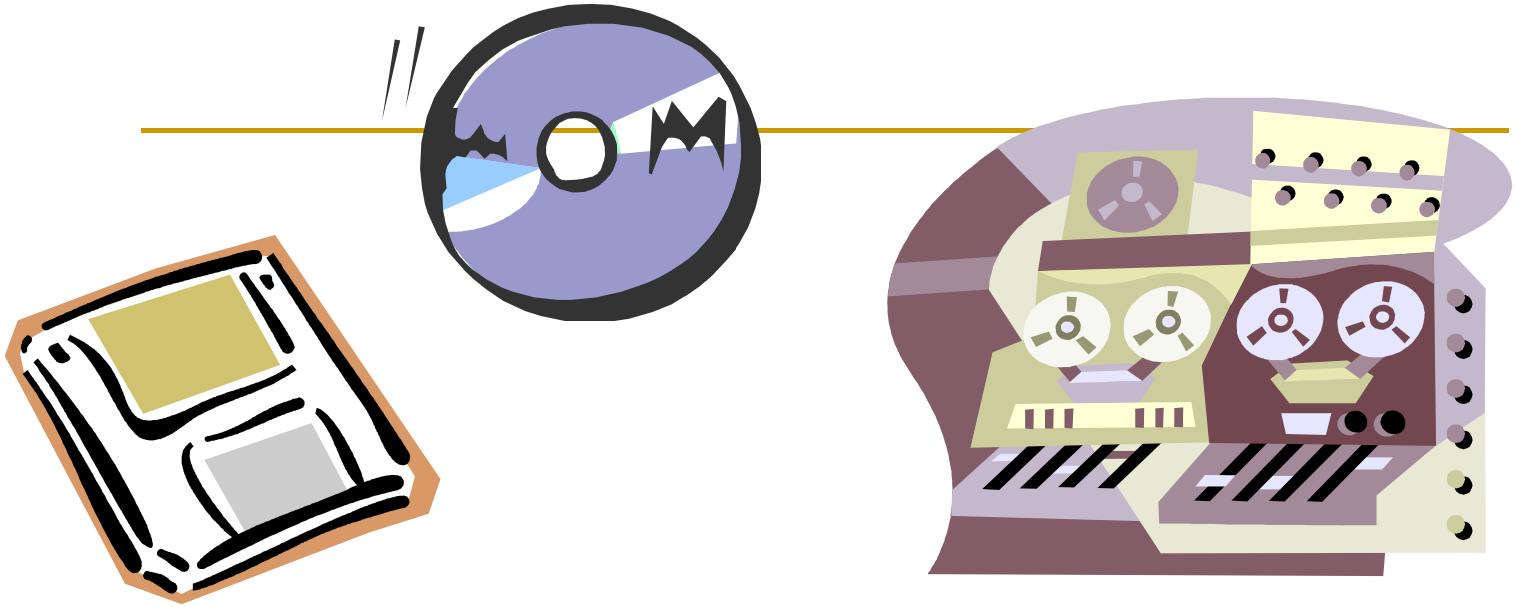
Sorting y Searching en memoria secundaria



Motivaciones

- Los problemas abstractos siguen siendo los mismos
- Pero el volumen de los datos hace que los mismos se encuentren necesariamente, en memoria secundaria
- En general, entran a jugar ciertos elementos “de sistema” además de los “algorítmicos”.
- Tiempo de acceso a los elementos es mucho mayor que en memoria principal (típicamente, 1000000 de veces más lento)
- Costo principal: entrada-salida
  - p. ej. invertir un gran archivo ya es complicado, aún cuando no hay que hacer comparaciones entre elementos ni nada de eso.
- Queremos minimizar la cantidad de veces que un elemento hace el viaje memoria secundaria – memoria principal

Motivaciones

- Tiempo de acceso dominado por el posicionamiento (más que la transferencia en sí misma): conviene acceder a bloques de datos simultáneamente
- A veces sólo se puede acceder a los elementos en forma secuencial (p. ej. en una cinta)
- Habitualmente, puede haber jerarquía de niveles de memoria
- Memoria virtual + manejo de memoria por parte del SO (caching) puede ser una buena solución si se usa un método con buena “localidad de referencia”
- Se requieren soluciones distintas

Ordenación-fusión

- Muchos métodos hacen pasada sobre el archivo, dividiéndolo en bloques que entren en memoria interna, ordenando esos bloques y luego fusionando.
- El acceso secuencial resulta apropiado a ese tipo de dispositivos.
- Objetivo: minimizar número de pasadas sobre el archivo

Métodos de ordenación para mem. secundaria

- Fusión Múltiple Equilibrada
- Selección por sustitución
- Fusión Polifásica

Fusión Múltiple Equilibrada

- Supongamos tenemos que ordenar los registros con valores
  - EJEMPLODEORDENACIONFUSION
- Los registros (o sus claves) están en una cinta, y sólo pueden ser accedidos secuencialmente.
- En memoria hay sólo espacio para un número constante pequeño de registros, pero disponemos de todas las cintas que necesitemos (vamos a suponer capacidad 3 registros)

Fusión Múltiple Equilibrada

- Primera pasada
  - Mientras haya elementos en la cinta de entrada
    - Para i desde 1 hasta 3 hacer:
      - leer 3 registros a la vez y escribirlos, ordenados, en la cinta i

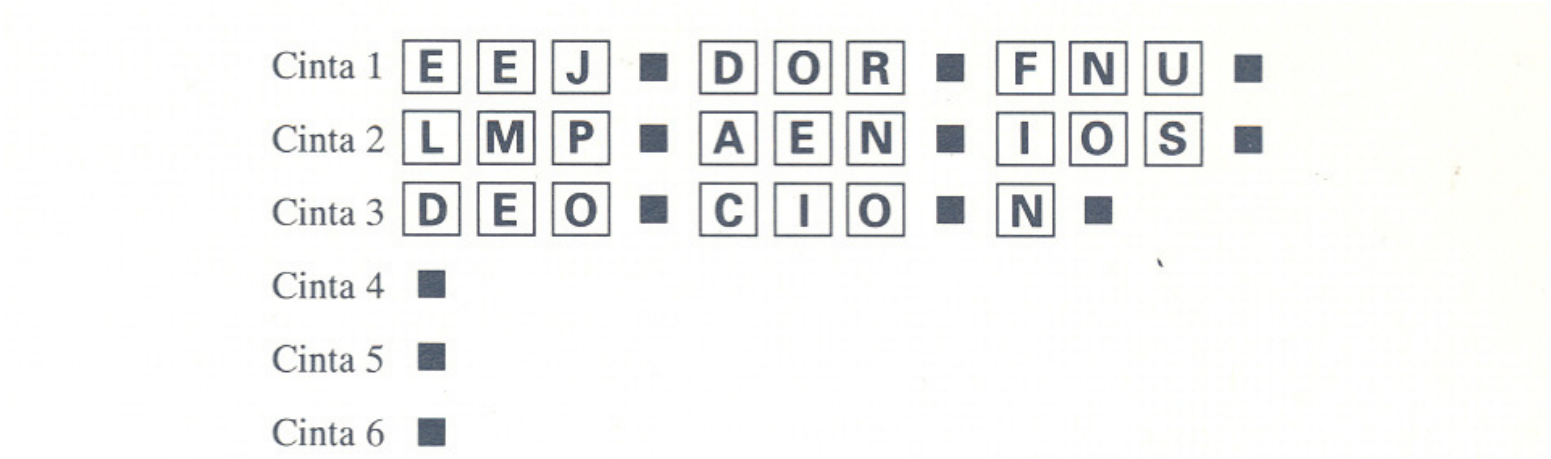


Figura 13.1 Fusión equilibrada de tres vías: resultado de la primera pasada.

Fusión Múltiple Equilibrada

- Segunda pasada
  - Mientras haya bloques de 3 en las 3 cintas hacer
    - Fusionar los 3 bloques de 3 armando uno de 9 en las cintas 4, 5 y 6 alternadamente.

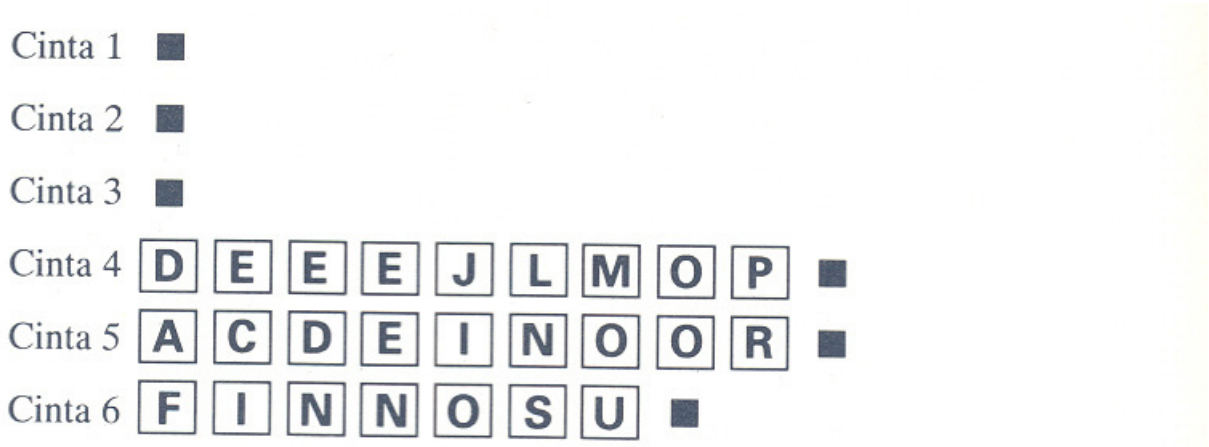


Figura 13.2 Fusión equilibrada de tres vías: resultado de la segunda pasada.

- Etcetera

Fusión Múltiple Equilibrada

- Complejidad (N registros, memoria interna de tamaño M, fusiones a P vías)
  - La primera pasada produce aprox. N/M bloques ordenados
  - Se hacen fusiones a P vías en cada paso posterior, lo que requiere aprox. $\log_p(N/M)$ pasadas
- Por ejemplo
  - Archivo de 200 millones de palabras
  - 1 millón de palabras de memoria (¿cómo las ordenamos?)
  - Fusión de 4 vías
  - No más de 5 pasadas.

Selección por sustitución

- Idea: usar una cola de prioridad de tamaño M para armar bloques, “pasar” los elementos a través de una CP escribiendo siempre el más pequeño y sustituyéndolo por el siguiente. Pero, si el nuevo es menor que el que acaba de salir, lo marcamos como si fuera mayor, hasta que alguno de esos nuevos llegue al primer lugar de la CP.
- EJEMPLODEORDENACIONFUSION

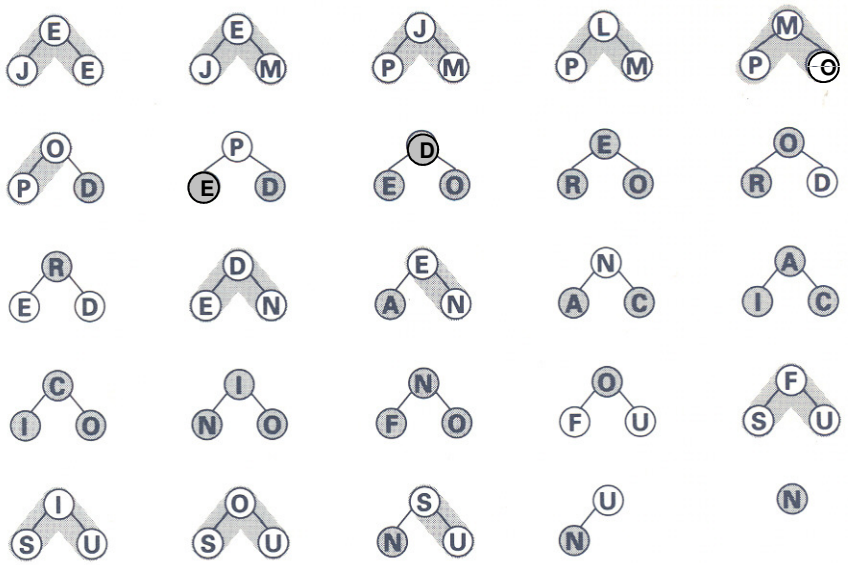


Figura 13.4 Creación de las secuencias iniciales de la selección por sustitución.

Selección por sustitución

- Tamaño de los bloques ordenados: 7, 4, 3, 5, 5, 1
- Propiedad: para secuencias aleatorias, las secuencias creadas así son aproximadamente el doble del tamaño del heap utilizado.
- Efecto práctico: ganamos una pasada sobre la secuencia (además de pagar log M por elemento en la primera pasada, pero eso no es grave, ya que de la otra manera también pagábamos N/M * M log M).

Fusión Polifásica

- Problema de los anteriores: gran cantidad de dispositivos necesarios: 2P, o como mínimo P+1 (a un costo de duplicar la cantidad de pasadas)
- Cuando se tienen menos cintas, se puede hacer lo siguiente:
  - Distribuir los datos en todas las cintas menos una, aplicando una estrategia de "fusión hasta el vaciado".
  - Cuando una cinta se vacía, se puede rebobinar y usar como cinta de salida

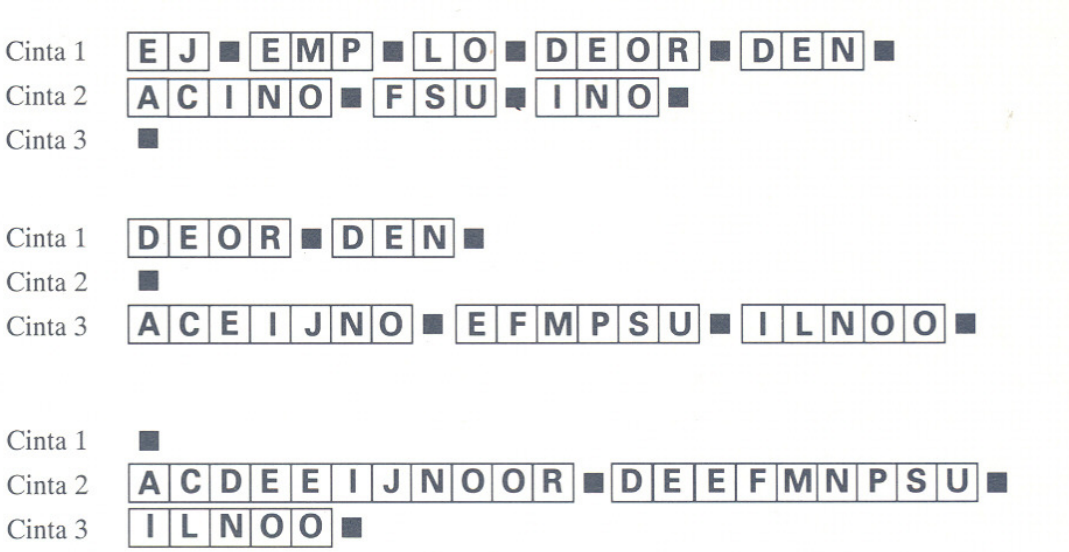


Figura 13.5 Etapas iniciales de una fusión polifásica con tres cintas.

Fusión Polifásica

- Secuencia ejemplo con 6 cintas:

Cinta 1	61	0	31	15	7	3	1	0	1
Cinta 2	0	61	30	14	6	2	0	1	0
Cinta 3	120	59	28	12	4	0	2	1	0
Cinta 4	116	55	24	8	0	4	2	1	0
Cinta 5	108	47	16	0	8	4	2	1	0
Cinta 6	92	31	0	16	8	4	2	1	0

Búsqueda/Diccionarios externa

- Idea: poder encontrar una palabra entre un millón con 2 o 3 accesos a disco.
- Más parecidos a los métodos de memoria principal que los métodos de ordenamiento.
- En Bases de Datos aparece esta problemática (también en versiones mucho más elaboradas: otros métodos).
- Acceso a páginas = bloques contiguos de datos a los que se puede acceder eficazmente (a todos juntos).
  - Modelo simplificado pero efectivo

Métodos

- Acceso Secuencial Indexado (ISAM)
- Árboles B
- Hashing Extensible

Acceso Secuencial Indexado

- Acceso secuencial puro - Ejemplo:
  - EJEMPLODEBUSQUEDAEXTERNA
  - Discos con 3 paginas de 4 registros cada una

Figura 18.1 Acceso secuencial.

- ¿Qué pasa si queremos buscar X? Problemas conocidos
- Solución: tener en cada disco un índice que establezca qué claves pertenecen a las páginas que están en ese disco
- La primera página de cada disco es su índice
- El índice dice cuál es la mayor clave que se encuentra en la página anterior

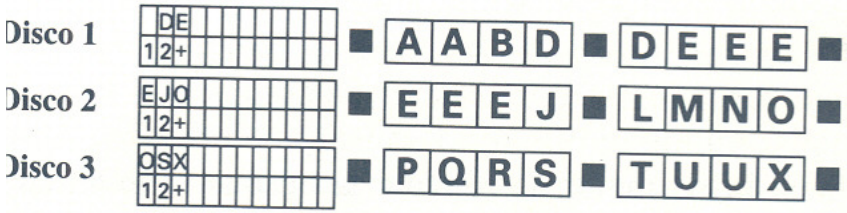


Figura 18.2 Acceso secuencial indexado.

Acceso Secuencial Indexado

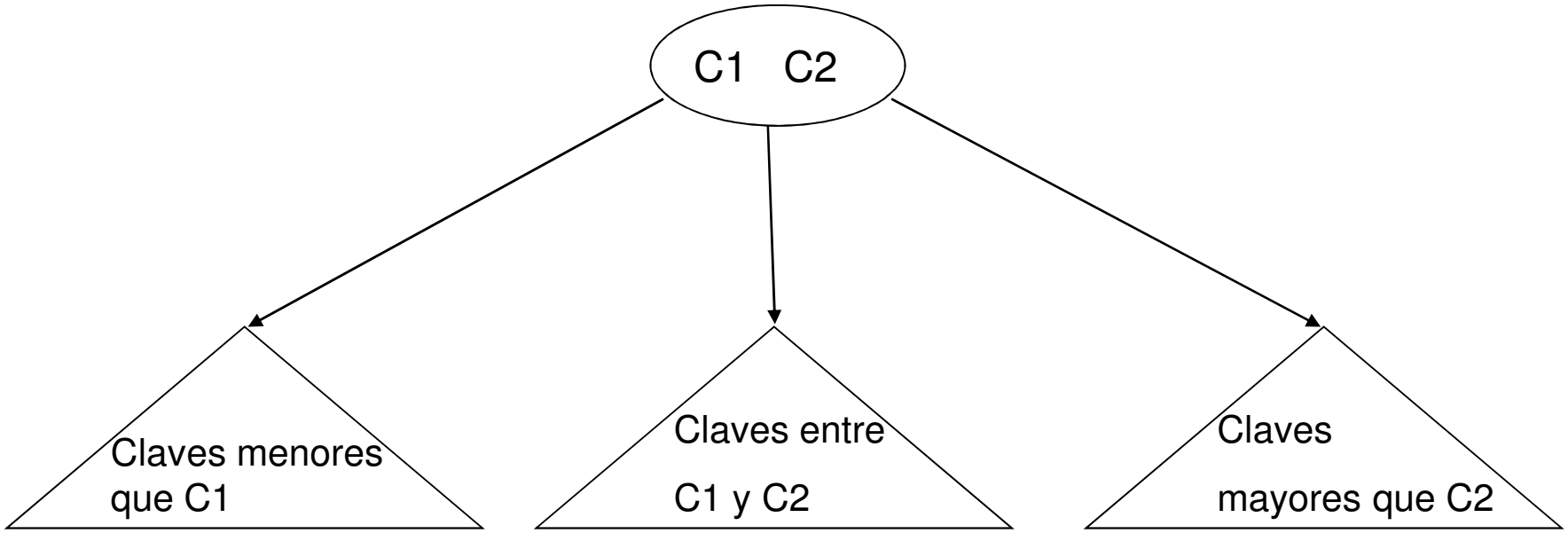
- Se puede acelerar más con un índice maestro, que dice qué claves están en cada disco.
- En el ejemplo anterior, el índice maestro diría que el disco 1 contiene las claves ≤ E, el disco 2 las claves ≤ O (pero no < E), el disco 3 las claves ≤ X (pero no < P)
- El índice maestro puede estar en memoria principal
- Problema de este método: muy rígido, si va a haber inserciones y/o borrados
- Complejidad:
  - Búsqueda O(1)
  - Inserción O(n) pues puede implicar una reorganización completa!

Árboles B

- Son árboles balanceados: todas las hojas están a la misma distancia de la raíz.
- Idea: acceder a una página de memoria por nivel
- Muchas claves por nodo/ mucha ramificación
- Inventados por Bayer & McCreight (1970)
- Variantes/Ejemplo/Antecedente: Árboles 2-3-4

Árboles 2-3-4

- Los nodos pueden contener 1, 2 o 3 claves
  - 1 clave → 2 enlaces: 2-nodos
  - 2 claves → 3 enlaces: 3-nodos
  - 3 claves → 4 enlaces: 4 nodos
- Ejemplo de 3-nodo



Árboles 2-3-4 - Ejemplo

- Claves EJEMPLODEBU

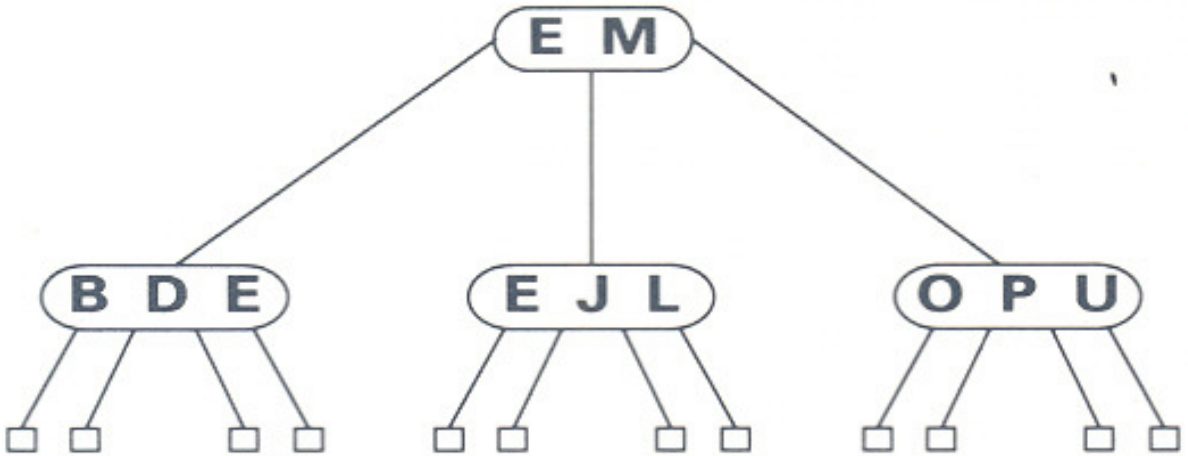


Figura 15.1 Un árbol 2-3-4.

- ¿Qué haríamos si quisiéramos insertar S?

Árboles 2-3-4 - Ejemplo

- inserción de S

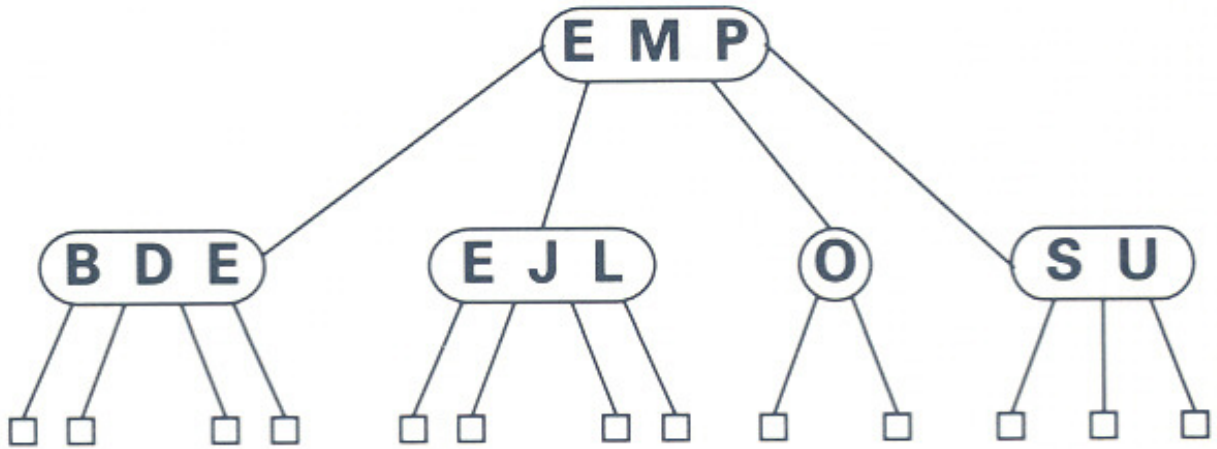


Figura 15.2 Inserción (de S) en un árbol 2-3-4.

- Para mantenerlo balanceado, “partimos” el 4-nodo OPU en dos 2-nodos (O y U) y mandamos la clave intermedia P hacia arriba

Árboles 2-3-4 – Ejemplo

- ¿Qué pasa si el padre también es un 4-nodo?
  - Primera solución: partirlo también a él e ir así hacia arriba
  - Segunda solución: en el camino de descenso, ir partiendo todos los 4 nodos que encontremos
- División de 4-nodos:
  - Usa sólo información local
  - Se puede hacer en tiempo constante

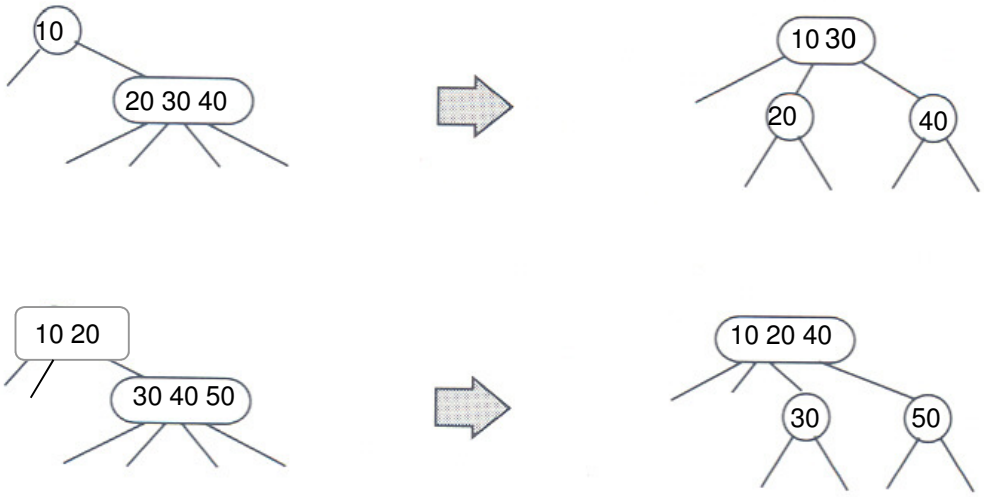
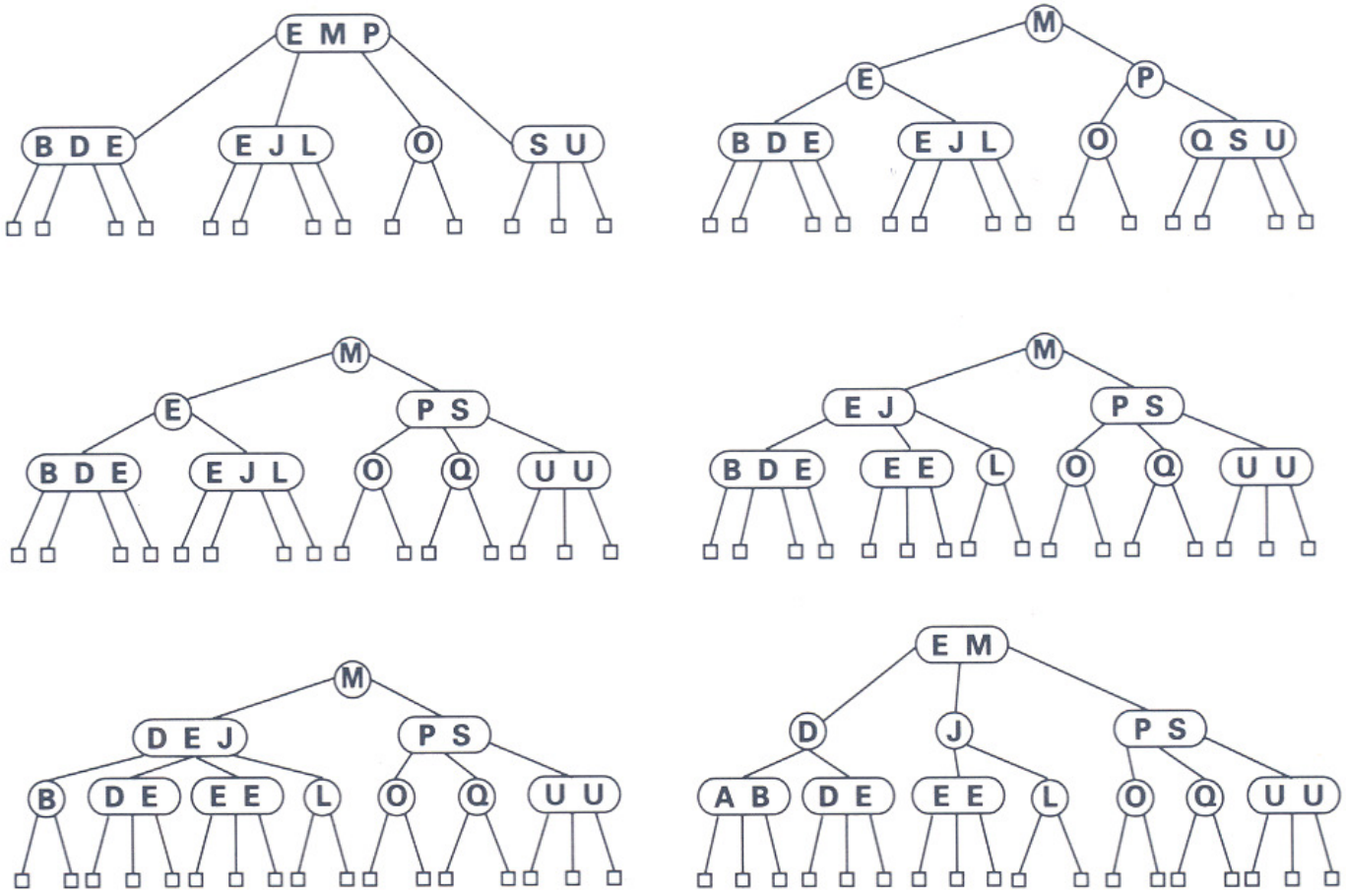


Figura 15.4 División de 4-nodos.

Árboles 2-3-4 - Ejemplo

- Secuencia de inserciones QUEDA



Construcción de un árbol 2-3-4.

Árboles 2-3-4 – Final

- Complejidad de las operaciones
  - Búsqueda O(log n)
  - Inserciones O(log n) (necesitan a lo sumo O(log N) divisiones de nodos)
  - Borrado: también se puede en O(log n) (pensarlo!)
- Problemas de implementación:
  - Algoritmos con cierta complicación
  - Más complicación proveniente del manejo de distintos tipos de nodos
- Implementación de árboles rojo-negros

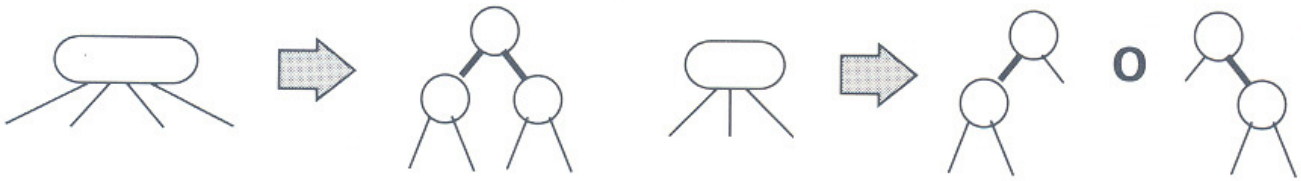


Figura 15.6 Representación rojinegra de 3-nodos y 4-nodos.

Volviendo a los Árboles B

- Similares a los anteriores, pero cada nodo puede tener hasta M-1 claves (o sea hasta M enlaces)
- Para desplazarse de un nodo al siguiente, hay que encontrar el intervalo apropiado de la clave buscada, y salir a través del enlace correspondiente.
- Similarmente a los árboles 2-3-4, es necesario dividir los nodos que están “llenos” a lo largo del camino: cada vez que se encuentra un k-nodo que es padre de un M-nodo, se lo reemplaza por un (k+1)-nodo asociado a dos (M/2)-nodos.
- La idea es que cada nodo corresponda a una página de disco
- Notar que eso incluye claves, datos y enlaces
- Notar también que M grande, implica mayor costo en cada nodo
- En realidad, se puede distinguir entre nodos internos y hojas, que no necesitan el espacio para los índices
- Hay muchas variantes y versiones distintas

Hashing Dinámico o Extensible

- A semejanza de los Árboles B, los registros se almacenan en páginas que, cuando se llenan, se dividen en dos partes
- Como en el acceso secuencial indexado, se mantiene un índice al que se accede para encontrar la página que contiene los registros que concuerdan con la clave
- Idea:
  - Todos los elementos con el mismo valor de h, van a parar a la misma página de memoria
  - Si por causa de una inserción, se llena una página, ésta se divide en 2, los elementos van a una u otra, dependiendo del valor de una nueva función hash.
- En la práctica, se va armando (y desarmando) un árbol de índices, cuyas hojas apuntan al archivo donde está el registro. Si el árbol de índices cabe en memoria principal, entonces con un sólo acceso a disco, alcanza.