Apunte Teorico Practico Conceptos de Bases de Datos

Bautista Garcia

April 28, 2024

Contents

1. Archivos 4
    1.1.1. Almacenamiento 4
    1.1.2. Niveles de Vision 4
    1.1.3. Indexado en Pascal 4
    1.1.4. Tipos de Archivos 4
    1.2. Organizacion de un Archivo 5
        1.2.1. Archivos de longitud variable 5
        1.2.2. Claves o Llaves 6
        1.2.3. Accesos 6
    1.3. Cambios en archivos 7
        1.3.1. Estaticos 7
        1.3.2. Volatiles 7
    1.4. Bajas 7
        1.4.1. Fragmentacion 7
    1.5. Performance en Archivos 8
    1.6. Ordenamiento 8
        1.6.1. Sort Interno 8
        1.6.2. Seleccion por Reemplazo 8
        1.6.3. Indizacion 10
            1.6.3.1. Indice Primario 10
            1.6.3.2. Indice secundario 11
            1.6.3.3. Conclusion de Indices 12
2. Arboles 12
    2.1. Arboles Binarios 12
        2.1.1.1. Performance en Arboles Binarios 13
    2.2. Arboles Balanceados 14
        2.2.1. AVL 14
        2.2.2. Arboles Binarios Paginados 14
        2.2.3. Arboles Multicamino 14
        2.2.4. Arboles B (Balanceados) 15
            2.2.4.1. Declaracion B 15
            2.2.4.2. Construccion B 16
            2.2.4.3. Busqueda en B 16
            2.2.4.4. Insercion 16
            2.2.4.5. Performance en la busqueda 16
            2.2.4.6. Eliminacion en B 16
                2.2.4.6.1. Underflow 17
        2.2.5. Arboles B* 17
            2.2.5.1. Insercion 17
        2.2.6. Arboles B+ 18
            2.2.6.1. Prefijos Simples 19
        2.2.7. B vs B+ 19
3. Dispersion 20

3.1. Dispersion Estatica 20
3.1.1. Densidad de Empaquetamiento 20
3.1.2. Tratamiento de Overflow 21
3.1.2.1. Saturacion Progresiva 21
3.1.2.2. Saturacion con Area Separada 21
3.1.3. Hash asistido por tabla 21
3.1.4. Espacio no disponible 22
Bibliography 23
Index of Figures 24

Base de Datos: Informacion dispuesta de manera adecuada (acceso rapido) para su tratamiento por una computadora. Los datos estan **interrelacionados**.

SGBD: Sistema de software para gestionar las bases de datos.
- Controlan **concurrencia** de los datos, multiples usuarios editando la misma informacion.
- Control sobre los datos y acceso a ellos en **todo momento.**
- Seguridad en el acceso, declarar quienes tienen permitidos el acceso a los datos.
- Backup y control de integridad. Es decir, que estos datos sean correctos.

Se crearon los archivos detalles (propios de cada aplicacion) para que no esten todas intentado acceder al maestro en simultaneo (muy lento). Cada detalle replica la informacion relevante para su aplicacion. La actualizacion maestro-detalle sirve para actualizar los maestros en base a las modificaciones realizadas por cada aplicacion.

1. Archivos

Un archivo es una **estructura homogenea** (registros del mismo tipo), que se almacenan en **almacenamiento secundario** y sobre los que se realizan operaciones.
- **Clasificacion por datos:** Pueden existir archivos de **bytes, campos o registros**.
- **Clasificacion por acceso:** Pueden existir archivos de **acceso secuencial, secuencial indizado, directo.**

1.1.1. Almacenamiento

- Primario(RAM): Almacenamiento limitado, volatil y de alto costo.
- Secundario(Discos, Cintas): Alta capacidad, no volatil pero mucho mas lento.

1.1.2. Niveles de Vision

Fisico: Archivo existente en el almacenamiento, conocido por SO.
Logico: Archivo existente en el programa. Se realizan operaciones sin conocer su ubicacion fisica.

Buffers E/S: Cada vez que se leen datos, se trae el registro en cuestion y otros registros contiguos, siguiendo el principio de localidad. Es muy probable que en la proximidad temporal se deban leer esos, por lo que debemos optimizar los accesos a disco. Luego para escribir, se escriben en RAM y cuando se llenan o despues de $t_{\text{prestablecido}}$ se escriben todos juntos en disco. *Estas decisiones de escritura son realizadas por **procesador E/S**, el cual es independiente de la CPU.*

1.1.3. Indexado en Pascal

- Los archivos son de index 0.

1.1.4. Tipos de Archivos

Archivo Maestro: Se denomina maestro al archivo que **resume información** sobre un dominio específico.
Archivo Detalle: Se denomina detalle al archivo que contiene **nueva información generada** por las diferentes aplicaciones.

1.2. Organizacion de un Archivo

Longitud Fija: Se reserva espacio para un registro, donde en realidad no se va a utilizar todo el espacio reservado. Por lo que hay **desperdicio de espacio** y tardamos mas en procesarlo.

- Esto se puede solucionar usando caracteres que sirven como **delimitadores**, para indicar cuando finaliza un campo (delimitador no puede ser un dato valido).

1.2.1. Archivos de longitud variable

*En pascal es posible definiendo archivos sin tipo o de tipo **FILE**.*

```
program ejemplo_4_5;
Var
 empleados: file; {archivo sin tipo}
 nombre,apellido,direccion,documento: string;

Begin
  Assign(empleados,'empleados.txt');
  Rewrite(empleados,1);
  writeln('Ingrese el Apellido');
  readln(apellido);
  writeln('Ingrese el Nombre');
  readln(nombre);
  writeln('Ingrese direccion');
  readln(direccion);
  writeln('Ingrese el documento');
  readln(documento);

  while apellido<>'zzz' do
    Begin
       BlockWrite(empleados,apellido,length(apellido)+1);
       BlockWrite(empleados,'#',1);
       BlockWrite(empleados,nombre,length(nombre)+1);
       BlockWrite(empleados,'#',1);
       BlockWrite(empleados,direccion,length(direccion)+1);
       BlockWrite(empleados,'#',1);
       BlockWrite(empleados,documento,length(documento)+1);
       BlockWrite(empleados,'@',1);
       writeln('Ingrese el Apellido');
       readln(apellido);
       writeln('Ingrese el Nombre');
       readln(nombre);
       writeln('Ingrese direccion');
       readln(direccion);
       writeln('Ingrese el documento');
       readln(documento);
    end;
  close(empleados);
end.
```

Figure 1: Longitud variable usando delimitadores

Se utiliza el **BlockWrite** para escribir de a bytes. Se usa length + 1, porque los strings utilizan un byte para indicar dimension fisica. En el siguiente codigo se visualiza como leer esos archivos.

```pascal
program ejemplo_4_6;
Var
empleados: file; {archivo sin tipo}
  campo, buffer :string;
Begin
  Assign(empleados,'empleados.txt');
  reset(empleados,1);
  while not eof(empleados) do
    Begin
      BlockRead(empleados,buffer,1);
      while (buffer<>'@') and not eof(empleados)do
        begin
          while ((buffer<>'@') and
                (buffer<>'#') and not eof(empleados))do
            begin
              campo := campo + buffer ;
              BlockRead(empleados,buffer,1);
            end;
          writeln(campo);
        end;
      if not eof(empleados) then
        BlockRead(empleados,nombre,1);
    end;
  close(empleados);
end.
```

Se puede observar que se usa @ para indicar el fin del registro y # para indicar fin del campo.

1.2.2. Claves o Llaves

Facilitar el acceso a un registro dentro de un archivo.
Claves Primarias: Identifican de manera **univoca** a los registros. Existe la forma canonica y a partir de reglas hacen que dos registros distintos no puedan tener la misma clave primaria.
Claves secundarias: Pueden identificar a mas de un registro.

1.2.3. Accesos

Acceso Secuencial: Para acceder a un determinado registro, se necesita pasar por todos los registros anteriores.
Acceso Directo: Permite saltar directamente a un registro en particular. *Util cuando necesitamos pocos registros.* Esta forma es posible solo con registros de **longitud fija**.
Secuencial Indizada: Acceso secuencial pero determinado por una estructura logica adicional y no por el orden fisico de los datos.
NRR: Posicion del registro con respecto al inicio del archivo.

*En archivos de longitud variable **no es posible** aplicar el NRR ya que no se conoce el tamaño de antemano.*

1.3. Cambios en archivos

1.3.1. Estaticos

Pocos cambios, estos se pueden tratar **por lotes**. Es decir, recorriendo el archivo completo y realizando todos los cambios alli.

1.3.2. Volatiles

Alta frecuencia de cambios

Para la insercion de registros, la **longitud fija o variable** no presentan problemas ya que se agregan al final. El problema aparece con la **modificacion de los registros en longitud variable.**

Al modificarse los registros:
- RegNuevo > RegViejo: No entra en el espacio asignado.
- RegNuevo < RegViejo: Sobra lugar en el espacio asignado.

1.4. Bajas

La **baja logica** indica que el registro sigue estando presente en el archivo, pero esta marcado como eliminado. Mientras que la **baja fisica** lo borra del almacenamiento.

Compactacion (Baja Fisica) : Se copian en un archivo nuevo todos los registros que no fueron eliminados (con marca logica) en un nuevo archivo y luego se elimina el viejo y se renombra el nuevo. *Util para archivos estaticos* . En el momento de la compactacion:

$$\text{Espacio necesario} = 2 * \text{Tamaño archivo} \tag{1}$$

Marca logica: Se marcan los registros eliminados con una marca logica. Luego los programas modifican su funcionamiento para no tomar como validos a los registros marcados. Ademas se debe aprovechar su espacio para insertar registros nuevos.

Lista Invertida / Pila: Se usa el NRR en algun campo indicando el proximo registro libre. *Esta lista inicia en un registro cabecera NRR = 0.*

*Se puede observar que el metodo de Lista invertida solo puede ser usado en **Longitud Fija** ya que hace uso del NRR.*

Lista en Longitud Variable: Se utiliza un campo binario que explícitamente indica el enlace. Ademas cada registro indica en su inicio la cantidad de bytes que.ocupa

1.4.1. Fragmentacion

Interna: Ocurre en **registros de longitud fija** (strings que no se ocupan por completo) o en **longitud variable** al reemplazar un registro de tamaño mayor por uno de tamaño menor.

Externa: Ocurre cuando los fragmentos de espacio libre, son muy pequeños como para almacenar un nuevo registro.
- Unir fragmentos libres adyacentes para armar un fragmento mas grande.

- Regenerar / compactar el archivo.
- Minimizar la fragmentacion usando alguna tecnica al agregar:

Primer ajuste: Primer lugar libre que pueda alojar al registro (se asigna espacio completo), esta tecnica minimiza busquedas. *Fragmentacion interna*

Mejor Ajuste: Busqueda por toda la lista, buscando el espacio que mejor se ajuste al registro nuevo. *Fragmentacion interna*

Peor ajuste: Se busca el espacio mas grande y al registro solo se le asigna lo necesario. *Fragmentacion externa*

1.5. Performance en Archivos

La busqueda secuencial es $O(n)$. Una alternativa seria un acceso directo por NRR, $O(1)$, pero esta no es una alternativa que este siempre disponible.

Busqueda Binaria: Se necesita un archivo ordenado de longitud fija. El orden es $O(\log_2 N)$. *La desventaja es que se debe mantener el archivo ordenado.*

1.6. Ordenamiento

1.6.1. Sort Interno

Para archivos realmente grandes surge una nueva alternativa, que consiste en los siguientes pasos:

1. Dividir el archivo en particiones de igual tamaño, de modo tal que cada par- ticion quepa en memoria principal.
2. Transferir las particiones (de una a una) a memoria principal. Esto implica realizar lecturas secuenciales sobre memoria secundaria, pero sin ocasionar mayores desplazamientos.
3. Ordenar cada particion en memoria principal y reescribirlas ordenadas en me- moria secundaria. Tambien en este caso la escritura es secuencial (estos tres pasos anteriores se denominan sort interno).
4. Realizar el merge o fusion de las particiones, generando un nuevo archivo ordenado. Esto implica la lectura secuencial de cada particion nuevamente y la reescritura secuencial del archivo completo.

Se puede concluir que la ordenacion es una operacion de $O(N^2)$ evaluada en terminos de desplazamiento. k particiones con k desplazamientos para leer cada una.

1.6.2. Seleccion por Reemplazo

Claves en memoria secundaria:

34, 19, 25, 59, 15, 18, 8, 22, 68, 13, 6, 48, 17

↑ Principio de la cadena de entrada

Resto de la entrada	Mem. principal	Partición generada
34, 19, 25, 59, 15, 18, 8, 22, 68, 13	6 48 17	-
34, 19, 25, 59, 15, 18, 8, 22, 68	13 48 17	6
34, 19, 25, 59, 15, 18, 8, 22	68 48 17	13, 6
34, 19, 25, 59, 15, 18, 8	68 48 22	17, 13, 6
34, 19, 25, 59, 15, 18	68 48 (8)	22, 17, 13, 6
34, 19, 25, 59, 15	68 (18) (8)	48 ,22, 17, 13, 6
34, 19, 25, 59	(15) (18) (8)	68, 48, 22, 17, 13, 6

continúa >>>

Primera partición completa; se inicia la construcción de la siguiente:

Resto de la entrada	Mem. principal	Partición generada
34, 19, 25, 59	(15) (18) (8)	-
34, 19, 25	15 18 59	8
34, 19	25 18 59	15, 8
34	25 19 59	18, 15, 8
	25 34 59	19, 18, 15, 8
	- 34 59	25, 19, 18, 15, 8
	- - 59	34, 25, 19, 18, 15, 8
	- - -	59, 34, 25, 19, 18, 15, 8

El metodo de seleccion por reemplazo agarra la cantidad de registros que entran en memoria principal y en cada paso manda el menor de ellos a memoria secundaria. A su vez cada vez que manda alguno a mem secundaria hace ingresar a otro proveniento de mem secundaria. Si uno de los elementos que viene de memoria secundaria es menor al elemento que envie recien se lo duerme (marcado con parentesis), ya que mandar este elemento romperia con el ordenamiento de la particion. Esta secuencia se realiza hasta que todos los registros de m.principal quedan **no disponibles**, donde se crea una nueva particion y se activan todos los registros **no disponibles** y luego repetimos el mismo proceso hasta que termine el archivo.

Este metodo duplica el tamaño de las particiones, haciendo que el merge requiera menor cantidad de desplazamientos.

Seleccion Natural: La seleccion natural nos permite tener un buffer en memoria secundaria, donde se mandan los "Registros Dormidos" ∴ entran mas registros en **memoria principal** aumentando todavia mas el tamaño de las particiones.

- **Ventajas**:
  - ***Sort* interno:** produce particiones de igual tamaño. Algorítmica simple. No es un detalle menor que las particiones tengan igual tamaño; cualquier variante del método de *merge* es más eficiente si todos los archivos que unifica son de igual tamaño.
  - **Selección natural y selección 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 costoso en términos de *performance*.
  - **Selección natural y selección por reemplazo:** tienden a generar muchos registros no disponibles. Las particiones no quedan, necesariamente, de igual tamaño.

Figure 2: Comparativa entre sort interno y seleccion

Otra alternativa que mejora la performance es el merge multietapa. Es decir realizar merge de conjunto de particiones y luego otro merge de esos conjuntos. La performance mejora para estos casos.

1.6.3. Indizacion

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 permanezcan ordenadas**.

1.6.3.1. Indice Primario

Clave	Ref.	Dir. reg.	Registro de datos
AME2323	248	15	WAR | 23 | Early Morning | A-ha | A-ha
ARI2313	313	36	SON | 13 | Just a Like a Corner | Cock Robin | Cock Robin
BMG11	83	83	BMG | 11 | Selva | La Portuaria | La Portuaria
RCA1313	275	118	SON | 15 | Take on Me | A-ha | A-ha
SON13	36	161	VIR | 2310 | Land of Confusion | Genesis | Phil Collins
SON15	118	209	VIR | 1323 | Summer Moved On | A-ha | A-ha
VIR1323	209	248	AME | 2323 | Africa | Toto | Toto
VIR2310	161	275	RCA | 1313 | Leave of New York | REM | REM
WAR23	15	313	ARI | 2313 | Here Come The Rain Again | Eurythmics | Annie Lennox

La tabla de la imagen superior muestra a la izquierda la clave primaria, creada a partir de la informacion de campos particulares, y la referencia a la direccion donde se encuentran el resto de los datos. Estos registros de datos son de longitud variable (con marca de separado).

Creacion: Al crear el archivo de datos, tambien se debe crear el archivo de indices, ordenado por clave.

Altas en indice primario: La operación de alta de un nuevo registro al archivo de datos consiste simplemente en agregar dicho registro al final del archivo. A partir de esta operación, con el

NRR o la dirección del primer byte, según corresponda, más la clave primaria, se genera un nuevo registro de datos a insertar en forma ordenada en el archivo índice.

Modificaciones indice primario: Si los registros de datos son de longitud variable y este debe cambiar de lugar (debido a espacios), cambiamos la referencia en la tabla de indices (rapido si se encuentra en RAM).

Bajas indice primario: Las bajas son un borrado fisico o logico en la tabla de indices. No tiene sentido recuperar ese espacio ya que debemos mantener el orden.

1.6.3.2. Indice secundario

La idea principal de un indice secundario, es que multiples registros pueden estar asociados a un mismo indice secundario. Estos indices secundarios son mejores para la busqueda, ya que pueden ser un campo textual, ej: nombre de una banda. Luego existe una tabla secundaria que asocia a las claves secundarias sus claves primarias. Si ocurre una de las operaciones indicadas arriba, solo se modifica el indice primario. *Puede ocurrir que un mismo indice secundario referencie a varias claves primarias, por lo que es comun crear otro archivo indice secundario, con otro criterio de busqueda.*

Clave secundaria	Clave primaria
A-ha	SON15
A-ha	VIR1323
A-ha	WAR23
Cock Robin	SON13
Eurythmics	ARI2313
Genesis	VIR2310
La Portuaria	BMG11
REM	RCA1313
Toto	AME2323

Modificaciones: La unica modificacion que implica un cambio en el indice secundario es el cambio de la propia clave secundaria.

Bajas: Las bajas de un registro implican borrar las referencias a ese registro de todos los indices secundarios. Representado por su clave primaria. *Una alternativa seria borrar la referencia en el indice primario, pero quedarian datos inservibles ocupando espacio en el indice secundario.*

$$\text{Indice secundario} \rightarrow \text{Indice primario} \rightarrow \text{Ref no existente} \tag{2}$$

Alternativas: Una de las alternativas que surgen es tener una entrada por cada clave secundaria y una lista asociada a ese campo donde se almacenen todas las **claves primarias** asociadas a esa **clave secundaria**.

En el siguiente ejemplo, John Lennon representa la clave secundaria con 4 claves primarias asociadas

John Lennon	EMI13920	EMI18807	SON2626	UNI3795

INDICE DE COMPOSITORES		
NRR	Clave Secundaria	Link
0	Coldplay	2
1	Coti	4
2	Jarabe de Palo	9
3	John Lennon	8
4	Kevin Johansen	6
5	Radiohead	0
6	Regina Spektor	7

LISTA DE CLAVES PRIMARIAS		
NRR	Clave Primaria	Link
0	UNI2312	-1
1	SON2626	3
2	WAR23699	-1
3	UNI3795	-1
4	BMG38358	-1
5	EMI18807	1
6	SON75016	-1
7	BMG31809	-1
8	EMI13920	5
9	SON245	-1

1.6.3.3. Conclusion de Indices

Ventajas:Permiten **busquedas binarias** incluso si el archivo es de registros de longitud variable. Los archivos indices son mucho mas pequeños que los datos ∴ pueden estar en RAM. Desventajas: La desventaja principal es el procesamiento adicional necesario para mantener todas las estructuras adicionales.

2. Arboles

Conceptos generales de los arboles

- Grado de un nodo: nro de descendientes directos
- Grado del árbol: mayor grado de sus nodos
  - Árbol binario → grado 2
  - Árbol multicamino → grado N
  - Lista → árbol degenerado de grado 1
- Profundidad de un nodo: nro de predecesores
- Profundidad del árbol (altura): profundidad máxima de sus nodos

Figure 3: Conceptos

2.1. Arboles Binarios

Un árbol binario es una estructura de datos dinámica no lineal, en la cual cada nodo puede tener **a lo sumo dos hijos**. Nos es de utilidad que los elementos esten ordenados, ya que estamos eligiendo esta estructura para **minimizar accesos a disco**.

Luego el archivo **indice binario** se veria asi, donde cada entero positivo representa una posicion en el arbol y un entero negativo representa que no tiene hijos.

FIGURA 6.3

Raíz → 0

	Clave	Hijo Izq	Hijo Der
0	MM	1	2
1	GT	3	4
2	ST	8	11
3	BC	5	6
4	JF	7	14
5	AB	-1	-1
6	CD	-1	-1
7	HI	-1	-1

	Clave	Hijo Izq	Hijo Der
8	PR	9	10
9	OP	-1	-1
10	RX	-1	-1
11	UV	12	13
12	TR	-1	-1
13	ZR	-1	-1
14	KL	-1	-1

2.1.1.1. Performance en Arboles Binarios

Como se puede observar en el codigo anterior, en cada comparacion donde chequeamos si el nodo es mayor o menor a la raiz estamos descartando la mitad del archivo restante. Es por esto que la busqueda es $O(\log_2(N))$. Puede existir casos en donde al agregar nodos, esta busqueda deje de ser logaritmica:

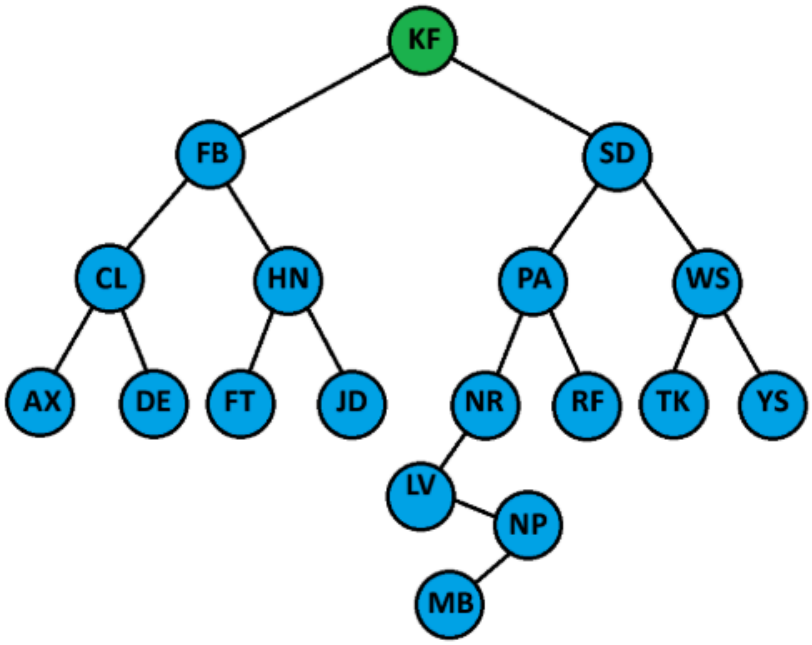


Figure 4: Se Requieren 7 accesos para acceder a MB

Este caso se termina pareciendo mas a una busqueda en una lista, aproximando el orden de las mismas a $O(n)$.

2.2. Arboles Balanceados

Un árbol balanceado (completamente) es un árbol en donde la altura de la trayectoria más corta hacia una hoja no difiere de la altura de la trayectoria más larga hacia una hoja. Para N claves:

$$\text{Desplazamientos} = \log_2(N+1) \tag{3}$$

2.2.1. AVL

La situación planteada anteriormente está resuelta. Los árboles balanceados en altura son árboles binarios cuya construcción se determina respetando un precepto muy simple: *la diferencia entre el camino más corto y el más largo entre un nodo terminal y la raíz no puede diferir en más que un determinado delta*. Para los **AVL** $\rightarrow \Delta = 1$.

*Con otras palabras la **altura de dos hojas distintas de un arbol** no pueden diferir en mas de 1.*

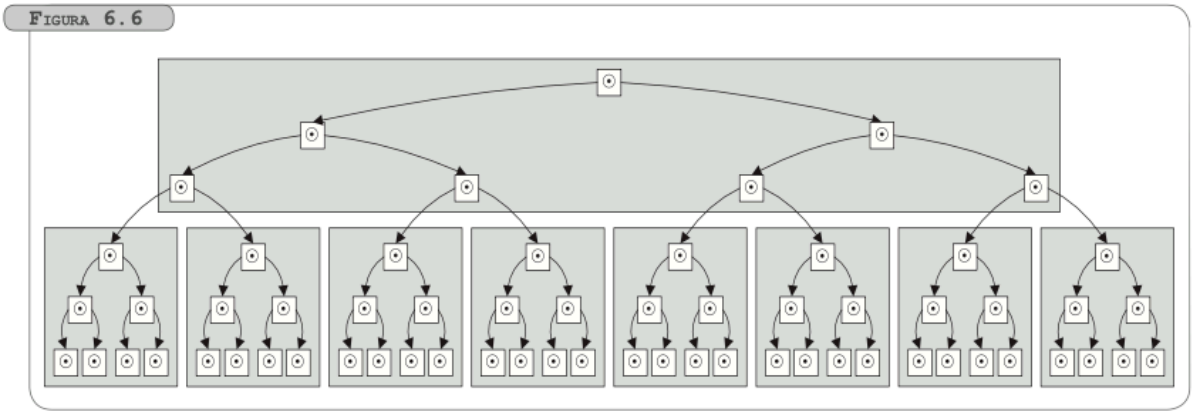
El principal problema de los AVL, termina siendo la cantidad de accesos a Disco. Para N claves:

$$\text{Desplazamientos} = 1.44 \log_2(N+2) \tag{4}$$

2.2.2. Arboles Binarios Paginados

Los arboles binarios paginados combinan los conceptos de: paginado usando **buffer** (transferencia de datos multiples en cada operacion) y la facilidad en busqueda, insercion y eliminacion de los datos de los arboles binarios.

FIGURA 6.6



En la figura 6.6 se puede observar, como se realizan transferencias o busquedas en paginas con K nodos. La performance resultante termina siendo $\log_{k+1}(N)$, siendo k la cantidad de nodos que entran en un buffer.

El problema de los ABP reside en su implementacion. Para que el uso de paginas tenga sentido, entonces **cada nodo debe ir a la pagina que le corresponde (como se ve en 6.6) y no a la primer pagina disponible**.

2.2.3. Arboles Multicamino

Se define a los **arboles multicamino** como una estructura de datos en la cual cada nodo puede contener k elementos y $k+1$ hijos.

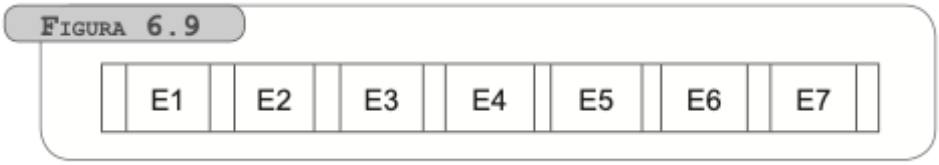


Figure 5:

Se define el **orden** de un multicamino como la maxima cantidad de descendientes posibles de un nodo.

La implementacion de los multicamino se puede realizar con:

- **Estructuras Estaticas**: como puede ser un arreglo (no nos sirve si numero de hijos es variable).
- **Estructuras Dinamicas**: Listas enlazadas de hijos.

El problema vuelve a ser el mismo que en los arboles binarios, si estos arboles no estan balanceados, entonces la busqueda deja de ser logaritmica.

2.2.4. Arboles B (Balanceados)

Los árboles B son árboles multicamino con una construcción especial que permite mantenerlos balanceados a bajo costo. A diferencia de los vistos anteriormente estos arboles se construyen **hacia arriba**.

Propiedades de los Arboles B (orden M):

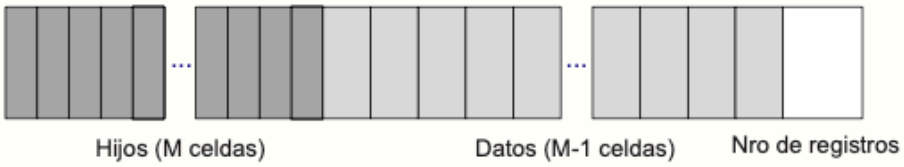
1. Cada nodo del árbol puede contener, como máximo, M descendientes y $M - 1$ elementos.
2. La raíz no posee descendientes directos o tiene al menos dos.
3. Un nodo con x descendientes directos contiene $x - 1$ elementos.
4. Los nodos terminales (hojas) tienen, como mínimo, $\left[\frac{M}{2}\right] - 1$ elementos, y como máximo, $M - 1$ elementos.
5. Los nodos que no son terminales ni raíz tienen, como mínimo, $\left[\frac{M}{2}\right]$ hijos.
6. Las hojas se encuentran todas en el mismo nivel.

2.2.4.1. Declaracion B

```pascal
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;
```

*Prestar atencion a la declaracion de ambos arreglos, se respeta la **propiedad 1**.*

FIGURA 7.1



Formato del nodo para archivo del índice árbol b

Formato gráfico del nodo del índice árbol B

2.2.4.2. Construccion B

2.2.4.3. Busqueda en B

$$H < 1 + \log_{[\frac{M}{2}]}\left(\frac{N+1}{2}\right) \tag{5}$$

Debemos acotar H, ya que en estos arboles los accesos necesarios, se analizan en base a la altura del arbol. Para comparar la eficiencia de este arbol contra los anteriores, debemos acotar H en funcion de N (claves en el arbol), para poder saber que altura tendra un arbol de orden M, con N claves.

Para llegar a esa cota, se usa el axioma que un arbol B tendra N + 1 punteros nulos en el ultimo nivel, si N es la cantidad de registros que contiene.

2.2.4.4. Insercion

*Orden **M** y altura **H***
Mejor Caso: Leemos hasta el nodo hoja y no se produce overflow $\therefore$ tenemos **H lecturas** y **1 escritura**.
Peor Caso: Overflow propagado hasta la raiz. $2H + 1$ escrituras y H lecturas. *De acuerdo a estudios realizados, a mayor orden, menor es el porcentaje de **overflow**.*

2.2.4.5. Performance en la busqueda

Las busquedas resultaran en 1 acceso si el dato se encuentra en la raiz y en H (altura) accesos si el dato se encuentra en una hoja. La cota encontrada para la altura es:

$$H < 1 + \log\left[\frac{M}{2}\right]\left(\frac{N+1}{2}\right) \tag{6}$$

2.2.4.6. Eliminacion en B

Para poder eliminar un elemento, este debe estar en un **nodo terminal**. Algorítmicamente se deberá intercambiar el elemento a borrar con *la mayor de sus claves menores (MN), o la menor de sus claves mayores (UW)* (Ejemplo Figure 12). Si al finalizar el proceso de baja se

sigue cumpliendo la **propiedad 4** de B, se da por finalizado el proceso. Pero si no se cumple se produce **UNDERFLOW**.

2.2.4.6.1. Underflow

Nodos Hermanos: Son aquellos nodos que comparten el mismo padre.
Nodos Hermanos Adyacentes: Son aquellos nodos que comparten el mismo padre y dependen de **punteros consecutivos** .

- Mejor caso (borra de un nodo terminal)
  - H lecturas
  - 1 escritura (el nodo sin el elemento borrado)

- Peor caso (concatenación lleva a decrementar el nivel del árbol)
  - 2H - 1 lecturas
  - H + 1 escrituras

2.2.5. Arboles B∗

Un **árbol *B**** es un árbol balanceado con las siguientes propiedades especiales:

- 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, [(2*M*-1)/3] -1 elementos, y como máximo, *M*-1 elementos.
- Los nodos que no son terminales ni raíz tienen, como mínimo, [(2*M*-1) / 3] descendientes.
- Todos los nodos terminales se encuentran al mismo nivel.

Figure 6:

Ventajas B*: Los árboles B* exigen que sus nodos se mantengan más llenos que los árboles B ∴ presentan **menor altura**. Como ya vimos las busquedas, inserciones y eliminaciones tienen su performance completamente vinculada a la **altura**

2.2.5.1. Insercion

- **Inserción**: redistribución / división. Tres políticas:
  - Derecha (de un lado):
    - Redistribuir con nodo adyacente hermano derecho (o izq. si es el último). Si esta lleno se realiza la división **usando ambos nodos**.
  - Derecha o Izquierda (de un lado u otro)
    - Si el nodo derecho está lleno, se intenta hacerlo con el izquierdo. Si ambos están llenos **se usa el nodo en overflow y su hermano derecho** para realizar la división.
  - Derecha e Izquierda (de un lado y otro)
    - Similar anterior, pero al realizar la división **se usan los tres nodos**, generando cuatro nodos que tendrán 3/4 parte llena.

Figure 7:

En el caso de overflow se aplican las politicas de arriba, usando redistribucion siempre que sea posible. Es decir, se intenta postergar la division lo mas que se pueda.

2.2.6. Arboles B+

Estos arboles son similares a los **Arboles B**, pero presentan una propiedad adicional:
- Los nodos terminales representan un conjunto de datos y son enlazados entre ellos.
- Los nodos interiores no contienen datos, solo punteros.

Estos conservan la propiedad de **accesos aleatorios rapidos** de los arboles B y agregan el **recorrido secuencial ordenado rapido**. Para esto el arbol B cuenta con:
- Indices: En la raíz y nodos interiores se duplican las claves necesarias para definir los caminos de búsqueda.
- Secuencia: Todas las claves se encuentran en los nodos hoja, que a su vez estan vinculados entre si para **mejorar acceso secuencial**.

Espacio: Esta claro que al comparar un arbol B contra un B+ con las mismas claves, el B+ ocupa mas espacio ya que tiene los indices duplicados.

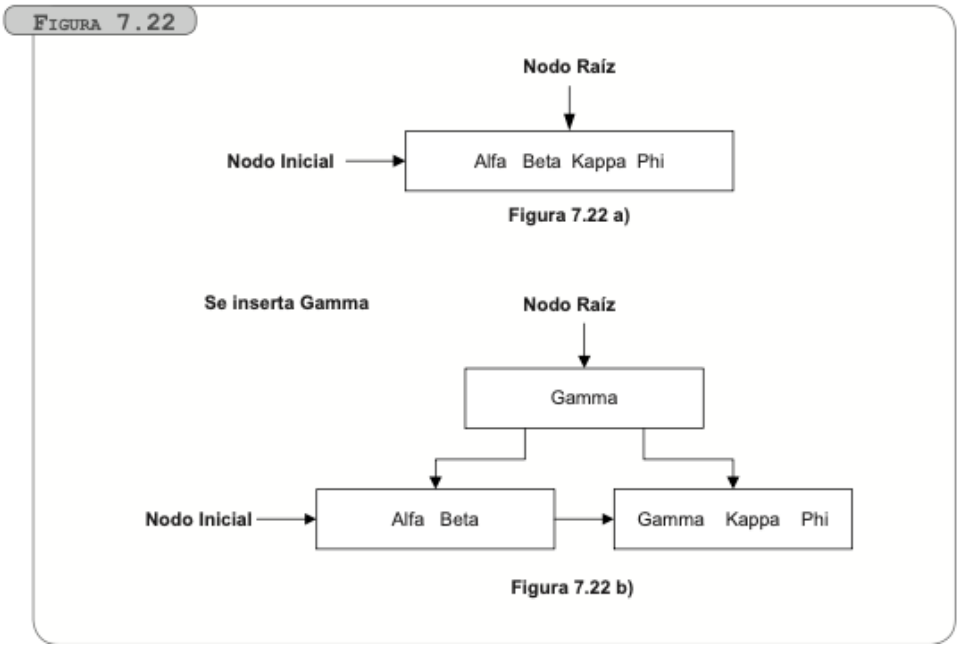


Figure 8: B+

Ahora en las **busquedas** no debemos deternos al encontrar el dato (puede ser un separador) sino que debemos seguir profundizando hasta llegar al nodo hoja (alli estan las claves).

Los elementos siempre se **insertan** en nodos terminales. Si se produce una saturación, el nodo se divide y se promociona una copia (aquí está la diferencia) del menor de los elementos mayores, hacia el nodo padre. Si el padre no tiene espacio para contenerlo, se dividirá nuevamente.

Para **borrar** un elemento de un árbol B+, siempre se borra de un nodo terminal, y si hubiese una copia de ese elemento en un nodo no ter- minal, esta copia se mantendría porque sigue actuando como sepa- rador.

2.2.6.1. Prefijos Simples

El arbol de prefijos simples trae la modificacion de que en los separadores no van a estar las claves completas, sino que solo el minimo prefijo que permita diferenciar claves mayores y menores (en los hijos).

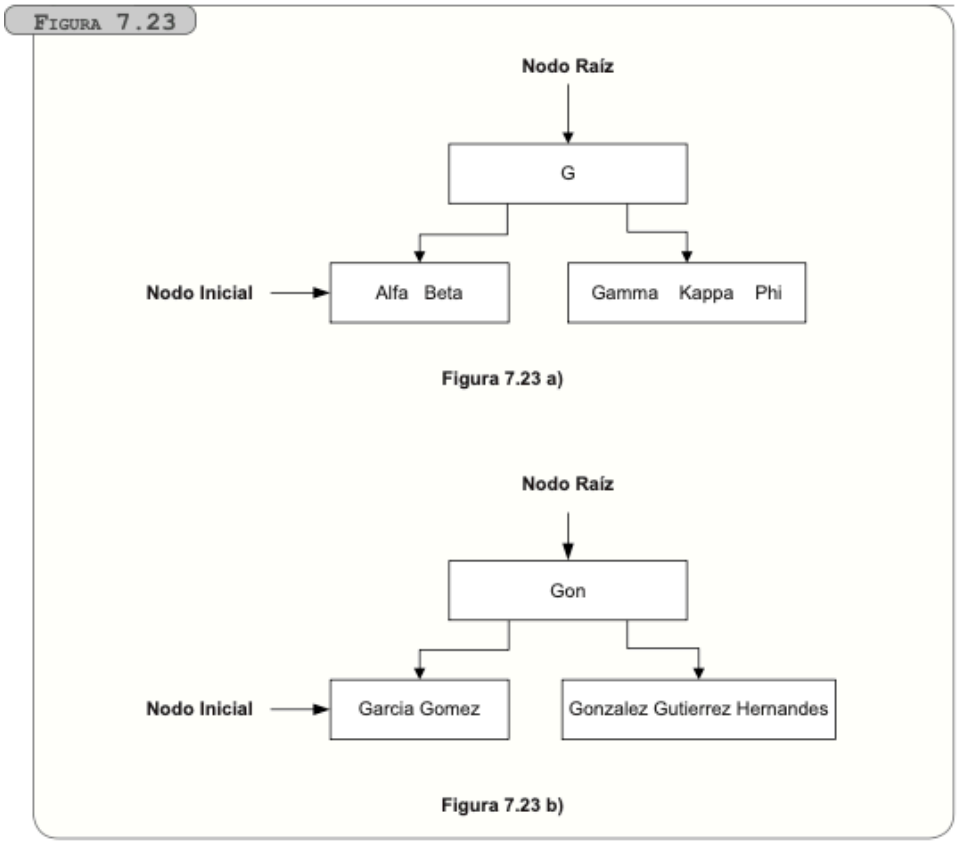


Figure 9:

2.2.7. B vs B+

	Árbol B	Árbol B+
Ubicación de datos	Nodos (cualquiera)	Nodos terminales
Tiempo de búsqueda	Equivalente (puede ser algo mejor)	Equivalente
Procesamiento secuencial ordenado	Lento (complejo)	Rápido (con punteros)
Inserción/Eliminación	Equivalente	Equivalente

Figure 10: Cuadro comparativo

3. Dispersion

Técnica que convierte la clave del registro en un número aleatorio, el que sirve después para determinar **donde se almacena el registro**.

3.1. Dispersion Estatica

Funcion de Dispersion: Tomar algunas claves del problema y simular el comportamiento con algunos métodos, y luego elegir el que mejor se comporte. Esta eleccion se va a basar en cual es el metodo que me retorna la mayor **uniformidad**.

Colision: Registros se asigna a una direccion ya ocupada. Claves asignadas a una misma direccion son denominadas **sinonimos**. *Para disminuir estas se debe:*

- Buscar el metodo mas uniforme.
- Usar mayor cantidad de direcciones, con la desventaja de **desperdiciar espacio.**
- Usar mas de un registro por direccion → **cubetas**. A mayor cantidad de registros, menores seran las colisiones, pero se hace mas lento recuperar un dato.

3.1.1. Densidad de Empaquetamiento

$$DE = \frac{\text{Registros del archivo}}{\text{Capacidad total de cubetas}} \tag{7}$$

- Un DE menor significa menor saturacion y un mayor desperdicio de espacio.

A partir de DE y las probabilidades independientes de que a una clave se le asigne cierta cubeta, llegamos a la siguiente probabilidad:

$$P(I) = \frac{\left(\frac{K}{N}\right)^I . e^{-\frac{K}{N}}}{I!} \tag{8}$$

*La probabilidad de que una cubeta reciba **I** elementos de los **K** disponibles.*

	TAMAÑO COMPARTIMENTO				
DE	1	2	5	10	100
10%	4.8	0.6	0.0	0.0	0.0
20 %	9.4	2.2	0.1	0.0	0.0
30 %	13.6	4.5	0.4	0.0	0.0
40 %	17.6	7.3	1.1	0.1	0.0
50 %	21.4	10.4	2.5	0.4	0.0
60 %	24.8	13.7	4.5	1.3	0.0
70 %	28.1	17.0	7.1	2.9	0.0
75 %	29.6	18.7	8.6	4.0	0.0
80 %	31.2	20.4	10.3	5.3	0.1
90 %	34.1	23.8	13.8	8.9	0.8
100 %	36.8	27.1	17.6	12.5	4.0

Figure 11: Relacion entre DE y registros por cubeta

3.1.2. Tratamiento de Overflow

3.1.2.1. Saturacion Progresiva

Se inserta en la direccion base o proximo lugar libre. Al buscarla se finaliza cuando se encuentra la clave, se encuentra espacio vacio o se vuelve a la direccion de arranque.

- Marca de borrado cuando proxima posicion no esta vacia, para indicar que la busqueda no debe finalizar alli.

3.1.2.2. Saturacion con Area Separada

El problema con el area separada reside en que si este area separada esta en otro cilindro del disco, entonces el proceso se torna lento.

3.1.3. Hash asistido por tabla

El hash asistido tiene la particularidad de **inserciones lentas** y **recuperaciones rapidas.** *Para cuando se recupera mas de lo que se inserta.*

Clave	F1H(Clave)	F2H(Clave)	F3H(Clave)
Alfa	50	3	0001
Beta	51	4	0011
Gamma	52	7	0101
Delta	50	3	0110
Epsilon	52	3	1000
Rho	51	5	0100
Pi	50	2	0010
Psi	50	9	1010

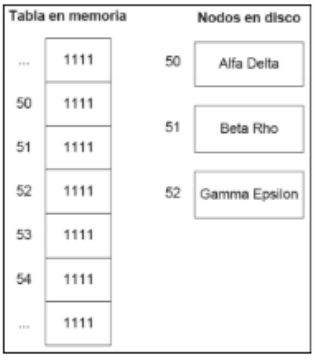


Figure 12: Hash tabla

Siempre y cuando hay espacio se usa la primer funcion de dispersion. En caso de haber saturacion (*insertar PI*) se comparan las secuencias de la **F3H**. Las secuencias menores se quedan en el nodo actual y la mayor de todas se desplaza **F2H** lugares. A su vez en la tabla de memoria se escribe la secuencia **F3H** de la que desplazamos. Esto sirve para que al recuperar el dato comparemos si la **F3H** que estoy buscando es mayor o menor a la de la tabla.

Eliminacion: En la mayoria de los casos el elemento a borrar se saca del nodo y se reescribe el nodo sin el elemento borrado. Hay que tener cuidado porque si se produjo saturacion en ese nodo y se intenta insertar, en la tabla habra un valor **F3H** que referencie al nodo desplazado en saturacion ∴ se debe seguir cumpliendo que:

$$\text{F3H}(\text{ nuevo elemento }) < \text{Tabla}(\text{ dirección del nodo}) \tag{9}$$

*Esto se logra reordenando **Clave a insertar** y **Clave desplazada anteriormente**.*

3.1.4. Espacio no disponible

Cuando se aproxima a una $DE > 75\%$ se empiezan a producir muchos **Overflow**. Es por eso que en ese momento **se le asigna mas espacio**. *La funcion de dispersion anterior ya no sirve*:

1. Obtener mas espacio.
2. Actualizar funcion de dispersion.
3. Redispersar el archivo.

El problema con esta tecnica es que mientras se redispersa el archivo, este no puede ser usado. Ademas los tiempos de redispersion suelen ser muy largos.

Bibliography

Index of Figures

Figure 1: Longitud variable usando delimitadores 5
Figure 2: Comparativa entre sort interno y seleccion 10
Figure 3: Conceptos 12
Figure 4: Se Requieren 7 accesos para acceder a MB 13
Figure 5: 15
Figure 6: 17
Figure 7: 18
Figure 8: B+ 18
Figure 9: 19
Figure 10: Cuadro comparativo 20
Figure 11: Relacion entre DE y registros por cubeta 21
Figure 12: Hash tabla 21