CIS 5930 Advanced Topics in Data Management

Feifei Li

Fall 2008

(Many slides were made available by Ke Yi)
External Memory Data Structures

- Names:
  - I/O-efficient data structures
  - Disk-based data structures (index structures) used in DB
  - Disk-resilient data structures (index structures) used in DB
  - Secondary indexes used in DB

- Other Data structures
  - Queue, stack
    * O(N/B) space, O(1/B) push, O(1/B) pop
  - Priority queue
    * O(N/B) space, O(1/B \cdot \log_{M/B} N/B) insert, delete-max

Mainly used in algorithms
External Memory Data Structures

• General-purpose data structures
  – Space: linear or near-linear (very important)
  – Query: logarithmic in B or 2 for any query (very important)
  – Update: logarithmic in B or 2 (important)
• In some sense, more useful than I/O-algorithms
  – Structure stored in disk most of the time
  – DB typically maintains many data structures for many different data sets: can’t load all of them to memory
  – Nearly all index structures in large DB are disk based
External Search Trees

- **Binary search tree:**
  - Standard method for search among \( N \) elements
  - We assume elements in leaves

\[
O(\log_2 N) \Rightarrow \text{Search in } O(\log_2 N) \text{ I/Os}
\]
\[
\Rightarrow \text{Rangesearch in } O(\log_2 N + T) \text{ I/Os}
\]
External Search Trees

- **Bottom-up BFS blocking:**
  - Block height $O(\log_2 N) / O(\log_2 B) = O(\log_B N)$
  - Output elements blocked

\[\downarrow\]

- **Optimal:** $O(N/B)$ space and $O(\log_B N + T/B)$ query
External Search Trees

- Maintaining BFS blocking during updates?
  - Balance normally maintained in search trees using rotations

- Seems very difficult to maintain BFS blocking during rotation
  - Also need to make sure output (leaves) is blocked!
B-trees

• BFS-blocking naturally corresponds to tree with fan-out $\Theta(B)$

• B-trees balanced by allowing node degree to vary
  – Rebalancing performed by splitting and merging nodes
(a,b)-tree

- $T$ is an $(a,b)$-tree ($a \geq 2$ and $b \geq 2a - 1$)
  - All leaves on the same level (contain between $a$ and $b$ elements)
  - Except for the root, all nodes have degree between $a$ and $b$
  - Root has degree between 2 and $b$

- $(a,b)$-tree uses linear space and has height $O(\log_a N)$

Choosing $a, b = \Theta(B)$ each node/leaf stored in one disk block

$O(N/B)$ space and $O(\log_B N + T/B)$ query
**(a,b)-Tree Insert**

- **Insert:**

  Search and insert element in leaf $v$
  
  DO $v$ has $b+1$ elements/children

  **Split $v$:**
  
  - make nodes $v'$ and $v''$ with
    
    $\left\lfloor \frac{b+1}{2} \right\rfloor \leq b$ and $\left\lceil \frac{b+1}{2} \right\rceil \geq a$ elements
  
  - insert element (ref) in $parent(v)$
  
    (make new root if necessary)

  $v = parent(v)$

- Insert touch $O(\log_a N)$ nodes
$(a,b)$-Tree Insert
(a,b)-Tree Delete

- **Delete:**

  Search and delete element from leaf $v$
  
  **DO** $v$ has $a$-1 elements/children
  
  **Fuse** $v$ with sibling $v'$:
    - move children of $v'$ to $v$
    - delete element (ref) from $parent(v)$
      - (delete root if necessary)
    
  If $v$ has $b$ (and $\leq a+b-1 < 2b$) children split $v$
  
  $v = parent(v)$

- Delete touch $O(\log_a N)$ nodes
(a,b)-Tree Delete
External Searching: B-Tree

- Each node (except root) has fan-out between $B/2$ and $B$
- Size: $O(N/B)$ blocks on disk
- Search: $O(\log_B N)$ I/Os following a root-to-leaf path
- Insertion and deletion: $O(\log_B N)$ I/Os
Summary/Conclusion: B-tree

- **B-trees**: $(a,b)$-trees with $a,b = \Theta(B)$
  - $O(N/B)$ space
  - $O(\log_B N+T/B)$ query
  - $O(\log_B N)$ update

- **B-trees with elements in the leaves** sometimes called **B⁺-tree**
  - Now B-tree and B⁺-tree are synonyms

- **Construction in** $O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os
  - Sort elements and construct leaves
  - Build tree level-by-level bottom-up
2D Range Searching
Quadtree

Adaptive quadtree where no square contains more than 1 particle

- No worst-case bound!
- Hard to block!
• **kd-tree:**
  – Recursive subdivision of point-set into two half using vertical/horizontal line
  – Horizontal line on even levels, vertical on uneven levels
  – One point in each leaf

Linear space and logarithmic height
kd-Tree: Query

- **Query**
  - Recursively visit nodes corresponding to regions intersecting query
  - Report point in trees/nodes completely contained in query

- **Query analysis**
  - Horizontal line intersect $Q(N) = 2 + 2Q(N/4) = O(\sqrt{N})$ regions
  - Query covers $T$ regions
  $\Rightarrow O(\sqrt{N} + T)$ I/Os worst-case
The `$\text{kdB-tree}$` has the following properties:

- **kdB-tree:**
  - Bottom-up BFS blocking
  - Same as B-tree

- **Query** as before
  - Analysis as before but each region now contains $\Theta(B)$ points

The I/O query complexity is $O(\sqrt{\frac{N}{B}} + \frac{T}{B})$.
Construction of kdB-tree

- Simple $O \left( \frac{N}{B} \log \left( \frac{N}{B} \right) \right)$ algorithm
  - Find median of $y$-coordinates (construct root)
  - Distribute point based on median
  - Recursively build subtrees
  - Construct BFS-blocking top-down (can compute the height in advance)

- Idea in improved $O \left( \frac{N}{B} \log \left( \frac{M}{B} \frac{N}{B} \right) \right)$ algorithm
  - Construct $\log \sqrt{\frac{M}{B}}$ levels at a time using $O(N/B)$ I/Os
**Construction of kdB-tree**

- Sort $N$ points by $x$- and by $y$-coordinates using $O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os.
- Building $\log \sqrt{\frac{M}{B}}$ levels ($\sqrt{\frac{M}{B}}$ nodes) in $O(N/B)$ I/Os:
  1. Construct $\sqrt{\frac{M}{B}}$ by $\sqrt{\frac{M}{B}}$ grid with $\sqrt{\frac{N}{M/B}}$ points in each slab.
  2. Count number of points in each grid cell and store in memory.
  3. Find slab $s$ with median $x$-coordinate.
  4. Scan slab $s$ to find median $x$-coordinate and construct node.
  5. Split slab containing median $x$-coordinate and update counts.
  6. Recurse on each side of median $x$-coordinate using grid (step 3).

$\Rightarrow$ Grid grows to $\frac{M}{B} + \sqrt{\frac{M}{B}} \cdot \sqrt{\frac{M}{B}} = \Theta\left(\frac{M}{B}\right)$ during algorithm.

$\Rightarrow$ Each node constructed in $O(\frac{N}{\sqrt{\frac{M}{B} \cdot B}})$ I/Os.
• kdB-tree:
  – Linear space
  – Query in $O(\sqrt{\frac{N}{B}} + \frac{T}{B})$ I/Os
  – Construction in $O(\text{sort}(N))$ I/Os
  – Height $O(\log_B N)$
• Dynamic?
  – Difficult to do splits/merges or rotations …