

CONCEPTOS DE BASES DE DATOS

CLASE 5



Árboles

- El uso de **índices** presenta ciertos problemas:
 - Se necesita un **procesamiento adicional** para reacomodar los archivos de índices cuando hay cambios en el archivo de datos
 - Índice primario
 - Índices secundarios
 - Archivos de listas invertidas
 - Si se manejan **archivos de datos muy grandes**, estos archivos adicionales probablemente se deberán manejar en **memoria secundaria**

Árboles

- Quedan planteados dos grandes problemas a resolver:
 - La búsqueda binaria es muy costosa en almacenamiento secundario → demasiados desplazamientos
 - Mantener el índice ordenado es muy costoso → ante cada cambio se debe reordenar el índice completo

Árboles

- Para **mantener el orden de forma más eficiente**, se requiere un tipo de estructura diferente a las utilizadas convencionalmente para los archivos de índices
 - Se necesita que ante un cambio no se deban hacer **reorganizaciones masivas**, sino **locales**
- Un **árbol** es una estructura de datos que se puede aplicar en los índices:
 - Permite **localizar en forma rápida** la información de un archivo
 - Ante cambios, sólo debe **reorganizarse localmente**

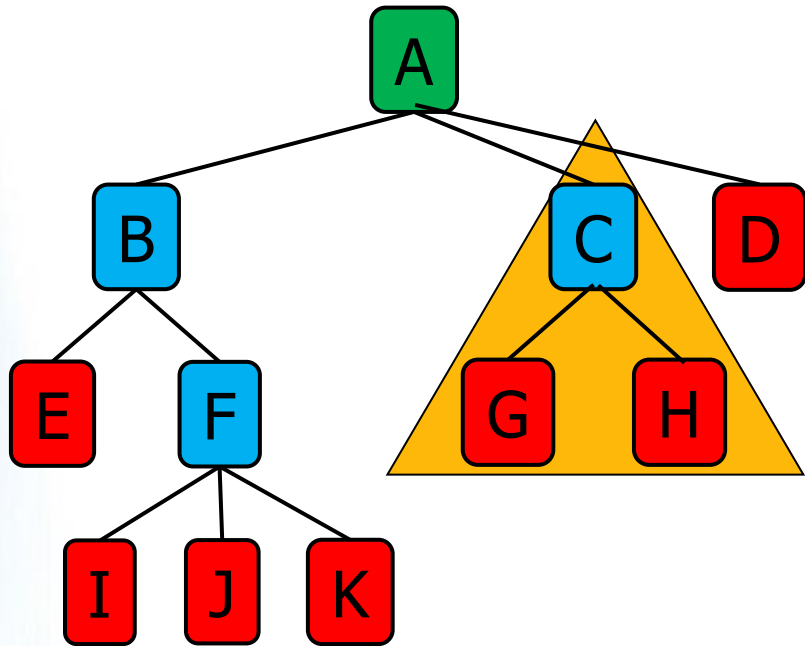
Árboles

- Se analizarán las diferentes soluciones que fueron surgiendo para el almacenamiento de índices:
 - Árboles Binarios
 - Árboles AVL
 - Árboles Multicamino
 - Árboles Balanceados
 - Árbol B
 - Árbol B*
 - Árbol B+

Árboles

- Un **árbol** está formado por una serie de nodos conectados por aristas, que verifican:
 - Hay un **único nodo raíz**
 - Cada nodo tiene un **único padre** (excepto la raíz)
 - Hay un **único camino** desde la raíz hasta cada nodo
 - Puede estar **ordenado** o no

Árboles

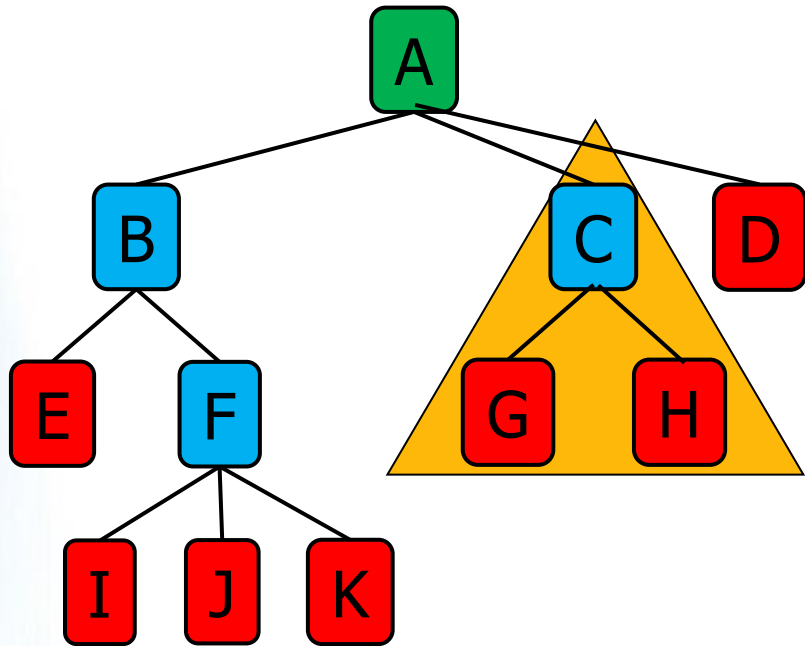


- **Raíz:** único nodo sin padre
- **Nodo interno:** tiene al menos un hijo
- **Nodo hoja (externo):** no tiene hijos
- **Descendiente directo:** hijo
- **Descendientes:** hijo, nieto...
- **Subárbol:** árbol formado por un nodo y sus descendientes

Árboles

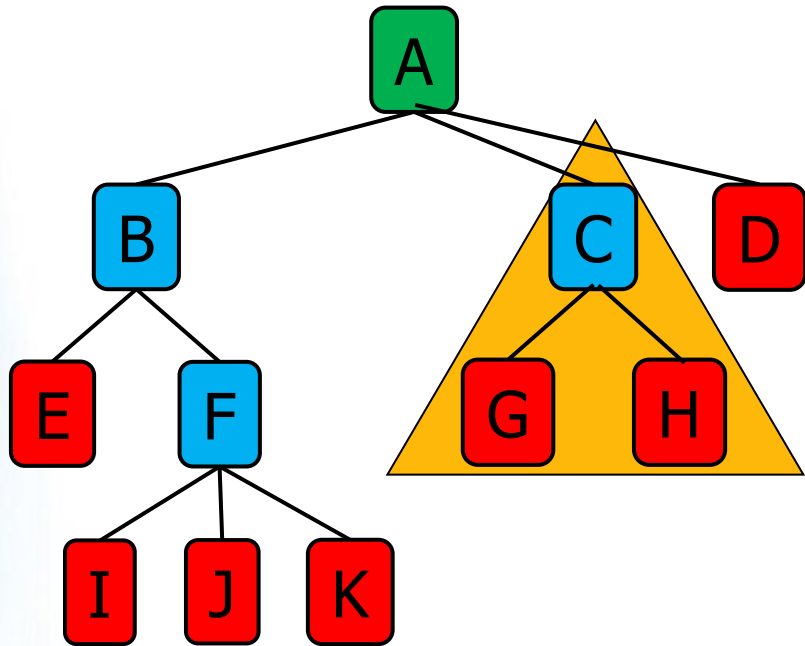
- **Grado** de un **nodo**: nro de descendientes directos
- **Grado** del **árbol**: mayor grado de sus nodos
 - Árbol binario \rightarrow grado 2
 - Árbol multcamino \rightarrow grado N
 - Lista \rightarrow árbol degenerado de grado 1
- **Profundidad** de un **nodo**: nro de predecesores
- **Profundidad**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

Árboles



- $\text{Grado}(\text{B}) \rightarrow ?$
- $\text{Grado del árbol} \rightarrow ?$
- $\text{Profundidad}(\text{A}) \rightarrow ?$
- $\text{Profundidad}(\text{H}) \rightarrow ?$
- $\text{Altura del árbol} \rightarrow ?$

Árboles



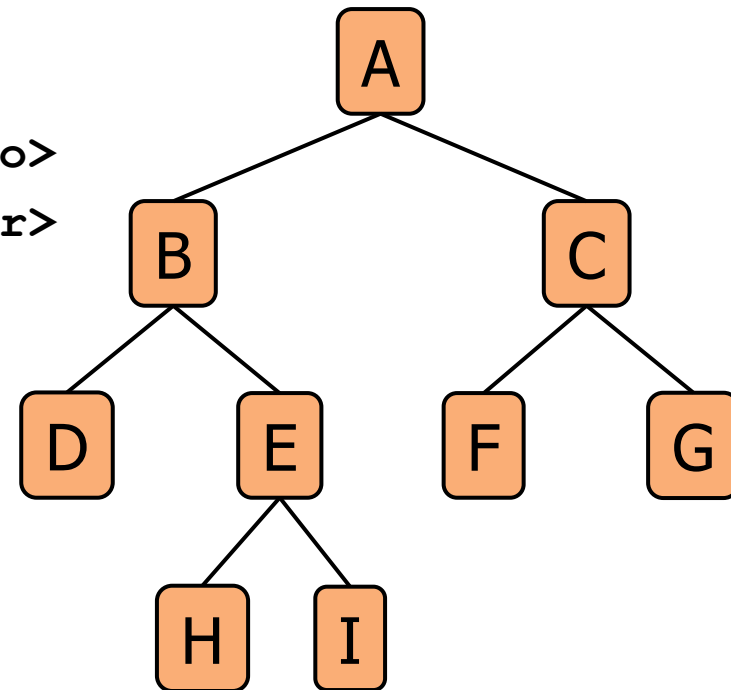
- **Camino**: existe un camino del nodo **X** al nodo **Y**, si existe una **sucesión de nodos** que permitan llegar desde **X** a **Y**
- $\text{Camino}(\text{A}, \text{K}) = \{\text{A}, \text{B}, \text{F}, \text{K}\}$
- $\text{Camino}(\text{C}, \text{K}) = \{\}$

Árboles

Binario

- El **árbol binario** es un árbol de grado 2
- Cada nodo tiene de 0 a 2 descendientes directos: el **hijo izquierdo** y el **hijo derecho**

```
<arbol> ::= <nulo> | <nodo>  
<nodo> ::= <info> <izq> <der>  
<izq> ::= <arbol>  
<der> ::= <arbol>
```

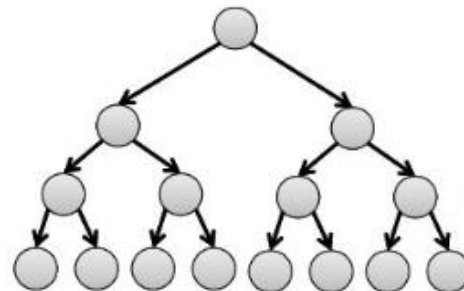


Árboles

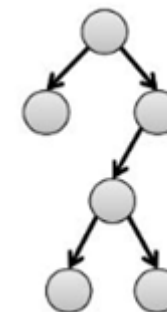
Binario

- El árbol binario tiene diferentes aplicaciones tales como expresiones aritméticas y árboles de decisión.
- También es posible usarlos para búsqueda de información, y se denominan **Árboles Binarios de Búsqueda (ABB)**.
- En algunos casos se exige que el árbol sea **completo** → **todo nodo interno debe tener dos descendientes**

Árbol binario completo



Árbol binario no completo



- Estructura de datos utilizada para el almacenamiento de un árbol binario
 - Archivo con registros de longitud fija
 - La información en el archivo no está ordenada
 - Cada nodo es un registro: elemento, hijoIzq, hijoDer
 - Existe un desperdicio de espacio de almacenamiento → muchos campos vacíos

Árboles

Binario

- Estructura de datos utilizada para el almacenamiento de un árbol binario

Type regAB = record

 elemento: tipoDato; *{clave índice}*

 hijoIZQ: integer; *{dir. hijo menor (NRR)}*

 hijoDER: integer; *{dir. hijo mayor (NRR)}*

end;

indiceBinario = file of regAB;

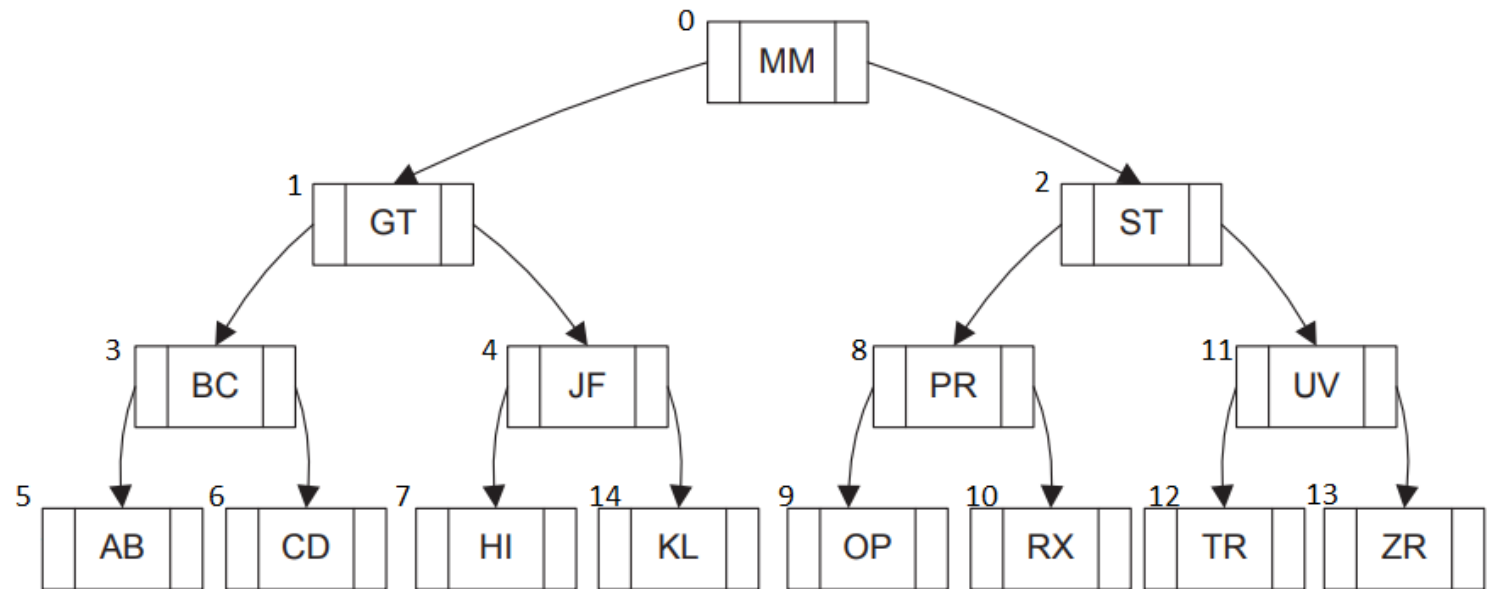
- Operaciones
 - Búsqueda
 - Inserción
 - Eliminación

Árboles

Binario

Raíz → 0

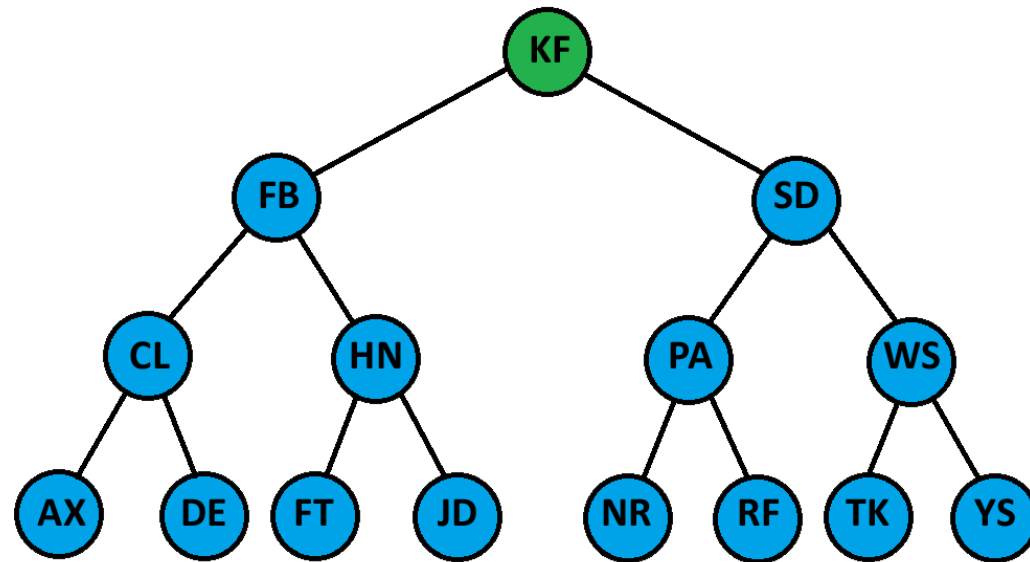
	Clave	Hijo Izq	Hijo Der
0	MM	1	2
1	GT	3	4
2	ST	8	11
3	BC	5	6
4	JF	7	14
5	AB	-1	-1
6	CD	-1	-1
7	HI	-1	-1
8	PR	9	10
9	OP	-1	-1
10	RX	-1	-1
11	UV	12	13
12	TR	-1	-1
13	ZR	-1	-1
14	KL	-1	-1



Árboles

Binario

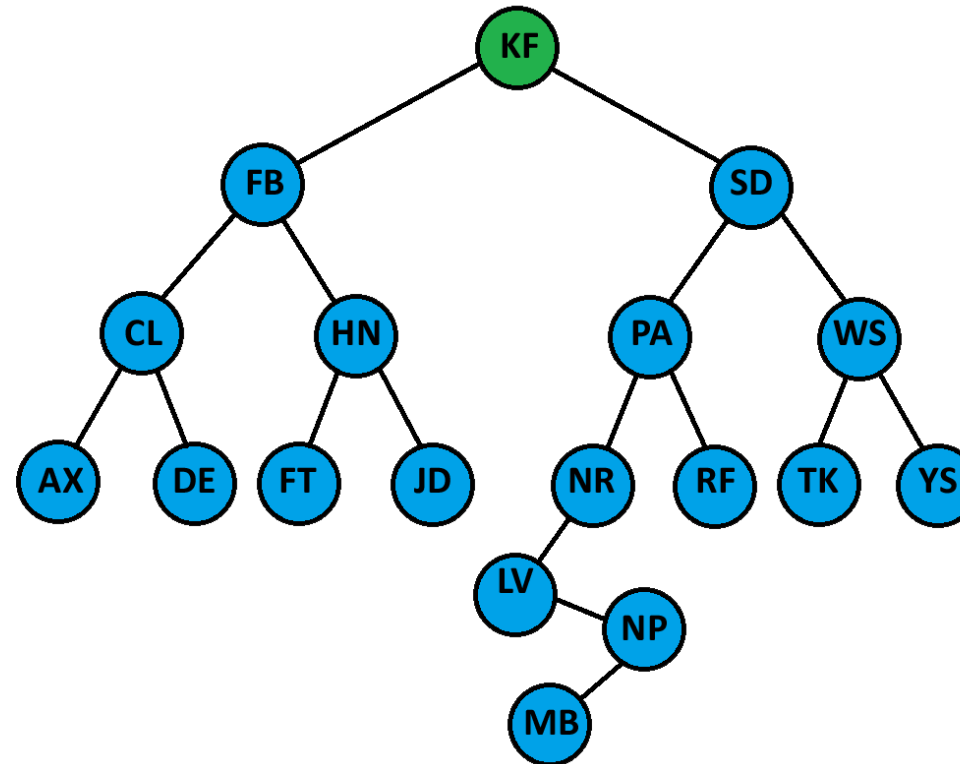
- Ejemplo: se construye el árbol dada la siguiente lista ordenada de claves
 - **AX CL DE FB FT HN JD KF NR PA RF SD TK WS YS**



Árboles

Binario

- Pero luego llegan las siguientes claves → **LV NP MB**



Árboles

Binario

- Para acceder al nodo **MB** son necesarios 7 accesos → la performance de la búsqueda **ya no puede considerarse logarítmica**
- Para árboles binarios con **cientos de claves** se puede llegar a requerir **más de 30 accesos** para alcanzar algunos de sus elementos
- No alcanza a ser lo suficientemente eficiente

Árboles

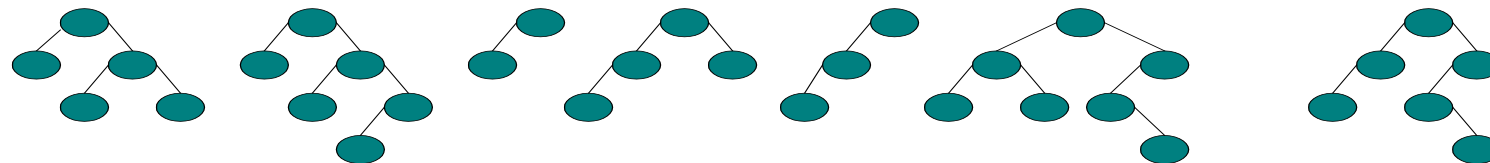
Binario

- 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
- Los árboles binarios se desbalancean fácilmente → **búsquedas más costosas** (mayor cantidad de desplazamientos)
 - Una solución que se encontró fue reorganizar los nodos del árbol a medida que se reciben las claves → **Árboles AVL**

Árboles

AVL

- Un **árbol AVL** es un árbol binario balanceado
 - No está balanceado completamente → la **diferencia máxima de altura** entre las alturas de cualquiera de dos subárboles que comparten raíz común es **1**
 - Se lo denomina entonces **árbol BA(1)** → árbol balanceado en altura, con diferencia máxima 1
 - Clase general BA(k): máxima diferencia de altura es k
 - ¿Cuáles de los siguientes son árboles AVL?



Árboles

AVL

- La estructura debe ser respetada → al insertar y eliminar nodos se realizan **rotaciones restringidas a un área local** del árbol
 - Algoritmos de rebalanceo sencillos
- Pero al aplicarlo a índices, se encontraron algunas desventajas:
 - El costo de acceso a disco adicional (inserción / eliminación de nodos)
 - Como son árboles binarios, si tienen muchos elementos son muy profundos → **muchos niveles**

Árboles

AVL

- Si se cuenta con N claves
 - En un árbol binario completamente balanceado, el peor caso de búsqueda para encontrar una clave, busca en $\log_2(N+1)$ niveles del árbol
 - En un árbol AVL el peor caso de búsqueda podría ser buscar en $1.44 \log_2(N+2)$ niveles del árbol

Árboles

AVL

- Para 1.000.000 claves
 - Un árbol binario completamente balanceado, requiere desplazamiento en 20 niveles para buscar alguna de las claves
 - En un árbol AVL el número máximo de niveles a buscar es de 28
 - Para almacenamiento secundario son valores inaceptables → máximo 5 o 6 desplazamientos

Árboles

- Volviendo a los dos problemas planteados inicialmente
 - La búsqueda binaria requiere demasiados desplazamientos
 - Mantener un índice en orden es muy costoso
- Los árboles balanceados en altura proporcionan una solución admisible al segundo problema

Árboles

Binarios Paginados

- El desplazamientos en memoria secundaria tiene un **costo de tiempo relativamente alto**
- Pero una vez en posición, la lectura o escritura de un conjunto de bytes contiguos es **rápida**
- Desplazamiento lento + transferencia rápida → **paginación**

Árboles

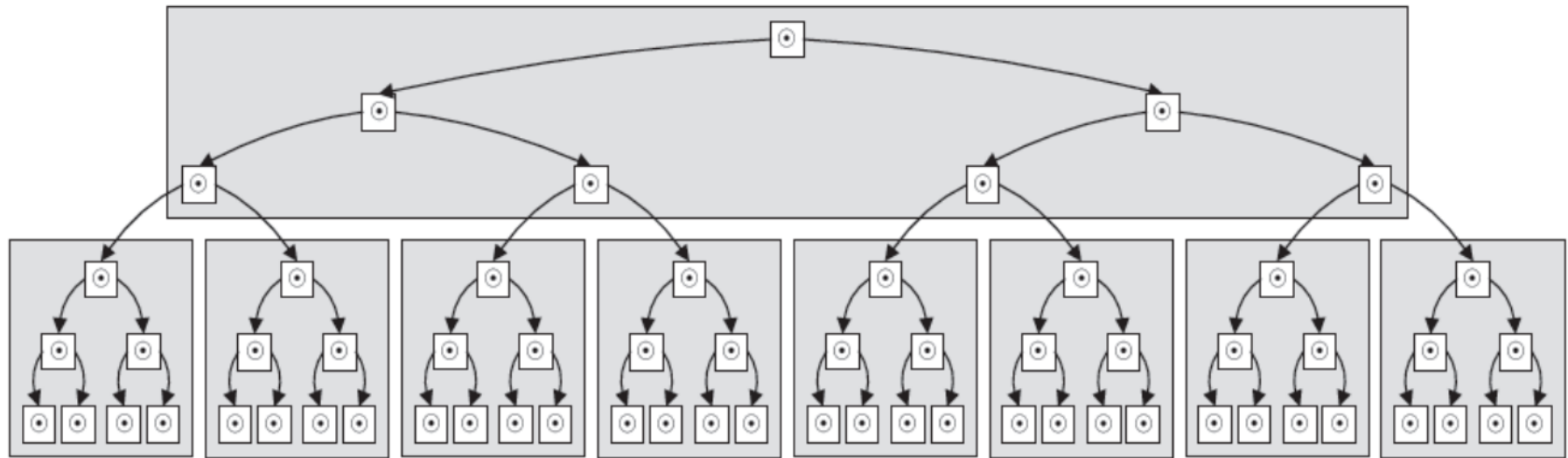
Binarios Paginados

- Estrategia
 - Dividir el árbol binario en páginas
 - Almacenar cada página en un bloque de direcciones contiguas en disco
- Se puede reducir el número de desplazamientos considerablemente
 - Permite búsquedas más rápidas en almacenamiento secundario
 - **Solución potencial** al problema de búsqueda eficiente

Árboles

Binarios Paginados

- Ejemplo
 - Posibilidad de acceder a 63 nodos con sólo 2 accesos a disco



Árboles

Binarios Paginados

- Uso de páginas grandes
 - Suposición de árbol **completamente balanceado**
 - Páginas de 8 Kb → **511 claves por página**
 - Para buscar cualquiera de **134.217.727 claves** se requieren **sólo 3 desplazamientos**
- Cada acceso a una página requiere transmitir muchos datos → la mayoría no usados
 - Hay un tiempo de transmisión adicional, pero **se ahorran muchos desplazamientos** que consumirían mucho más tiempo

Árboles

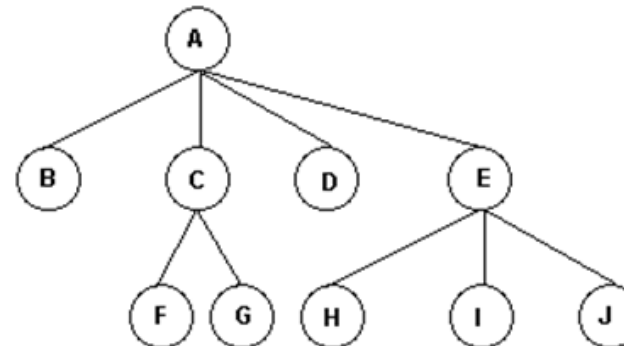
Binarios Paginados

- Problemas
 - ¿Cómo construirlo?
 - ¿Cómo elegir la raíz?
 - ¿Cómo mantenerlo balanceado?
 - La idea de agrupar claves en páginas era muy buena, pero no se encontró la forma de hacerlo eficientemente
- Sin embargo, se adopta la idea de manipular más de un registro a la vez para crear nuevas estructuras

Árboles

Multicamino

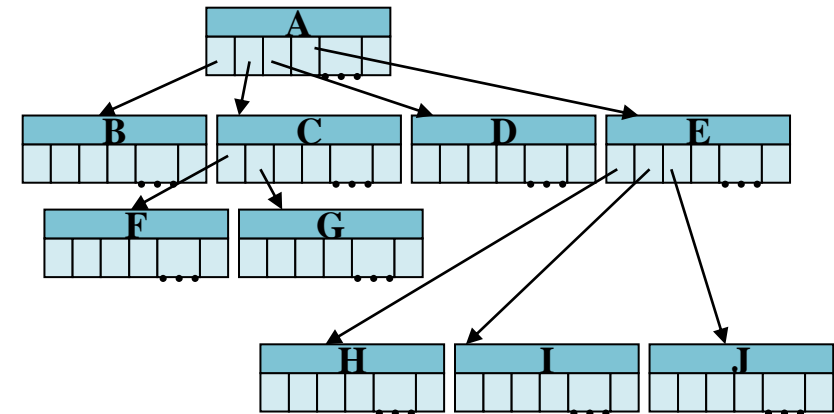
- Un **árbol N-ario** o **multicamino** es aquel en el que cualquier nodo puede tener cualquier número de hijos
 - Por definición, un árbol multicamino es todo árbol N-ario, con $N \geq 2$
 - En general, al nombrar a un árbol multicamino se hará referencia a un árbol con $N > 2$, para diferenciarlos de los árboles binarios que ya fueron analizados



Árboles

Multicamino

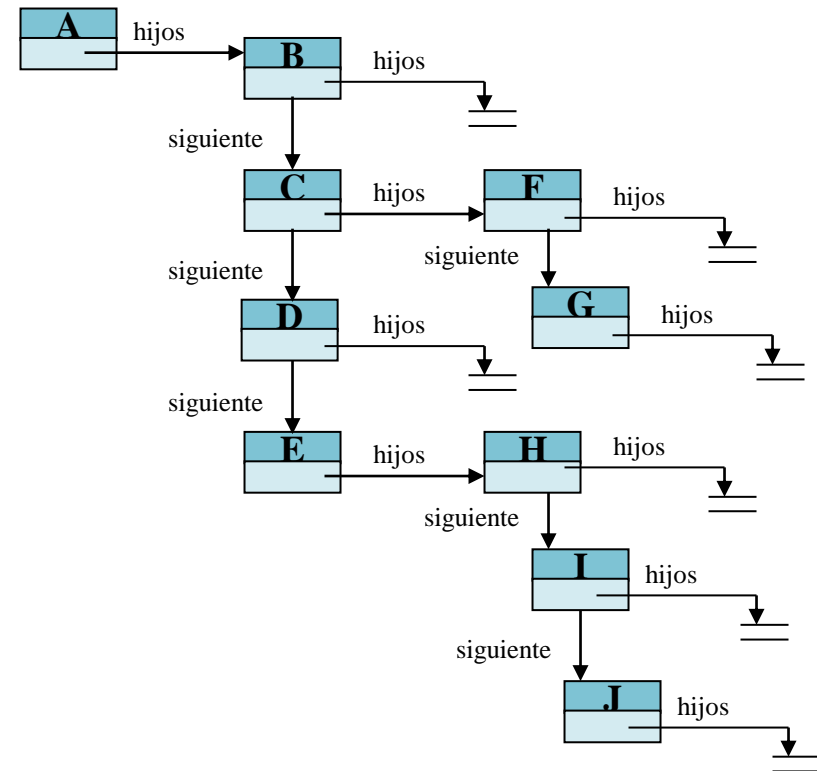
- Implementación 1: hijos como **arreglo de referencias**
- Estructura estática
 - Uso sencillo
 - Si el número de hijos es muy variable se desaprovecha espacio de almacenamiento
 - No puede usarse si el número de hijos es ilimitado



Árboles

Multicamino

- Implementación 2: hijos como **lista enlazada**
- Estructura dinámica
 - Permite la cantidad necesaria de hijos en cada caso
 - No desperdicia espacio de almacenamiento
 - Se implementa con el uso de punteros, lo que hace su manejo más complejo



Árboles

Multicamino

- Los árboles **multicamino** surgieron como una buena opción que resuelve uno de los problemas que tienen los árboles binarios → **cantidad de niveles**
- Sin embargo, se debe mantener el árbol **balanceado** para asegurar la eficiencia
 - Si el árbol está desbalanceado, no se puede asegurar que las búsquedas realizadas tengan un costo logarítmico

Árboles

Balanceados (B)

- El problema del desbalanceo surge porque, hasta ahora, en todas las soluciones vistas se han construido los árboles desde la raíz hacia abajo
 - Problemas para elegir la raíz
 - Problemas para mantener el árbol balanceado
- **Alternativa:** utilizar árboles que se construyan hacia arriba

Árboles

Balanceados (B)

- Árboles B (balanceados)
 - Son árboles **multicamino**
 - Se construyen en **forma ascendente**, partiendo desde la base
 - La raíz emerge con la misma construcción del árbol
 - Esta forma especial de construcción permite mantenerlo **balanceado a bajo costo**
 - Su **orden** indica la **cantidad máxima de descendientes directos** que cada nodo puede tener

- Propiedades de un **árbol B** de orden **M**
 - **Cantidad de hijos por nodo:**
 - Máximo: **M** hijos
 - Mínimo: **$\lceil M/2 \rceil$** hijos (menos raíz y terminales)
 - Mínimo raíz: **2** hijos (o sino **ninguno**)
 - **Nodos no terminales:** **K** hijos \rightarrow **K-1** claves
 - **Nodos terminales (hoja):** todos están al mismo nivel
 - Máximo: **M-1** claves
 - Mínimo: **$\lceil M/2 \rceil - 1$** claves
 - **Cantidad de punteros:** **cantidad claves + 1**

Árboles

Balanceados (B)

- Propiedades de un árbol B de orden M
 - **Formato del nodo:** cada $R_{i-1} < R_i < R_{i+1}$

PO	R1	P1	R2	P2	R3	P3	R4	P4	R5	P5	Nro de registros
----	----	----	----	----	----	----	----	----	----	----	------------------

- **Estructura**
 - Archivo con registros de longitud fija
 - Cada registro contiene un nodo
 - **Type regArbolB = record**
 - hijos: array [1..M] of integer;
 - claves: array [1..(M-1)] of char;
 - cantClaves: integer;
 - end**

Árboles

Balanceados (B)

- Nodo **adyacente hermano**
 - Dos nodos son adyacentes hermanos si tienen **el mismo padre y son apuntados por punteros adyacentes** en el padre
- **Operaciones**
 - Creación e inserción
 - Búsqueda
 - Borrado
 - Modificación (se trata como una baja seguida de un alta)

- **Inserción (creación)**
 - Comienza con una búsqueda que llega hasta el nivel hoja → los registros se insertan en un nodo terminal
 - Después de encontrar el lugar de inserción, el trabajo de inserción, división y promoción continúa en forma ascendente desde abajo

- **Inserción (creación)**
 - Si el registro **tiene lugar** en el nodo terminal → **no se produce overflow**
 - Sólo se hacen **reacomodamientos internos** en el nodo
 - Si el registro **no tiene lugar** en el nodo terminal → **se produce overflow**
 - El nodo se **divide** y los elementos se **reparten** entre los nodos
 - Hay una **promoción** al nivel superior, y ésta **puede propagarse, incluso hasta generar una nueva raíz**

- **Inserción (creación) → Performance**
 - Orden = M , Altura = H
 - **Mejor caso** (sin overflow)
 - H lecturas
 - 1 escritura
 - **Peor caso** (overflow hasta la raíz → aumenta nivel del árbol)
 - H lecturas
 - $2H + 1$ escrituras (dos por nivel + la raíz)
 - Estudios realizados
 - $M = 10 \rightarrow$ **25% overflow** (75% MC, 25% puede o no ser PC)
 - $M = 100 \rightarrow$ **2% overflow** (98% MC, 2% puede o no ser PC)

- **Búsqueda de información**
 - Comienza la búsqueda desde el nodo raíz
 - Busca la clave en el nodo
 - Si no la localiza se toma el **puntero correspondiente entre las claves existentes**
 - Si el puntero **no es nulo** → se toma ese nodo y se repite la búsqueda
 - Si el puntero **es nulo** → el elemento **no se encuentra** en el árbol

- **Búsqueda de información** → Performance

- Orden = M , Altura = H
- Axioma: $N = \text{\#claves} \rightarrow N+1$ punteros nulos en nodos terminales
- Accesos
 - Mejor caso: 1 lecturas Peor caso: H lecturas
- ¿Cómo se puede acotar H (en función de M y N) ?

Nivel	#mínimo de descendientes
1	2
2	$2 * [M/2]$
3	$2 * [M/2] * [M/2]$
<hr/>	
H	$2 * [M/2]^{H-1}$

- **Búsqueda de información** → Performance

- Orden = M , Altura = H , #Claves = N

$$\begin{aligned} N+1 &\geq 2 * [M/2]^{H-1} \\ (N+1)/2 &\geq [M/2]^{H-1} \\ \log_{[M/2]} ((N+1)/2) &\geq H - 1 \end{aligned}$$

$$H \leq 1 + \log_{[M/2]} ((N+1)/2)$$

- $M = 512$ y $N = 1.000.000 \rightarrow H \leq 3,37$
- En sólo 4 lecturas se encuentra cualquier registro del archivo

- **Eliminación**

- **Mejor caso: no se produce underflow** al borrar el elemento del nodo (sólo reacomodos) →
 $\#claves \geq [M/2]-1$
 - **Caso 1:** nodo terminal
 - **Caso 2:** nodo no terminal → llevar a un nodo terminal
- **Peor caso: se produce underflow** al borrar el elemento del nodo → $\#claves < [M/2]-1$
 - **Caso 3:** redistribución
 - **Caso 4:** concatenación

- **Eliminación**

- **Redistribución**: cuando un nodo tiene underflow es posible redistribuir las claves entre dicho nodo y el **nodo adyacente hermano** → sólo en caso de que éste tenga **suficientes elementos**
- **Concatenación**: si un **nodo adyacente hermano** está al **mínimo** no se puede redistribuir → se concatena con un nodo adyacente disminuyendo el número de nodos. En algunos casos también **puede llevar a disminuir la altura del árbol**

- **Eliminación** → Performance
 - Orden = M , Altura = H
 - **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