RESUMEN FOD – FINAL

CAPITULO 1 (Base de datos y gestores de bases de datos)

- **Base de datos**
  Es una colección o conjunto de datos interrelacionados con un propósito específico vinculado a la resolución de un problema del mundo real.
- **Gestores de BD**
  Un sistema de gestión de BD (SGBD) consiste en un conjunto de programas necesarios para acceder y administrar una BD.
  - Lenguaje de definición de datos (LDD): esto implica el diseño de la estructura que tendrá efectivamente la BD; describir los datos, las relaciones entre ellos, la semántica asociada y las restricciones de consistencia. El resultado de compilar lo escrito con el LDD es un archivo llamado diccionario de datos, este tiene datos acerca de los datos.
  - Lenguaje de manipulación de datos (LMD): mediante este se puede recuperar o agregar nueva información y modificar o borrar datos existentes. Tipos:
    - LMD Procedimentales: el nuevo usuario debe especificar QUE datos requiere y COMO obtener esos datos.
    - LMD no procedimentales: el usuario debe especificar QUE datos requiere, sin detallar COMO obtenerlos.
  - Objetivos de un SGBD:
    - Controlar la concurrencia: varios usuarios pueden acceder a la misma información en un mismo periodo de tiempo y si más de uno quiere actualizar el mismo dato a la vez, se puede llegar a un estado de inconsistencia.
    - Tener control centralizado: tanto de los datos como de los programas que acceden a los datos.
    - Facilitar el acceso a los datos: dado que provee un lenguaje de consulta para recuperación rápida de información.
    - Proveer seguridad para imponer restricciones de acceso: se debe definir quiénes son los usuarios autorizados a acceder a la BD.
    - Mantener la integridad de los datos: implica que los datos incluidos en la BD respeten las condiciones establecidas al definir la estructura de la BD y que, ante una falla, se posea la capacidad de restauración a la situación previa.
- **Niveles de visión de los datos**
  - Nivel de vista: corresponde al nivel más alto de la abstracción. En este se describe parcialmente la BD, solo la parte que se desea ver.
  - Nivel lógico: en este nivel se describe la BD completa, indicando que datos se almacenaran y las relaciones existentes entre esos datos.
  - Nivel físico: este es el nivel más bajo de la abstracción, en el cual se describe como se almacenaran realmente los datos.
- **Modelo de datos**
  Es un conjunto de herramientas conceptuales que permite describir los datos y su semántica, sus relaciones y las restricciones de integridad y consistencia. Existen tres grupos: - Modelos lógicos basados en objetos. – Modelos lógicos basados en registros. – Modelos físicos.
  Los modelos basados en objetos son utilizados para describir datos en los niveles de vista y lógicos; y los basados en registros son utilizados para describir datos en los niveles lógico y físico.
  - Modelos lógicos basados en objetos: son utilizados para representar información de un problema del mundo real con un esquema de alto nivel de abstracción. Dentro de estos se destacan el modelo de

entidad y relaciones, y el modelo orientado a objetos. Actualmente el modelo de datos ER es el más utilizado debido a su simplicidad y en potencia expresiva. El modelo OO representa la realidad con objetos de similares características a los objetos del mundo real.

- Modelos lógicos basados en registros: utilizan la estructura de datos de registro para almacenar la información en un BD. Existen tres modelos:
  - Modelo jerárquico: consiste en utilizar registros vinculados entre si formando una estructura de árbol.
  - Modelo en red: utiliza registros con vínculos que generan una estructura de grafo.
  - Modelo relacional: utiliza un conjunto de tablas para representar la información. Estas tablas se vinculan entre si y establecen relaciones de integridad entre los datos que la componen.

- **Categorías de usuarios de BD**
  - Administrador de BD: es la persona que tiene el control centralizado de la BD a través de SGBD. Sus funciones más importantes son definir el esquema de la BD, otorgar derechos de acceso a otros usuarios y establecer las restricciones de integridad entre los datos.
  - Programadores de aplicaciones: son los profesionales del ámbito informático que desarrollan sistema de software.
  - Usuarios sofisticados: realizan consultas a la BD a través de un lenguaje de consulta.
  - Usuarios especializados: aquellos usuarios que desarrollan aplicaciones no tradicionales (sistemas expertos, etc.).
  - Usuarios normales: usuarios clásicos que utilizan los sistemas de software desarrollados por los usuarios antes descriptos.

CAPITULO 2 (Archivos, estructura y operaciones básicas)

- **Archivo**

  Un archivo es una colección de registros que abarcan entidades con un aspecto común y originado para algún propósito particular.

  Es una estructura de datos homogénea y se caracterizan por e crecimiento y las modificaciones que se efectúan sobre estos.

- **Aspectos físicos**

  Cuando es necesario tener persistencia en la información que maneja un algoritmo se debe almacenar en archivos.

  Los archivos residen en dispositivos de memoria secundaria o almacenamiento secundario, los cuales están ubicados fuera de la memoria RAM y son capaces de retener información. Ejemplos de almacenamiento secundario son los discos rígidos, diskettes, discos ópticos, cintas, etc.

  Existen dos diferencias básicas entre las estructuras de datos que residen en la RAM y las que van en almacenamiento secundario. Capacidad (la RAM tiene capacidad más limitada) y el tiempo de acceso (la RAM mide el acceso en nanosegundos y un disco rígido lo hace en milisegundos).

- **Administración de archivos**

  Se deben considerar dos aspectos:
  - Visión física: tiene que ver con el almacenamiento de este en memoria secundaria. Es responsabilidad del SO resolver cuestiones relativas al lugar de almacenamiento, como se ubicara la información y como será recuperada.
  - Visión lógica: desde el punto de vista práctico lo utiliza como si estuviera almacenado en la memoria RAM. Desde un algoritmo se establece una “conexión” con el SO, y este determinara el lugar de residencia y su administración.

A partir de estas visiones se distinguen dos conceptos diferentes pero interrelacionados:

- Archivo físico: es el archivo residente en la memoria secundaria y es administrado por el SO.
- Archivo lógico: es el archivo utilizado desde el algoritmo. Se genera una conexión con el SO para que este lo administre.

- **Tipos de archivos**

  Por la información que contiene es una estructura homogénea. Pueden ser de longitud física o variable.

- **Acceso a información contenida en los archivos**
  - Secuencial: el acceso a cada elemento de datos se realiza luego de haber accedido a su inmediato anterior. El recorrido es desde el primer hasta el último elemento, siguiendo el orden físico.
  - Secuencial indizado: el acceso a los elementos de un archivo se realiza teniendo presente algún tipo de organización previa, sin tener en cuenta el orden físico.
  - Directo: es posible recuperar un elemento de dato de un archivo con un solo acceso, conociendo sus características, mas allá de que exista un orden físico o lógico predeterminado.

- **Operaciones básicas sobre archivos**

  Estas incluyen:

  - La definición del archivo lógico que utilizara el algoritmo y la relación del nombre lógico del archivo con su almacenamiento en el disco rígido.
  - La definición de la forma de trago del archivo que puede ser la creación inicial de un archivo o la utilización de uno ya existente.
  - La administración de datos (lectura y escritura de información).

  - Definición: se reserva la palabra 'file' para indicar la definición.

    Var archivo_logico: file of tipo_de_dato;

    Donde archivo_logico es el nombre de la variable, file es la palabra clave reservada y tipo_de_dato indica el tipo de información que contendrá el archivo.
  - Correspondencia archivoLogico-archivoFisico: se reserva la palabra 'assign'.

    Assign (nombre_logico, nombre_fisico);

    Donde nombre_logico e sla variable definida en el algoritmo y nombre_fisico es una cadena de caracteres que representa el camino donde quedara el archivo y nombre del mismo.
  - Apertura y Creación: para abrir un archivo existen dos posibilidades:
    - que el archivo aun no exista porque no fue creado.
    - Que el archivo exista y se quiera operar con él.

    Para esto se disponen dos operaciones:
    - Rewrite indica que el archivo va a ser creado, y la única operación valida sobre el mismo es escribir información. "rewrite (nombre_logico);"
    - Reset indica que el archivo ya existe y las operaciones sobre el mismo son lectura/escritura de información. "reset (nombre_logico);"
  - Cierre: cerrar un archivo significa una serie de eventos.
    - siempre debe tener un fin. La marca de fin se cono por su sigla en ingles EOF (end of file) y servirá posteriormente para controlar las lecturas sobre el archivo y no exceder la longitud del mismo.
    - Trasferir definitivamente la información volcada sobre el archivo a disco, es decir, trasferir el buffer a disco.

    La operación es: close (nombre_logico);
  - Lectura y Escritura:

    Instrucción de lectura: read (nombre_logico, var_dato);

    Intruccion de escritura: Write (nombre_logico, var_dato);

Nombre_logico corresponde al nombre lógico del archivo donde se desea leer o escribir, y var_datos es la variable del algoritmo que contendrá la información a transferir al archivo o la variable que recibirá la información desde el archivo. Estos deben coincidir.

- **Buffers de memoria**
  Las lecturas y escrituras desde o hacia archivos se realizan sobre buffers. Se denomina buffers a una memoria intermedia (ubicada en la RAM) entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados en memoria secundaria. La forma de trabajar con buffer optimiza la performance de lecturas y escrituras sobre archivos.
- **Operaciones adicionales**
  - Control de fin de datos: función "eof (nombre_logico);" es la encargada de resolver este control. la función retornara verdadero si el puntero del archivo referencia a eof, y falso en caso contrario.
  - Control de tamaño del archivo: función "filesize (nombre_logico);" y retorna un valor entero indicando cuantos elementos se encuentran almacenados en el archivo.
  - Control de posición de trabajo: función "filepos (nombre_logico);" y retorna un nuemero intero indicando la posición actual del puntero. Siempre comprende entre cero y la cantidad de elementos que tiene el archivo (filesize).
  - Ubicación física en alguna posición: con cada operación de lectura y escritura avanza automáticamente el puntero lógico del archivo. Para modificar esta posición, se dispone de la instrucción "seek (nombre_logico, posición);" donde posición es un valor entero dentro del rango de cero y filesize, e indicara el lugar donde se ubicara el puntero.

CAPITULO 3 (Algorítmica clásica sobre archivos)

- **Archivo maestro**
  Es un archivo que resume información sobre un dominio de problema especifico. Ejemplo: el archivo de productos de una empresa que contiene el stock actual de cada producto.
- **Archivo detalle**
  Es un archivo que contiene novedades o movimientos realizados sobre la información almacenada en el maestro. Ejemplo: el archivo con todas las ventas de los productos de la empresa realizadas en un dia particular.
- **Proceso de actualización**
  - Actualización de un archivo maestro con un archivo detalle
  - Merge: es el proceso por el cual se genera un nuevo archivo a partir de otros archivos existentes. También llamado unión, y la principal diferencia con la actualización anterior es que el archivo maestro no existe, y por lo tanto debe ser generado.
  - Corte de control: es el proceso mediante el cual la información de un archivo es presentada en forma organizada de acuerdo con la estructura que tiene el archivo.

CAPITULO 4 (Eliminación de datos. Archivos con registros de longitud variable)

- **Proceso de baja**
  Es aquel proceso que permite quitar información de un archivo. Tiene dos perspectivas: una ligada con la algorítmica y performance necesarias para borrar la información, y otra que tiene que ver con la necesidad real de quitar información de un archivo en el contexto informático actual.
  Se puede llevar a cabo de dos maneras:
  1) Baja física: consiste en borrar efectivamente la información del archivo, recuperando el espacio físico.
  2) Baja lógica: consiste en borrar la información del archivo, pero sin recuperar el espacio físico.

- Baja física: esto se realiza cuando un elemento es quitado del archivo y es reemplazado por otro elemento del mismo archivo, decrementando en uno su cantidad de elementos.
  Ventajas: administrar en cada momento un archivo de datos que ocupe el lugar mínimo necesario.
  Desventajas: tiene que ver con la performance final de los algoritmos que se implementan.
  Existen dos técnicas algorítmicas para este proceso:
    - Baja física generando un nuevo archivo: Generar un nuevo archivo con los elementos válidos, es decir, sin copiar los que se desean eliminar. El análisis de performance básico determina que este método necesita leer tantos datos como tenga el archivo original y escribir todos los datos, menos el que se elimina. Suponiendo que el archivo tiene N registros, N lecturas y N-1 escrituras; las lecturas y escrituras se realizan en forma secuencial.
      Una vez finalizado este proceso, coexisten en el disco rígido dos archivos, por esto se debe disponer en memoria secundaria la capacidad de almacenamiento suficiente.
    - Baja física utilizando el mismo archivo: Utilizar el mismo archivo de datos, generando los reacomodamientos que sean necesarios. El análisis de performance determina que la cantidad de lecturas a realizar es N, en tanto que la cantidad de escrituras dependerá del lugar donde se encuentra el elemento a borrar. En el peor de los casos, se realizaran N-1 escrituras, si el elemento a borrar aparece en la primera ubicación.
- Baja lógica: esto se realiza cuando el elemento que se sedea quitar es marcado como borrado, pero sigue ocupando el espacio dentro del archivo. Ventajas: tiene que ver con la performance, basta con localizar el registro a eliminar y colocar sobre él una marca que indique que se encuentra no disponible. Desventaja: está relacionada con el espacio en disco. Al no recuperarse el espacio borrado, el tamaño del archivo tiende a crecer continuamente.
- Recuperación de espacio: periódicamente utilizar el proceso de baja física para realizar un proceso de compactación de archivo consiste en quitar todos aquellos registros marcados como borrados, utilizando para ello cualquiera de los algoritmos para borrado físico.
- Reasignación de espacio: consiste en reutilizar el espacio indicado como borrado para que nuevos registros se inserten en dichos lugares. Utilizaríamos el proceso de alta pero en vez de avanzar sobre la última posición, se debe localizar alguna posición marcada como borrada para insertar el nuevo elemento.

- **Campos y registros con longitud variable**
  - Archivos de longitud fija: la información siempre es homogénea, los datos almacenados son del mismo tipo, y por ende son del mismo tamaño. Ventajas: el proceso de entrada y salida de información, desde y hacia los buffers, es responsabilidad del SO.
  - Archivos de longitud variable: en estos archivos la cantidad de espacio utilizada por cada elemento no está determinada a priori. Cada elemento de dato debe descomponerse en cada uno de sus elementos constitutivos y así, elemento a elemento, guardarse en el archivo. Es necesario colocar marcas de inicio y fin de cada campo o registro. Características:
    - El archivo se define como file y permite realizar la trasferencia de la información campo a campo.
    - Cuando se termina de insertar un campo, se utiliza como marca de fin el carácter #.
    - Cuando se termina de insertar un registro, se utiliza como marca de fin el carácter @.
  - Eliminación con registros de longitud variable: un elemento puede ser borrado de manera físico o lógica. Para recuperar el espacio borrado lógicamente con nuevo elementos se debe tener en cuenta el espacio disponible.
    El proceso de inserción debe localizar el lugar dentro del archivo más adecuado al nuevo elemento.
    Existen tres formas para esto:

        - Primer ajuste: consiste en seleccionar el primer espacio disponible donde quepa el registro a insertar. Genera fragmentación interna.
        - Mejor ajuste: consiste en seleccionar el espacio más adecuado para el registro. Se considera el espacio más adecuado como aquel de menor tamaño donde quepa el registro. Genera fragmentación interna.
        - Peor ajuste: consiste en seleccionar el espacio de mayor tamaño, asignando para el registro solo los bytes necesario. Genera fragmentación externa.
    - Fragmentación: existen dos tipos:
        - Frag. Interna: es aquella que se produce cuando a un elemento de dato se le asigna mayor espacio del necesario. Re reserva lugar que no se utiliza.
        - Frag. Externa: es el espacio disponible entre dos registros, pero que es demasiado pequeño para poder ser reutilizado.
    - Modificación con registros de longitud variable: el problema se genera cuando el dato a modificar tiene menos espacio que el que se quiere insertar. Para evitar estoy y facilitar el algoritmo se estila dividir el proceso de modificación en dos etapas: en la primera se da de baja al elemento de dato viejo, mientras que en la segunda el nuevo registro es insertado de acuerdo con la política de recuperación de espacio determinada.

CAPITULO 5 (Búsqueda de información. Manejo de índices)

- **Proceso de búsqueda**

  Cuando se realiza la búsqueda de un dato, se debe considerar la cantidad de accesos a disco en pos de encontrar esa información, y en cada acceso, la verifación de si el dato obtenido es el buscado (comparación).
  Este proceso implica un análisis de situaciones en función del tipo de archivo sobre el que se quiere buscar información.
    - Búsqueda secuencial:
        - Archivo desordenado: hay que recorrer el archivo desde el primer registro hasta que se encuentre o finalice sin encontrarlo. Mejor caso → 1 acceso. Peor caso → N accesos. Promedio → N/2 accesos.
        - Archivo ordenado: si el argumento de búsqueda coincide con el criterio de ordenación, el proceso se detiene en el registro cuyo dato es “mayor” al buscado si no se encuentra.

      En ambos casos la performance es de orden N.
    - Búsqueda binaria: el archivo debe estar ordenado. La primera comparación del dato que se pretende localizar es contra el registro medio del archivo (NRR = N/2). Si el registro no contiene ese dato, se descarta la mitad mayor o menor, según corresponda, reduciendo el espacio de búsqueda a los N/2 registros restantes. Nuevamente se realiza la comparación con esta mitad restante, repitiendo así el proceso hasta reducir el espacio a un registro. La performance es de orden Log2(N).
- **Ordenamiento de archivos**
    - Alternativas:
        - Si el archivo a ordenar puede almacenarse en forma completa en memoria RAM, una alternativa es trasladar el archivo compuesto desde memoria secundaria a memoria principal y luego ordenarlo ahí. En cuanto a performance es la mejor alternativa pero solo sirve con archivos pequeños.
        - Si el archivo no cabe en memoria RAM, otra alternativa seria trasferir a memoria principal de cada registro solo la clave por la que se desea ordenar, junto con los NRRs, a los registros correspondientes en memoria secundaria.

        - Surge cuando el archivo es realmente grande. Implica los siguientes pasos:
            - Dividir el archivo en particiones de igual tamaño, de modo que cada partición quepa en memoria principal.
            - Transferir las particiones (de a una) a memoria principal. Esto implica realizar lecturas secuenciales sobre memoria secundaria, pero sin ocasionar mayores desplazamientos.
            - Ordenar cada partición en memoria principal y reescribirlas ordenadas en memoria secundaria. La escritura es secuencial. (estos tres puntos se llaman sort interno).
            - Realizar el merge o fusión de las particiones, generando un nuevo archivo ordenado.
    - Selección por reemplazo: este método se basa en seleccionar siempre de memoria principal la clave menor, enviarla a memoria secundaria y reemplazarla por una nueva clave que está esperando ingresar a memoria principal. Pasos a cumplir:
        - 1) Leer desde memoria secundaria tantos registros como quepan en memoria principal.
        - 2) Iniciar una nueva partición.
        - 3) Seleccionar, de los registros disponibles en memoria principal, el registro cuya es menor.
        - 4) Transferir el registro elegido a una partición en memoria secundaria.
        - 5) Reemplazar el registro elegido por otro leído desde memoria secundaria. Si la clave de este registro es menor que la clave del registro recientemente transferido a memoria secundaria, este nuevo registro se lo guarda como no disponible.
        - 6) Repetir desde el paso 3 hasta que todos los registros en memoria principal estén no disponibles.
        - 7) Iniciar una nueva partición activando todos los registros no disponibles en memoria principal y repetir desde el paso 3 hasta agotar los elementos del archivo a ordenar.
    - Selección natural: es una variante que reserva y utiliza memoria secundaria en la cual se insertan los registros no disponibles. De este modo, la generación de una partición finaliza cuando este nuevo espacio reservado esta en overflow (completo).
    - Comparación de los tres métodos:

      Ventajas:
        - Sort Interno: produce particiones de igual tamaño. Algorítmicamente simple. Cualquier variante del método de merge es más eficiente si todos los archivos que unifica son de igual tamaño.
        - Selección natural y por reemplazo: producen particiones con tamaño promedio igual o mayor al doble de la cantidad de registro, que caben en memoria principal.

      Desventajas:
        - Sort Interno: es el más costo en términos de performance.
        - Selección natural y por reemplazo: tienden a generar muchos registros no disponibles. Las particiones no quedan, necesariamente, de igual tamaño.

- **Indización**

Un índice es una estructura de datos adicional que permite agilizar el acceso a la información almacenada en un archivo. En dicha estructura se almacenan las claves de los registros del archivo, junto a la referencia de acceso a cada registro asociado a la clave. Es necesario que las claves estén ordenadas. Tiene registros de longitud fija. La característica fundamental es que posibilita imponer orden en un archivo sin que realmente este se reacomode.

- Índice primario:
    - Creación: al crearse el archivo, se crea también el índice asociado, ambos vacíos, solo con el registro encabezado.
    - Altas: consiste simplemente en agregar dicho registro al final del archivo. A partir de acá, con el NRR o la dirección del primer byte, más la clave primaria, se genera un nuevo registro de datos a insertar en forma ordenada en el índice.

        - Modificaciones: se considera la posibilidad de cambiar cualquier parte del registro excepto la clave primaria. Si el archivo es de longitud fija, el índice no se altera. Si el archivo es de longitud variable y el registro modificado no cambia de longitud, el índice no se altera. Si el registro modificado cambia de longitud, agrandando su tamaño, este debe cambiar de posición, reubicarse.
        - Bajas: se debe borrar física o lógicamente el registro correspondiente en el índice.
        - Ventajas: posibilita mejorar la performance de búsqueda. Puede realizar búsqueda binaria.
        - Índices para claves candidatas: las claves candidatas son claves que no admiten repeticiones de valores para sus atributos, similares a una clave primaria.
    - Índice secundario: es una estructura adicional que permite relacionar una clave secundaria con una o más claves primarias, dado que varios registros pueden contener la misma clave secundaria.
        - Creación: al implantarse el archivo de datos, se deben crear todos los índices secundarios asociados, naturalmente vacíos, solo con el registro encabezado.
        - Altas: cualquier alta en el archivo genera una inserción en el índice secundario, que implica reacomodar el archivo en el cual se almacena. En términos de performance es de bajo costo si el índice puede estar en memoria principal, pero es muy costoso si esta en memoria secundaria.
        - Modificaciones: se analizan dos alternativas. La primera ocurre cuando se produce un cambio en la clave secundaria, acá se debe reacomodar el índice secundario, con los costos que ello implica. Y la segunda ocurre cuando cambia el resto del registro de datos (excepto clave primaria), no generando así ningún cambio en el índice secundario.
        - Bajas: implica eliminar la referencia a ese registro del índice primario más todas las referencias en índices secundarios. La eliminación puede ser física o lógica.
        - Alternativas de organización: la primera alternativa implica almacenar en un mismo registro todas las ocurrencias de la misma clave secundaria. El problema es la elección del tamaño del registro (long fija) ya que puede haber casos en que resulte insuficiente o que sobre espacio.
          Otra alternativa es pensar en una lista de claves primarias asociada a cada clave secundaria, dicha lista, llamada lista invertida, esta almacenada en otro archivo, el cual se recorre de acuerdo con el camino establecido por el atributo enlace, que indica cual es el próximo registro asociado y en caso de no existir tiene puntero nulo.
          Ventajas: - El único reacomodamiento en el índice se produce cuando se agrega una clave primaria. – Si se agregan o borran datos de una clave secundaria ya existente, solo se debe modificar el archivo que contiene la lista. – Dado que se generan los archivos, uno de ellos podría residir en memoria secundaria, liberando memoria principal.
    - Índices selectivos: consiste en disponer de índices que incluyan solo claves asociadas a una parte de la información existen, aquella información que tenga mayor interés de acceso.

CAPITULO 6 (ARBOLES)

- **Arboles binarios**

  Es una estructura de datos dinámica no lineal, en la cual cada nodo puede tener a lo sumo dos hijos. En general tiene sentido cuando esta ordenando.
    - Búsqueda: se realiza a partir del nodo raíz y se recorre hacia los nodos hoja. Se chequea un nodo, si es el deseado finaliza o si no lo es continúa hacia la izquierda o derecha descartando una mitad.
      Por este motivo, la búsqueda es de orden logarítmico, orden log2(N), siendo N la cantidad de elementos distribuidos en el árbol.

- Definición: Type

```
regitroArbolBinario=record
    elemento_de_dato : tipo_de_dato;
    hijo_izq : integer;
    hijo_der : intenger;
    end;
indiceBinario = file of registroArbolBinario;
```

  Donde 'elemento_de_dato' contendrá la clave del índice, y 'hijo_izq' e 'hijo_der' definen la dirección del hijo menor y mayor.
- Insertar nuevo elemento: pasos:
  - Agregar el nuevo el nuevo elemento de datos al final del archivo.
  - Buscar al padre de dicho elemento. Para ellos se recorre el archivo desde la raíz hasta llegar a un nodo terminal.
  - Actualizar el padre, haciendo referencia a la dirección del nuevo hijo.

  Desde el punto de vista de accesos a disco la performance es: se deben realizar log2(N) lecturas para localizar al padre y dos operaciones de escritura (nuevo elemento y actualización).
- Borrado de un elemento: debe ser necesariamente un elemento terminal. Si no lo es debe intercambiarse el elemento en cuestión con el menor de sus hijos mayores. La performance es igual al insertar.
- Arboles balanceados: es aquel árbol donde la trayectoria de la raíz a cada una de las hojas está representada por igual cantidad de nodos.
- Árbol AVL: es un árbol balanceado en altura donde el delta determinado es uno, es decir, el máximo desbalanceo posible es uno.

- **Paginación de árboles binarios**

  Dicho árbol se divide en páginas, es decir, se página y cada página contiene un conjunto de nodos, los cuales están ubicados en direcciones físicas cercanas. Así cuando se transfieren datos, se transmite una página completa. Al dividir un árbol binario en páginas, es posible realizar búsquedas más rápidas de datos almacenados en memoria secundaria.

  La performance final de búsqueda sería del orden log k+1(N), siendo N la cantidad de claves del archivo y k la cantidad de nodos por página.
- **Arboles multicaminos**

  Es una estructura de datos en la cual cada nodo puede contener k elementos y k+1 hijos.

  Se define el concepto de orden de un árbol multicamino como la máxima cantidad de descendientes psibles en un nodo.

CAPITULO 7 (FAMILIA DE ARBOLES BALANCEADOS)

- **Arboles B (balanceados)**

  Son árboles multicamino con una construcción especial que permite mantenerlos balanceados a bajo costo. Un árbol B de orden M posee las siguientes propiedades básicas:
  - Cada nodo del árbol puede contener, como máximo, M descendientes y M-1 elementos.
  - La raíz no posee descendientes directos o tiene al menos dos.
  - Un nodo con x descendientes directos contiene x-1 elementos.
  - Los nodos terminales (hojas) tienen como mínimo, [M/2]-1 elementos, y como máximo, M-1 elementos.
  - Los nodos que no son terminales ni raíz tiene, como mínimo, [M/2] elementos.
  - Todos los nodos terminales se encuentran al mismo nivel.

- Definición: const
  Orden=255;
  Type
  Reg_arbol_b=record
  Hijos:array[0..orden] of integer;
  Claves:array[1..orden] of tipo_de_dato;
  Nro_registros:integer;
  End;

  Donde 'hijos' es un arreglo que contiene la dirección de los nodos que son descendientes directos, 'claves' es un arreglo que contiene las claves que forman el índice del árbol y número de registros indica la dimensión efectiva en uso de cada nodo.
- Creación de árbol B: el proceso de creación comienza con una estructura vacía. En el momento inicial, el único nodo existente en el árbol será un nodo raíz, el cual no contendrá ningún elemento, y el archivo índice se encontrara vacío. Cuando llega el primer elemento se inserta en la primera posición libre del nodo raíz. Los elementos siguientes comenzaran siempre el proceso de inserción a partir del nodo raíz.
- Inserción en un árbol B: si el nodo raíz tiene lugar se procederá a insertarlos en este nodo teniendo en cuenta que los elementos de datos deben quedar ordenados dentro del nodo.
  Cuando llega un elemento nuevo y no hay más capacidad dentro del nodo se produce lo que llamamos overflow, ante esto se hace:
  - Se crea un nodo nuevo.
  - La primera mitad de las claves se mantiene en el nodo viejo.
  - La segunda mitad de las claves se trasladan al nodo nuevo.
  - La menor de las claves de la segunda mitad se promociona al nodo padre.
- Búsqueda en un árbol B: comienza desde el nodo raíz a buscar el elemento. En caso de encontrarlo en dicho nodo, se retorna una condición de éxito (implica retornar la dirección física en memoria secundaria, asociada al registro que contiene la clave encontrada). Si no se encuentra, se procede a buscar en el nodo inmediato siguiente que debería contener al elemento, procediendo de esa forma hasta encontrarlo, o hasta encontrar un nodo sin hijos que no incluya el elemento.
- Eficiencia de búsqueda en un árbol B: consiste en contar los accesos al archivo de datos que se requieren para localizar un elemento o para determinar que el elemento no se encuentra. Hay que establecer una cota para la altura o niveles del árbol en función de los registros que el árbol contiene. Se debe analizar siempre la peor situación posible. Cota deseada: $H < 1 + \log [M/2] ((N+1)/2)$ donde H es la altura y N cantidad de registros.
- Eficiencia de inserción en un árbol B: pueden surgir dos alternativas. Si el nodo a realizar la operación dispone de lugar, no se produce overflow, solo sería necesaria una escritura en el nodo terminal con el nuevo elemento y se da finalizado el proceso.
  Si se produce overflow, el peor de los casos es aquel donde el overflow se propaga hacia la raíz, haciendo aumentar en uno el nivel del árbol.
  Resumiendo, la performance es: H lecturas 1 escritura (mejor caso) H lecturas (2*H)+1 escrituras (peor caso).
- Eliminación en arboles B: existen dos situaciones:
  - La primera se presenta cuando, al borrar el elemento del nodo terminal, la cantidad de elementos restantes no está por debajo de la cantidad mínima ([M/2]-1). En este caso, no se genera un overflow en el nodo, y el proceso de baja finaliza.

        - La segunda situación posible es aquella que genera una situación de underflow. En este caso, el nodo al que se le quita un elemento deja de cumplir la condición de contener al menos [M/2[-1 elementos. Existen varias acciones frente a esto:
            - La acción más inmediata consiste en la opuesta a la división en un caso de overflow, la cual se denomina concatenación.
            - Otra alternativa es la redistribución, que plantea utilizar algún nodo adyacente hermano del nodo conflictivo, permitiendo que dicho nodo adyacente hermano ceda elementos al nodo que presentan insuficiencia. Generalizando, la mitad de los elementos queda en un nodo, la otra mitad en el segundo nodo, y se toma un elemento que actué como separador y quede ubicado en el nodo padre.
    - Eficiencia de eliminación:
        - Mejor caso: se intenta eliminar un elemento que está en un nodo terminal y cuyo borrado no genera insuficiencia. Entonces serán necesarias tantas lecturas como niveles tenga el árbol y una sola escritura (la correspondiente al nodo terminal sin el elemento borrado).
        - Peor caso: quedara representado cuando la operación de borrado necesite concatenar. Implica leer un nodo adyacente hermano. Por cada nivel que se deba concatenar habrá dos lecturas, salvo en el nodo raíz, el cual carece de hermanos. La cantidad de escrituras se limita a una por nivel.
    - Modificación en arboles B: se procede tomando un cambio como una baja del elemento anterior y un alta del nuevo elemento del nuevo elemento. El análisis de performance es el obtenido de sumar una operación de baja seguida de un alta.
    - Nodos hermanos: son aquellos nodos que tienen el mismo nodo padre.
    - Nodos hermanos adyacentes: aquellos nodos que, siendo hermanos, son además dependientes de punteros consecutivos del padre.

- **Arboles B***

Es un árbol balanceado con las siguientes propiedades:

- Cada nodo del árbol puede contener, como máximo, M descendientes y M-1 elementos.
- La raíz no posee descendientes o tiene al menos dos.
- Un nodo con x descendientes contiene x-1 elementos.
- Los nodos terminales tienen, como mínimo, [(2M-1)/3]-1 elementos, y como máximo, M-1 elementos.
- Los nodos que no son terminales ni raíz tienen, como mínimo, [(2M-1)/3] descendientes.
- Todos los nodos terminales se encuentras al mismo nivel.

    - Búsqueda: la operación de búsqueda es similar a la de árboles B, comienza en el nodo raíz y se avanza hasta encontrar el elemento, o hasta llegar a un nodo terminal y no poder continuar con el proceso.
    - Baja: esta operación es similar a los arboles B, tanto para los casos donde no se genera underflow como para aquellos donde sí se genera insuficiencia. La primera opción consiste en redistribuir, o en su defecto en concatenar.
    - Inserción: puede ser regulado con tres políticas básicas:
        - Política de un lado: determina que el nodo adyacente hermano considerado será uno solo, definiendo la política de izquierda, o en su defecto, la política de derecha. En caso de completar un nodo, intenta redistribuir con el hermano indicado. En caso de no ser posible porque el hermano también está completo, tanto el nodo que genera overflow como dicho hermano son divididos de dos nodos llenos a tres nodos dos tercios llenos.
        - Política de un lado u otro lado: en caso de producirse una saturación en un nodo, se intenta primero redistribuir con un adyacente hermano. De no ser posible, se intenta con el otro

adyacente hermano. Si nuevamente no es posible, la alternativa es dividir de dos nodos llenos a tres nodos tercios llenos.

- Política de un lado y otro lado: la forma de trabajo es similar al anterior. Primero, redistribuir hacia un lado, y si no es posible, con el otro hermano. La diferencia aparece cuando los tres nodos están completos. Aquí se toman los tres nodos y se generan cuatro nodos con tres cuartas partes completas cada uno.

El análisis de performance depende de cada política. Ante la ocurrencia de overflow como mínimo requieren dos lecturas y tres escrituras. Si realiza una división, la política de un lado necesita cuatro escrituras al igual que las otras dos políticas.

- **Manejo de buffers. Arboles B virtuales**

  Cuando se decide la capacidad de cada nodo, esta está determinada por la cantidad de información que podrá manejar el SO, a través del uso de buffers. Se debe tener en cuenta que un SO administra un número importante de buffers y que es posible que aquellos buffers más utilizados se mantengan en memoria principal, a fin de minimizar el número de accesos a disco.

- **Acceso secuencial indizado**

  Es un archivo que permite dos formas para visualizar la información:

  1) Indizada: el archivo puede verse como un conjunto de registros ordenados por una clave o llave.
  2) Secuencial: se puede acceder secuencialmente al archivo, con registros físicamente contiguos y ordenados nuevamente por una clave o llave.

- **Arboles B+**

  Es un árbol multicamino e incorpora las características de árboles B, además del tratamiento secuencial ordenado del archivo. Propiedades:

  - Cada nodo del árbol puede contener, como máximo, M descendientes y M-1 elementos.
  - La raíz no posee descendientes o tiene al menos dos.
  - Un nodo con x descendientes contiene x-1 elementos.
  - Los nodos terminales tienen, como mínimo, [M/2]-1 elementos, y como máximo, M-1 elementos.
  - Los nodos que no son terminales ni raíz tienen, como mínimo, [M/2] descendientes.
  - Todos los nodos terminales se encuentras al mismo nivel.
  - Los nodos terminales representan un conjunto de datos y son enlazados entre ellos.

  Los elementos siempre se insertan en nodos terminales. Si se produce una saturación, el nodo se divide y se promociona una copia del menor de los elementos mayores hacia el nodo padre. Si el padre no tiene espacio, se dividirá nuevamente. Para borrar un elemento siempre se borra de un nodo terminal, y si hubiese una copia en un nodo no terminal, esta copia se mantiene como separador.

- **Arboles B+ de prefijos simples**

  Es un árbol B+ donde los separadores están representados por la mínima expresión posible de la clave, que permita decidir si la búsqueda se realiza hacia la izquierda o derecha.

CAPITULO 8 (DISPERSION. HASHING)

- **Conceptos de dispersión. Métodos de búsqueda mas eficientes**

  Hashing o dispersión es un método que mejora la eficiencia obtenida con árboles balanceados, asegurando en promedio un acceso para recuperar la información.

  Algunas definiciones:

  - Técnica para generar una dirección base única para una clave dada. La dispersión se usa cuando se requiere acceso rápido mediante una clave.

- Técnica que convierte la clave asociada a un registro de datos en un numero aleatorio, el cual posteriormente es utilizado para determinar donde se almacena dicho registro.
- Técnica de almacenamiento y recuperación que usa una función para mapear registros en direcciones de almacenamiento en memoria secundaria.

Atributos:

- No se requiere almacenamiento adicional. Dignifica que cuando se elige la opción de dispersión como método de organización de archivos, es el archivo de datos el que resulta disperso.
- Facilita la inserción y eliminación rápida de registros en el archivo.
- Localiza registros dentro del archivo con un solo acceso a disco.

Limitaciones:

- No es posible aplicarla en archivos con registros de longitud variable.
- No es posible obtener un orden lógico de los datos.
- No es posible tratar con claves duplicadas. No es aplicable la función hash sobre una clave secundaria.

- **Tipos de dispersión**
  - Hashing con espacio de direccionamiento estático: es aquella política donde el espacio disponible para dispersar los registros de un archivo de datos está fijado previamente. Así, la función de hash aplicada a una clave da como resultado una dirección física posible dentro del espacio disponible para el archivo.
  - Hashing con espacio de direccionamiento dinámico: es aquella política donde el espacio disponible para dispersar los registros de un archivo de datos aumenta o disminuye en función de las necesidades de espacio que encada momento tiene el archivo. Así, la función de hash aplicada a una clave da como resultado un valor intermedio, que será utilizado para obtener una dirección física posible para el archivo. Estas direcciones no están definidas a priori y son generadas de manera dinámica.
- **Parámetros de la dispersión**
  - Función de hash: es una función que transforma un valor, que representa una llave primaria de un registro, en otro valor dentro de un determinado rango, que se utiliza como dirección física de acceso para insertar un registro en un archivo de datos.
    No es posible almacenar dos registros en el mismo espacio físico. Existen varias alternativas para este problema:
    - Elegir un algoritmo de dispersión perfecto, que no genere colisiones. Este debe asegurar que dadas dos claves diferentes siempre se obtendrán dos direcciones físicas diferentes. Este método es extremadamente difícil.
    - Minimizar el número de colisiones a una cantidad aceptable y de esta manera tratarse como una condición excepcional. Modos de reducir colisiones:
      - Distribuir los registros de la forma más aleatoria posible.
      - Utilizar más espacio de disco.
      - Ubicar o almacenar más de un registro por cada dirección física en el archivo.
  - Tamaño de cada nodo de almacenamiento: la capacidad del nodo queda determinada por la posibilidad de transferencia de información en cada operación de entrada/salida desde RAM hacia disco y viceversa.
  - Densidad de empaquetamiento (DE): se la define como la relación entre el espacio disponible para el archivo de datos y la cantidad de registros que integran dicho archivo.
    Formula: DE = r / RPN*n
    Donde r es la cantidad de registros que componen a un archivo y RPN*r es el espacio disponible para almacenar el archivo, donde n es la cantidad de nodos direccionables por la función de hash y RPN la cantidad de registros que cada nodo puede almacenar.

Cuando mayor sea la DE, mayor será la posibilidad de colisiones, dado que en ese caso se dispone de menos espacio para esparcir registros.
Si la DE se mantiene baja, se dispone de mayor espacio, por ende, disminuye la probabilidad de colisiones. La DE no es constante.

- Métodos de tratamiento de desbordes (overflow): un desborde u overflow ocurre cuando un registro es direccionado a un nodo que no dispone de capacidad para almacenarlo. Cuando ocurre esto, hay dos acciones a realizar: encontrar lugar para el registro en otra dirección y asegurarse que el registro posteriormente sea encontrado en esa dirección.
- Estudio de la ocurrencia de overflow: para determinar la probabilidad de que un nodo reciba i registros se utilizara la fórmula: $P(i) = \frac{K! \ (1/N) \text{ a la i} * (1-1/N) \text{ a la k-i}}{i! * (k-i)!}$

Donde N representa el número de direcciones de nodos disponibles en memoria secundaria.
Donde K determina la cantidad de registros a dispersar.
Donde i determina la cantidad de registros que contendrá un nodo en un momento especifico.
Donde C determina la capacidad de cada nodo.
Poisson demostró que la probabilidad anterior se puede acotar a la siguiente fórmula:
$P(i) = \frac{(k/N) \text{ a la i} * e \text{ a la } -(K/N)}{I!}$

- **Resolución de colisiones con overflow**

Aquí se presentan cuatro métodos para reubicar a aquellos registros que no pueden ser almacenados en la dirección base obtenida a partir de la función de hash.

- Saturación progresiva: este método consiste en almacenar el registro en la dirección siguiente más próxima al nodo donde se produce saturación. Podría requerir chequear todas las direcciones disponibles en un caso extremo, para poder localizar un registro. También necesita indicar que si una dirección estuvo compuesta anteriormente, debe ser marcada como dirección ya saturada a fin de no impedir la búsqueda potencial de registros.
- Saturación progresiva encadenada: en líneas generales el método funciona igual al anterior. Un elemento que se intenta ubicar en una dirección completa es direccionado a la inmediata siguiente con espacio disponible. La diferencia radica en que, una vez localizada la nueva dirección, esta se encadena o enlaza con la dirección base inicial, generando una cadena de búsqueda de elementos.
- Doble dispersión: el método consiste en disponer de dos funciones de hash. La primera obtiene a partir de la llave la dirección de base, en el cual el registro será ubicado. De producirse overflow, se utilizara la segunda función de hash. Esta segunda no retorna una dirección, sino que su resultado es un desplazamiento. Este desplazamiento se suma a la dirección base obtenida con la primera función, generando así la nueva dirección donde se intentara ubicar el registro.
- Área de desborde por separado: aquí se distinguen dos tipos de nodos: aquellos direccionables por la función de hash y aquellos de resera, que solo podrán ser utilizados en caso de saturación pero que no son alcanzables por la función de hash.

- **Hash asistido por tabla**

Es una alternativa que utiliza espacio de direccionamiento estático y que, aun así, asegura acceder en un solo acceso a un registro de datos. Para poder ser implementada requiere que una de las propiedades no sea cumplida; necesita una estructura adicional.
El método utiliza tres funciones de hash:

- La primera retorna la dirección física del nodo donde el registro debería residir (F1H).
- La segunda retorna un desplazamiento, similar al método de doble dispersión (F2H).
- La tercera retorna una secuencia de bits que no pueden ser todos uno (F3H).

El método comienza con una tabla con tantas entradas como direcciones de nodos se tengan disponibles. Cada entrada tendrá sus bits en uno.

- El proceso de eliminación: a fines prácticos se estable el siguiente proceso para dar de baja un registro del archivo:
  - 1) se localiza el registro de acuerdo con el proceso de búsqueda definido anteriormente.
  - 2) si el paso anterior encontró la llave buscada, se reescribe el nodo en cuestión sin el elemento a borrar.

  Si el nodo ya había producido un overflow, la tabla en memoria contiene un valor correspondiente a la F3H de la llave que produjo saturación, en este caso el nuevo elemento debe cumplir que: F3H (nuevo elemento) < tabla (dirección del nodo).

- **Hash con espacio de direccionamiento dinámico**

Este hash dispersa las claves en función de las direcciones disponibles en cada momento, y la cantidad de direcciones puede crecer, a priori sin límites, en función de las necesidades de cada archivo particular.
Existen varias alternativas de implementación para hash con espacio dinámico: hash virtual, hash dinámico y hash extensible, entre otras.

- Has extensible: el principio del método consiste en comenzar a trabajar con un único nodo para almacenar registros e ir aumentando la cantidad de direcciones disponibles a medida que los nodos se completan.

  Este método no utiliza el concepto de DE ya que el espacio en disco utilizado aumenta o disminuye en función de la cantidad de registros de que dispone el archivo en cada momento.

  Para este método la función de hash retorna un string de bits, y estos determinan la cantidad máxima de direcciones a las que puede acceder el método.

  Para su implementación este método necesita de una estructura auxiliar, que es una tabla que se administra en memoria principal. Esta estructura contiene la dirección física de cada nodo.

  El tratamiento de dispersión con la política de hash extensible comienza con un solo nodo en disco, y una tabla que solamente contiene una dirección, la del único nodo disponible.

  Resumiendo, este método trabaja según las siguientes pautas:
  - Se utilizan solo los bits necesarios de acuerdo con cada instancia del archivo.
  - Los bits tomados forman la dirección del nodo que se utilizara.
  - Si se intenta insertar en un nodo lleno, deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nuevo; para ello se toma un bits más.
  - La tabla tendrá tantas entradas (direcciones de nodo) como 2 a la i, siendo i el número de bits actuales para el sistema.

  El proceso de búsqueda asegura encontrar cada registro en un solo acceso. Se calcula la secuencia de bits para la llave, se toman tantos bits de esa llave como indique el valor asociado a la tabla, y la dirección del nodo contenida en la celda respectiva debería contener el registro buscado. En caso de no encontrar el registro en dicho nodo, el elemento no forma parte del archivo de datos.