CSE 373 Lecture 9: B-Trees and Binary Heaps

- ✦ Today's Topics:
  - More on B-Trees
    - Insert/Delete Examples and Run Time Analysis
  - Introduction to Heaps and Priority Queues
    - Binary Heaps
- ✦ Covered in Chapters 4 and 6 in the text

B-Trees

B-Trees are multi-way search trees commonly used in database systems or other applications where data is stored externally on disks and keeping the tree shallow is important.

A B-Tree of order M has the following properties:

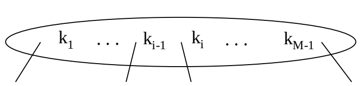
1. The root is either a leaf or has between 2 and M children.
2. All nonleaf nodes (except the root) have between $\lceil M/2 \rceil$ and M children.
3. All leaves are at the same depth.

All data records are stored at the leaves.
Leaves store between $\lceil M/2 \rceil$ and M data records.

B-Tree Details

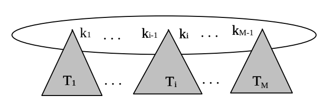
Each internal node of a B-tree has:

- Between $\lceil M/2 \rceil$ and M children.
- up to M-1 keys $k_1 < k_2 < ... < k_{M-1}$



Keys are ordered so that:
$k_1 < k_2 < ... < k_{M-1}$

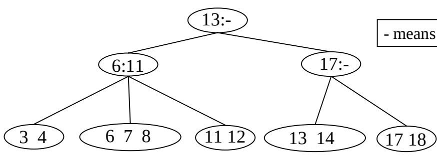
Properties of B-Trees



Children of each internal node are "between" the items in that node.
Suppose subtree T_i is the *i*th child of the node:
all keys in T_i must be between keys k_{i-1} and k_i
i.e. $k_{i-1} \leq T_i < k_i$
k_{i-1} is the smallest key in T_i
All keys in first subtree $T_1 < k_1$
All keys in last subtree $T_M \geq k_{M-1}$

Example: Searching in B-trees

- B-tree of order 3: also known as 2-3 tree (2 to 3 children)

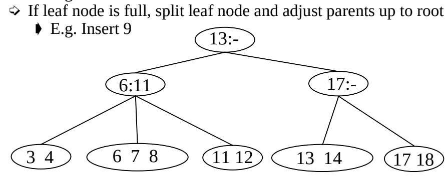


- means empty slot

- Examples: Search for 9, 14, 12
- Note: If leaf nodes are connected as a Linked List, B-tree is called a B+ tree – Allows sorted list to be accessed easily

Inserting and Deleting Items in B-Trees

- Insert X: Do a Find on X and find appropriate leaf node
  - If leaf node is not full, fill in empty slot with X
    - E.g. Insert 5
  - If leaf node is full, split leaf node and adjust parents up to root node
    - E.g. Insert 9



- Delete X: Do a Find on X and delete value from leaf node
  - May have to combine leaf nodes and adjust parents up to root node
    - E.g. Delete 17 (after Insert 9)

Run Time Analysis of B-Tree Operations

- For a B-Tree of order M
  - Each internal node has up to M-1 keys to search
  - Each internal node has between $\lceil M/2 \rceil$ and M children
  - Depth of B-Tree storing N items is $O(\log_{\lceil M/2 \rceil} N)$
- Find: Run time is:
  - O(log M) to binary search which branch to take at each node
  - Total time to find an item is O(depth*log M) = O(log N)

Run Time Analysis of B-Tree Operations

- For a B-Tree of order M
  - Depth of B-Tree storing N items is $O(\log_{\lceil M/2 \rceil} N)$
- Find: Run time is:
  - Total time to find an item is O(depth*log M) = O(log N)
- Insert and Delete: Run time is:
  - O(M) to handle splitting or combining keys in nodes
  - Total time is O(depth*M) = $O((\log N/\log\lceil M/2 \rceil)*M)$ = O((M/log M)*log N)
- Tree in internal memory → M = 3 or 4
- Tree on Disk → M = 32 to 256. Interior and leaf nodes fit on 1 disk block.
  - Depth = 2 or 3 → allows very fast access to data in database systems.

Summary of Search Trees

- Problem with Search Trees: Must keep tree balanced to allow fast access to stored items
- AVL trees: Insert/Delete operations keep tree balanced
- Splay trees: Repeated Find operations produce balanced trees
- Multi-way search trees (e.g. B-Trees): More than two children per node allows shallow trees; all leaves are at the same depth keeping tree balanced at all times

A New Problem…

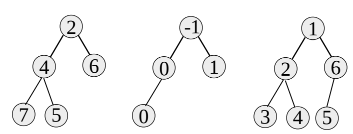
- Instead of finding any item (as in a search tree), suppose we want to find only the smallest (highest priority) item quickly. Examples:
  - Operating system needs to schedule jobs according to priority
  - Doctors in ER take patients according to severity of injuries
  - Event simulation (bank customers arriving and departing, ordered according to when the event happened)
- We want an ADT that can efficiently perform:
  - FindMin (or DeleteMin)
  - Insert
- What if we use…
  - Lists: If sorted, what is the run time for Insert/DeleteMin? Unsorted?
  - Binary Search Trees: What is the run time for Insert/DeleteMin?

Using the Data Structures we know…

- Suppose we have N items.
- Lists
  - If sorted: DeleteMin is O(1) but Insert is O(N)
  - If not sorted: Insert is O(1) but DeleteMin is O(N)
- Binary Search Trees (BSTs)
  - Insert is O(log N) and DeleteMin is O(log N)
- BSTs look good but…
  - BSTs are designed to be efficient for Find, not just FindMin
  - We only need FindMin/DeleteMin
- We can do better than BSTs!
  - O(1) FindMin and O(log N) Insert
  - How?

Heaps

- A binary heap is a binary tree that is:
  1. Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  2. Satisfies the heap order property: every node is smaller than (or equal to) its children
- Therefore, the root node is always the smallest in a heap



Which of these is not a heap?

Next Class:

More Heaps

To Do:

Read Chapter 6

Homework # 2 due on Monday

Have a great weekend!