Unidad 5: Algoritmos de memoria externa

Ordenamiento y búsqueda

Gibran Fuentes Pineda

Junio 2021

Ordenamiento por distribución para memoria externa (1)

- Se usan $S - 1$ elementos $e_1, \ldots e_{S-1}$ para dividir problema en subarreglos S (cubetas)
- El i-ésimo subarreglo consiste en todos los elementos con un valor entre $[e_{i-1}, e_i)$
- División, ordenamiento y concatenación recursivos de los subarreglos hasta tener todos los elementos ordenados
- La proceso recursivo termina cuando el subarreglo es de B elementos (cabe en un bloque de tamaño B)

Ordenamiento por distribución para memoria externa (2)

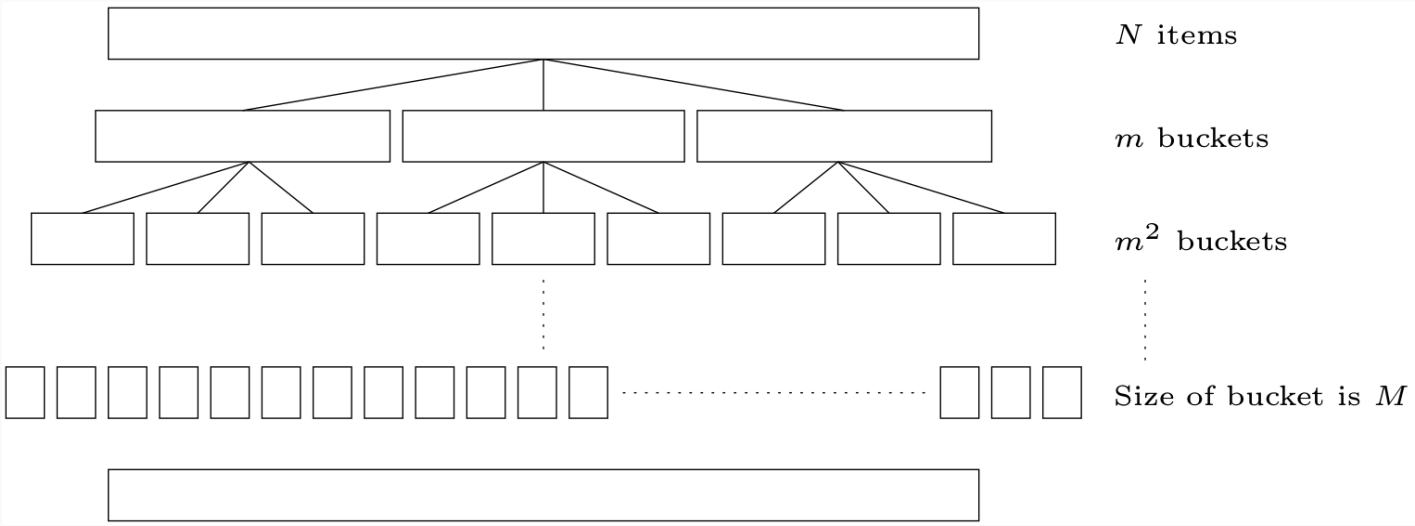

Imagen tomada de Lars Arge: *External-Memory Algorithms with Applications in Geographic Information Systems*, 1996.

Ordenamiento por distribución para memoria externa (3)

- Retos
  - Encontrar elementos que generen subarreglos de tamaño similar
  - Cuando $D > 1$, balancear la carga entre los discos
  - Estrategia de escritura de subarreglos en disco

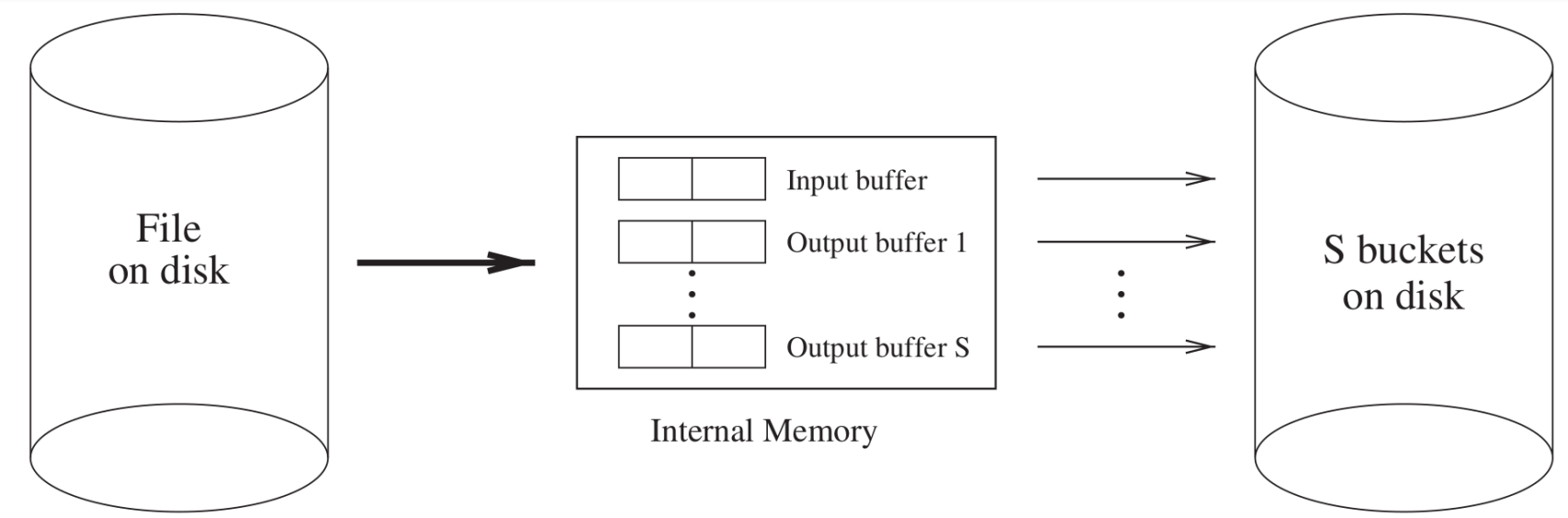

Imagen tomada de Vitter: *Algorithms and Data Structures for External Memory*, 2008.

Ordenamiento por mezcla en memoria interna (1)

- Algoritmo de ordenamiento basado comparaciones diseñado por John von Neumann

- Paradigma *divide y vencerás*
  1. Divide recursivamente el arreglo de N elementos hasta obtener N subarreglos de un solo elemento
  2. Ordenar y mezclar repetidamente los subarreglos ordenados hasta que quede un solo arreglo

- Complejidad $O(N \log N)$: $\log N$ niveles y N comparaciones en cada nivel

Ordenamiento por mezcla en memoria interna (2)

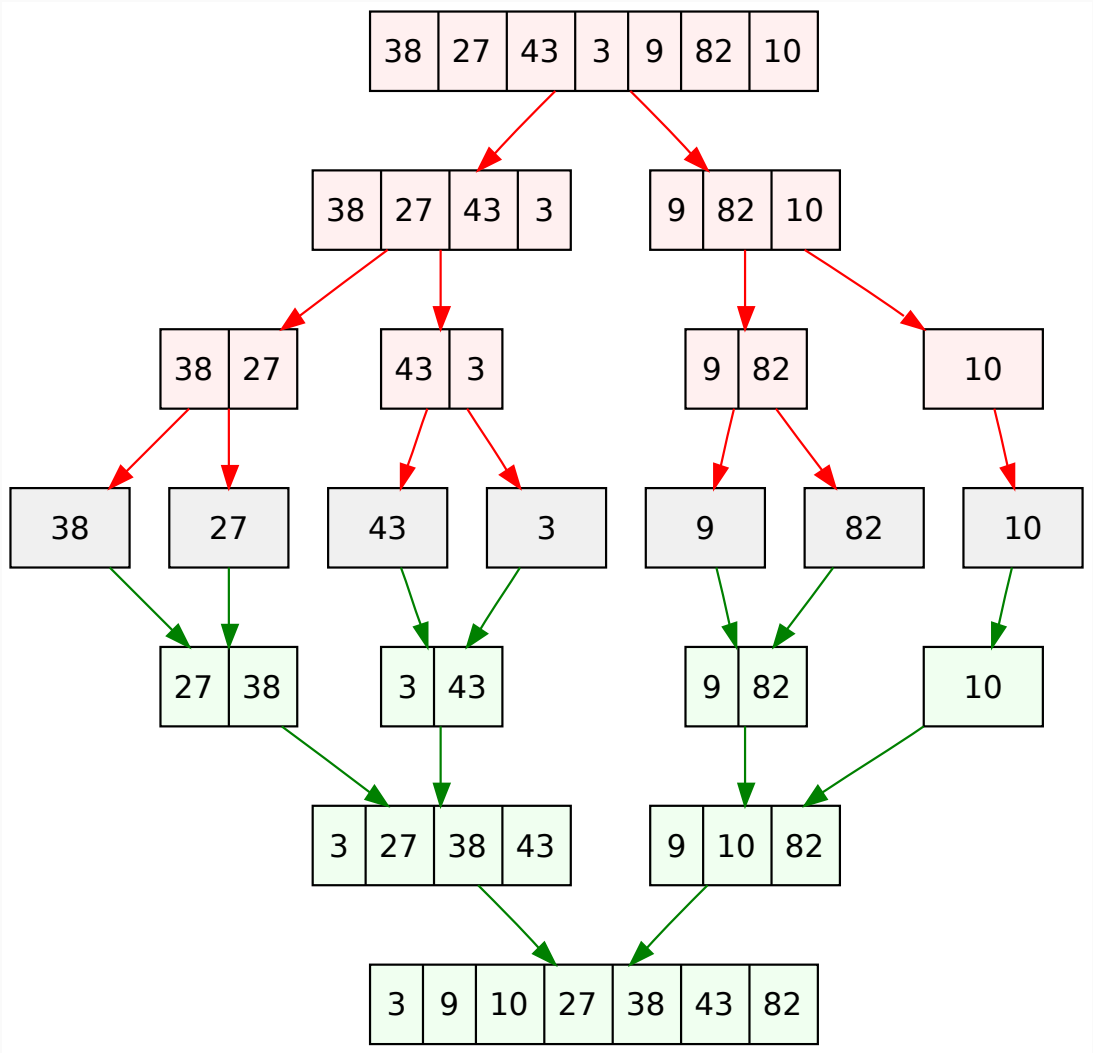

Imagen tomada de Wikipedia (Merge Sort)

Ordenamiento por mezcla en memoria externa: división

- Divide arreglo en N/M subarreglos de tamaño M
- Ordena cada subarreglo de forma independiente (trabajo)
- $2N/B$ transferencias, $2M/B$ por subarreglo
- $N \log M$ comparaciones, $M \log M$ por subarreglo

Ordenamiento por mezcla en memoria externa: mezcla

- Mezcla 2 trabajos R y S de tamaño L en uno solo T de tamaño $2L$ (conocido como *mezcla de 2 caminos*)
  1. Carga primeros bloques $\hat{R}$ y $\hat{S}$ de R y S
  2. Aloja el primer bloque $\hat{T}$ de T
  3. Mientras haya elementos en R y S
     a. Mezcla todos los posibles elementos de $\hat{R}$ y $\hat{S}$ en $\hat{T}$
     b. Si ya no hay elementos en $\hat{R}$ o $\hat{S}$, carga un nuevo bloque
     c. Si $\hat{T}$ se llena, cópialo a T
  4. Copia el resto de elementos de R o S a T

- Transferencias: $\frac{2L}{B}$ lecturas, $\frac{2L}{B}$ escrituras y $2L$ comparaciones

Mezcla de K caminos

- Mezcla K trabajos de forma eficiente (por ej. con montículos mínimos)
- $\frac{2KL}{B}$ transferencias por mezcla
- Maximiza K para reducir transferencias
  - (K lecturas + 1 escritura) B = M
  - Total de transferencias $O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$

Ordenamiento por mezcla en memoria externa (2)

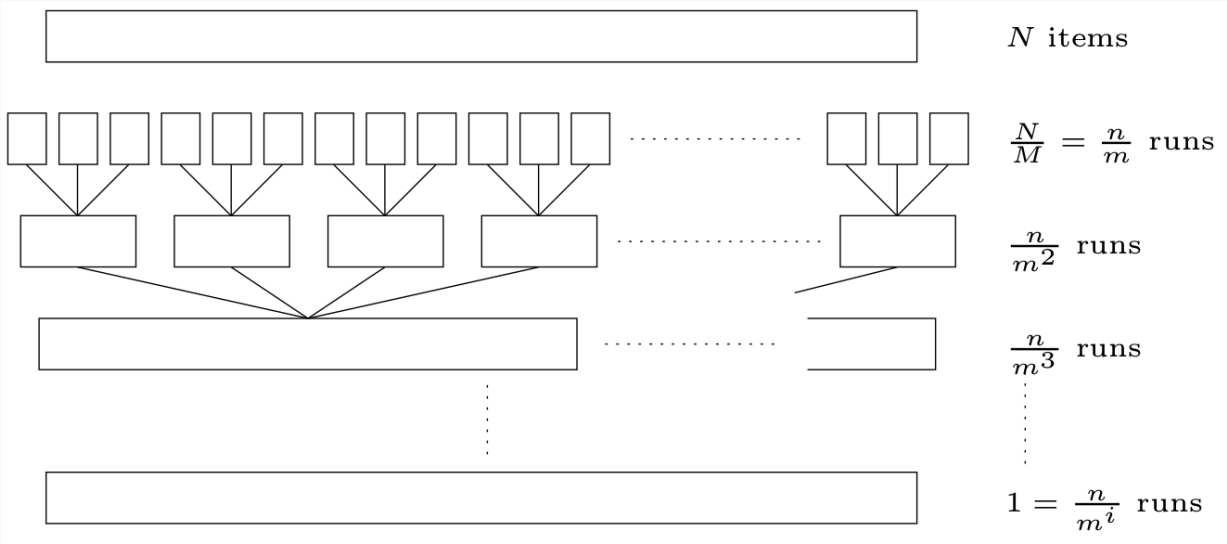

Imagen tomada de Lars Arge: *External-Memory Algorithms with Applications in Geographic Information Systems*, 1996.

Ordenamiento por mezcla en memoria externa (3)

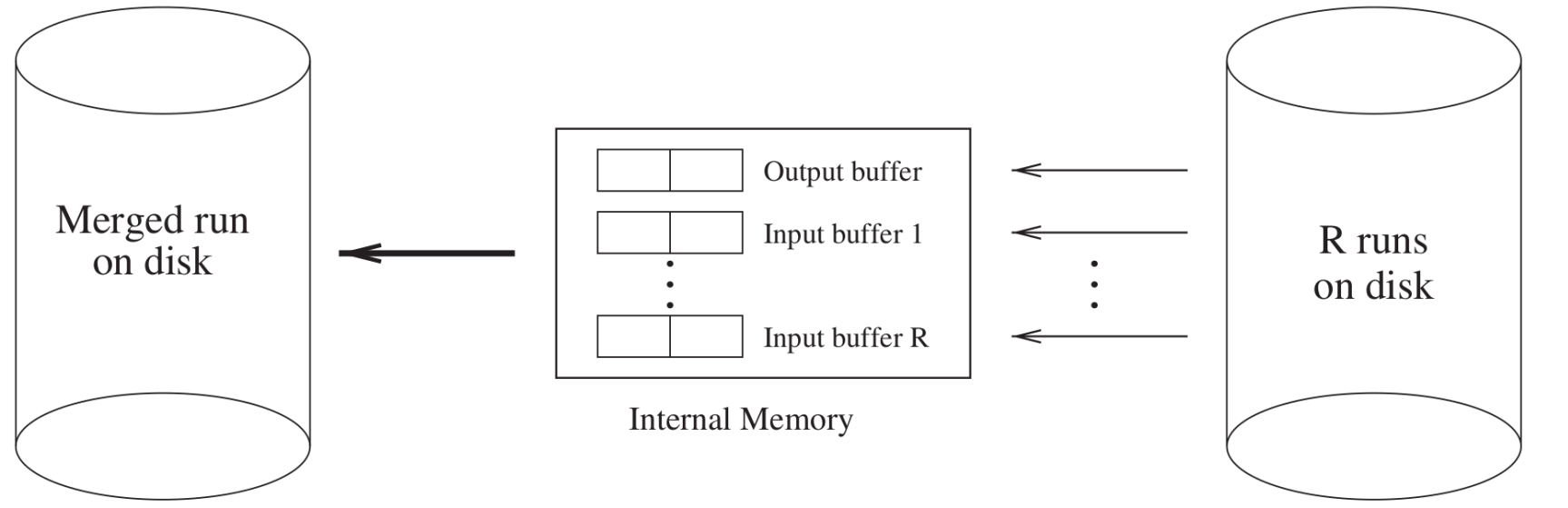

Imagen tomada de Vitter: *Algorithms and Data Structures for External Memory*, 2008.

Ordenamiento en modelo de inconsciente de caché

- Es posible usar el ordenamiento por mezcla de 2 caminos pero no es muy eficiente

- Alternativa más eficiente: ordenamiento de embudo
  1. Divide el arreglo en $K = N^{1/3}$ subarreglos contiguos (cada uno de tamaño $N/K = N^{2/3}$) y los ordena de forma recursiva
  2. Mezcla los K subarreglos ordenados de forma eficiente usando K-embudos[1]

- Ordenamiento de embudo requiere $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ transferencias

[1] Demaine: *Cache-Oblivious Algorithms and Data Structures*, 2002

Búsqueda de elementos

- Dado un arreglo de N elementos, determinar si un elemento de consulta q está en el arreglo
- Recorriendo todos los elementos tomaría $O(N/B)$ operaciones E/S
- Si están ordenados podríamos usar búsqueda binaria y tomaría $O\left(\log_2 \frac{N}{B}\right)$ operaciones E/S

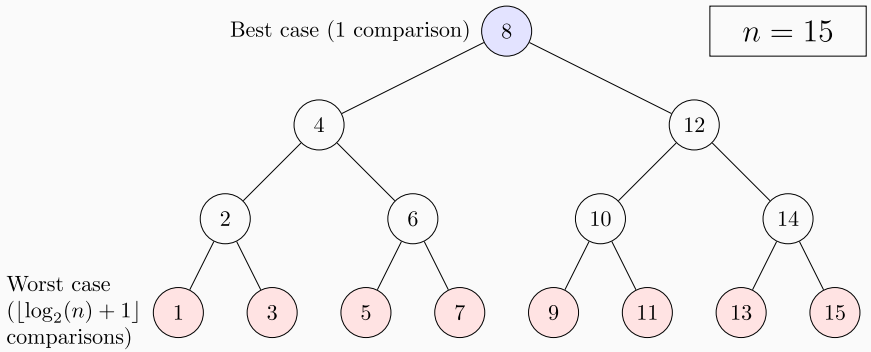

Imagen tomada de Wikipedia (Binary search algorithm)

Algoritmos de búsqueda para memoria externa

- *Hashing*
  - Métodos con directorios
  - Métodos sin directorios
- Algoritmos de árboles
  - Árboles-B y variantes
  - Árboles de *buffer*

Hashing extendible

- Directorio
  - Consiste de una tabla con 2^d referencias, donde $d \geq 0$
  - La ubicación en la tabla de un elemento está dada por los d bits menos significativos de su valor *hash*

$$hash_d(x) = hash(x) \mod 2^d$$

  - Cada ubicación de la tabla contiene una referencia a un bloque donde se almacenan los elementos
  - Una ubicación comparte el mismo bloque con las ubicaciones con los mismos k bits menos significativos

$$hash_k(x) = hash_d(x) \mod 2^k$$

- Búsquedas requieren 2 operaciones E/S si el directorio reside en disco
  - Acceso a ubicación de la tabla
  - Acceso a bloque asignado a ubicación

Profundidad global y local

- d es la profundidad global
  - Se elige el número más pequeño con el cual cada ubicación de la tabla tiene a lo mucho B elementos
- k es la profundidad local
  - Se elige el número más pequeño para que los elementos asignados entren en un solo bloque
- Cuando un bloque se desborda, d y k se recalculan
  - Se divide el bloque y se redistribuyen sus elementos
  - El tamaño del directorio crece (se duplica cuando d se incremente en 1)
  - Permite adaptarse al crecimiento en el número de elementos N

Ejemplo de hashing extendible con bloque $B = 3$

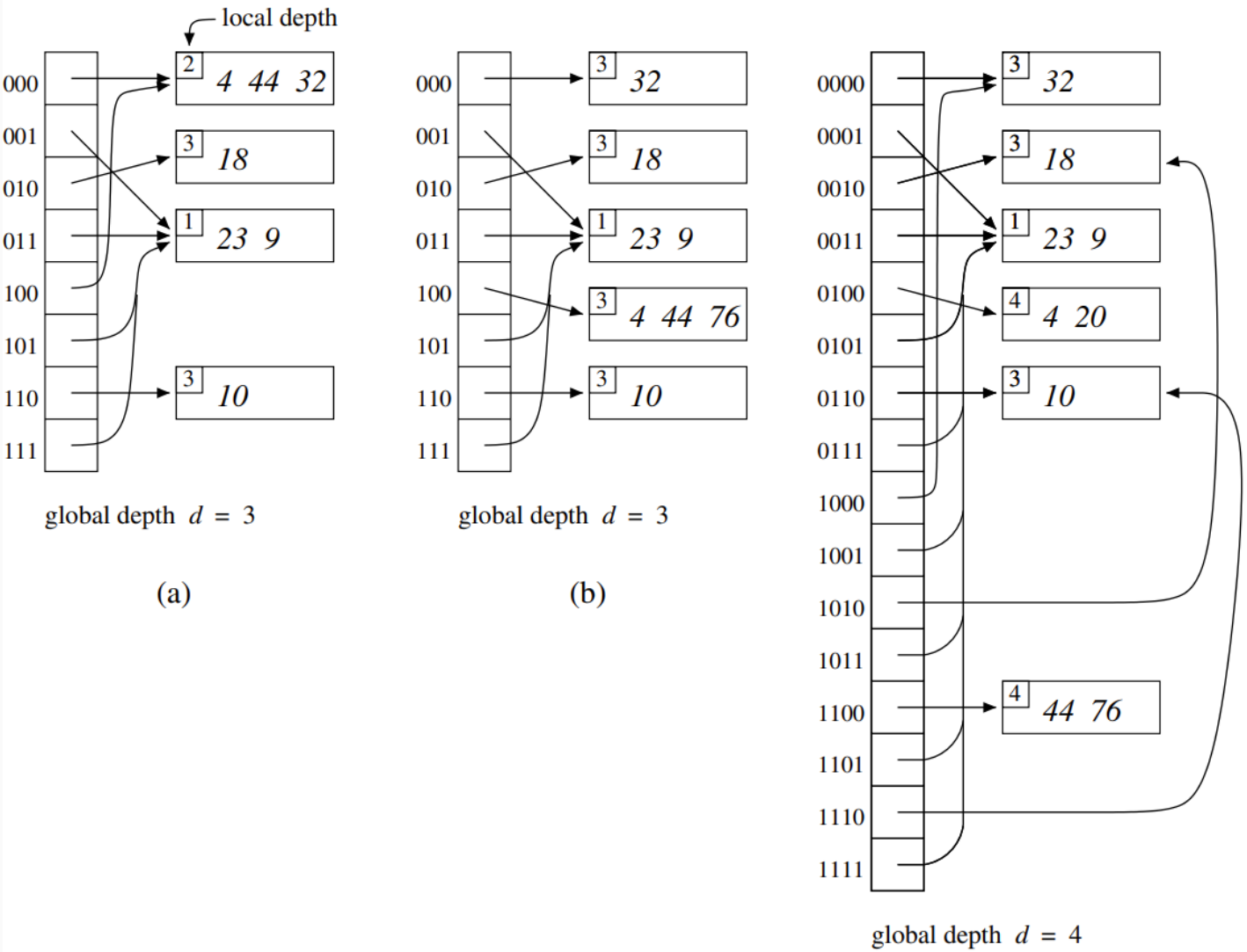

Imagen tomada de Vitter. *Algorithms and Data Structures for External Memory*, 2008.

Hashing linear

- No necesita directorio: cada ubicación de la tabla corresponde a un bloque donde se almacenan elementos
- Cuando hay desbordamiento, un bloque predefinido de la tabla se divide y sus elementos se redistribuyen
  - El bloque que se divide no es necesariamente el que se desbordó, por lo que los bloques desbordados necesitan listas auxiliares para almacenar elementos desbordados
- Búsquedas requieren usualmente una sola operación E/S

Árboles-B

- Árboles balanceados que mantienen sus elementos ordenados
  - Operaciones: escaneo secuencial, inserción, eliminación y búsqueda
- Definición de árbol-B de orden m[2]
  - Cada nodo tiene a lo mucho m hijos
  - Cada nodo no hoja tiene al menos $\lceil m/2 \rceil$ hijos (excepto la raíz)
  - Si no es hoja, el nodo raíz tiene al menos 2 hijos
  - Un nodo no hoja con k hijos contiene $k - 1$ valores de separación (llaves)
  - Los nodos hoja están en el mismo nivel del árbol
  - Elementos de un nodo están ordenados

[2] De acuerdo a D. Knuth: *Sorting and Searching, The Art of Computer Programming*, vol. 3.

Ejemplo de árbol-B

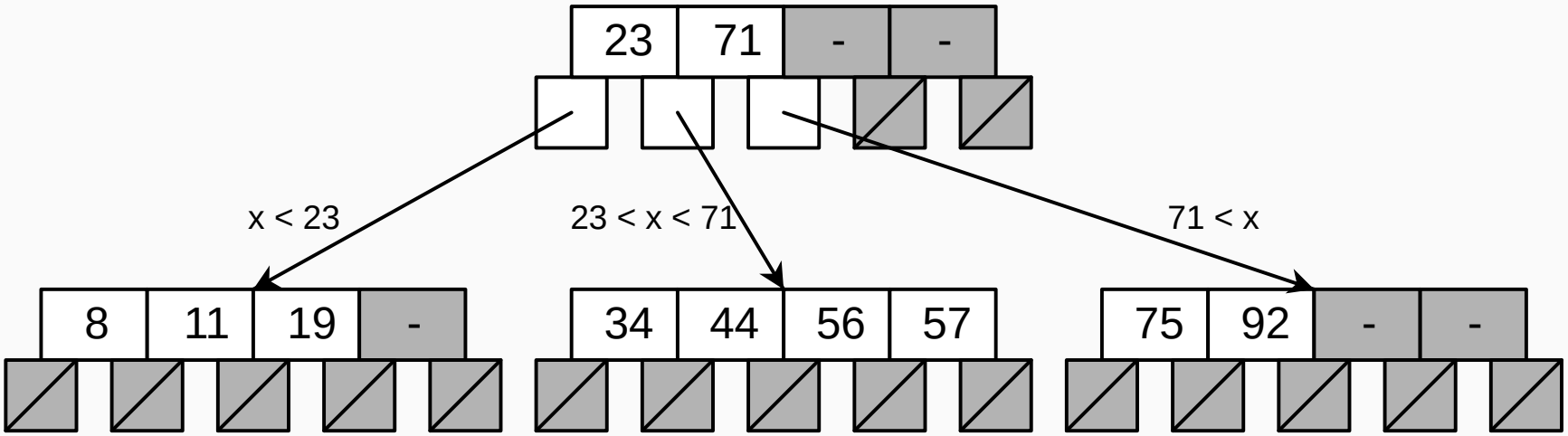

Imagen tomada de Wikipedia (Árbol-B)

Inserción en árboles-B

- Se hacen de nodos hoja a raíz
  1. Se encuentra el nodo hoja donde insertar el elemento
  2. Si el nodo hoja tiene menos elementos que el máximo, se inserta en ese nodo
  3. En caso contrario, se divide en 2
     a) Escoge mediana de elementos del nodo y nuevo elemento
     b) Valores menores a mediana se agregan en hijo izquierdo y mayores en hijo derecho
     c) La mediana se inserta en nodo padre, si excede capacidad se divide en 2 (se repite a)–c))
     d) Si se llega a raíz, se crea nueva

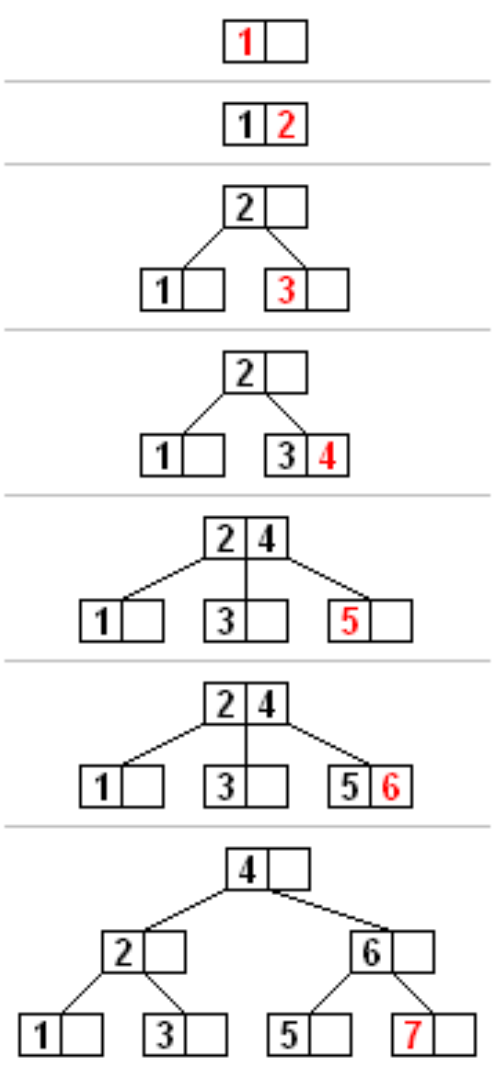

Imagen tomada de Wikipedia (B-tree)

Eliminación en árboles-B

- Eliminación en un nodo hoja
  1. Se busca el elemento a eliminar
  2. Si el valor se encuentra en un nodo hoja, se elimina
  3. Si queda con muy pocos elementos, se rebalancea

- Eliminación en un nodo no hoja (interno)
  1. Encuentra un nuevo separador que substituya al elemento
  2. Toma elemento más grande del hijo izquierdo o el más pequeño del derecho, elimínalo del sub-árbol correspondiente y reemplaza con este el elemento a eliminar
  3. Rebalancea sub-árbol, si es necesario

Búsqueda en árboles-B

1. Se recorre el árbol de la raíz a las hojas

2. Si el elemento está almacenado en el nodo la búsqueda termina y si no se continúa con el sub-árbol correspondiente al valor a buscar

3. Si estamos en un nodo hoja y el elemento no está almacenado, entonces no se encontró

Árboles-B+

- Árboles-B+ son una variante en la cual todos los elementos se almacenan en las hojas y los nodos internos solo almacenan llaves y referencias

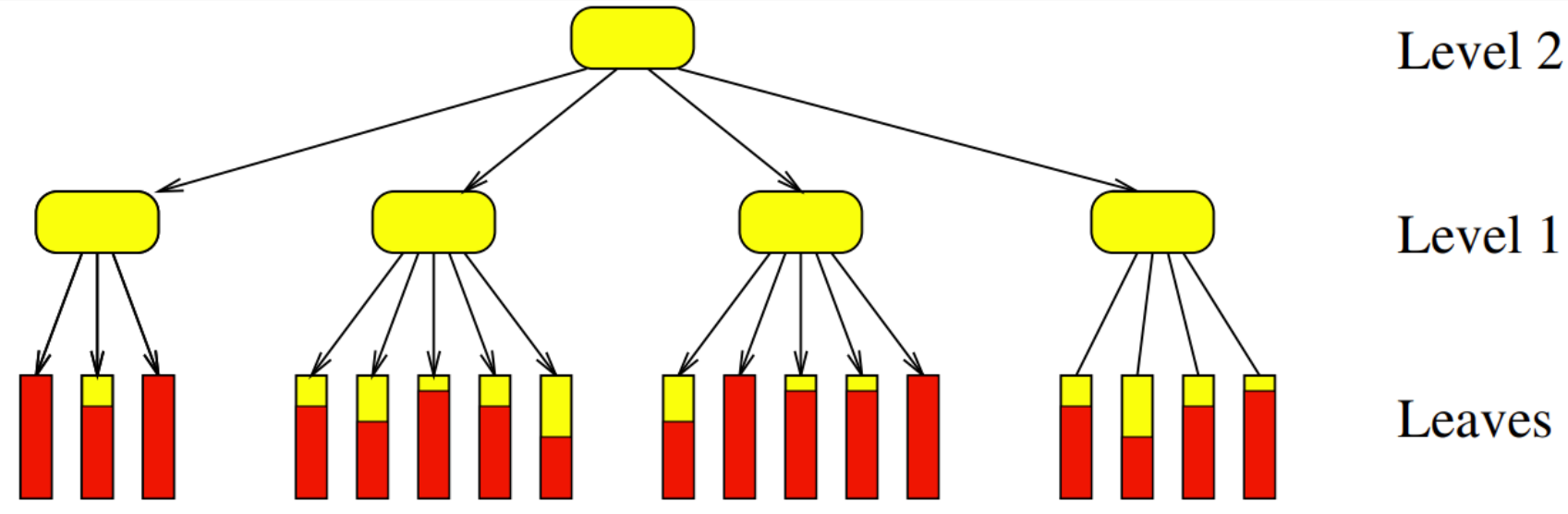

Imagen tomada de Vitter. *Algorithms and Data Structures for External Memory*, 2008.