Fundamentos de Organización de Datos – Cursada 2025

Bibliografía

- Introducción a las Bases de Datos. Conceptos básicos (Bertone, Thomas).
- Estructuras de Archivos (Folk-Zoellick).
- Files & Databases: An introduction (Smith-Barnes).

1) Archivos

Persistencia de datos

Una **base de datos** es una colección de datos relacionados, específicamente de archivos diseñados para servir a múltiples aplicaciones. Estos datos representan hechos conocidos que pueden registrarse y que tienen un resultado implícito.

Propiedades implícitas de una BD

Una base de datos…

1. ...representa algunos aspectos del mundo real, a veces denominado Universo de Discurso.
2. ...es una colección coherente de datos con significados inherentes. Un conjunto aleatorio de datos no puede considerarse una BD. O sea los datos deben tener cierta lógica.
3. ...se diseña, construye y completa de datos para un propósito específico. Está destinada a un grupo de usuarios concretos y tiene algunas aplicaciones preconcebidas en las cuales están interesados los usuarios.
4. ...está sustentada físicamente en archivos en dispositivos de almacenamiento persistente de datos.

Definiciones de archivo

Un archivo es una colección de...

1. ...registros guardados en almacenamiento secundario.
2. ...datos almacenados en dispositivos secundarios de memoria.
3. ...registros que abarcan entidades con un aspecto común y originadas para algún propósito particular.

Hardware

1. Almacenamiento primario: Memoria RAM

@author Federico Dobal

2. Almacenamiento secundario (DR): platos, superficies, pistas, sectores, cilindros.

Organización de un archivo

1. En una **secuencia de bytes:** archivos de texto dónde se leen o recuperan caracteres sin formato previo. Una palabra se determina por un conjunto de caracteres finalizados en blanco (convención).
2. En **registros y campos**: un campo es la unidad más pequeña lógicamente significativa de un archivo; un registro es un conjunto de campos agrupados que definen un elemento del archivo.

Acceso de archivos

1. Secuencial físico: acceso a los registros uno tras otro y en el orden físico en el que están guardados (sin orden específico, simplemente en como llegaron).
2. Secuencial indizado (lógico): acceso a los registros de acuerdo al orden establecido por otra estructura o criterio.
3. Directo: se accede a un registro determinado sin necesidad de haber accedido a los predecesores.

Tipos de archivos

Se determinan de acuerdo a su forma de acceso:

1. Serie: cada registro es accesible solo luego de procesar su antecesor, simples de acceder (acceso secuencial físico).
2. Secuencial: los registros son accesibles en orden de alguna clave (acceso secuencial indizado/secuencial lógico).
3. Directo: se accede al registro deseado (acceso directo).

Operaciones con archivos

Los **archivos físicos** aparecen en el disco y están a cargo del SO. En cambio, los **archivos lógicos** se definen dentro del programa. El archivo se puede definir de dos formas:

- Como variable:

  ```
  Var archivo: file of Tipo_de_dato;
  ```
- Como tipo:

  ```
  Type archivo: file of Tipo_de_dato;
  Var arch: archivo;
  ```

Relación con el SO: se debe asignar la correspondencia entre el nombre físico y el lógico:

```
Assign(n_logico,n_fisico);
```

`Rewrite`: de solo escritura (creación):

```
Rewrite(n_logico);
```

`Reset:` lectura/escritura (apertura del archivo):

```
Reset(n_logico);
```

`Close:` cierre de archivo. Luego del ultimo dato del archivo se coloca la marca EOF (End Of file), es decir, al final del archivo:

```
Close(n_logico);
```

`Read:` leer un archivo sobre una variable del mismo tipo del archivo:

```
Read(n_logico, variable);
```

`Write:` escribo en el buffer:

```
Write(n_logico, variable);
```

Estas operaciones (read y write) leen y/o escriben sobre los buffers relacionados a los archivos No se realizan directamente sobre la memoria secundaria.

Bajas

Para eliminar registros de un archivo puedo hacer una baja **física** o una baja **lógica**.

- Baja física: se recupera espacio. Usualmente se copia a un archivo nuevo todos los registros excepto los que se quieran bajar. Se suele usar ante la necesidad de espacio y cada determinado tiempo.
- Baja lógica: se provee una forma de reconocer el registro una vez eliminado. Así puedo anular cualquier eliminación de forma fácil. Por ejemplo, colocar una marca especial en el registro eliminado.

Ejemplo 1: generar archivo

```
program Generar_Archivo;
type archivo = file of integer; {definición del tipo de dato para el
archivo }
var
    arc_logico: archivo; {variable que define el nombre lógico del archivo}
    nro: integer; {nro será utilizada para obtener la información de
    teclado}
    arc_fisico: string[12]; {utilizada para obtener el nombre físico del
archivo
    desde teclado}

begin
    write( 'Ingrese el nombre del archivo:' );
    read( arc_fisico ); { se obtiene el nombre del archivo}
    assign( arc_logico, arc_fisico );
    rewrite( arc_logico ); { se crea el archivo }
    read( nro ); { se obtiene de teclado el primer valor }
    while nro <> 0 do begin
        write( arc_logico, nro ); { se escribe en el archivo cada número }
        read( nro );
    end;
    close( arc_logico ); { se cierra el archivo abierto oportunamente con la
instrucción
```

```
        rewrite }
    end.
```

EOF: fin del archivo:

```
    EOF(n_logico); // true o false
```

FileSize: tamaño del archivo:

```
    FileSize(n_logico); // tamaño del archivo
```

FilePos: posición dentro del archivo:

```
    FilePos(n_logico); // posición (integer)
```

Seek: ir a una posición del archivo. Siempre se cuenta desde el comienzo del archivo (principio = 0):

```
    Seek(n_logico, posicion); // procedimiento
```

Ejemplo 2: agregar datos a un archivo existente

```
    procedure agregar (var emp: empleados);
    var
        e:registro;
    begin
        reset(emp);
        seek(emp, filesize(emp)); // se posiciona al final del archivo
        leer(e);
        while(e.nombre <> ' ') do begin
            write(emp,e);
            leer(e);
        end;
        close(emp);
    end;
```

Archivos maestro y detalle

La relación entre estos archivos debe ser compatible para poder actualizar el maestro a partir del detalle. En general, hay **un archivo maestro por cada N archivos detalle**.

Precondiciones

1. Ambos “tipos” de archivos están ordenados por el mismo criterio.
2. En el archivo detalle solo aparece la información que existe en el archivo maestro.
3. Cada estructura de datos del archivo maestro solo puede aparecer a lo sumo una vez en su correspondiente archivo detalle.

Ejemplo 1: actualizar maestro a partir de detalle

```
    program actualizar;
    type
        emp = record
            nombre:string[30];
            direccion:string[30];
```

```
        cht:integer;
    end;
    e_diario = record
        nombre:string[30];
        cht:integer;
    end;
    detalle = file of e_diario;
    maestro = file of emp;
var
    regm:emp; regd:e_diario;
    mae1:maestro; det1:detalle;
begin
    assign(mae1,'maestro');
    assign(det1,'detalle');
    reset(mae1);
    reset(det1);
    while(not eof(det1)) do begin
        read(mae1,regm);
        read(det1,regd);
        while(regm.nombre <> regd.nombre) do
            read(mae1,regm);
        regm.cht := regm.cht + regm.cht;
        seek(mae1, filepos(mae1)-1);
        write(mae1,regm);
    end;
end.
```

Ejemplo 2: actualizar maestro a partir de detalle con corte de control

Cada bloque del archivo maestro contiene a lo sumo una estructura de datos (un registro, por ejemplo) de sus correspondientes archivos detalles (**precondición 3**), por lo que no puede haber información repetida en el maestro. Para actualizar el maestro sin repetir información de los detalles se utilizarán cortes de control:

```
program actualizar;
const VALOR_ALTO=9999;
type
    str4 = string[4];
    prod = record
        cod:str4;
        descripcion:string[30];
        pu:real;
        cant:integer;
    end;
    v_prod = record
        cod:str4;
        ov:integer; {cantidad vendida}
    end;
    detalle = file of v_prod;
    maestro = file of prod:
var
    regm:prod; regd:v_prod;
    mael:maestro; det1:detalle;
    total:integer;
begin
    assign(mae1,'maestro');
```

```
    assign(det1,'detalle'); {proceso principal}
    reset(mae1);
    reset(det1);
    while (not EOF(det1)) do begin
        read(mae1,regm);
        read(det1,regd);
        while (regm.cod <> regd.cod) do
            read(mae1,regm);
        while(not EOF(det1) and (regm.cod = regd.cod)) do begin
            regm.cant := regm.cant – regd.cv;
            read(det1,regd); // (!)
        end;
        if(not EOF(det1))
            seek(det1, filepos(det1)-1);
        seek(mae1, filepos(mae1)-1);
        write(mae1,regm);
    end;
end.
```

(!) Nótese que si estoy ante el ultimo registro del archivo, al evaluar aquel **`while`** el **`read`** ya avanzó y me perdería este último registro. Para abstraerse de esta situación, podemos crear un procedimiento **`leer(archivoX,registroX)`**:

```
procedure leer(var archivo:detalle; var dato:v_prod);
begin
    if(not EOF(archivo)) then
        read(archivo,dato)
    else
        dato.cod := VALOR_ALTO;
end;
```

Y en el programa principal suprimimos **`not (EOF(det1):`**

```
...
while(regm.cod = regd.cod)) do begin
      regm.cant := regm.cant - regd.cv:
      leer(det1,regd); // (OK)
end;
...
```

Múltiples archivos detalle

Las precondiciones y la declaración de tipos son los mismos. El problema es que al tener N archivos detalles (cada uno ordenado), se hace más complejo encontrar el primer elemento entre todos (el menor) para actualizar su respectivo en el archivo maestro. Por ejemplo, no puedo actualizar un elemento y luego uno menor que este mismo. Para ello, se deben recorrer los N archivos detalle y encontrar el elemento mínimo para arrancar a actualizar su archivo maestro a partir de él.

```
begin // Programa principal
    assign(mae1, 'maestro'); assign(det1, 'detalle1');
    assign(det2, 'detalle2'); assign(det3, 'detalle3');
    reset(mae1); reset(det1); reset(det2); reset(det3);
    leer(det1, regd1); leer(det2, regd2); leer(det3, regd3);
    minimo(regd1, regd2, regd3, min);
    while (min.cod <> valoralto) do begin
```

```
                read(mae1,regm);
                while (regm.cod <> min.cod) do
                    read(mae1,regm);
                while (regm.cod = min.cod ) do begin
                    regm.cant:=regm.cant - min.cantvendida;
                    minimo(regd1, regd2, regd3, min);
                end;
                seek (mae1, filepos(mae1)-1);
                write(mae1,regm);
            end;
        end.
```

Merge

Implica fusionae o unir archivos con contenido similar.

Precondiciones

1. Todos los archivos detalle tienen la misma estructura.
2. Todos los archivos están ordenados por el mismo criterio.

1) Fusión sin repeticiones

Ejemplo: CADP inscribe a los alumnos que cursarán la materia en tres computadoras separadas. C/U de ellas genera un archivo con los datos personales de los estudiantes, luego son ordenados físicamente por otro proceso. El problema que tienen los JTP es genera un archivo maestro de la asignatura.

```
procedure minimo (var r1,r2,r3:alumno; var min:alumno);
begin
    if (r1.nombre<r2.nombre) and (r1.nombre<r3.nombre) then begin
        min := r1;
        leer(det1,r1)
    end else if (r2.nombre<r3.nombre) then begin
        min := r2;
        leer(det2,r2)
    end else begin
        min := r3;
        leer(det3,r3)
    end;
end;

begin
    assign (det1, 'det1');
    assign (det2, 'det2');
    assign (det3, 'det3');
    assign (maestro, 'maestro');
    rewrite (maestro);
    reset (det1); reset (det2); reset (det3);

    leer(det1, regd1); leer(det2, regd2); leer(det3, regd3);
    minimo(regd1, regd2, regd3, min);

    { se procesan los tres archivos }
    while (min.nombre <> valoralto) do
    begin
```

```
        write (maestro,min);
        minimo(regd1,regd2,regd3,min);
    end;
    close (maestro);
end.
```

2) Fusión con repeticiones

En este ejemplo, los vendedores de cierto comercio asientan las ventas realizadas. Sin embargo, el archivo maestro deberá resumir los archivos detalles en código de vendedor y monto total entre todas las ventas. Es decir, los detalles y el maestro tienen diferente estructura (distintos registros).

```
begin
    assign (det1, 'det1');
    assign (det2, 'det2');
    assign (det3, 'det3');
    assign (mae1, 'maestro');
    reset (det1); reset (det2); reset (det3);
    rewrite (mae1);

    leer (det1, regd1); leer (det2, regd2); leer (det3, regd3);
    minimo (regd1, regd2, regd3, min);
    { se procesan los archivos de detalles }
    while (min.cod <> valoralto) do begin
        {se asignan valores para registro del archivo maestro}
        regm.cod := min.cod;
        regm.total := 0;
        {se procesan todos los registros de un mismo vendedor}
        while (regm.cod = min.cod ) do begin
            regm.total := regm.total+ min.montoVenta;
            minimo (regd1, regd2, regd3, min);
        end;
        { se guarda en el archivo maestro}
        write(mae1, regm);
    end;
end.
```

Fusión de N archivos con repeticiones

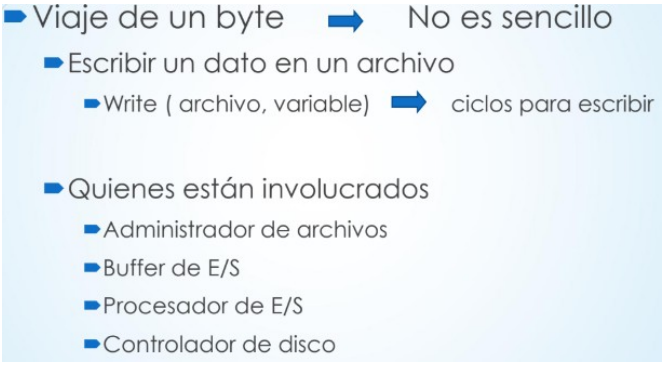
```
    program union_de_N_archivos;
    const valoralto = '9999';
    type
        vendedor = record
            cod: string[4];
            producto: string[10];
            montoVenta: real;
        end;
        ventas = record
            cod: string[4];
            total: real;
        end;
        maestro = file of ventas;
        arc_detalle = array[1..100] of file of vendedor; // asumir como válido
        reg_detalle = array[1..100] of vendedor; // ídem
    var
        min: vendedor;
        deta: arc_detalle;
```

```
    reg_det: reg_detalle;
    mae1: maestro;
    regm: ventas;
    i,n: integer;
procedure leer (var archivo:detalle; var dato:vendedor);
begin
    if (not eof( archivo )) then
        read (archivo, dato)
    else
        dato.cod := valoralto;
end;
procedure minimo (var reg_det: reg_detalle; var min:vendedor; var
deta:arc_detalle);
var i: integer;
begin
    { busco el mínimo elemento del
    vector reg_det en el campo cod,
    supongamos que es el índice del for es i }
    min = reg_det[i];
    leer(deta[i], reg_det[i]);
end;

begin
    Read(n)
    for i:= 1 to n do begin
        assign (deta[i], 'det'+i);
        { ojo lo anterior es incompatible en tipos}
        reset( deta[i] );
        leer( deta[i], reg_det[i] );
    end;
    assign (mae1, 'maestro'); rewrite (mae1);
    minimo (reg_det, min, deta);
    { se procesan los archivos de detalles }
    while (min.cod <> valoralto) do begin
        {se asignan valores para registro del archivo maestro}
        regm.cod := min.cod;
        regm.total := 0;

        {se procesan todos los registros de un mismo vendedor}
        while (regm.cod = min.cod ) do begin
        regm.total := regm.total+ min.montoVenta;
        minimo (regd1, regd2, regd3, min);
        end;

        { se guarda en el archivo maestro}
        write(mae1, regm);
    end;
end.
```

Organización de datos

Comparado a la memoria RAM, el almacenamiento secundario es más “lento” pero menos costoso para la misma cantidad de bytes. De esta manera, a la hora de recuperar información de una unidad de almacenamiento secundario, se espera que esta venga de una vez o en pocos intentos.

Un **archivo** es una colección de bytes que representa información (ver Definiciones de archivo).



Administrador de archivos

Es un conjunto de programas del S.O. (capas de procedimientos) que tratan aspectos relacionados con archivos y dispositivos de E/S.

- En Capas Superiores: aspectos lógicos de datos (tabla)
  - Establecer si las características del archivo son compatibles con la operación deseada (1).
- En Capas Inferiores: aspectos físicos (FAT)
  - Determinar donde se guarda el dato (cilíndro, superficie, sector) (2).
  - Si el sector está ubicado en RAM se utiliza, caso contrario debe traerse previamente. (3).

Los **buffers de E/S** agilizan la entrada y salida de datos. Manejarlos implica trabajar con grandes grupos de datos en RAM, para reducir el acceso a almacenamiento secundario.

Un **procesador de E/S** es un dispositivo utilizado para la transmisión desde o hacia almacenamiento externo. Independiente de la CPU (3).

Un **controlador de disco** se encarga de controlar las operaciones del disco. Requiere:

- Colocarse en la pista.
- Colocarse en el sector.
- Transferir a disco.

Capas del protocolo de transmisión de un byte

1. El Programa pide al S.O. escribir el contenido de una variable en un archivo.
2. El S.O. transfiere el trabajo al Administrador de archivos.
3. El Adm. busca el archivo en su tabla de archivos y verifica las características.
4. El Adm. obtiene de la FAT la ubicación física del sector del archivo donde se guardará el byte.
5. El Adm se asegura que el sector del archivo está en un buffer y graba el dato donde va dentro del sector en el buffer.

6. El Adm. de archivos da instruccciones al procesador de E/S (donde está el byte en RAM y en que parte del disco deberá almacenarse).
7. El procesador de E/S encuentra el momento para transmitir el dato a disco, la CPU se libera.
8. El procesador de E/S envía el dato al controlador de disco (con la dirección de escritura).
9. El controlador prepara la escritura y transfiere el dato bit por bit en la superficie del disco.

Archivos como secuencia de bytes

- No se puede determinar fácilmente comienzo y fin de cada dato.
- **Ejemplo: archivos de texto.**

Archivos estructurados

Están dotados de registros y campos, con longitud fija o variable.

Los **campos** son la unidad lógica significativa más pequeña de un archivo, separa la información. Identidad de campos:

- Longitud predecible o fija: se desperdicia espacio ya que no todos los datos ocupan la misma cantidad de espacio.
- Longitud indicada: se indica al principio de cada campo.
- Delimitador al final de cada campo: carácter especial no usado como dato.

Los **registros** se caracterizan de la siguiente manera:

- Longitud predecible o fija: en cantidad de bytes o cantidad de campos (estos pueden ser de longitud fija o variable también).
- Longitud variable:
  - Indicador de longitud puesto al comienzo.
  - Un segundo archivo que mantiene la información de la dirección del byte de inicio de cada registro.
  - Un delimitador que indica el final.

Claves

Las claves permiten la identificación de los registros. Además deben permitir generar orden en el archivo por ese criterio.

- Clave **primaria/unívoca:** identifican un solo elemento en particular.
- Clave **secundaria:** generalmente no identifican un único elemento.

Performance en acceso secuencial

- Mejor caso: leer 1 registro.
- Peor caso: leer N registros.

- Promedio: N/2 comparaciones (lecturas).
- De O(N).

Performance en acceso directo

- Mejor y peor caso: leer 1 registro.
- De O(1).
- Se requiere saber donde acceder.
- Aplicable a registros de longitud fija.

El acceso directo es preferible sólo cuando se necesitan pocos registros específicos, pero este método NO siempre es el más apropiado para la extracción de información.

Por ejemplo: generar cheques de pago a partir de un archivo de registros de empleados. Como todos los registros se deben procesar ➔ es más rápido y sencillo leer registro a registro desde el principio hasta el final, y NO calcular la posición en cada caso para acceder directamente.

Número relativo de registro (NRR)

- Indica la posición relativa con respecto al principio del archivo.
- Solo aplicable con registros de longitud fija.
- Por ejemplo: NRR 546 y longitud de cada registro 128 bytes ➔ distancia en bytes= 546 * 128 = 69.888.

2) Árboles

Árbol binario

Es una estructura de datos donde cada nodo tiene a lo sumo dos hijos. En memoria se organizan por un índice 0 <= i <= n-1 dónde n es la cantidad de nodos del árbol. Cada referencia a dónde apunta el índice contiene la clave del nodo, y las referencias a sus hijos izq. y der:

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

Árbol AVL

Árbol balanceado: un árbol está balanceado cuando la altura de la trayectoria más corta hacia una hoja no difiere de la altura de la trayectoria más grande.

Un **árbol AVL** es un árbol binario balanceado en altura (BA(1)) en el que las inserciones y eliminaciones se efectúan con un mínimo de accesos. 1 es el nivel de balance, es decir, que la mayor diferencia que puede haber entre las alturas de cualesquiera nodos diferentes **es 1**.

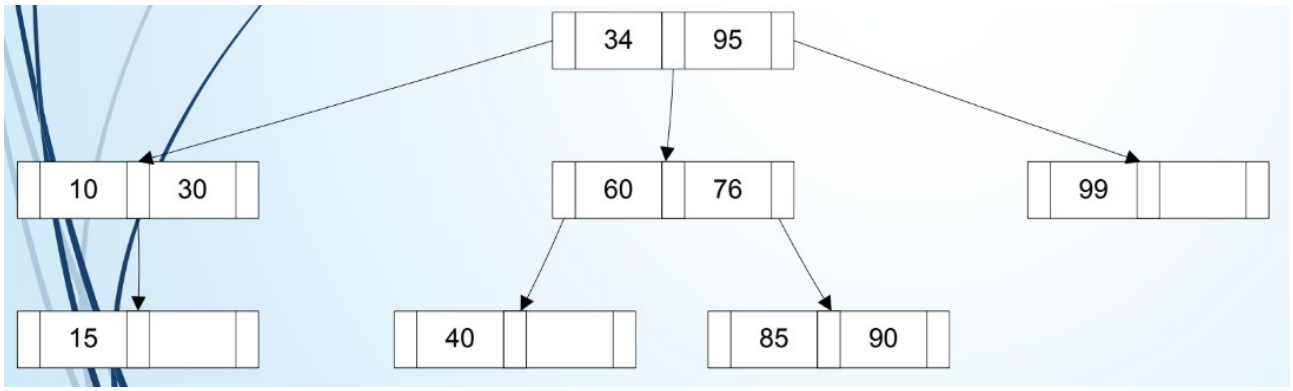
En la performance del peor caso posible:

Binario: → Búsqueda: log2(N+1)

AVL: → Búsqueda: 1.44 * log2(N+2)

Árbol multicamino

Un **árbol multicamino** es una estructura de datos en la cual cada nodo puede contener k elementos y k+1 hijos. El **orden** de un árbol multicamino es la máxima cantidad de descendientes posibles de un nodo.



Árboles balanceados

Árbol B

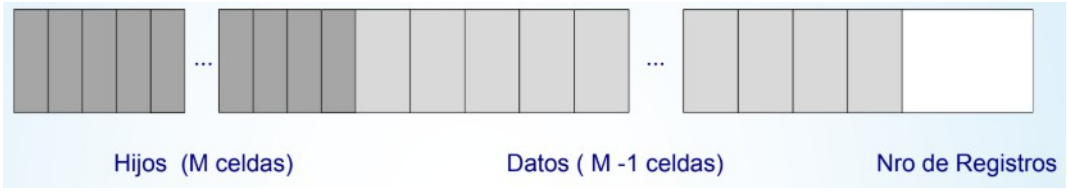
Son árboles multicamino con una construcción especial en forma ascendente que permite mantenerlo balanceado a bajo costo.

Propiedades (orden = M):

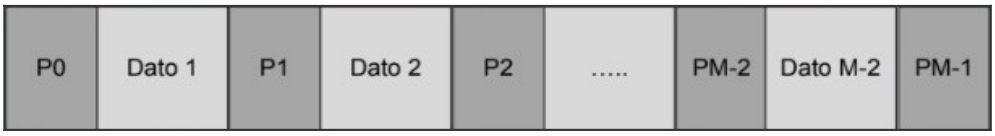
- Ningún nodo tiene más de M hijos
- C/nodo (menos raíz y los terminales) tienen como mínimo [M/2] hijos
- La raíz tiene como mínimo 2 hijos (o sino ninguno)
- Todos los nodos terminales a igual nivel
- Nodos no terminales con K hijos contienen K-1 registros. Los nodos terminales tienen:
- Mínimo [M/2] –1 registros
- Máximo M – 1 registros

Cada nodo de un árbol B se puede representar de la siguiente manera:

- Formato del nodo:

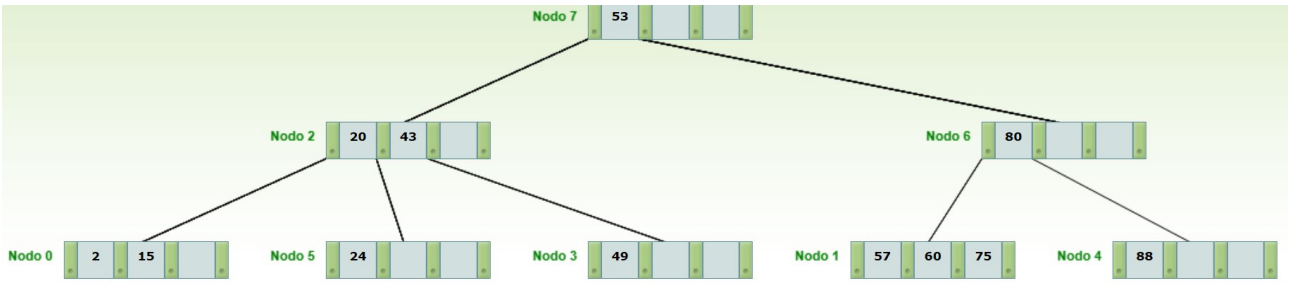


- Formato gráfico del nodo:



Ejemplo:

- Árbol resultante de agregar 12 claves (43, 2, 53, 88, 75, 80, 15, 49, 60, 20, 57, 24):



- Historial de operaciones (de más reciente a más vieja:

  - División de la raíz, aumento de altura
  - Desborde del nodo 0. División y promoción del elemento 20
  - Inserción elemento 57 en el nodo 1
  - Inserción elemento 20 en el nodo 0
  - Desborde del nodo 1. División y promoción del elemento 80
  - Desborde del nodo 0. División y promoción del elemento 43
  - Inserción elemento 15 en el nodo 0
  - Inserción elemento 80 en el nodo 1
  - Inserción elemento 75 en el nodo 1
  - División de la raíz, aumento de altura
  - Inserción elemento 53 en el nodo 0
  - Inserción elemento 2 en el nodo 0
  - Inserción elemento 43 en el nodo 0

- Se creo el árbol B con un orden 4