


Grafos

- **Nodos**
 - Grado: Cantidad de arcos salientes
 - Left(nodo X): Conjunto de nodos, que llegan a nodo X (con un solo paso)
 - Right(nodo X): Conjunto de nodos, que llegan desde nodo X (con un solo paso)
 - Indegree(nodo x): Cardinalidad left
 - Outdegree(nodo x): Cardinalidad right
- **Grafos**
 - Maximales: Conjuntos de nodos que no salen flechas
 - Míniales: Conjunto de nodos que no reciben flechas
 - Grado del grafo: El máximo grado de los nodos
 - **Caracterización de Grafos**
 - Simple: Un grafo es simple si a lo sumo un arco une dos nodos cualquiera.
 - Complejo: Grafo no simple
 - Conexos: un grafo es conexo si cada par de nodos está conectado por un camino.
 - Completo: un grafo simple es completo si existen arcos uniendo todos los pares posibles de nodos.
 - Bipartito: es un grafo en el cual el conjunto de puntos se divide en dos subconjuntos disjuntos, de tal manera que no exista un arco entre dos nodos que pertenezcan a un mismo conjunto.
 - **Clasificación de grafos**
 - En función de su dirección
 - Dirigidos / no dirigidos
 - En función de restricciones
 - Restringidos / no restringidos
- Camino: Existe un camino entre X e Y cuando de X puedo llegar a Y (es un bool, si o no)
- Paso: Ídem camino, pero respetando sentido flechas
- Paso simple: Si todos los nodos intervinientes en ese camino son distintos. (y no hay loop)
- Longitud de paso: Cantidad de arcos del paso
- Walk(x,y): Camino sin dirección
- **Representación computacional de los grafos**
 - Métodos estáticos (mayor performance, se utiliza más espacio)
 - Matriz adyacencia (Matriz de nodo x nodo, y se completa con la cantidad de aristas que unen nodoA con nodoB).
 - Matriz incidencia (Matriz (fila X columna) Arcos x Nodo, se pone 1 si el arco es incidente con Nodo))
 - Métodos Dinámicos
 - Punteros
 - Base de datos relacional
 - Manera mixta
- Complejidad espacial vs Complejidad
 - Son inversamente proporcional
 - Se prefiere disminuir la complejidad computacional




Arboles

- Un árbol es un grafo que cumple:
 - Aciclico
 - Sumatoria de nodos = Sumatoria de arcos + 1
 - Todo arco es desconectante
 - Existe y es único un walk para todo par de nodos
- Grado del árbol: Esta dado por el grado de salida del nodo con mayor grado
- Profundidad de un nodo: distancia entre un nodo y la raíz 
- Nivel: Clase de equivalencia de los nodos respecto a su profundidad. (La raíz es el cero)
- Nivel de un árbol: máximo nivel
- Altura de un nodo: Cantidad de aristas desde el nodo hasta la hoja mas profunda bajo el.
- Árbol completo: Para todo nodo que no sean maximales, el grado de salida coincide
- Árbol lleno: Árbol completo que tiene a todas sus hojas (maximales) en el mismo nivel
- Características arboles binarios
 - Árbol binario perfecto
 - Tiene exactamente $2^{(h+1)}-1$ nodos
 - En el peor de los casos complejidad de búsqueda: $O(\log n)$
 - Árbol binario completo
 - árbol binario en el que todos los niveles están llenos, excepto posiblemente el ultimo nivel, el cual es completado de izquierda a derecha
 - Tiene al menos $[2^h ; 2^{(h+1)}-1]$ nodos
 - Árbol binario balanceado
 - Un árbol binario está completamente balanceado si esta vacío, o ambos subárboles están completamente balanceados y tienen la misma altura
 - Si para todo nodo la cardinalidad izquierda = carnalidad derecho (o difiere en uno)
 - Árbol binario AVL
 - Para todo nodo, nivel subárbol izquierdo = nivel subárbol derecho (o difiere en uno)
- Tipos de barridos
 - Pre-orden: Raiz, Izq, Der [Voy bajando por la izquierda]
 - In-Orden / Simétrico: Izq Raiz Der [Se empieza desde la hoja mas bajo izquierda]
 - Post-orden: Izq, Der, Raiz [Se empieza desde la hoja mas abajo izquierda]
 - Primero en anchura / por niveles: [Se escriben los nodos de izquierda da derecha desde el nivel 0 (raíz)]
 - Se usa para representarlo en array
- Árbol de Expresión
 - Todas las hojas son operandos
 - Todos los nodos que no son hojas son operadores
 - Binario y Completo
- Árbol de búsqueda
 - Se llama árbol de búsqueda a un árbol que soporta las operaciones de búsqueda, inserción y eliminación de forma eficiente.

- hay un criterio de orden que determina donde va cada clave
- Árbol binario de búsqueda (ABB)
 - El tiempo de ejecución de muchas de sus operaciones es de $O(\log N)$ en el caso medio, pero en el peor de los casos puede llegar a $O(N)$
 - Se cumple que para cada nodo X del árbol, los valores de todas las claves de su subárbol izquierdo son menores que la clave de X y los valores de todas las claves de su subárbol derecho son mayores que la clave de X .
- Árboles M-arios
 - Un árbol M-ario es consiste de n subárboles $T_0 \dots T_n$ y $(n-1)$ claves $\{K_1 \dots K_{n-1}\}$ donde $2 \leq n \leq M$. $\{T_0, K_1, T_1, K_2 \dots \text{ect}\}$
 - Y se satisface lo siguiente
 - Las claves son distintas y están ordenadas de menor a mayor
 - Las claves contenidas en el subárbol izquierdo de la clave K_i son menores, y en el derecho mayores
- Árboles-B (La b es de balanceado, no de binario)
 - Son árboles M-arios diseñados para discos magnéticos u otros. El objetivo principal es minimizar las operaciones de entrada y salida hacia el disco.
 - Al imponer la condición de balance, el árbol es restringido de manera tal que se garantice que la búsqueda, la inserción y la eliminación de sean todos de tiempo $O(\log N)$.
 - Árbol B es más lento que Hashing para la creación de índices ya que para ello debe crear toda la estructura en memoria.
 - Por como es su estructura, siempre va quedando balanceado. Consta de $n-1$ claves y n arcos. En cada hoja, pongo varias claves(y varios punteros). Estas hojas apuntan a los datos.
 - Si el nodo hoja tiene $M-1$ elementos, y se agrega uno, se divide el nodo a la mitad
- Algoritmo para balancear árbol
 - Rotación derecha
 - Rotación izquierda
- Transforma de KNUTH
 - Knuth transforma un árbol r -ario en un árbol binario de forma tal que, dado un nodo, coloca como hijo izquierdo el primer elemento de su right y como hijo derecho el siguiente nodo del mismo nivel del mismo padre.

Métodos de Clasificación

- **Conceptos**
 - Estabilidad: Un ordenamiento se considera estable si mantiene el orden relativo que tenían originalmente los elementos con claves iguales.
 - In situ: Los métodos in situ son los que transforman una estructura de datos usando una cantidad extra de memoria, siendo esta pequeña y constante. Generalmente la entrada es sobrescrita por la salida a medida que se ejecuta el algoritmo.
 - Clasificación interna y externa
 - Interna: Si el archivo a ordenar cabe en memoria principal. Acceso fácil a los elementos y de forma aleatoria.

- Externa: si ordenamos archivos desde un disco u otro dispositivo. Se debe acceder a los elementos de forma secuencial o al menos en grandes bloques de datos.
- Teoría de la complejidad u orden de crecimiento de los algoritmos
 - Estudia los recursos requeridos durante el cálculo para resolver un problema. (Tiempo y Espacio)
 - Complejidad: se denota con $O(\text{función})$
 - La búsqueda binaria en una estructura ordenada tiene complejidad logarítmica $O(\ln n)$
 - Clases de complejidad
 - La clase P: contiene a aquellos problemas que son solubles en tiempo polinómico por una máquina de Turing determinista.
 - La clase NP: No determinística en tiempo polinómico. Se utilizan Máquinas de Turing no deterministas
- **Métodos de Ordenamiento**
 - **Burbuja / Bubble Sort**: Compara cada elemento con el siguiente, si el siguiente es menor lo intercambia, así barriendo todos. [Lo repite hasta que este ordenado, luego de K pasadas, los últimos K elementos ya nos barre ya que están ordenados]
 -  **Selection Sort**: Busca el elemento más pequeño y lo intercambia con el que esta en primera posición, luego se busca el segundo elemento más pequeño y se lo intercambia en la segunda posición. Se continua con este proceso hasta que todo el array este ordenado.
 -  **Insertion Sort**: Recorre cada elemento, lo quita y lo ubica de forma ordenada en el subconjunto ordenado de elementos izquierdo de donde se encuentra el puntero en el array.
 - **Shell Sort**: Mejora de insertion sort, intercambia elementos que están muy distantes. Primero compara los elementos que más lejanos y luego va comparando elementos más cercanos para finalmente realizar un Insertion Sort.
 - Se agrupan de a H elementos, los grupos se ponen uno debajo de otro y se ordenan de menor a mayor cada columna, luego se arma el nuevo array y se aplica Insertion Sort
 -  **Merge Sort**: Divide y vencerás. Divide las secuencia en dos; Divide recursivamente cada grupo, hasta que queden indivisible y los ordena por grupo ; luego junta todo.
 - **Heap Sort**: Es un árbol binario completo representado mediante un array. Utiliza como estructura auxiliar un árbol binario donde establece un orden parcial entre sus nodos, en donde el padre será mayor que sus hijos. El vector ordenado se construye seleccionando el elemento más grande, quitándolo del montículo y colocándolo en la secuencia ordenada.
 - 1- Se crea el árbol desde arriba de izquierda a derecha con los elementos del array
 - 2- Se hace cumplir que el nodo padre es mayor que los hijos, comparándose cada grupo de abajo hacia arriba
 - 3- Cuando cada grupo esta ordenado (padre mayor que los hijos), se saca el nodo raíz poniendo al final del array (elemento más grande) y

en su lugar se pone el elemento del árbol que más abajo y a la derecha esta

- **Quick Sort:** Ordenamiento con partición, divide y vencerás. El algoritmo original es recursivo, pero se utilizan versiones iterativas para mejorar su rendimiento (los algoritmos recursivos son en general más lentos que los iterativos, y consumen más recursos.)
 - 1- Se elige un pivote cualquiera y se ponen del lado izquierdo los elementos menores al pivote, y del derecho los mayores. (Se utiliza una Estrategia de partición)
 - 2- Se divide en dos grupos GRUPO1 pivote GRUPO2 y se vuelve a aplicar {1,2} recursivamente
 - Luego se une todo en GRUPO IZQ PIVOTE GRUPO DER

Nombre	Mejor caso	Caso promedio	Peor caso	Comentario
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	Estable. El mas lento. Método intercambio.
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Estable. Tiempo constante. Bueno para archivos grandes y claves pequeñas.
Insertion Sort	$O(n)$	$O(n)$	$O(n^2)$	Estable. Conviene cuando el array esta casi ordenado y tiene pocos elementos. Método Inserción
Shell Sort	$O(n^{5/4})$	$O(n^{3/2})$	$O(n^2)$	NO estable. Dependiente del orden y secuencia de incremento. Método inserción.
Merge Sort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	Estable. Adecuado para trabajos en paralelo. Requiere de memoria extra proporcional a los elementos del array. Método mezcla
Heap Sort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	NO estable. No depende de como vienen los datos. Acotado en el tiempo, muy utilizado en grandes volúmenes de datos. No requiere memoria adicional. Método selección.
Quick Sort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	NO estable. Depende de como vienen los datos. El mas rápido en la practica, es in situ ya que solo usa

				una pila auxiliar. Método Partición

Algoritmo de Huffman

- Es un algoritmo que puede ser usado para compresión de datos
- Huffman se basa en asignar códigos de distinta longitud de bits a cada uno de los caracteres de un archivo. Si se asignan códigos más cortos a los caracteres que aparecen más a menudo se consigue una compresión del archivo.
- Utiliza árbol binario
- Tiene
 - Tabla de frecuencia (flag, carácter, puntero árbol, código)
 - Pila
- Pasos Comprimir
 - Se crea la tabla de frecuencia, ordenada con los caracteres que mas aparecen arriba
 - Se toma de la tabla de frecuencia los dos con menor frecuencia (puede ser carácter o nodo), y se unen por un nodo (a la izquierda se pone el de menor frecuencia y a la derecha el de mayor frecuencia). Se escribe en la tabla el nuevo nodo creado, (alfa n) y en frecuencia se le pone la suma de la frecuencia de los nodos hijos.
 - Se arma el árbol, y se completa la columna código de la tabla de frecuencia
 - En el árbol a las aristas izquierda se le ponen 0, y a las derechas 1
 - El código de un caracteres es ir escribiendo 0 o 1 según corresponda por cada aristas que se pasa desde la raíz hasta la hoja (que es el carácter)
 - Formato archivo comprimido
 - Archivo comprimido
 - Cabecera(ASCII)
 - # caracteres diferentes
 - Tabla frecuencia original
 - Cuerpo (se graba de a byte, se completa con ceros)
- Descomprimir
 - Se tiene la tabla de frecuencia
 - Inicia variable de caracteres a descomprimir
 - Detecta los caracteres porque son hojas (decrementa la variable de caracteres a descomprimir)

Acceso a datos

- Acceso a datos
 - Hashing
 - Árbol B <- El mas utilizado
- Hashing
 - Que sucede cuando hay colisión

- Se aplica una función de rehash

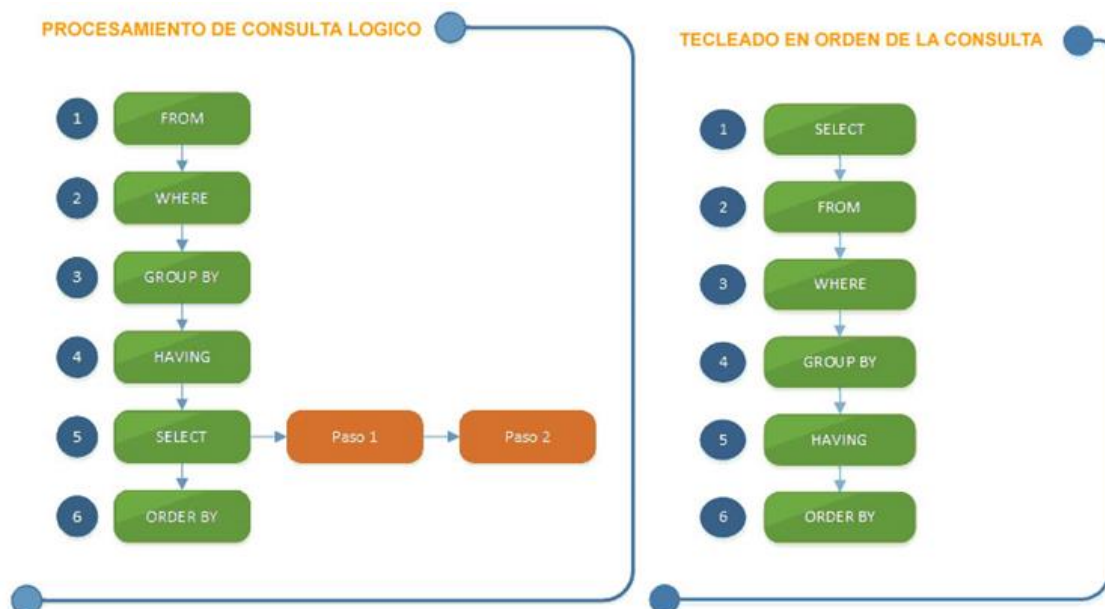
Objetos de la base de datos

- Objetos:
 - Tablas
 - Tablas temporales (locales y globales)
 - Constraint
 - Integridad de entidades (PK)
 - Integridad referencial (FK)
 - Integridad semántica
 - Secuencial
 - Vistas
 - Índices
 - Snapshots
 - Funciones propias del motor
 - Funciones agregadas
 - Funciones escalares
 - Funciones de tablas
 - Funciones de usuarios
 - Stored Procedures
 - Triggers
 - Insert
 - Update
 - Delete

Concepto de base de datos

- Ventajas
 - Aplicar políticas de seguridad
 - Aplicar normas de negocio
 - Disminuir redundancia
 - Evitar inconsistencia
 - Compartir los datos
 - Mantener la integridad
 - Criterio de organización
 - Independencia de los datos
 - Lógica
 - Física
- DDL: Data Definition/Description Language
 - Crear, eliminar, borrar estructuras. (no toca los datos)
- DML: Data Manipulation Language
 - Trabaja con los datos, leer, crear, borrar, modificar
- DCL: Data Control Language
 - Dar o revocar permisos a los usuarios
- Funciones del motor de base de datos y del DBMS
 - Diccionario de datos:

- Es lo que se llama “metadatos”. Es una base de datos que define a los objetos. Se almacenan las diversas vistas (internas/externas). Esta integrado en la BD.
- Control de seguridad
 - Permisos de los objetos
 - Vistas
- Mecanismo para garantizar la integridad y consistencia
 - Constraints: garantizan integridad, ej (PK, FK)
 - Triggers
 - Transacciones
 - Logical Logs
- Backup and Recovery
 - Recovery: Se utilizan los logs transaccionales
 - Backup
 - Backup full
 - Backup en caliente. (con usuarios)
 - Backup en frio. (sin usuarios)
 - Backup Log transaccional:
 - Backup Diferencial
 - Incremental
 - Acumulativo
- Concurrencia
- Facilidad auditoria
- Logs



Transacciones y niveles de aislamientos

- Las transacciones deben cumplir el concepto ACID
 - Atomicidad
 - Consistencia

- Isolation (Aislamiento)
- Durabilidad (Persistencia)
- Niveles de aislamientos
 - **Lectura sucia:** Una lectura sucia ocurre cuando se le permite a una transacción la lectura de una fila que ha sido modificada por otra transacción concurrente pero todavía no ha sido cometida.
 - **Lectura no repetible:** Una lectura no repetible ocurre cuando en el curso de una transacción una fila se lee dos veces y los valores no coinciden.
 - **Lectura fantasma:** Cuando A y B hacen el mismo SELECT pero en el medio se pusieron mas registros, B va a obtener mas registros. (Una lectura fantasma ocurre cuando, durante una transacción, se ejecutan dos consultas idénticas, y los resultados de la segunda no son iguales a los de la primera.)

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura Fantasma
READ UNCOMMITTED	Sí	Sí	Sí
READ COMMITTED	No	Sí	Sí
REPEATABLE READ	No	No	Sí
SERIALIZABLE	No	No	No

Niveles de aislamiento vs Bloqueos [\[editar \]](#)

Nivel de aislamiento	Bloqueo de escritura	Bloqueo de lectura	Bloqueo de rango
Read Uncommitted	-	-	-
Read Committed	V	-	-
Repeatable Read	V	V	-
Serializable	V	V	V

"V" indica que el método bloquea esa operación, manteniéndolo hasta el final de la transacción.

Nota: Las operaciones de lectura (i.e. SELECT) pueden efectuar bloqueos (compartidos) en el nivel *Read Committed*, pero se liberan inmediatamente tras la operación de lectura.