Conjunto: Formado por elementos no repetidos de la misma naturaleza, es decir, elementos diferenciados entre sí pero que poseen en común ciertas propiedades o características, y que pueden tener entre ellos, o con elementos de otros conjuntos, ciertas relaciones.

Multiconjunto: Admite elementos repetidos.

Relación Binaria: Es una relación R existente entre dos elementos a y b, de dos conjuntos A y B respectivamente. Esta relación se puede denotar de diversas formas:

- (a, b)
- aRb

Si A, B son conjuntos, una relación de A en B es cualquier subconjunto de $A \times B$

Función: Para los conjuntos no vacíos A, B una función f de A en B ($f: A \to B$) es una relación de A en B en la que cada elemento de A aparece exactamente una vez como la primera componente de un par ordenado de una relación. Hay UNICIDAD y EXISTENCIA.

Por ejemplo:

Si $A = \{1,2,3\}$ y $B = \{w, x, y, z\}$, entonces, podemos definir a f como $\{(1, w), (2, x), (3, x)\}$.

Pero, por ejemplo, los siguientes conjuntos no son funciones:

- $f' = \{(1, w), (2, x)\}$
- $f' = \{(2, w), (2, x)\}$

Definición: Para la función $f: A \to B$, A es el dominio de f y B es el codominio de f. El subconjunto de B formado por aquellos elementos que aparecen como segundas componentes de los pares ordenados de f se conoce como la imagen de f y se denota también como $f(A)$.

Relación Compuesta: Si A, B, C son conjuntos y $R_1 \subseteq A \times B$ y $R_2 \subseteq B \times C$, entonces la relación compuesta $R_1 \circ R_2$ es una relación de A en C definida como:

$$R_1 \circ R_2 = \{(x, z) \mid x \in A, z \in C, y \in B \ tal \ que \ (x, y) \in R_1, (y, z) \in R_2\}$$

Relación Inversa:

$$R^{-1} = \{(y, x) \mid (x, y) \in R\}$$

Producto Cartesiano:

$$A \times B = \{(x, y) \mid x \in A \ y \ Y \in B\}$$

Complemento de R:

$$\bar{R} = \{(x, y) \mid (x, y) \ no \ pertenece \ a \ R \ y \ (x, y) \in X^2\}$$

Identidad:

$$\Delta = \{(\boldsymbol{x}, \boldsymbol{x}) \mid x \in X\}$$

Propiedades de una relación

- $\boldsymbol{Reflexividad} \rightarrow \forall\, \boldsymbol{x} \in \boldsymbol{A}: (\boldsymbol{x}, \boldsymbol{x}) \in \boldsymbol{R}$
- $\boldsymbol{Antireflexividad} \rightarrow \forall\, \boldsymbol{x} \in \boldsymbol{A}: (\boldsymbol{x}, \boldsymbol{x})\ \boldsymbol{no\ pertenece\ a\ R}$
- $\boldsymbol{Simetria} \rightarrow \forall (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{A}: (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{R} \Longrightarrow (\boldsymbol{y}, \boldsymbol{x}) \in \boldsymbol{R}$
- $\boldsymbol{Antisimetria} \rightarrow \forall (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{A}: (\boldsymbol{x}, \boldsymbol{y})\ \boldsymbol{no\ pertenece\ a\ R} \Longrightarrow (\boldsymbol{y}, \boldsymbol{x}) \in \boldsymbol{R}$
- $\boldsymbol{Simetria\ Débil} \rightarrow \forall (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{A}: (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{R}\ \boldsymbol{y}\ \boldsymbol{x} \neq \boldsymbol{y} \Longrightarrow (\boldsymbol{y}, \boldsymbol{x})\ \boldsymbol{no\ pertenece\ a\ R}$
- $\boldsymbol{Transitividad} \rightarrow \forall\, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \boldsymbol{A}: (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{R}\ \boldsymbol{y}\ (\boldsymbol{y}, \boldsymbol{z}) \in \boldsymbol{R} \Longrightarrow (\boldsymbol{x}, \boldsymbol{z}) \in \boldsymbol{R}$
- $\boldsymbol{Comparabilidad} \rightarrow \forall\, \boldsymbol{x} \in \boldsymbol{A}: (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{R}\ \boldsymbol{o}\ (\boldsymbol{y}, \boldsymbol{x}) \in \boldsymbol{R}$

Conjunto de partes: $P(X)$ es un conjunto de partes tal que está formado por todos los subconjuntos del conjunto dado (en este caso seria X).

Por ejemplo:

Si $X = \{a, b, c\}$ entonces $P(X) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.

Cardinalidad: Es la cantidad de elementos de un conjunto. Si tomamos un conjunto X, su cardinalidad se la denota como $|X|$. Lo llamaremos n. P(X) tiene como cardinalidad 2^{n^2}.

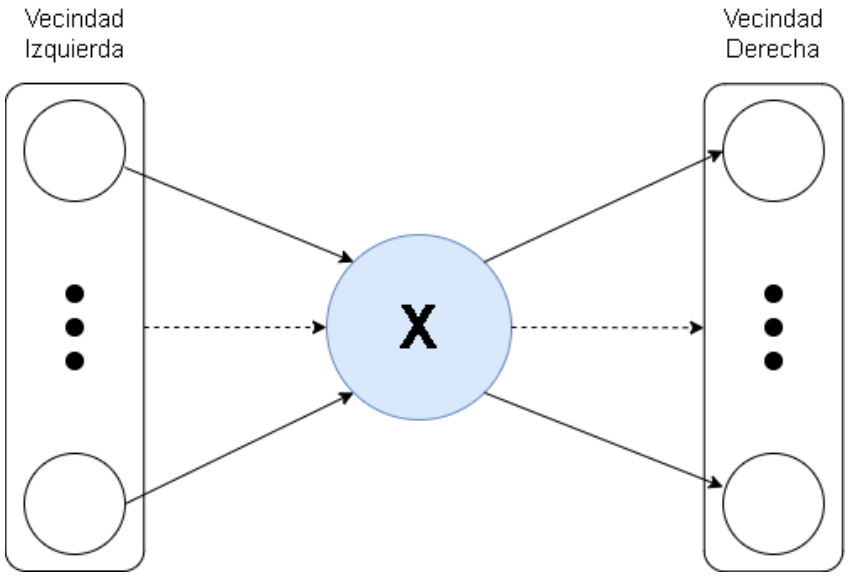
Clasificación de las relaciones

- **Orden Parcial:** Reflexiva, Antisimétrica y Transitiva.
- **Orden Total:** Reflexiva, Antisimétrica, Transitiva, y Comparable.
- **Relación de Equivalencia:** Reflexiva, Simétrica, y Transitiva.

Grafo Dirigido: Denotado como $G = (P, E)$ tal que P es un conjunto de nodos y E es una relación binaria en P (cada par ordenado se lo llama arco).

Operadores de Grafos:

- $Vecindad\ Izquierda \rightarrow L(x) = \{p \in x \mid (p, x) \in E\} \rightarrow Nodos\ que\ inciden\ sobre\ x$
- $Vecindad\ Derecha \rightarrow R(x) = \{p \in x \mid (x, p) \in E\} \rightarrow Nodos\ que\ x\ incide\ sobre\ ellos$



- $Loops(G) = \{p \in P: (p,p) \in E\}$
- $Min(G) = \{p \in P: L(p) = \emptyset\}$
- $Max(G) = \{p \in P: R(p) = \emptyset\}$
- $Aislados(G) = \{p \in P: L(p) = \emptyset \; y \; R(p) = \emptyset\}$
- $GradosDeEntrada(x) = |L(x)|$
- $GradosDeSalida(x) = |R(x)|$
- $Grado(x) = (GradosDeEntrada(x), GradosDeSalida(x))$
- $\rho(x,y) \rightarrow Paso \rightarrow Secuencia\ de\ nodos\ entre\ x\ e\ y\ tal\ que\ importa\ el\ sentido\ de\ las\ aristas$
- $\omega(x,y) \rightarrow Camino \rightarrow Secuencia\ de\ nodos\ entre\ x\ e\ y\ tal\ que\ no\ importa\ el\ sentido\ de\ las\ aristas$
- $Ideal\ Izquierdo \rightarrow \overline{L}(x) = \{w \in P \mid \rho(w,x)\ existe\}$
- $Ideal\ Derecho \rightarrow \overline{R}(x) = \{y \in P \mid \rho(x,y)\ existe\}$
- $\rho(R) \rightarrow Paso\ como\ relación$
- $\omega(R) \rightarrow Camino\ como\ relación$
- $Conectado(G) \rightarrow Para\ todo\ (x,y)\ existe\ \omega(x,y)$
- $FuertementeConectado(G) \rightarrow Para\ todo\ (x,y), existe\ \rho(x,y)$
- $Isomorfos(G_1, G_2) \rightarrow A\ continuación\ se\ habla\ en\ detalle\ a\ cerca\ del\ isomorfismo.$

Isomorfismo: Dos grafos $G_1 = (E_1, P_1)$ y $G_2 = (E_2, P_2)$ son isomorfos ($G_1 \cong G_2$) si existe una función $\varphi: P_1 \rightarrow P_2$ con la propiedad de que para todo $x, y \in P_1$, $(x,y) \in E_1$ si y solo si $(\varphi(x), \varphi(y)) \in E_2$.

Condiciones Necesarias para que dos grafos sean isomorfos:

- $|E_1| = |E_2|$
- $|P_1| = |P_2|$
- Los multiconjuntos de grado (multiconjunto formado por los grados de cada nodo de cada grafo) de ambos grafos deben ser iguales.
- $|Loops(G_1)| = |Loops(G_2)|$

Algoritmo de paso: Me da una secuencia de nodos que describe cual es la secuencia a seguir para ir de un nodo a otro (no necesariamente es el más corto).

Paso(A,B):

$OPEN \leftarrow (A, -)$

Mientras OPEN no se encuentre vacío:

$(Z, Y) \leftarrow OPEN$

$CLOSED \leftarrow (Z, Y)$

Para los elementos t en la vecindad derecha de Z (R(Z)):

Si t=B,

$CLOSED \leftarrow (B, Z). El\ algoritmo\ termina$

Para los elementos w en R(Z) - ($\pi_1(OPEN) \cup \pi_1(CLOSED)$) :

$$OPEN \leftarrow (W, Z)$$

Matriz de Adyacencia: Es una forma de representar un grafo. Si $G = (P, E)$, para todo $i, j \in P$:

$$\begin{cases} a_{ij} = 1, & (i,j) \in E \\ a_{ij} = 0, & En\ Caso\ Contrario \end{cases}$$

Estructura De Datos: Es un grafo dirigido $G = (P, E)$ con funciones de asignación:

- $F_i(x) \rightarrow Función\ de\ asignación\ a\ punto$
$$F_i: P \rightarrow V$$
- $G_i(x) \rightarrow Función\ de\ asignación\ a\ arco$
$$G_i: E \rightarrow V'$$

V y V' pueden ser los conjuntos de los reales, de los enteros, de los autos: de cualquier cosa.

$$ED = (P, E, F_1, \dots, F_n, G_1, \dots, G_n)$$

Una ED denota las relaciones (si las hay) entre ítems individuales.

Datos: Conjunto de hechos, números, o símbolos que funcionan como operandos en una computadora.

Las estructuras de dato tienen una **validación** (demostrar un alto grado de correspondencia con la realidad) y **utilidad** (poder obtener buenos resultados de lo que queremos investigar).

Relación de Camino: Si $H = (P, E)$ es un grafo. Entonces la relación ρ_H definida $(x, z) \in \rho_H$ si y solo si $\rho_H(x, z)$ existe. Es una relación tal que es reflexiva (un nodo puede ir a si mismo) y transitiva (si puedo ir de A a B y de B a C, entonces puedo ir de A a C).

Clausura Transitiva: Grafo tal que $G^T = (P, \rho_G)$. G^T es un grafo inducido de G. $G^T = G$ solo si E es transitiva.

Grafo G es fuertemente conectado: G es fuertemente conectado si ρ_G es una relación de equivalente de una única clase de equivalencia. Esto quiere decir que existe $\rho(x, z)$ para cualquier par de nodos (x, y).

$$Clase\ de\ Equivalencia \rightarrow R_x = \{y_i \mid (x, y_i) \in R\}$$

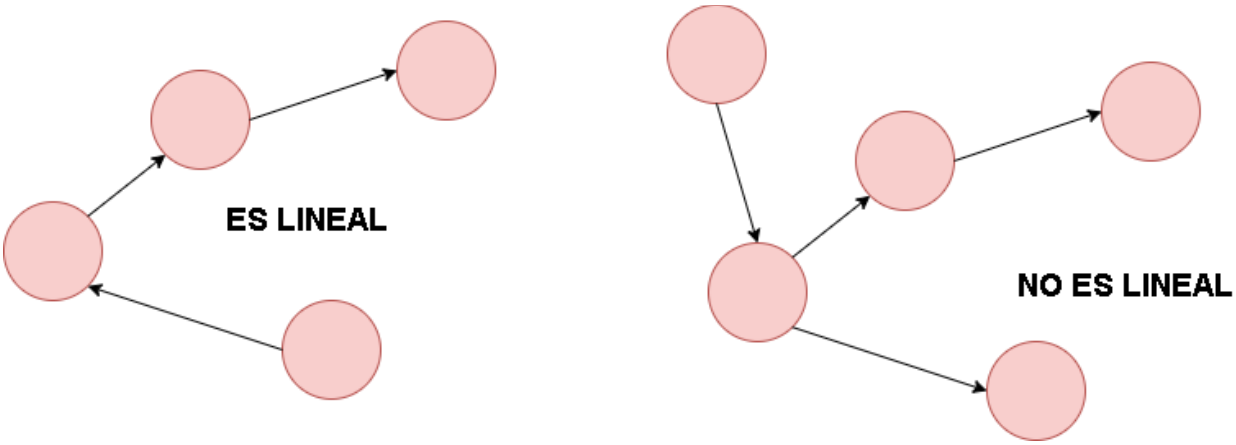
TDA (Tipo de Dato Abstracto): Es un modelo matemático con una serie de operaciones definidas en ese modelo. Un ejemplo sencillo de TDA son los conjuntos de números enteros con las operaciones UNION, INTERSECCIÓN, y DIFERENCIA.

- Tipo de Datos De Una Variable: Conjunto de valores que ésta puede tomar
- Estructura de Datos: Con esto representamos a un TDA. Son conjuntos de variables conectadas entre sí. Tiene como componente básico a la celda, lugar capaz de almacenar un valor tomado de algún tipo de datos básicos o compuestos.

Estructura de datos lineal: Un grafo es lineal si la ρ_G es de orden total y G es básico.

Si $G(P, E)$ es lineal:

- $\exists\, x\ \in P: L(x) \neq \{\emptyset\}$
- $\forall\, y\ \in P: |L(y)| \leq 1\ y\ |R(y)| \leq 1$
- G es conectado



Estructura de Datos Lineales:

- **Arreglo:** Según un índice, obtener un valor. Es de largo fijo y se puede cambiar sus valores. Es una sucesión de celdas de un tipo de dato determinado. Se lo puede ver como transformación $CONJUNTO\_INDICE -> TIPO\_DE\_CELDA$.
  Al representar relaciones de celdas podemos usar **punteros** (celda cuyo valor indica o señala otra) y **cursores** (celda de valor entero que se utiliza como punto a un arreglo).
- **Lista:** Secuencia de cero o más elementos de un tipo determinado:

  $$a_1, a_2, \dots, a_n$$

  Si $n \geq 0$ y cada a_i es de un tipo determinado y si $n = 0$, la lista está vacía. a_1 es el primer elemento y a_n.

  Es dinámica y es 'moldeable'. Una lista se puede cambiar en cualquier parte.

  Se dice que a_i precede a a_{i+1} y sucede a a_{i-1}.

  Operaciones Con Listas:

  - $\boldsymbol{INSERTA(x, p, L)}$**:** Inserta x en la posición p pasando los elementos de la posición p y siguientes a la posición inmediata posterior. Si la posición no existe, el resultado es indefinido.
  - $\boldsymbol{LOCALIZA(x, L)}$**:** Devuelve la posición de x en la lista L. Si x se repite, se devuelve la primera posición. Si x no está, se devuelve $FIN(L)$.

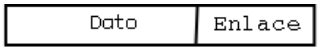
- $\boldsymbol{RECUPERA(p, L)}$: Devuelve el elemento que está en la posición p de L. El resultado es indefinido si $p = FIN(L)$ o si L no tiene posición p.
- $\boldsymbol{SUPRIME(p, L)}$: Elimina el elemento en la posición p de la lista L. El resultado no está definido si L no tiene posición p o si $p = FIN(L)$.
- $\boldsymbol{SIGUIENTE(p, L)}$:Devuelve la posición siguiente a p (Indefinido si $p = FIN(L)$)
- $\boldsymbol{ANTERIOR(p, L)}$**:** Devuelve la posición anterior a p (Indefinido si $p = 0$)
- $\boldsymbol{ANULA(L)}$: Hace que L se vacié y devuelve $FIN(L)$.
- $\boldsymbol{PRIMERO(L)}$: Devuelve la primera posición de la lista L. Si L está vacía, la posición que se devuelve es $FIN(L)$.
- $\boldsymbol{FIN(L)}$: Devuelve la posición que le sigue a la posición N.

Las listas pueden ser implementadas por arreglos. Los elementos de la lista se almacenan en celdas contiguas de un arreglo. Hace fácil acceder y agregar elementos nuevos al final, pero agregar en la mitad de la lista se complica. También en el caso de eliminar un elemento. Requiere desplazamientos de elementos para llenar espacios vacíos.

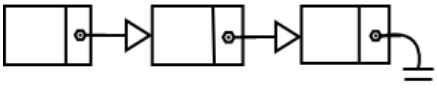
Al usar arreglos usamos un registro de dos elementos: uno un arreglo y despues la posición del último elemento.

En vez de usar arreglos se pueden usar punteros (o, mejor dicho, apuntadores) donde tenemos celdas enlazadas sencillas utilizando punteros para enlazar elementos consecutivos.

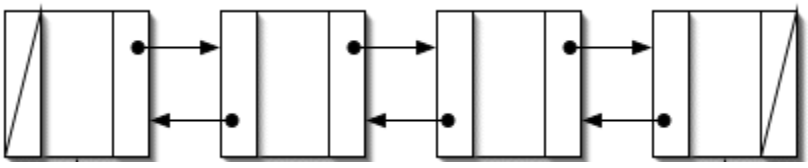
Si tenemos la lista $a_1, a_2, \dots, a_n$; la celda contiene a_i tiene un elemento de la lista y un puntero a la a_{i+1}. La celda a_n tiene un puntero nil (nil representaría "vacío").
Si no hay punteros, si simulan puntadores mediante cursores: enteros que indican posiciones.



Estructura de un nodo


Lista enlazada

- **Lista Doblemente Enlazadas:** Se tiene posibilidad de ir hacia adelante o hacia atrás en una lista (puntero al anterior y al que sigue).



- **Pila:** Tipo especial de lista en la que todas las inserciones y supresiones se dan en un extremo llamado **tope:**

También se las llama listas **LIFO (Last In, First Out).**
Tiene los siguientes operadores:

- $\boldsymbol{ANULA(P)}$: Convierte a P en una pila vacia
- $\boldsymbol{TOPE(P)}$: Devuelve el valor del elemento superior de la pila P.
- $\boldsymbol{SACA(P)}$: Suprime el elemento superior de la pila. Se lo conoce como ***POP***. Mirando las operaciones de Lista seria $SUPRIME(PRIMERO(P), P)$.
- $\boldsymbol{METE(x, P)}$: Inserta el elemento $x$en la parte superior de la pila. Se lo conoce como ***PUSH***. Mirando las operaciones de Lista seria $INSERTAR(X, PRIMERO(P), P)$.
- $\boldsymbol{VACIA(P)}$: Devuelve verdadero si la pila P esta vacía. Si no, falso.

Se pueden utilizar punteros o cursores.

- **Cola:** Lista donde se insertan elementos en un extremo (el posterior) y se suprimen en el otro (frente). Son listas del tipo FIFO (First In, First Out).

Tenemos los siguientes operadores:

- $\boldsymbol{ANULA(C)}$: Convierte a C en una pila vacia
- $\boldsymbol{FRENTE(C)}$: Devuelve el valor del primer elemento de la cola C. Mirando las operaciones de Lista seria $RECUPERA(PRIMERO(C), C)$.
- $\boldsymbol{PONE\_EN\_COLA(x, C)}$: Inserta elemento x al final de la cola. Se lo conoce como ***QUEUE***. Mirando las operaciones de Lista seria $INSERTA(x, FIN(P), C)$.
- $\boldsymbol{QUITA\_DE\_COLA(C)}$: Suprime al primer elemento de C. Se lo conoce como ***DEQUEUE***. Mirando las operaciones de Lista seria $SUPRIME(PRIMERO(C), C)$.
- $\boldsymbol{VACIA(C)}$: Devuelve verdadero si la pila C esta vacía. Si no, falso.

- **Cola de doble entrada:** La bicola o doble cola es un tipo de cola especial que permiten la inserción y eliminación de elementos de ambos extremos de la cola. Puede representarse a partir de un vector y dos índices, siendo su representación más frecuente una lista circular doblemente enlazada. Esta estructura es una cola bidimensional en que las inserciones y eliminaciones se pueden realizar en cualquiera de los dos extremos de la bicola. Gráficamente representamos una bicola de la siguiente manera:

Orden Lexicográfico: Relación de orden definida sobre el producto cartesiano de conjuntos ordenados.

Sea X una cadena de símbolos $X = (x_p, x_{p-1}, \dots, x_1)$. Dadas las cadenas X_i y X_j:

- $X_i = (x_{ip}, x_{ip-1}, \dots)$
- $X_j = (x_{jp}, x_{jp-1}, \dots)$

$X_i < X_j$ sí y solo si $\exists\ t \leq p$ tal que $X_{it} < X_{jt}$ y $\forall\ w > t$ se cumple que $x_{iw} = x_{jw}$.

Radix Sort: Algoritmo para ordenar cadenas de igual longitud sin comparación.

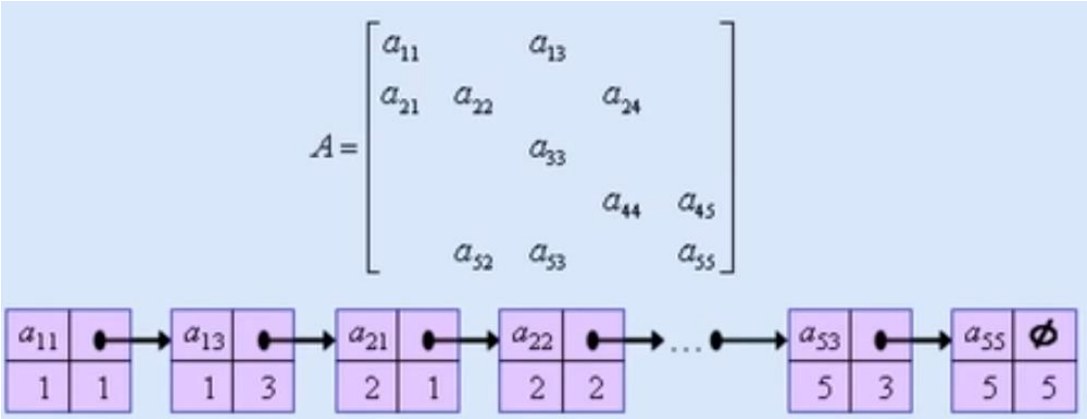
Tenemos los alfabetos $\Sigma_1, \Sigma_2, \Sigma_3, \dots, \Sigma_m$ tal que $\Sigma_i = < y_1^i\ y_2^i\ y_3^i\ \dots\ y_{r_i}^i >$. También tenemos la cola de palabras Q tal que cada palabra tiene la siguiente forma $X^j = (x_1^j, x_2^j, \dots, x_n^j)$ tal que todas las palabras tienen la misma longitud y x_i pertenece a algún alfabeto (todas las palabras siguen una estructura determinada).

Para $w = n$ hasta 1 inclusive:
Dado que x_w de todas las palabras es del alfabeto Σ_i :
Vaciamos las colas $Q_{y_1^i}, Q_{y_2^i}, Q_{y_3^i}, \dots, Q_{y_{r_i}^i}$
Mientras Q no quede vacía:
$X^j = DEQUEUE(Q)$.
Si x_w^j es el elemento y_k^i, entonces:
$QUEUE(X^j, Q_{y_k^i})$.

Concatenamos todas las colas $Q_{y_1^i}, Q_{y_2^i}, Q_{y_3^i}, \dots, Q_{y_{r_i}^i}$ en Q.

Matrices Rala: Son matrices de GRAN dimensión tal que tienen un alto porcentaje de ceros en sus celdas. Si hablamos de una matriz de adyacencia, esto ocurre cuando $|E| \ll |P|$ (por ejemplo, $|E| = 10^3$ y $|P| = 10^6$) El problema de estas matrices es que ocupan demasiado espacio y se emplean varias alternativas para lograr que no ocupe mucho espacio y que aritméticamente tenga bajo costo.

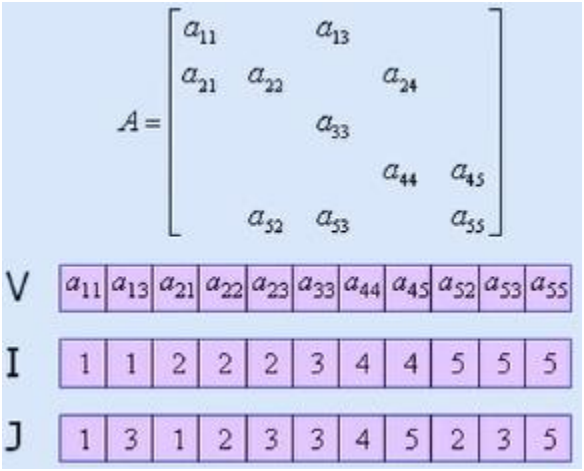
- **Enfoque de lista enlazada:** Cada elemento no nulo de la matriz en un nodo de la lista. Almacenamos el valor numérico, su posición en la matriz y un puntero al siguiente nodo.



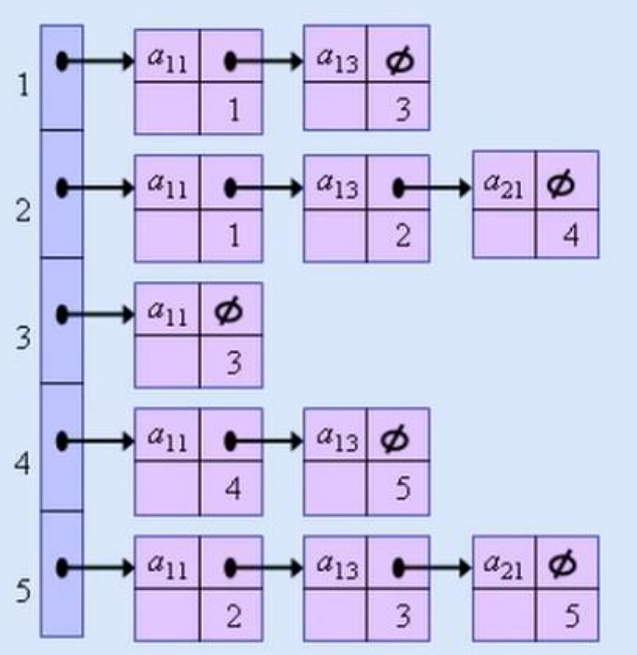
Es muy ineficiente salvo para insertar nuevos elementos. El tema es que científicamente no nos interesa inserción de datos, sino que el acceso a los mismos.

- **Enfoque Coordenado:** Se almacena la información en 3 array estáticos:
  - $V(1,2,\dots,n_z)$: $Valores\ Numéricos$
  - $I(1,2,\dots,n_z)$: $Indices\ de\ Fila$
  - $J(1,2,\dots,n_z)$: $Indices\ de\ Columna$

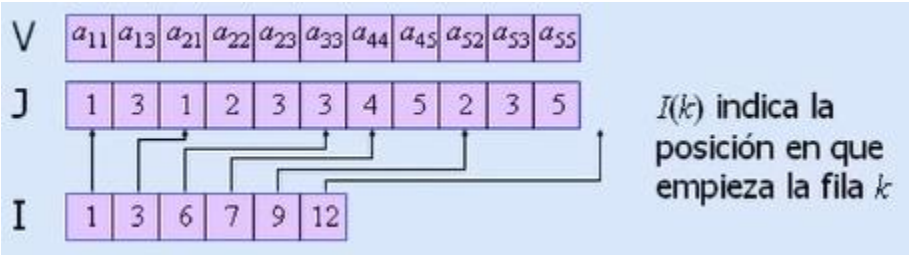
  n_z es la cantidad de elementos no nulos. Acceso secuencial muy ineficiente.



- **Enfoque Lista Enlaza Por Fila:** Hay una lista enlazada por cada fila de la matriz. La localización de un elemento es más eficiente. No es necesario almacenar explícitamente el **índice de fila**.



- **Enfoque Comprimido por Filas (CSR, Compressed Sparse Row):** Los índices de fila no se almacenan explícitamente:
  - $V(1,2,\dots,n_z)$: $Valores\ Numéricos$
  - $I(1,2,\dots,n+1)$: $Punteros\ a\ inicio\ de\ fila$
  - $J(1,2,\dots,n_z)$: $Indices\ de\ Columna$



También esta el enfoque Comprimido por Columnas (CSC, Compressed Sparse Column).

Sort Topológico (ST): Dado un grafo $G = (P, E)$ ***acíclico*** **(**si el grafo es acíclico, en un momento nos vamos a quedar sin minimales**)**, ST devuelve una estructura con los nodos del grafo G tal que cada arco dirigido (u, v), u aparece antes de v en la estructura.

Tenemos dos tipos de algoritmos ST**: el destructivo** (rompe las relaciones del grafo) **y el no destructivo** (no rompe las relaciones del grafo).

SortTopologicoDestructivo(G):

Mientras P no se encuentre vacío:

$ST \leftarrow MIN(G)$

$P = P - MIN(G)$

$E = E_P$ (E_P hace referencia a los arcos que quedan con los nodos que se quedaron)

SortTopologicoNoDestructivo(G):

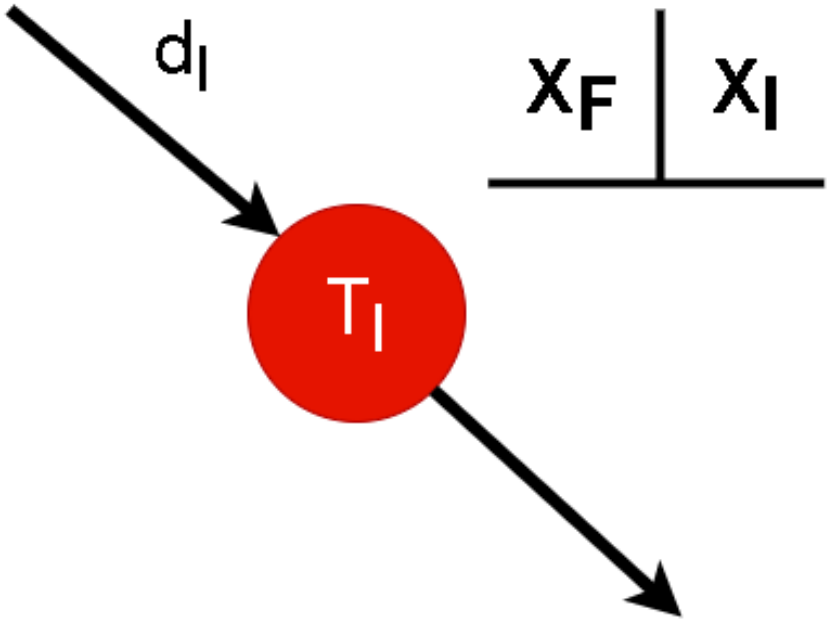
Mientras $P - P_{VISITADAS} \neq \emptyset$ y $MIN(G(P_{NO\ VISITADAS}, E_{|\ P_{NO\ VISITADAS}})) \neq \emptyset$:

Para todo x en $MIN(G(P_{NO\ VISITADAS}, E_{|\ P_{NO\ VISITADAS}}))$:

$ST \leftarrow x$

$P_{VISITADAS} \leftarrow x$

Red CPM (Critical Path Method): Nos permite evaluar una dependencia de tareas. Podemos determinar la red critica: un subgrafo en el cual un retraso en cualquier tarea haría que el proyecto total se retrase. La idea es que cada nodo sea un estado del proyecto y los arcos son tareas que llevan al proyecto de un estado a otro. Cada arco tiene asignado unidades de tiempo. Se recorre el grafo de manera secuencial. Para eso, hacemos sort topológico.



Se va recorriendo el grafo desde el primer nodo de la estructura obtenida en el ST y vamos actualizando el X_I de manera que vamos acumulando el tiempo que nos lleva ir hasta la tarea T_I. Nos queda que $X_I = d_I + X_{I-1}$. Una vez que llegamos al último nodo, hacemos que $X_F = X_I$ vamos para atrás y vamos restando el valor de los $X_F = X_{F-1} - d_{I+1}$. Cuando digo -1 me refiero al anterior siempre. Aquellos nodos que tienen $X_F = X_I$ son parte de la red crítica.

SortTopologicoDeCaminoCrítico(G):

Tenemos a ST como una lista tal que en cada nodo se conserva (v, t_i, t_f)

$x \leftarrow Min(G)$

$ST \leftarrow (x, 0, 0)$

$P_{VISITADAS} \leftarrow x$

Tener en cuenta que $P_{NO\ VISITADAS} = P - P_{VISITADAS}$

Mientras $P_{NO\ VISITADAS} \neq \emptyset$:

Para todo x en $MIN(G(P_{NO\ VISITADAS}, E_{|\ P_{NO\ VISITADAS}}))$:

Para todo v en $L(x)$:

$t_i \leftarrow Maximo(t_i, inicio(v) + costo(v, x))$

$ST \leftarrow (x, t_i, -1)$

$P_{VISITADAS} \leftarrow x$

Mientras $P_{VISITADAS} \neq \emptyset$:

Para todo x de $MAX(G(P_{VISITADAS}, E_{|\ P_{VISITADAS}}))$:

Para todo v en $R(x)$:

$t_f \leftarrow Maximo(t_i, fin(v) - costo(v, x))$

Cambio ST en donde esta x y reemplazo el -1 por t_f

Sacamos de $P_{VISITADAS}$ a x

Arreglo K-Dimensionales

Tenemos un arreglo con k dimensiones con los tamaños $n_1, n_2, \dots, n_k$. Para acceder a un elemento específico del arreglo:

$$Indice = (i_1, i_2, i_3, \dots, i_k) = \begin{cases} 0 \leq i_1 \leq n_1 - 1 \\ 0 \leq i_2 \leq n_2 - 1 \\ \dots \\ 0 \leq i_k \leq n_k - 1 \end{cases}$$

Por lo general, estos arreglos se implementan sobre un arreglo unidimensional. Para acceder a un elemento de este arreglo según el índice del arreglo k-dimensional:

$$h(i_1, i_2, i_3, \dots, i_k) = i_1 * n_2 * n_3 * \dots * n_k + i_2 * n_3 * \dots * n_k + \dots + i_{k-1} * n_k + i_k$$

Si queremos la inversa:

h^{-1}*(indiceUnidimensional):*

Tenemos un arreglo $coordenadas$ *de k posiciones*

Desde $i = k - 1$ *hacia:*

$$coordenadas_i = indiceUnidimensional \,\%\, n_i$$

$$indiceUnidimensional = indiceUnidimensional // n_i$$

$$coordenadas_k = indiceUnidimensional \,\%\, n_k$$

Árbol: Un árbol es una estructura no lineal. Definimos a un árbol como un conjunto finito T de uno o más nodos tal que:

- Hay un nodo que llamamos $raiz$ (también lo podemos ver cómo r o $root(T)$)
- Los demás nodos están particionados en conjuntos $T_1, T_2, \dots, T_m$. Estos conjuntos se los llama subárboles de la raíz.

Es una definición recursiva. Definimos al árbol en términos de árboles. Cada nodo de un árbol es raíz de un subárbol contenido en el árbol entero. La cantidad de subárboles de un nodo se le llama el **grado** del nodo. Un nodo con grado cero se lo llama nodo terminal (*leaf, son los terminales*). Un nodo con grado distinto de cero (*branch*).

Para dar una definición más formal: $G = (P, E)$ es árbol si es conectado y $|P| = |E| + 1$.

Tenemos las siguientes propiedades:

- Cualquier camino en G es simple no hay ciclos
- Todo árbol es acíclico
- Todo árbol tiene $|MIN(G)| \geq 1$ y $|MAX(G)| \geq 1$

G es árbol si cumple algunas de las siguientes propiedades:

- G es conectado y sin circuitos
- G no tiene circuitos y cualquier arco adicional lo crearía
- G no tiene circuitos y $|P| = |E| + 1$

Árbol Principal Derecho: Dado $G = (P, E)$, un árbol principal derecho (APD) es $\overline{R(x)}$. Hay un minimal y es al que llamamos **Raíz**.

Árbol Principal Izquierdo: Dado $G = (P, E)$, un árbol principal derecho (APD) es $\overline{L(x)}$.

Grado de un árbol: Es el máximo grado de salida de sus nodos.

Barrido: Es una de las operaciones más importante sobre árboles. Es moverse a través de todos los nodos del árbol visitando a cada uno una única vez y en algún orden determinado. Dado $G = (P, E)$ un APD, un barrido sobre G es cualquier relación lineal E' en P.

Barrido Pre-Orden: Visitamos al nodo corriente antes que sus subárboles, el contenido de la raíz aparece antes que el contenido de los subárboles de los nodos hijos.

r raíz del APD

$S \leftarrow r$ (cuando dice S hacemos referencia a una pila: "STACK")

Mientras $S \neq \emptyset$:

$x \leftarrow S$

$Visitar(x)$

Para $i = GradoDeSalida(X)$ hasta 1:

$S \leftarrow R^i(x)$ *(R^i hace referencia al elemento i de la vecindad derecha de x)*

Barrido Post-Orden: Visita el nodo corriente después de visitar sus subárboles, el contenido de los subárboles de los nodos hijos aparece antes que el contenido de la raíz.

r raíz del APD

$S \leftarrow (r, 0)$ (cuando dice S hacemos referencia a una pila: "STACK"; el 0 hace referencia al "estado"; si es 0, no se recorrió, en caso contrario, es 1)

Mientras $S \neq \emptyset$:

$(x, e) \leftarrow S$

Si $e = 0$:

$S \leftarrow (x, 1)$

Para $i = GradoDeSalida(X)$ hasta 1:

$S \leftarrow R^i(x)$

En Caso Contrario:

$Visitar(x)$

Barrido por niveles: Se usa una cola. Es muy sencillo.

r raíz del APD

$Q \leftarrow r$

Mientras $Q \neq \emptyset$:

$x \leftarrow S$

$Visitar(x)$

$Q \leftarrow R(x)$

Barrido Simétrico: Es solamente para arboles binarios.

r raíz del APD

$S \leftarrow (r, 0)$

Mientras $S \neq \emptyset$*:*

$(x, e) \leftarrow S$

Si $e = 0$*:*

$S \leftarrow (x, 1)$

$S \leftarrow (R^1(x), 0)$

En Caso Contrario:

$S \leftarrow (R^2(x), 0)$

$visitar(x)$

Árbol r-ario Completo: Si $|R(x)| = r$ para todo nodo no maximal.

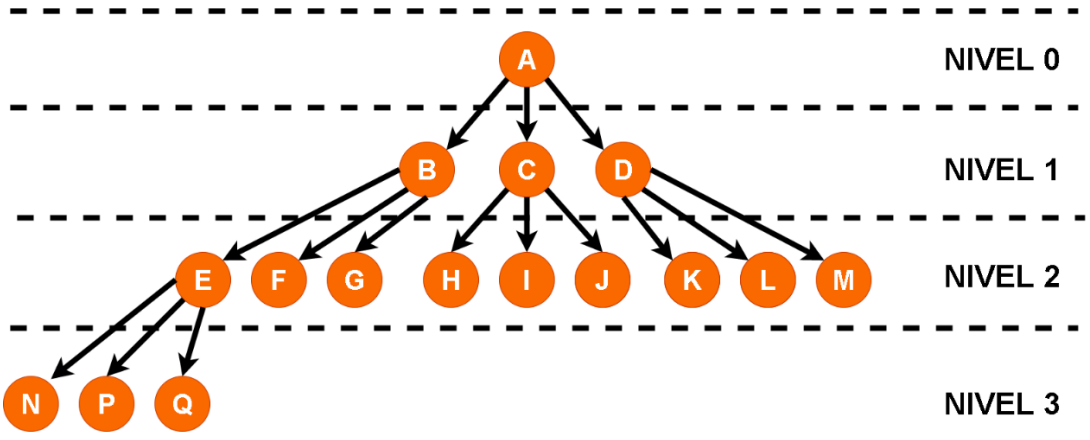
Árbol r-ario Lleno: Si es completo y todos los nodos maximales están en la misma altura.

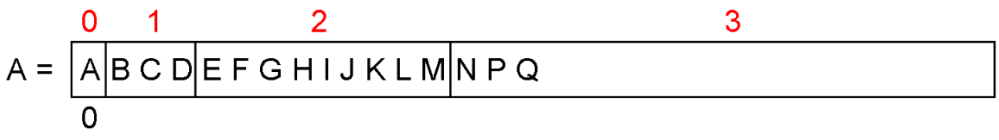
La cantidad de nodos va a estar dada por la suma de una serie geométrica. La altura de un árbol es logarítmica, es decir, que la cantidad de niveles de un árbol r-ario lleno es $\log_r(n)$, siendo $n = |P|$.

Representación de Árboles llenos sobre arreglos

Necesitamos la cantidad de nodos para poder definir el arreglo. Necesitamos un operador que me diga dado un nodo, donde empieza su vecindad derecha y cuál es su vecindad izquierda. Es como recorremos el árbol.

Por ejemplo:





Tener en cuenta que $y \in R(x)$ y $y = A_t$.

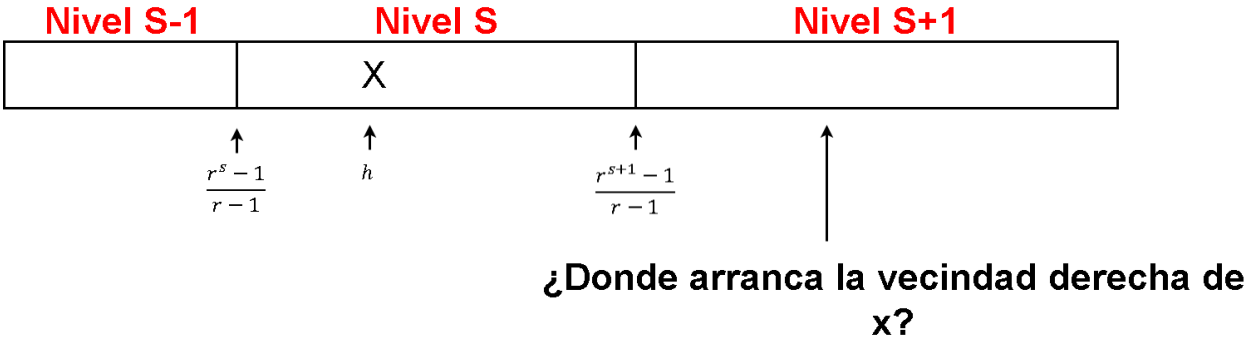
En el arreglo deberiamos saber cómo diferenciar los niveles. La vecindad izquierda queda determinada por (llamamos a A_i al elemento en la posición i del arreglo A y t a donde se ubica el nodo y dentro del arreglo):

$$L(y) = \begin{cases} A_{\lfloor t/r \rfloor} & si\ no\ se\ cumple\ que\ t \equiv 0\ mod\ r \\ A_{\lfloor t/r \rfloor - 1} & si\ se\ cumple\ que\ t \equiv 0\ mod\ r \end{cases}$$

¿Como sacamos la vecindad derecha? Si vemos a la vecindad derecha de un nodo determinado:

$$R(x) = \{A_{rh+1}, \dots, A_{rh+r}\}$$

Supongamos que queremos sacar la vecindad derecha del elemento x que se encuentra en la posición h. En el nivel i tenemos r^i nodos. La cantidad de nodos que tenemos del subárbol hasta el nivel i es $\frac{r^{i+1}-1}{r-1}$. La última división es parte de la serie geométrica.



Primero, lo que quiero saber es mi número de elemento dentro de mi nivel. Eso es igual a:

$$h - \frac{r^s - 1}{r - 1}$$

El último termino hace referencia a la posición donde arranca el nivel. El próximo nivel arranca en:

$$\frac{r^{s+1} - 1}{r - 1}$$

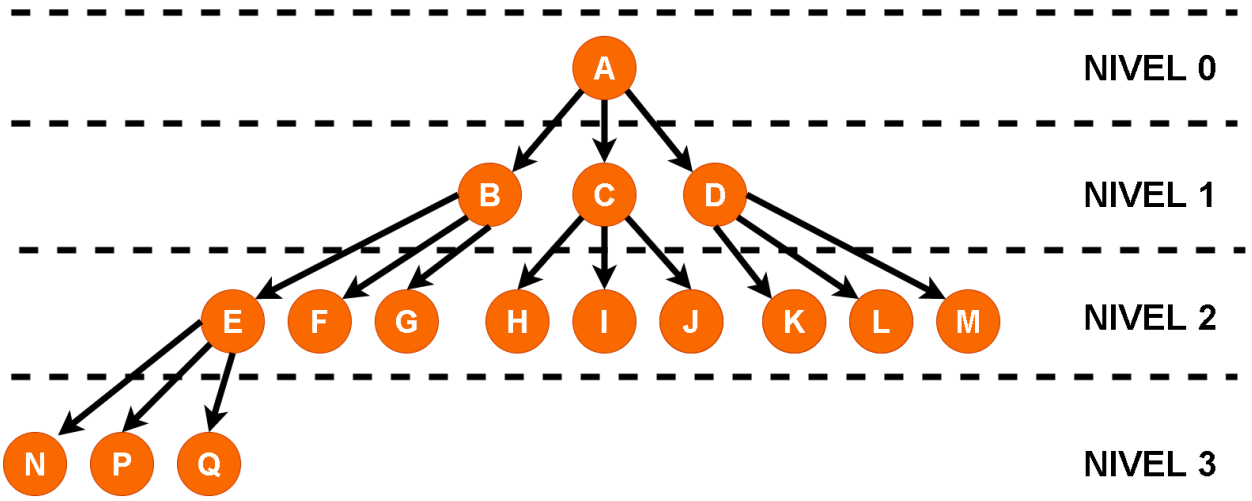
La posición donde arranca mi vecindad derecha es:

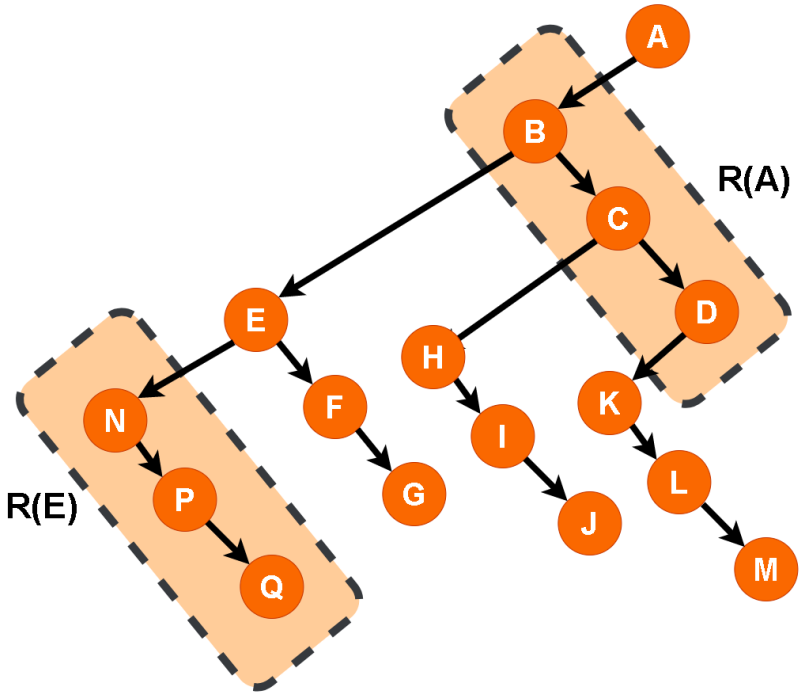
$$t = \frac{r^{s+1} - 1}{r - 1} + r * \left(h - \frac{r^s - 1}{r - 1} \right)$$

$$\boldsymbol{t = rh + 1}$$

Transformada de Knuth: Es otra forma de representación de un árbol. Knuth representa un árbol r-ario en un árbol binario, en el cual la vecindad derecha de un nodo x de un APD está representado como una lista que comienza en $R^1(x)$. Y esta lista está representada por $R^2(y)$, siendo y y los componentes de $R(x)$ en el modelo. Recordar que un bario Pre-Orden en el modelo es Pre-Orden en Transformada de Knuth y Post-Orden en el modelo es un barrido simétrico en Transformada de Knuth.

Tomando el ejemplo anterior:





Árbol binario de búsqueda: Es un árbol tal que se respeta:

$$f\left(R^1(x)\right) < f(x) \leq f\left(R^2(x)\right)$$

f es una función de asignación a punto.

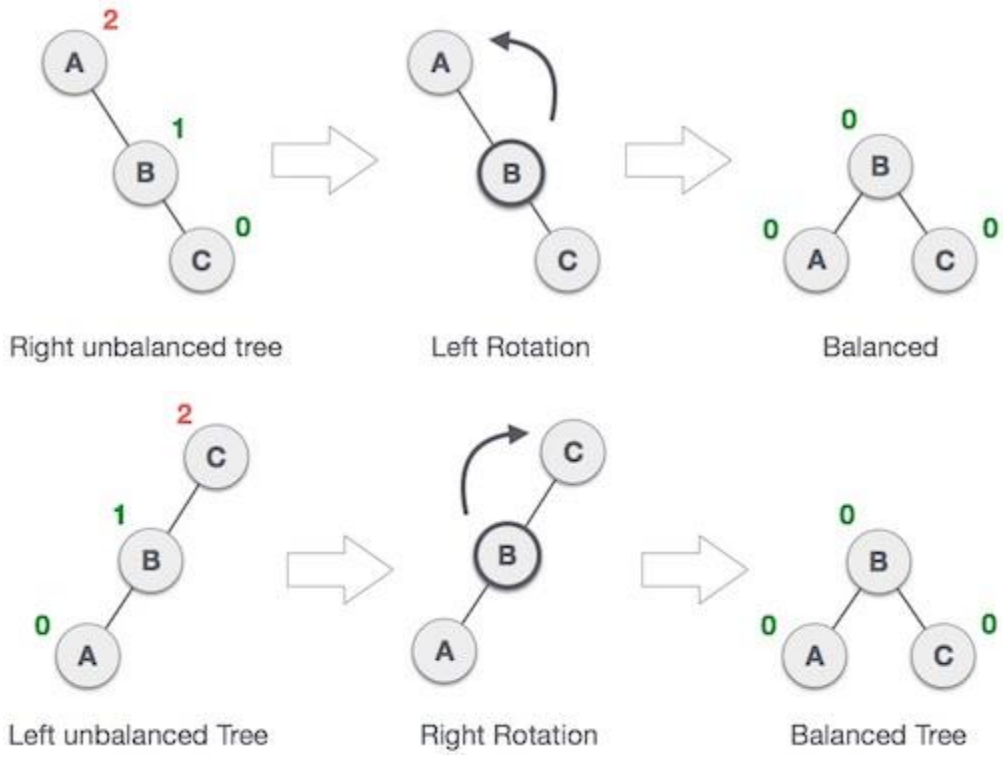
Un árbol desbalanceado hace que haya más niveles de las que podría haber y la idea es tener la menor cantidad de niveles posibles.

Árbol AVL: Son arboles binarios de búsqueda cuyos operadores de inserción aseguran que nunca haya un desequilibrio mayor a dos niveles entre SAI y SAD (Subarboles derecho e izquierdo). Deben analizarlo los casos especiales y las rotaciones necesarias del algoritmo.

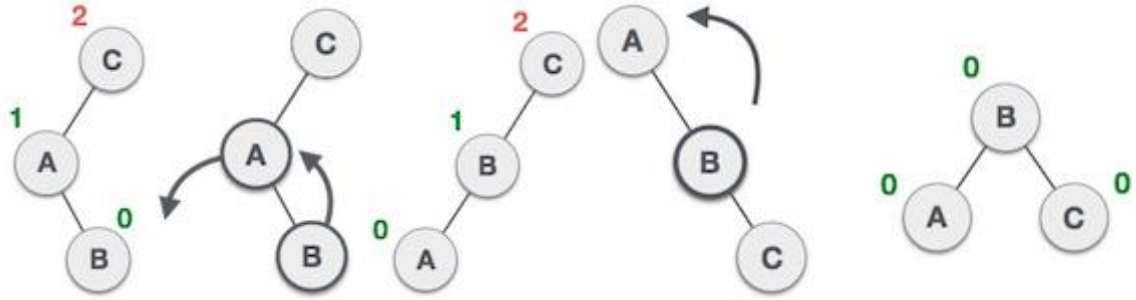
Un ABB es AVL si, dada r raíz del árbol:

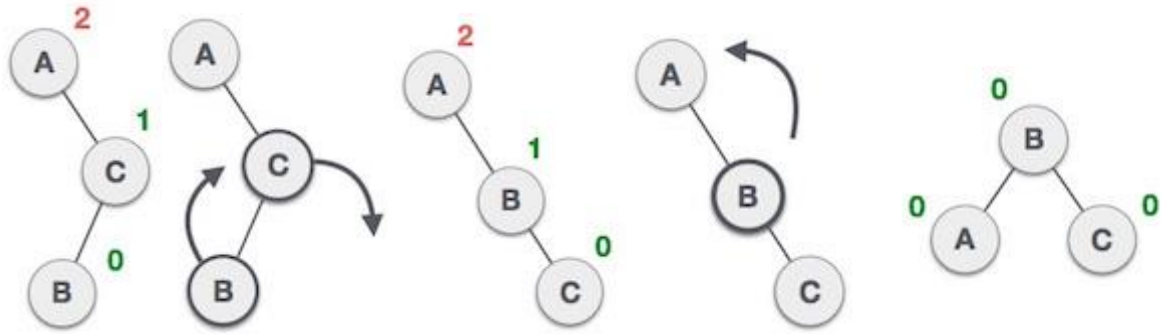
- SAI(r) es AVL
- SAD(r) es AVL
- $|h(SAI) - h(SAD)| = 0$ (h es igual a la altura)

Al hacer inserciones en un Árbol AVL, tenemos dos casos de **rotación: por izquierda y por derecha:**

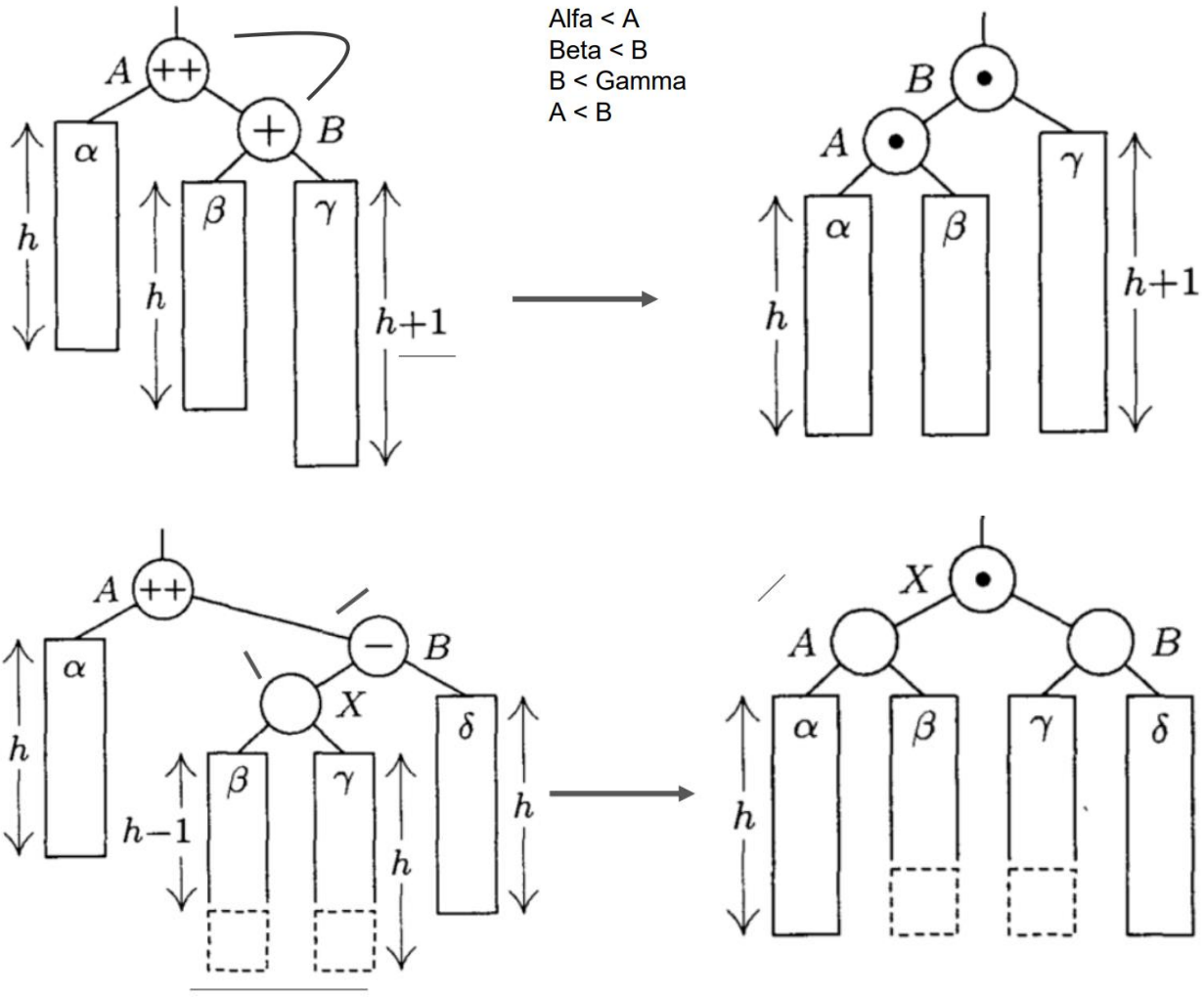


También tenemos los Left-right y Right-Left:

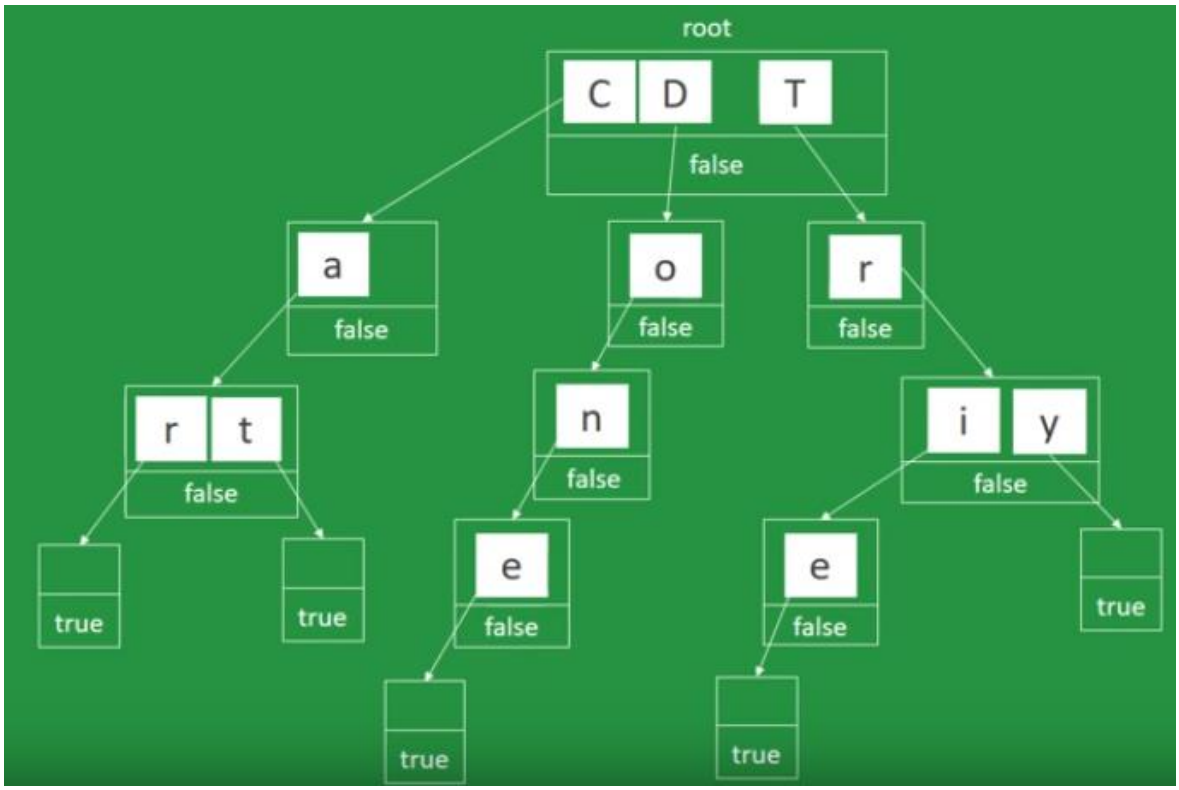




Todos estos casos se pueden resumir en los siguientes:



TRIE: Es un árbol para guardar cadenas de caracteres (Strings) existe un nodo por cada prefijo común. Un prefijo es el comienzo de los caracteres de un string. Por ejemplo:



Por ejemplo, tenemos la palabra Car y la palabra Cat. Pero la palabra Ca no. Ya que no hay un "true" en A en el segundo nivel. Esta la palabra Done, pero no Do, esta la palabra Trie, pero no la palabra Tri. También esta Try.

Algoritmo de Siklóssy: Basado en conceptos vertidos anteriormente sobre estructuras lineales utilizando doble linkeo, es válido pensar que el espacio utilizado para la representación de una cola con linkeo doble es mayor al utilizado por una cola de linkeo simple, aunque el costo que se asume es el de no poder recorrer la cola en ambos sentidos. El problema que se presenta nuevamente es asumir el costo que representa un mayor tiempo de acceso o un mayor espacio utilizado. El algoritmo de Siklóssy presenta la posibilidad de minimizar el espacio ocupado, utilizando un único campo link, sin aumentar el tiempo de acceso, mediante el concepto algebraico del OR exclusivo.

Si:

p	q	p+q
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

De donde:

$$(p + q) + p = q$$

$$(p + q) + q = p$$

p	q	p+q	(p+q)+p	(p+q)+q
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	1	1

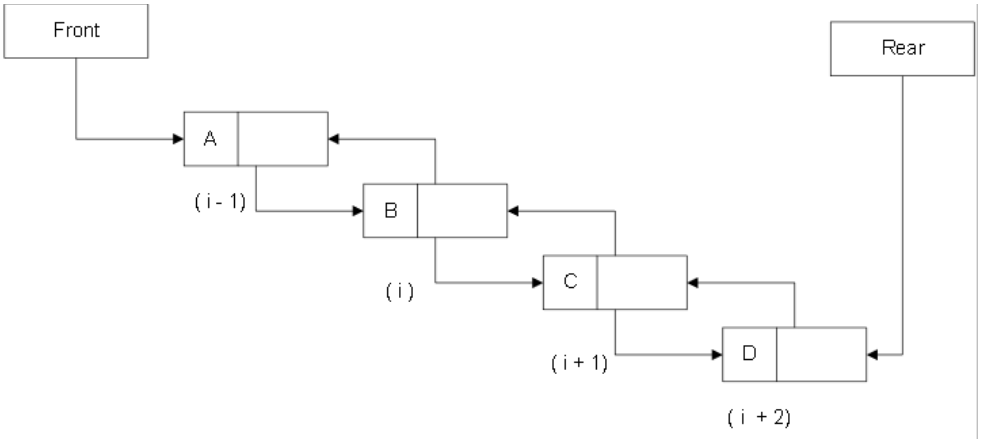
Si $p = 001101$ y $q = 100111$, entonces:

$$(p + q) + p = 100111$$

$$(p + q) + q = 001101$$

La idea de Siklóssy es evitar el doble linkeo de la siguiente manera. Propone almacenar en un mismo campo de dirección un valor que permita recorrer la estructura en ambos sentidos basándose en la suma del or exclusivo.

Supongamos que tenemos:



Si un nodo de la estructura lineal ocupa la posición i, por ejemplo, deber tener un campo tipo pointer con un valor que permita conocer la posición del nodo anterior de la estructura (posición $i - 1$) y la dirección del nodo posterior (posición $i + 1$).

Si el nodo B ocupa la posición i, es posible identificar a p como la dirección de memoria que contiene el nodo de posición $i - 1$, es decir $pX = dir(i - 1)$, donde X es el nodo en consideración, en este caso $X = B$. De la misma manera se identifica a q como la dirección de memoria que contiene el nodo de posición $i + 1$, es decir $qX = dir(i + 1)$. Por lo tanto, p se identifica como la dirección del nodo anterior y q con la dirección del nodo posterior. En el ejemplo:

$$pB = dir(A)\ y\ qB = dir(C)$$

Si se analiza la posibilidad de ir desde el nodo B al nodo anterior o posterior, sabiendo que $(p+q)+q=p$ y que $(p+q)+p=q$, se deduce que, conociendo la dirección del nodo anterior al que se este analizando, y realizando la suma con el valor $(p+q)$, es posible obtener la dirección del nodo posterior al que se esté analizando.

Al ir moviéndonos en nodo en nodo, se pasa el valor que se tiene al otro nodo y este lo utiliza para ir al próximo.

Algoritmo de Warshall: El algoritmo de Warshall partiendo de la matriz de adyacencia nos da como resultado la matriz correspondiente (A) en la clausura transitiva mediante el siguiente algoritmo. Computa la clausura transitiva de una relación en $O(n^3)$.

Warshall(A):

$Desde\ i = 0\ hasta\ n-1$:

$Desde\ j = 0\ hasta\ n-1$:

$Desde\ k = 0\ hasta\ n-1$:

$A_{ij} = A_{ij}\ |\ (A_{ik}\ \&\ A_{kj})$

$Retornar\ A$

Algoritmo de Floyd: El algoritmo de Floyd devuelve el menor costo para ir de un vértice a otro siempre que haya paso entre ellos. Devuelve una matriz del siguiente estilo:

$$\begin{pmatrix} 0 & 26 & 8 \\ 14 & 0 & 17 \\ 9 & 21 & 0 \end{pmatrix}$$

Y se obtiene:

Floyd(A):

$D\ es\ una\ matriz\ de\ mismas\ dimensiones\ que\ A\ con\ todas\ sus\ celdas\ en\ infinito$:

$Desde\ i = 0\ hasta\ n-1$:

$Desde\ j = 0\ hasta\ n-1$:

$Desde\ k = 0\ hasta\ n-1$:

$D_{ij} = Min\{D_{ik} + D_{kj}, D_{iJ}\}$

$Retornar\ D$

Algoritmo de Dijkstra: Busca paso más corto desde un nodo origen a todos los demás, contando con información de costo.

Y se obtiene:

Dijkstra(G):

$G = (P, E)$

$longitud: E \rightarrow \mathbb{R}^+$

$longitud(x, x) = 0 \; \forall \; x \in P$

$\forall \; x, \forall \; y, x \neq y, longitud(x, y) = \begin{cases} v \; en \; \mathbb{R}^+ si \; (x, y) \in P \\ +\infty \; si \; (x, y) \notin P \end{cases}$

$P = \{P_0, \dots, P_{n-1}\}$

$S \leftarrow \{P_0\}$

$D(P_0) \leftarrow 0$

$\forall \; x \in P - S: D(x) \leftarrow \rho(P_0, x)$

$Mientras \; S \neq P$:

$Elegir \; un \; w \; en \; P - S \; tal \; que \; D(w) sea \; mínimo$

$S \leftarrow S \cup \{w\}$

$\forall \; x \in P - S: D(x) \leftarrow Min\{D(x), D(w) + longitud(w, x)\}$

Excentricidad: Definimos excentricidad de un vértice como el costo máximo para llegar a ese vértice.

Centro de un grafo: Definimos centro de un grafo al nodo de menor excentricidad.

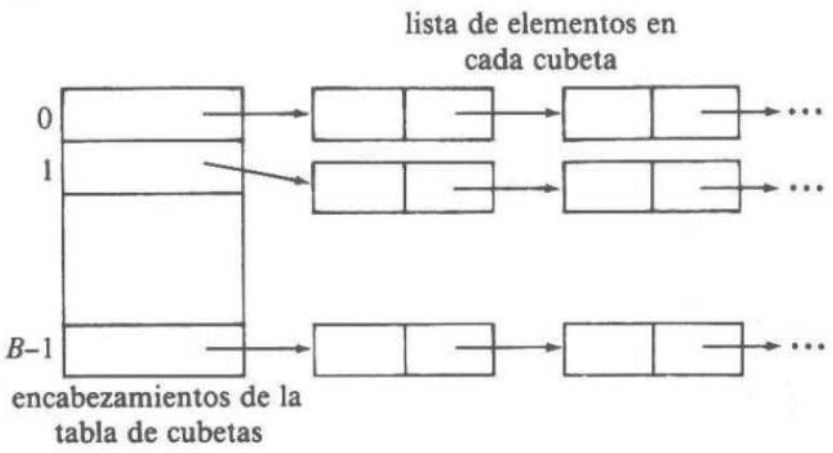
Hashing o Dispersión: Utiliza tiempo constante por operación, en promedio, y no existe la exigencia de que los conjuntos sean subconjuntos de algún conjunto universal finito. En el peor caso, este método requiere, para cada operación, un tiempo proporcional al tamaño del conjunto, como sucede con las realizaciones mediante arreglos y listas. Sin embargo, con un diseño cuidadoso es posible hacer que sea arbitrariamente pequeña la probabilidad de que la dispersión demande más de un tiempo constante para cada operación. Tenemos:

- **Dispersión Abierta o Externa:** Permite que el conjunto se almacene en un espacio potencialmente ilimitado, por lo que no impone un límite al tamaño del conjunto.
- **Dispersión Cerrada o Interna:** Usa un espacio fijo para el almacenamiento, por lo que limita el tamaño de los conjuntos.

Colisión: Llamamos colisión cuando dos datos tienen como resultado la misma cubeta. Dependiendo del Hashing, se las trata de manera distinta.

Dispersión abierta: La idea fundamental es que el conjunto (posiblemente infinito) de miembros potenciales se divide en número finito de clases. Si se desea tener B clases, numeradas de 0 a $B - 1$, se

usa una función de dispersión h tal que para cada objeto x del tipo de datos de los miembros del conjunto que se va a representar, $h(x)$ sea uno de los enteros de 0 a $B-1$. Lógicamente, el valor de $h(x)$ es la clase a la cual x pertenece. A menudo se da a x el nombre de **clave** y a $h(x)$ el de **valor de dispersión** de x. A las ***clases*** se les da el nombre de ***cubetas*** y se dice que x pertenece a la cubeta $h(x)$.



Organización de datos en la dispersión abierta.

La **tabla de cubetas** es un arreglo. En cada posición de este se encuentran los encabezamientos de las listas donde la lista i se encuentran aquellos elementos x tal que $h(x) = i$. La idea está en que todas las cubetas tengan el mismo tamaño y que sea corto.

Si hay N elementos en el conjunto, en promedio, cada cubeta tendrá N/B miembros. Si se puede estimar N y elegir B aproximadamente igual de grande, entonces una cubeta tendrá en promedio solo uno o dos miembros, y las operaciones con el diccionario tendrán, en promedio, un número pequeño y contante de pasos, independiente del valor que tenga N. La idea es que $h(x)$ sea un valor "aleatorio" que no dependa de x de ninguna forma trivial.

En el Hashing abierto cada cubeta es una lista simplemente enlazada, entonces cada colisión es añadir un elemento en la lista.

Dispersión cerrada: La tabla de dispersión cerrada guarda los miembros del diccionario en la tabla de cubetas, en vez de usar esa tabla para almacena encabezamientos de listas. En consecuencia, sólo es posible colocar un elemento en una cubeta, sin embargo, la dispersión cerrada tiene asociada una **estrategia de redispersión**. Al haber colisión, se "re-hashea" el dato sumándole 1 o random hasta encontrar un bucket vacío.

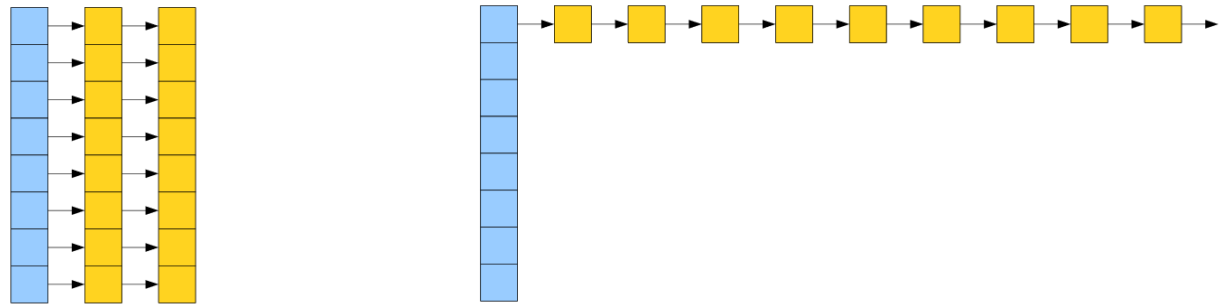
Una estrategia de redispersión es la dispersión lineal: $h_i(x) = (h(x) + i)\ mod\ B$.

¿Qué es Hashing (dispersión)?

Función que se define como un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres de longitud fija. Es una función de un solo sentido, es decir, una vez hasheada es casi imposible recuperar la original.

Operations	Sorted Array	Balanced BST	Hashing
Insertion	O(*n*)	O(log *n*)	O(1) avg
Deletion	O(*n*)	O(log *n*)	O(1) avg
Retrieval	O(log *n*)	O(log *n*)	O(1) avg

Al distribuir las claves en las cubetas queremos que la distribución sea lo más random posible.



Buena distribución

Mala Distribución

Hash Table (Tabla de Hashing): Es una estructura de datos que guarda datos de una manera asociativa. En una tabla de hash, los datos se guardan en un formato de array donde cada dato tiene su propio valor de índice. El acceso de los datos se vuelve muy rápido si sabemos su índice.

Búsqueda sobre arreglos ordenados

- **Búsqueda Binaria:** esta búsqueda se basa en mirar el valor ubicado en la mitad del arreglo en caso de que este sea menor al valor buscado repetimos esto en la parte siguiente del arreglo si es menor será en la parte izquierda. Esta búsqueda es del orden de $\log_2(n)$ siendo n la cantidad de elementos.
- **Búsqueda por interpolación:** Para usar este método debo conocer sobre el arreglo el máximo valor y el mínimo, además que este este debería ser más o menos uniforme. Esta búsqueda se basa en "tirar" índices y cortar el arreglo. Los índices están dados por

$$Factor\ de\ Interpolación = \frac{k - k_l}{k_u - k_l} * n$$

Donde k es el valor buscado, k_l es el menor valor, k_u es el mayor valor y n la cantidad de elementos.

Teniendo el arreglo $k_l, \dots, k_{i-1}, k_i, k_{i+1}, \dots, k_u$. k_i es el valor que esta en la posición dada por el factor de interpolación y hay 3 casos posibles.

1. $k_i = k \rightarrow Terminamos\ la\ busqueda$
2. $k_i > k \rightarrow Se\ repetira\ el\ procedimiento\ con\ la\ siguiente\ parte\ del\ arreglo$ $k_l, \dots, k_{i-1}$ siendo ahora k_{i-1} el nuevo k_u y n el largo del nuevo arreglo
3. $k_i < k \rightarrow Se\ repetira\ el\ procedimiento\ con\ la\ siguiente\ parte\ del\ arreglo$ $k_{i+1}, \dots, k_u$ siendo ahora k_{i+1} el nuevo k_l y n el largo del nuevo arreglo

- **Búsqueda con heurísticas:** Consiste en generar reglas para un determinado arreglo, una especie de función partida. Por ejemplo:

$$100,200,300,10300,10400,10500,98000,99000$$

$$h(x) = \begin{cases} Si\ x < 10000 \rightarrow [0,2] \\ Si\ x < 10000 \rightarrow [3,5] \\ Si\ x > 90000 \rightarrow [5,-] \end{cases}$$

Programación Dinámica: Brinda un procedimiento sistemático de encontrar una solución conociendo un punto de partida y a donde se quiere llegar.

***k*-vecindad izquierda de *y*:** Los elementos del ideal izquierdo hasta cierta longitud de paso.

$$L^k(y) = \{x : |\rho(x,y)| \leq k\}$$

***k*-frontera izquierda de *y*:** Los elementos que están exactamente a una distancia k.

$$\boldsymbol{L}^k(y) = \{x : |\rho(x,y)| = k\}$$

***k*-vecindad derecha de *y*:** Los elementos del ideal derecho hasta cierta longitud de paso.

$$R^k(y) = \{x : |\rho(y,x)| \leq k\}$$

***k*-frontera derecha de *y*:** Los elementos que están exactamente a una distancia k.

$$\boldsymbol{R}^k(y) = \{x : |\rho(y,x)| = k\}$$

Teorema: Si hay paso entre x (estado inicial) y z (estado final), existe un par de valores k_1 y k_2 para los cuales la frontera derecha del nodo origen y la frontera izquierda del destino van a tener una intersección no nula.

Puede haber más de una solución. Debo afirmar que hay un conjunto de soluciones óptimas y que la que encontré (más rápido) es una de ella. Para evitar generar estados (función exponencial) puedo utilizar un indicador de si ya pasé por ese estado o en un estado guardar información de los anteriores.

Se hace el siguiente algoritmo para ver si existe solución:

x es donde partimos e y es a donde queremos llegar.

Repetimos constantemente:

$Si\ \boldsymbol{R}^{k_1}(x) \cap \boldsymbol{L}^{k_2}(y) \neq \emptyset$:

    $Break\ por\ que\ existe\ solución$

$K_1 += 1\ (Se\ extiende\ la\ frontera\ derecha)$

$Si\ \boldsymbol{R}^{k_1}(x) \cap \boldsymbol{L}^{k_2}(y) \neq \emptyset$:

    $Break\ por\ que\ existe\ solución$

$k += 1\ (Se\ extiende\ la\ frontera\ izquierda)$

Para encontrar una:

$resolver(estadoInicial, estadoFinal)$:

    $listaDeEstadosExplorados$

    $colaDeNodos$

    $solucion\ es\ una\ lista$

    $nodoActual = (estadoInicial, -)$

    $estado = (estadoInicial)$

    $Mientras\ estado\ no\ sea\ el\ estado\ final$:

        $Si\ estado\ no\ se\ encuentra\ en\ listaDeEstadosExplorados$:

            $Para\ cada\ estado\ j\ en\ estadosSiguientes(estado)$:

                $colaDeNodos \leftarrow (j, nodoActual)$

        $nodoActual \leftarrow colaDeNodos$

        $estado = nodoActual[0]$

    $Mientras\ nodoActual[1]\ no\ sea\ -$:

        $solucion \leftarrow nodoActual[0]$

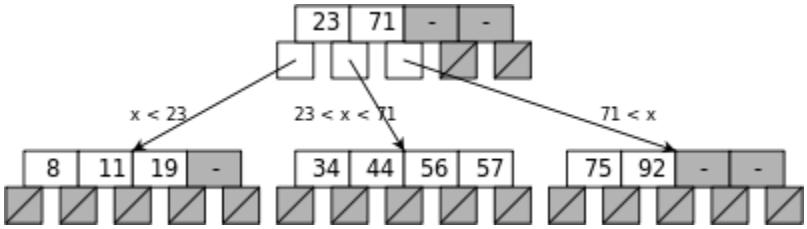
        $nodoActual = nodoActual[1]$

    $Retornar\ solucion$

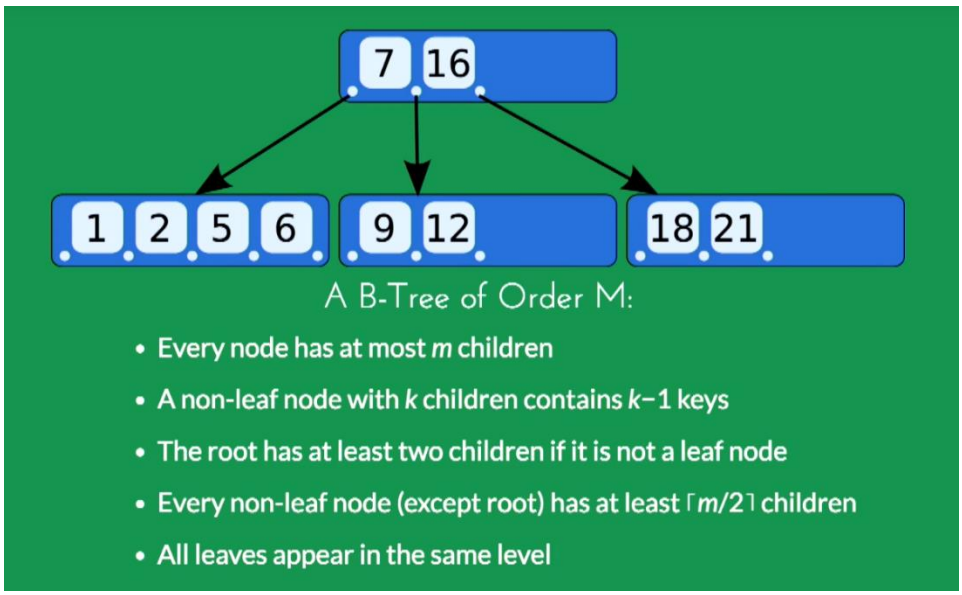
Buscar en un árbol de poca altura (Árbol B): Se siguen ciertas reglas:

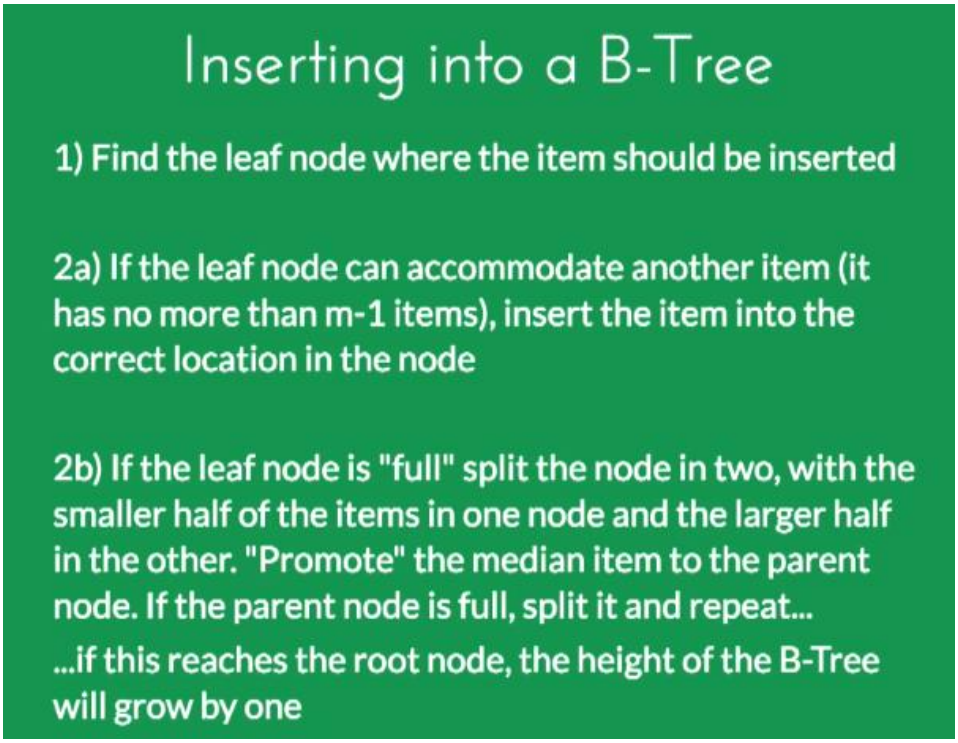
- **Los nodos de un árbol tienen entre $m/2$ y m hijos:** Esta regla hace que yo represente nodo y valores dentro de los nodos que no son vacíos, al menos 5 valores por nodo. Se llena la estructura.
- **Se busca que las hojas estén en el mismo nivel:** Los nodos hojas tienen valores sin punteros a los hijos y los nodos intermedios y raíz tiene punteros a sus hijos.
- **Crece horizontalmente:** Al insertar valores (no nodos) cuando se me llena un nodo, lo divido en 2. Si se llena una hoja, se realiza un **Split:** División de un nodo en dos partes.

La definición de un Árbol B es que arboles balanceados de búsqueda, pero cada nodo puede poseer más de dos hijos.



Los nodos internos deben tener un numero variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se juntan o se parten. Un árbol-B se mantiene balanceado porque requiere que todos los nodos hoja se encuentren a la misma altura.





Modelo: representación simplificada de un sistema real. Si la representación es adecuada, podremos interrogar al modelo si deseamos conocer alguna propiedad del sistema real.

Base de datos: Modelo del conjunto de los datos relativos a una organización o al menos a una aplicación en esa organización.

Modelo relacional de datos: La construcción principal del modelo relacional de datos es la relación.

Relación: Consiste en un esquema y una instancia.

- **Esquema:** Define el nombre, atributos, y dominio de la relación. Llamamos a la cantidad de atributos en el esquema como el **grado de la relación**.

$$Hotel(registro: String, dirección: String, id_{cadena}: int)$$

- **Instancia:** Conjunto de tuplas, cada una de las cuales respeta el esquema.

$$\{(\text{Don jose}, Rivadavia\ 522, 12), (\text{Caesar Palace}, Avellaneda\ 366, 2)\}$$

Restricciones de integridad: Son restricciones sobre el esquema, que impacta sobre las instancias. Una instancia es “legal” o “válida” si cumple las restricciones de su esquema. Un DBMS implementa las restricciones de integridad en momento de creación de las tablas, o en la inserción/actualización de datos.

Restricción de clave primaria (Primary Key): Es la elección de un mínimo subconjunto de atributos que identifican a las tuplas de una relación. El conjunto de atributos que identifica a las tuplas es una clave candidata.

- Dos tuplas distintas que no pueden tener valores iguales para todos los atributos de una clave.
- Ningún subconjunto de la clave identifica a las tuplas. Sólo la clave completa.

Una relación puede tener varias claves candidatas, pero la primaria debe servir para cualquier tupla que pueda existir.

Restricción de clave foránea (Foreign Key): Es común que haya relaciones conectadas entre sí de modo que, si hay un cambio en los datos de la primera, se necesita replicar en la segunda.

$$Reserva(registro: String, fecha: Date, cant_{mayores}: Int, cant_{menores}: int)$$

Registro se refiere a la Primary Key del hotel, luego es una Foreign Key que se refiere a la relación hotel. Esta restricción hace que los registros deban existir en Hotel una Foreign Key se puede referir a la misma relación.

Algebra Relacional: Consiste en un conjunto de operadores que se aplican a una o dos relaciones y producen como resultado otra relación. Una consulta se formaliza en este lenguaje como una expresión compuesta de operadores, variables relaciónales que representa los operandos, y constantes. Es de **carácter procedural y describe paso a paso cómo realizar una consulta.** Se encarga de componer operadores para obtener consultas complejas.

Cálculo Relacional: Brinda una notación declarativa para expresar lo mismo.

Algebra Relacional

De los operadores vamos a estudiar que devuelven.

Operadores Unarios:

- **Operador Proyección π:**

$$\boldsymbol{\pi[lista\ de\ atributos](Relacion) = Relación \mid esquema\ lista\ de\ atributos}$$

Devuelve una instancia cuyo esquema lo compone el subconjunto de atributos definidos como parámetro.

- **Operador Selección σ:**

$$\boldsymbol{\sigma[condición](Relacion) = Relación\ con\ tuplas\ de\ R\ que\ cumplen\ el\ predicado}$$

Devuelve una instancia con esquema igual al de la relación parámetro, conteniendo las tuplas que cumplan con la condición indicada.

Operadores Binarios Booleanos:

- **Operador Unión ($\boldsymbol{A} \cup \boldsymbol{B}$):** Necesita como condición que el esquema de A sea compatible con el esquema de B. Es decir, que atributos de A tengan igual dominio y obviamente tengan la misma cardinalidad que los de B. (Misma cantidad de atributos y mismo dominio de manera ordenada). Devuelve el esquema de A y la instancia $instancia(A) \cup instancia(B)$.

Del operador unión salen el operador intersección $\boldsymbol{A} \cap \boldsymbol{B}$ y diferencia $\boldsymbol{A} - \boldsymbol{B}$. Solo cambia el operador de cómo se obtienen las instancias.

Otros Operadores:

- **Operador Cartesiano ($A \times B$):** Se aplica a dos relaciones sin restricción de esquemas. El esquema resultante es la concatenación del esquema de A y la de B y como instancia queda todas las posibles combinaciones de tuplas de A con tuplas de B.
- **Operador de Junta ($A \bowtie B$):** No hay restricciones en los esquemas. Da como esquema la concatenación de ambos esquemas y como instancia todas las posibles combinaciones de tuplas de A con tuplas de B que cumplan con la condición de junta (condición que se pasa por parámetro). Representa un producto cartesiano con luego una selección.
- **Operador de División o Cociente (R/S):** En cierta forma, resulta la inversa del producto cartesiano. Tenemos a la relación R con el esquema $A_1, \dots, A_P, B_1, \dots, B_Q$ y S con el esquema $B_1, \dots, B_Q$. Se define R/S como la relación T que tiene atributos $A_1, \dots, A_P$ con el mayor conjunto posible de tuplas y se cumple que $T \times S \subseteq R$: cada tupla de T verifica que concatenada con toda tupla de$S$ da alguna tupla de R.
- **Renombre ('nombre_alumno'<- 'nombre'):** Nos sirve para renombrar un atributo determinado.

El cociente se lo puede expresar en función de otros operadores de la siguiente forma

$$T = R/S$$

$$T_1 = \pi_X(R)$$

$$T_2 = \pi_X((T_1 \times S) - R)$$

$$T = T_1 - T_2$$

Calculo Relacional

En el cálculo relacional, una consulta consiste en una especificación, formalizada en lógica de primer orden, de todas las tuplas que deseamos extraer de la base de datos. La forma general de una consulta será:

$$\{ x_1, \dots, x_n \mid \phi(x_1, \dots, x_n) \}$$

El valor de esta consulta es una relación: el conjunto de todas las tuplas $< x_1, \dots, x_n >$ de rango n que satisfacen la formula lógica ϕ. Como los x_i representan valores de componentes relacionales, esta versión del cálculo relacional se lama cálculo de dominios. Existe otra versión llamada cálculo de tuplas en las que las variables representan tuplas, no componentes de éstas. Es fácil demostrar que el cálculo de dominios puede expresar exactamente la misma clase de consultas; en este trabajo no limitaremos a tratar el cálculo de dominios.

Por ejemplo, si queremos los hospitales con más de 800 camas:

$$\{c\ n\ m \mid Hospital(c\ n\ m)\ y\ m > 800\}$$

Se interpreta como “el conjunto de las triplas <c,n,m> tales que <c,n,m> aparece en la relación Hospital y m es mayor que 800”.

Otro ejemplo: La consulta “personas que se aman a sí mismas”:

$$\{a \mid Ama(a, a)\}$$

Soundex: Es un algoritmo que aplica una serie de reglas a un string para generar un código de cuatro caracteres. Los pasos son los siguientes (haremos un ejemplo con MISSISSIPPI)

- Borrar todos los caracteres en el string excepto las letras de la A hasta la Z.

MISSISSIPPI

- Reemplazamos las consonantes con los siguientes dígitos (H,Y,W quedan fuera de la codificación):
  - 1: B, F, P, V
  - 2: C, G, J, K, Q, S, X, Z
  - 3: D, T
  - 4: L
  - 5: M, N
  - 6: R

MI22I22I11I

- Sacamos todas las vocales.

M222211

- Todos los números se unen en uno solo: Si tengo XX, queda solo X:

M21

- Como el código es de 4 caracteres, llenamos lo que nos queda de 0 hasta que nos quede de 4 caracteres:

M210

Estructuras de datos extra (Definiciones del NIST)

Árbol binario: Árbol donde cada nodo tiene como máximo dos nodos hijos. La definición formal es la siguiente:

- Árbol vacío (sin nodos)
- Tiene un nodo raíz, con un árbol binario a la izquierda y un árbol binario a la derecha.

Árbol binario de búsqueda: Es un árbol tal que se respeta:

$$f\left(R^1(x)\right) < f(x) \leq f\left(R^2(x)\right)$$

f es una función de asignación a punto.

Árbol binario balanceado: Árbol donde los nodos sin hijos (sin vecindad derecha, leaf nodes) no están más de una cierta cantidad más lejos de la raíz que cualquier otra.

Árbol Fibonacci: Es una variante del árbol binario donde el árbol es de orden n ($n > 1$) (orden hace referencia a la altura) tiene un subárbol a la izquierda de orden $n - 1$ y un subárbol a la derecha de orden $n - 2$.

Árbol Red Black: Un árbol Red Black es aquel que tiene n nodos y tiene altura máxima $2 * \log_2(n + 1)$. Son arboles binarios con un campo adicional en ellos nodos que es un color rojo o negro. Se tienen reglas que indican como el árbol debe ser coloreado. Las reglas de coloración aseguran que en el árbol no hay ningún camino que sea el doble de otro. El árbol es balanceado. Las reglas que se siguen son las siguientes:

- Cada nodo es coloreado de **rojo o negro**.
- La raíz de **color negro**.
- Cada hoja (NIL) es de **color negro**.
- Si un nodo es de color rojo, entonces sus hijos son de color negro.
- Para cada nodo, todos los caminos desde el nodo hasta sus hojas contienen el mismo número de nodos de color negro.

Árbol Splay: Son arboles binarios de búsqueda en donde las operaciones para acceder a los nodos restructuran el árbol.

Árbol Heap: Árbol completo (árbol donde en cada nivel, excepto del ultimo, esta “lleno”) donde cada nodo tiene una clave (seria el valor de $f(nodo)$) mayor, menor o igual a la clave de su padre. Es un caso muy especial de árbol balanceado. Se pueden cumplir cualquiera de las dos siguientes reglas:

MAX HEAP -> *Si α tiene como nodo hijo a β:*

$$f(\alpha) \geq f(\beta)$$

MIN HEAP -> *Si α tiene como nodo hijo a β:*

$$f(\alpha) \leq f(\beta)$$

Ejemplo de árbol Max Heap: