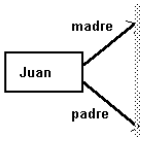
4. ESTRUCTURAS DE INFORMACIÓN DINÁMICAS

4.1 TIPOS DE DATOS RECURSIVOS (RECORD..END): son estructuras de variables que van cambiando durante el proceso de cálculo. Los valores del tipo de datos recursivos contendrán una o más solicitudes de él, esto es, una expresión *(hijo)* consta de un término *(padre)*, seguido de un operador *(hijo de...(flechas))* y después un término *(madre)*. Un término es una variables o bien una expresión entre paréntesis. Ejemplo: Padres de..



NOTA IMPORTANTE: los ejemplos de la página 184 y 185 NO FUNCIONAN en Modula

4.2 APUNTADORES (POINTER TO...). Una propiedad de las **estructuras recursivas** es su capacidad de cambiar de tamaño. Por consiguiente, es imposible asignar una cantidad fija de espacio memoria a una estructura definida recursivamente y, en consecuencia, un compilador no puede asociar direcciones específicas a las componentes de estas variables.

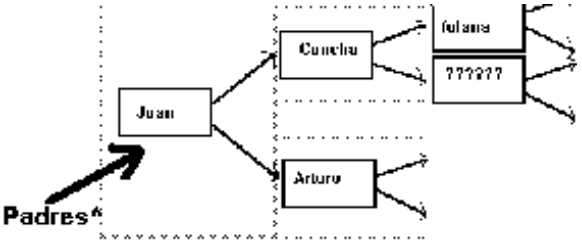
La técnica asociada a estas estructuras de datos recursivos se llama **asignación dinámica,** consiste en que el compilador asigna una cantidad fija para contener la dirección de la componente asignada dinámicamente y NO de la componente misma, esto es, **guarda el lugar donde está almacenada** en lugar de guardar el valor mismo.

Todos los apuntadores tienen el valor común NULO o NIL, sirve para apuntar a NINGÚN SITIO, este valor acaba una posible lista o posibilita una cardinalidad finita.

Se puede construir una estructura variante mediante CASE, posibilitando la modificación de los datos

```
MODULE DatosRecursivos;
  TYPE T= POINTER TO T0;
    TYPE T0 =   RECORD
                  v1:t1
                  ……
                  CASE b:tvOF
                     b1:|
                     b2:|
                  END;
                END;
```

La variable que apunta al lugar donde está guardada otra variable, se llama **apuntador**. La notación será: TYPE T = **POINTER TO** T0; significa que los valores de T son apuntadores de datos de T0. Del ejemplo anterior: Padres e...



- La estructura de datos será: los valores de Padres apuntan a los datos **Hijo**, y como estos últimos datos contienen a su vez el apuntador (y sólo el apuntador, no los valores siguientes) Padres, el tipo de datos **Hijo** será recursivo, pues a su vez apuntan a otro dato.

```
MODULE DatosRecursivos;
   TYPE Padres= POINTER TO Hijo;
   TYPE Hijo =   RECORD
            nombre:ARRAY[0..10] OF CHAR;
            padre,madre:Padres;
         END;
END DatosRecursivos.
```

- El valor NIL estará en los apuntadores contenidos en *Arturo* y *fulana* porque son últimos datos.

- Se puede cambiar la estructura de datos, por otra más flexible mediante una variante pues el valor *Concha* no es el último y además el padre es desconocido, ésta es:

```
MODULE DatosRecursivos;
   TYPE Padres1= POINTER TO Hijo1;
      TYPE Hijo1 =   RECORD
                  nombre:ARRAY[0..10] OF CHAR;
                  madre:Padres1;
                  CASE HijoPuta:BOOLEAN OF
                     TRUE: | (* padre desconocido, por tanto, ¡no tendrá!*)
                     FALSE:padre:Padres1|
                  END;
               END;
END DatosRecursivos.
```

4.3 LISTAS LINEALES, consisten en datos vinculados uno a uno; necesitando, por cada dato, sólo un apuntador para el sucesor.

4.3.1 OPERACIONES BÁSICAS

(VER APUNTES TEMA 3)

4.3.2 LISTAS ORDENADAS Y REORGANIZACIÓN DE LISTAS

- búsqueda en una lista desordenada con el algoritmo de búsqueda directa

- "" " " " " " " " " " " " " " " " " con centinela

- búsqueda en una lista ordenada con reordemanimento parcial

4.3.3 UNA APLICACIÓN: CLASIFICACIÓN TOPOLÓGICA (GRAFOS), es un proceso de clasificación de elementos mediante un ordenamiento PARCIAL:

- definición:

- propiedades :
  - transitividad: si "x precede a y" e "y precede a z" entonces "x precede a z"
  - asimetría: si "x precede a y", entonces " y NO precede a z"
  - irreflexibilidad: "z NO precede a z"

  Por tanto, en un ordenamiento parcial no puede existir ciclos o repeticiones

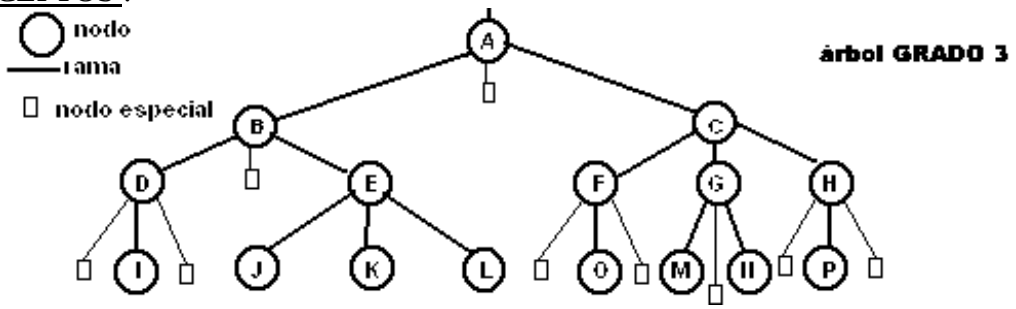
- todos los elementos tienen llave de identificación, conjunto de sucesores (mediante una lista) y un conteo de su predecesores

4.4 ESTRUCTURAS DE ÁRBOL

4.4.1 DEFINICION :

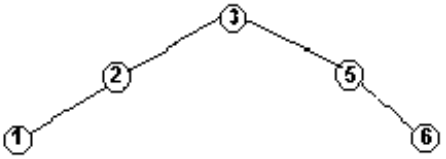
- una estructura de árbol con tipo base T es
  - la estructura vacía
  - un nodo de tipo T con un número finito de estructura de árbol disjuntas asociadas de tipo T, llamadas subárboles
- gráficamente se representa al revés, esto es, el tronco hacia arriba y las ramas hacia abajo

CONCEPTOS :



B es decendiente de A; A es el ancestro de B; A es la raíz; B,I,J,k,l,…P son nodos terminales; A,B,B,E,..H son nodos interiores; A tiene nivel 0 y D tiene nivel 2; G tiene grado 2 (sin contar con la rama del nodo especial) pues tiene dos ramas M y H; la longitud de trayectoria interna es 36 y estas son: A,AB,ABD,ABDI,ABE,ABEJ ,ABEK, ABEL,…. ; la longitud de trayectoria externa es 120 y estas son: A,AB,ABD,ABD,ABDI,ABDI,ABDI,,

- **degenerado**, es una lista o secuencia pues se considera un árbol en el que cada árbol tienen como máximo un subárbol



- **ordenado**: cuando las ramas de cada nodo tienen un orden preestablecido, pudiendo ser: preorden, en orden, postorden (NOTA: se amplía más adelante)
- **raíz** del árbol es la parte superior o cabeza del árbol, estando éste representado cabeza abajo.
- **descendiente** de *x*, es un nodo *y* que está directamente por debajo del nodo *x*
- **ancestro** de *y*, es un nodo *x* que está directamente por arriba del nodo *y*
- **nivel** del nodo *x*, es la distancia entre dos nodos que están en una misma rama. La raíz tiene un nivel 0.
- **profundidad** o **altura** es nivel máximo de cualquier elemento del árbol. La raíz tiene una profundidad de 0
- **nodo terminal** u **hoja**, si no tiene descendientes (apuntadores = NIL).
- **nodo interior**, si tiene descendientes, esto es, no es terminal
- **grado** de un nodo, es el número de descendientes (directos) o ramas
- **longitud de trayectoria** de *x*, al número de ramas que tienen que ser recorridas desde el nodo de la raíz hasta un nodo *x*. La raíz tiene trayectoria 0.
- **longitud de trayectoria interna**, la suma de las longitudes de trayectoria de

todos sus nodos

- longitud de trayectoria externa, suma de las longitudes de trayectoria en todos sus **nodos especiales** (= ampliación de las ramas de un nodo si éstas no alcanzan el fondo del árbol, resultando todos los nodos del mismo grado)

- caso particular: árboles de grado 2 o binarios

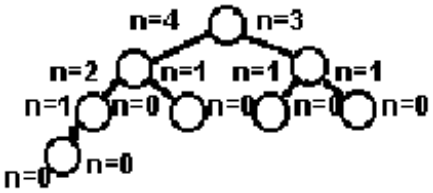
ToniWeb(http://www.jazzcyber.com/jazz5/toniweb/); E-mail: toniweb@jazzcyber.com

4.4.2 OPERACIONES BÁSICAS CON ÁRBOLES BINARIOS:

- Definición:

```
TYPE   Ptr    = POINTER TO Node;
       Node   = RECORD
                   op:CHAR;
                   left,right:Ptr;
                END;
```

- **ARBOL PERFECTAMENTE BALANCEADO** (ó Regla de igual distribución): en cada nodo los nº de nodos (n en el dibujo) en sus subárboles **izquierdo y derecho difieren a lo sumo en UNO**



- utilizar un nodo raíz
- generar el subárbol izquierda nL= n DIV 2 nodos
- generar el subárbol derecha nR= n - nL -1 nodos

```
PROCEDURE tree(n:INTEGER):Ptr; (* ARBOL BALANCEADO SIN ORDEN ENTRE ELEMENTOS *)
VAR newNode:Ptr;
   x:CHAR;
   nL,nR:INTEGER;
BEGIN
IF n = 0 THEN newNode:= NIL;
ELSE
   nL := n DIV 2;
   nR := n - nL - 1;
   WriteString("dato:");Read(x);WriteString(x);WriteLn;
   NEW (newNode);
   WITH newNode^ DO
      key:= x;
      left:= tree(nL);
      right:= tree(nR);
   END;
END;
RETURN newNode;
END tree;
```

- Ordenamientos en la forma de visitar los nodos y la operación entre ellos:

- **PRE**orden: **Operación**, rama Izquierda, rama Derecha

```
PROCEDURE PREorden(t:Ptr);
BEGIN
IF t # NIL THEN
   Write(t^.key);PREorden(t^.left);PREorden(t^.right);
END;
END PREorden;
```

- **EN** orden: rama Izquierda **Operación**, rama Derecha

```
PROCEDURE ENorden(t:Ptr);
BEGIN
 IF t # NIL THEN
```

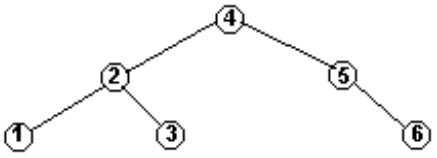
```
    ENorden(t^.left);Write(t^.key);ENorden(t^.right);
END;
END ENorden;
```

- **POST**orden: rama Izquierda, rama Derecha, **Operación**

```
PROCEDURE POSTorden(t:Ptr);
BEGIN
IF t # NIL THEN
        POSTorden(t^.left);POSTorden(t^.right);Write(t^.key);
      END;
END POSTorden;
```

ToniWeb(http://www.jazzcyber.com/jazz5/toniweb/); E-mail: toniweb@jazzcyber.com

4.4.3 BÚSQUEDA E INSERCIÓN EN ÁRBOL:



árbol de búsqueda, es un árbol binario **organizado** de forma que dada un **nodo cualquiera t_i** todas las llaves :

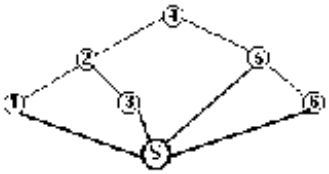
- en el subárbol **izquierdo** sean **menores** que t_i
- en el subárbol **derecho** sean **mayores** que t_i

Por tanto, un árbol de búsqueda de *n* elementos tendrá una altura máxima de *ln n*, realizando *ln n* comparaciones entre sus *n* elementos en el mejor de los casos; y en el peor, será un árbol degenerado o incluso parecido a un ARRAY

- búsqueda SIN centinela:

```
PROCEDURE locateSIN(x:CHAR;t:Ptr):Ptr;
BEGIN
WHILE (t # NIL) & (t^.key # x) DO
     IF t^.key < x THEN t:= t^.right ELSE t:= t^.left END;
END;
END locateSIN;
```

- búsqueda CON centinela:



el centinela se encuentra compartido entre todos los elementos y, por tanto, la estructura resultante no será un árbol. Esta búsqueda es más r

```
PROCEDURE
locateCON(x:CHAR;t:Ptr):Ptr; (*árbol búsqueda SIN balancear*)
BEGIN
s^.key:= x; (* centinela como variable global, cuando se inserten los nodos
          en vez de acabar en NIL, acabar n en s^  *)
WHILE (t # NIL) DO
   IF t^.key < x THEN t:= t^.right ELSE t:= t^.left END;
END;
END locateCON;
```

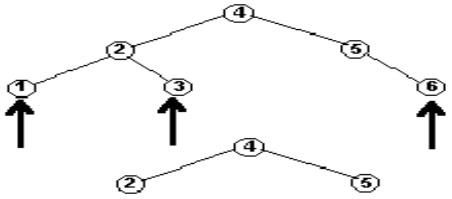
- inserción:

```
TYPE      WPtr    = POINTER TO Word;
          Word    =  RECORD
                        key:INTEGER; count:CARDINAL; left,right:WPtr;
                     END;
PROCEDURE search(x:INTEGER; VAR p:WPtr);
BEGIN
IF x < p^.key THEN (* si es menor *)
   search (x, p^.left);
ELSIF x < p^.key  THEN  (* si es mayor *)
```

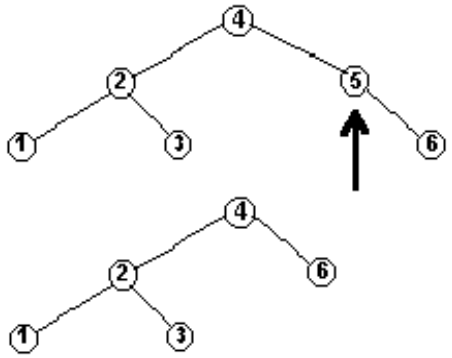
```
    search(x,p^.right);
  ELSIF p # c THEN (* si x = t^.key, pero no es fin de b£squeda *)
          p^.count:= p^.count +1; (* incrementa el contador *)
  ELSE            (* es fin de b£squeda e insertar  *)
    NEW(p);
    WITH p^ DO
      key:=   x;
      left:=  c;(* en vez de usar NIL, usa en centinela *)
      right:= c;(* en vez de usar NIL, usa en centinela *)
      count:= 1;(* inicializa el contador a un encuentro *)
    END;
  END;
  END search;
```


4.4.4 ELIMINACIÓN EN UN ÁRBOL: se elimina nodo a nodo según su posición:

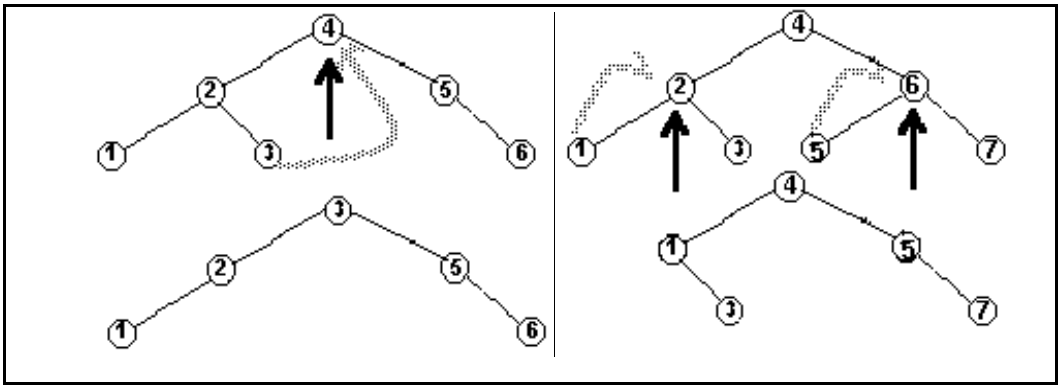
- nodo **terminal**: se elimina simplemente. Ejem:



- nodo interior con **un descendiente**, se elimina y se sustituye por el descendiente



- nodo interior con **dos descendientes** se elimina y se sustituye por el otro nodo que está más a la **derecha de su subárbol izquierdo**



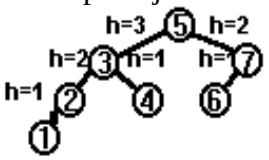
```
PROCEDURE delete(x:INTEGER; VAR p:WPtr); (* arbol búsqueda SIN balancear*)
  VAR q:WPtr;
    PROCEDURE del(VAR r:WPtr); BEGIN
    IF r^.right # NIL THEN del(r^.right)
    ELSE                    q^.key:= r^.key;
                            q^.count:=r^.count;
                            q:= r;
                            r:= r^.left;
    END;
```

```
    END del;
BEGIN
IF p= NIL THEN (* el nodo a eliminar no est  en el arbol *)
(* b£squeda del nodo*)
ELSIF x < p^.key THEN     delete(x, p^.left);
ELSIF x > p^.key  THEN    delete(x,p^.right);
ELSE (*  encuentra nodo *) q:=p;
   IF q^.right= NIL THEN p:= q^.left;
   ELSIF q^.left= NIL THEN p:= q^.right;
   ELSE   del(p^.left);
   END;
    (* Desasignar   q  *) DISPOSE(q);
END;
END delete;
```

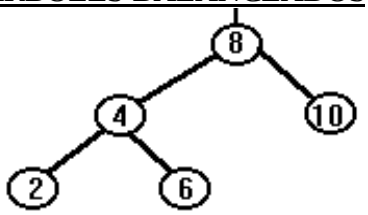

4.4.5 ANÁLISIS DE BÚSQUEDA E INSERCIÓN EN ÁRBOL:

- caso óptimo con un árbol de búsqueda perfectamente balanceado es *log n*
- el caso promedio con un árbol de búsqueda es *2 * (ln n + g - 1)* g_{Heuler}= 0.577..
- caso peor sería un árbol degenerado es *n/2*

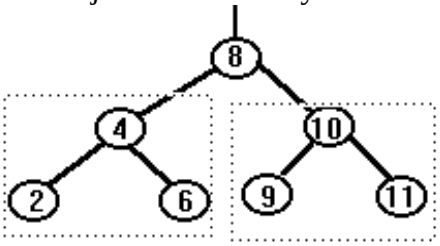
4.5 ARBOLES BALANCEADOS: un árbol está **balanceado** (o también **AVL**) si y sólo si en cada nodo, **las alturas** de sus dos subárboles difieren a lo **máximo en 1.** Un árbol balanceado NO es un árbol perfectamente balanceado; a la inversa sí se cumple. Ejem.:



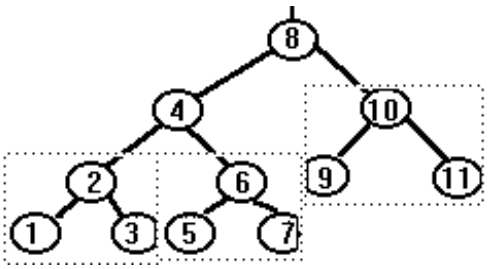
4.5.1 INSERCIÓN EN ÁRBOLES BALANCEADOS:



- seguir la trayectoria de la búsqueda y verificar que la llave NO esté en el árbol.
- INSERTAR NODO y determinar el FARTOR de EQUILIBRIO. Si se realiza en el subárbol L y las alturas anteriores de los subárboles tenían una relación:
  - $h_L < h_R$, al añadir un nodo a L, entonces L y R tendrán una altura **igual**. Mejera el equilibrio del árbol. Ejem.: insertar 9 ó/y 11

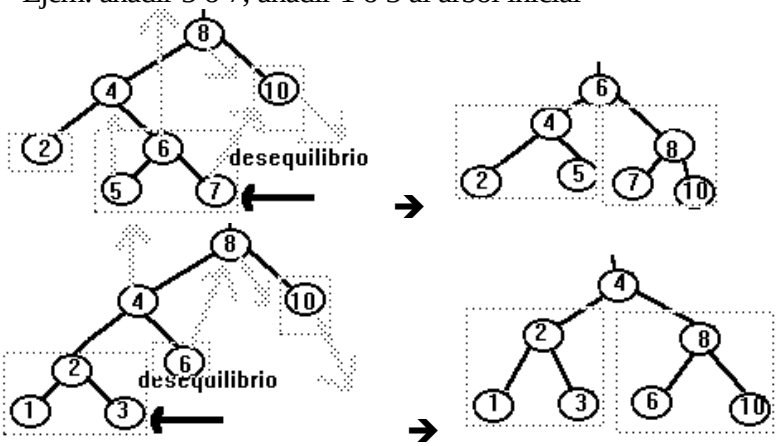


  - $h_L = h_R$, al añadir un nodo a L, entonces L y R tendrán una altura desigual, pero sin violar la condición de balanceo. Ejem.: añadir al anterior 1, 3, 5 y/ó 7



- $\mathbf{h_L > h_R}$, al añadir un nodo a L, entonces L y R tendrán una altura desigual y habrá desequilibrio en el árbol, por tanto, **habrá que rebalancear.** Estas operaciones se intercambian cíclicamente y dan por resultado una rotación sencilla o doble de los dos o tres nodos que intervienen

Ejem: añadir 5 ó 7, añadir 1 ó 3 al árbol inicial



- retroceder a lo largo de la trayectoria de búsqueda, verificando el factor de equilibrio en cada nodo. Realizando el rebalanceo en caso necesario.

4.5.2 ELIMINACIÓN EN ÁRBOLES BALANCEADOS, es como la inserción pero a la inversa en sus condiciones, esto es, si la condición de equilibrio inicial es $h_L < h_R$, y se elimina un nodo de L, entonces habrá violado la condición y se tendrá que rebalancear.

ANÁLISIS DE LOS ÁRBOLES AVL ó BALANCEADOS: según pruebas empíricas dan un resultado de *log (n) + c*, dando un rendimiento tan efectivo como el árbol perfectamente balanceado. Pero debido a sus costes de mantenimiento (inserción y eliminación) es sólo recomendable utilizar en las recuperaciones de información. Por tanto, el árbol de búsqueda es preferible para la clasificación

4.6 ARBOLES DE BÚSQUEDA ÓPTIMOS (ÁRBOLES EXCEPCIONALES), se caracterizan por **permanecer sus llaves inalteradas**, por tanto, no habrá inserciones o eliminaciones de nodos. El caso típico es el rastreados de un compilador.

Para construir este tipo de árbol se necesita saber, empíricamente, la frecuencia del uso de los nodos y, después diseñar el árbol. Para ello, se debe analizar **la longitud de la trayectoria pesada** (es la suma de todas las que van de la raíz a cada nodo) **ponderada por la probabilidad** o frecuencia de acceso del nodo.

4.7 ARBOLES B

4.7.1 ARBOLES B MULTICAMINOS ó DIRECIONES MÚLTIPLES (MEMORIA SECUNDARIA). Se trata de árboles muy extensos cuyos nodos deben guardarse en memoria secundaria (disco) por la gran cantidad de información que contienen. Así pues, la innovación son las **páginas**, estas son, grupos de subárboles que se acceden simultáneamente con unas **propiedades**:

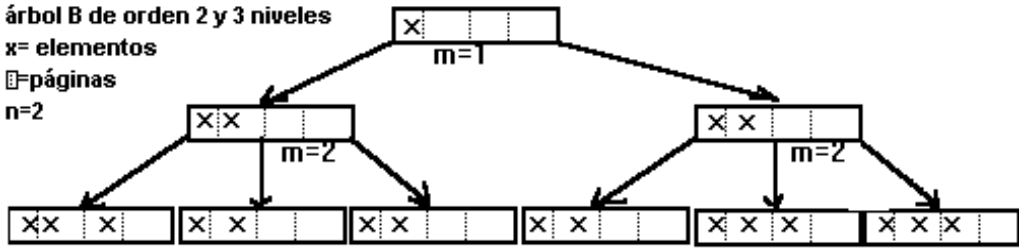
- cada página, contiene a lo sumo *2n* elementos (llaves)

- cada página excepto la raíz, contiene *n* elementos por lo menos

- cada página es una página de hoja, o sea que no tiene descendientes o tienen *m+1* descendientes, donde *m* es su número de llaves de esta página.

- todas las páginas de hoja aparecen al mismo nivel.
- se denomina ***n*** **al orden de árbol**

Ejemplo:



Inserción de una llave *X* en alguna página con un orden determinado:

- comprobación de que la llave NO está en el árbol:
  - se carga la página raíz en memoria primaria

  - se produce una búsqueda de la llave, por los procedimientos del tema 2 (binaria si *m* es grande ó secuencial si *m* es pequeña).

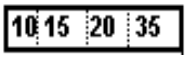
  - si la búsqueda fracasa y los apuntadores están activos, tendremos 3 casos:

    - el valor de la llave menor que el primer elemento, $X < k_1$, llamará a la primera página, p_0
    - el valor de la llave entre dos elemento, $k_i < X < k_{i+1}$, llamará a la página p_1
    - el valor de la llave mayor que el último elemento, $X < k_m$, llamará a la última página p_m
- INSERCIÓN en una página con apuntadores INactivos y NO está la llave, si:
  - la página está INcompleta, luego $m < 2n$, se inserta la llave con el orden
  - la página está completa ($m=2n$) y NO cabe la llave ($m + llave > 2n$) :
    - se DIVIDE la página en concreto en DOS
    - los *2*n+**1*** elementos se dividen:
      - (*2*n*) en *n* elementos por cada una de las dos páginas
      - **(*+1*) la llave de la mitad , <u>se sube un nivel</u>** (al contrario que los árboles en general) **y se repite la**

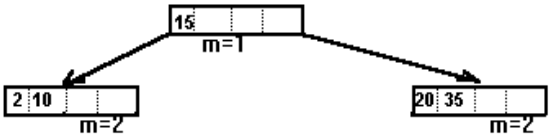
inserción hasta que no haya desbordamiento

Ejemplo: - inicio:

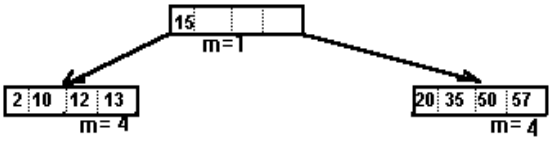
- se inserta 15, 35, 10:



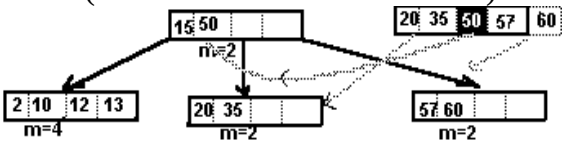
- se inserta 2(la llave de la mitad sube un nivel):



- se inserta 12,13,50,57:



- se inserta 60 (la llave de la mitad sube un nivel):



El árbol B multicamino crece de abajo hacia arriba, desde las hojas hacia la raíz, al contrario que los árboles.

ANÁLISIS: en el peor de los casos se necesitan $log_n N$ para N elementos y orden n, se aplicará en cada página el tipo de búsqueda en memoria primaria (tema 2)

4.7.2 **BB**, ARBOLES B BINARIOS ó ARBOL 2-3. Está construido con nodos de uno o dos elementos y por tanto, 2 ó 3 (m+1) apuntadores a los descendientes. Su funcionamiento es semejante al multicamino pero al tener pocos elementos por página puede almacenar los datos en memoria primaria.

ANÁLISIS DE BB: en el peor de los casos, se requieren $log_n N$ accesos de página (para N elementos en un árbol BB de orden n). La búsqueda en cada página también tendrá un coste, según el método empleado (tema 2)

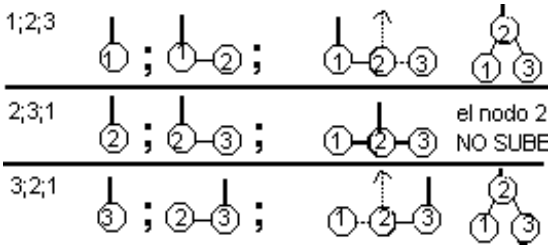
BBS, ÁRBOLES B BINARIOS SIMÉTRICOS, son los que poseen las siguientes propiedades:

- todo nodo contiene una llave y, al máximo dos subárboles (= apuntadores)
- todo apuntador es horizontal o vertical. **NO existen dos apuntadores consecutivos horizontales en ninguna trayectoria de búsqueda.**
- todos los nodos terminales aparecen en el mismo nivel.

Inserción: similar a los árboles BB, la diferenciación está en:

- si la raíz está en el **medio de tres nodos hermanos**, entonces un nodo puede tener **dos hermanos** (ejm:2,3,1)
- si la raíz está a un **lado de tres nodos hermanos**, entonces el **nodo central sube un nivel** y coloca allí su raíz. (ejms:1,2,3 y 3,2,1)

Ejemplos de inserción:



ANÁLISIS DE BBS: en el peor de los casos la longitud máxima de trayectoria es *2*logN* para *N* elementos. Los árboles BBS consiguen mejor rendimiento en la búsqueda que los BB, pero la inserción y la eliminación se complica bastante más los BBS.

4.8 ARBOLES DE BÚSQUEDA CON PRIORIDAD.