3
1
1
2
4

PREGUNTAS Y DEMOSTRACIONES

Un **Árbol** es un **grafo** que:

1) es ***conexo*** y conectado.
2) $|P| = |R| + 1$.
3) cualquier arco adicional crea un ***Circuito***.
4) Libre de ***loops***.
5) El ***camino*** entre un nodo y otro es *único*

¿Una *lista* es un *Grafo*?: si, se considera que la lista es un Grafo con restricción.

¿Un grafo puede ser una lista?: No

¿Un árbol es un grafo?: Si, es un grafo con restricciones definidas

¿Un grafo es un árbol?: Si. Se puede interpretar que un ***árbol B+*** también es un ***grafo***. El grafo no tiene restricciones y todo lo que hemos estado viendo han sido grafos con restricciones.

¿Una multilista es un Hipergrafo?: Considerando la definición de ***Hipergrafo***, la cual dice que para un mismo conjunto de puntos pueden existir varias relaciones y a su vez funciones de asignación a nodos y funciones de asignación a arcos, y la definición de multilista, que nos dice que es una lista que contiene varias listas en su interior; podemos decir que una ***multilista*** es un ***Hipergrafo***.

¿Las listas doblemente enlazadas son multilista?
No, ya que las listas doblemente enlazadas contienen dos campos de enlace y se puede avanzar y retroceder en la lista, precisamente porque contiene puntero al antecesor y al sucesor, cosa que no ocurre con las multilista, que contienen listas como información y no se puede retroceder sobre la lista.

¿Una lista doblemente enlazada es una estructura lineal?
Una lista doblemente enlazada no es una estructura lineal, ya que sus elementos tienen más de un antecesor y más de un sucesor.

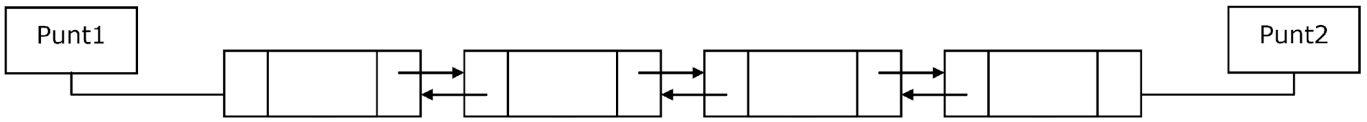
Una **estructura irrestricta** (por ejemplo, los diseños de celda definidos por Phaltz) puede utilizarse para implementar un ***árbol binario*** ya que toda estructura irrestricta es un grafo en el que no es restricción la forma en que los nodos están conectados

Integridad: característica de las *bases de datos* que indica la **ausencia** de ***datos inconsistentes***.

Sea el grafo G = (P, R), G es lineal ⇔ G es básico. Esta afirmación es falsa debido a que un grafo básico puede tener más de una entrada a un nodo. Por ejemplo si tenemos un grafo G = (P, R) donde P = {*a*, *b*, *c*}y R = {(*a*, *c*), (*b*, *c*)} entonces G es básico pero no lineal ya que |L(c)| = 2, con lo cual se viola una de las condiciones de Grafo Lineal.

Existen datos que se utilizan en distintos sitios, la **consistencia** implica que esos datos son iguales en todos los lugares de trabajo, al mismo tiempo.

Una ***cola de doble entrada*** **NO** es una estructura lineal, ya que sus elementos tienen más de un ***antecesor*** y más de un ***sucesor***



Es posible que el grado de entrada de un nodo x de un árbol sea ≥ 2, si estamos hablando de ***árbol principal izquierdo***.

Diferencia (funcionales y estructurales) entre AVL y Árbol B

AVL	Árbol B
Orden 2 (binario)	orden m
cada nodo almacena una única clave	cada nodo puede almacenar como máximo $(m-1)$ claves

La principal diferencia que tiene un **Árbol B** con un **AVL** es en cuanto al manejo de datos dentro de los nodos.

La condición de que todas las hojas se encuentren en un mismo nivel impone un comportamiento en árboles B, a diferencia de los árboles de búsqueda, ya que no pueden crecer en sus hijos y se ven obligados a crecer por la raíz.

Demostrar que $\exists\rho(x,z) \Leftrightarrow \overline{R}(x) \cap \overline{L}(z) \neq \phi$ a partir de las definiciones de la Teoría de Grafos.

No se puede demostrar una afirmación que no es válida para todos los casos:
Por ejemplo, si tenemos un grafo G = (P, R) donde P = {*x*, *z*} y R = {(*x*, *z*)} entonces
$\exists\rho(x,z);\ \overline{L}(z) = x$ y $\overline{R}(x) = z\ \therefore \overline{L}(z) \cap \overline{R}(x) = \phi$ con lo cual concluimos que no verifica la afirmación planteada.

11. Sea el grafo G = (P, R), G es árbol ⇒ G es básico

Si un grafo G es árbol entonces cuenta, entre otras, con las siguientes dos propiedades, cada arco es un arco desconectante, esto implica que todo árbol está ***libre de loops***, y dados dos puntos cualesquiera *x*, *z* ∈ a P, existe camino de *x* a *z* y ese camino es único, esto nos está indicando que ***no existen arcos redundantes***. Para concluir las dos condiciones que deben cumplirse para que un grafo sea básico son éstas, por lo tanto la afirmación es correcta.

Demostrar que $\exists\rho(x,z) \Leftrightarrow \overline{R}(x) \cap \overline{L}(z) \neq \phi$ sabiendo que P = {x, y, z} a partir de las definiciones de la Teoría de Grafos.

<u>1º Parte</u>: si $\exists\rho(x,z) \rightarrow \overline{R}(x) \cap \overline{L}(z) \neq \phi$

Aseveramos que $\exists\rho(x,z) = \{x_0, x_1, ...x_n\}$ donde $x = x_0$ y $z = x_n$ entonces podemos determinar que

$$\overline{R}(x) = \{^y / \exists\rho(x,y)\} = \{x_1, x_2, ..., x_n\} \text{ y que}$$

$$\overline{L}(z) = \{^y / \exists\rho(y,z)\} = \{x_{n-1}, ..., x_1, x_0\} \text{ y por lo tanto que}$$

$$\overline{R}(x) \cap \overline{L}(z) = \{x_1, x_2, ..., x_{n-1}\} \neq \phi$$

Concluimos entonces que esta primera parte es cierta pero se debe demostrar el otro camino.

<u>2º Parte</u>: si $\overline{R}(x) \cap \overline{L}(z) \neq \phi \rightarrow \exists\rho(x,z)$

Como $\overline{R}(x) \cap \overline{L}(z) \neq \phi$ entonces $\exists w / w \in \overline{R}(x) \wedge w \in \overline{L}(z)$ y por lo tanto al tener un punto en común existe una secuencia de puntos que permiten ir de *x* a *z*, es decir existe el conjunto $\{x, x_1, x_2, x_3, ..., w, ...z_n, z\}$ que define un paso de *x* a *z*.