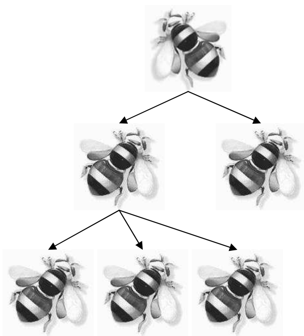
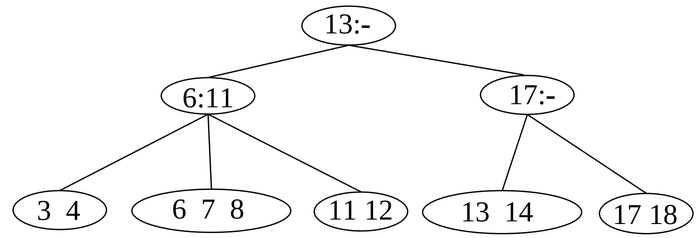
B-Trees and Rank Trees



CSE 326
Data Structures
Unit 7
Reading: Section 4.7
(B trees only)

Beyond Binary Search Trees: Multi-Way Trees

- Example: B-tree of order 3 has 2 or 3 children per node



- Search for 8

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:

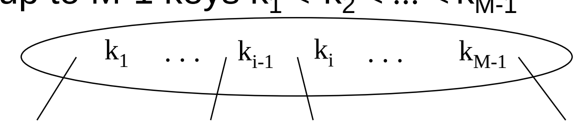
1. The root is either a leaf or has between 2 and M children.
2. All nonleaf nodes (except the root) have between $\lceil M/2 \rceil$ and M children.
3. All leaves are at the same depth.

All data records are stored at the leaves.
Internal nodes have “keys” guiding to the leaves.
Leaves store between $\lceil M/2 \rceil$ and M data records.

B-Tree Details

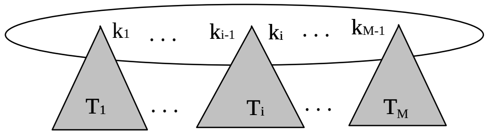
Each (non-leaf) internal node of a B-tree has:

› 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:

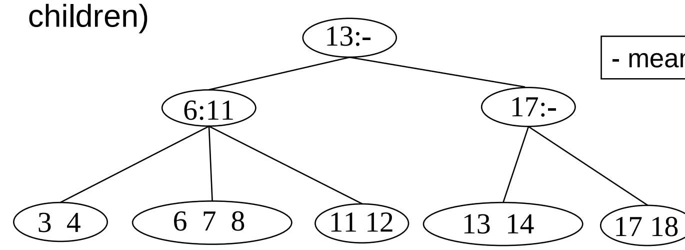
all keys in T_i must be between keys k_{i-1} and k_i

i.e. $k_{i-1} \le T_i < k_i$

k_{i-1} is the smallest key in T_i
All keys in first subtree $T_1 < k_1$
All keys in last subtree $T_M \ge k_{M-1}$

Example: Searching in B-trees

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

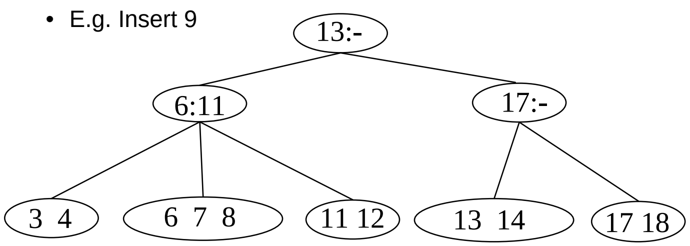


- means empty slot

- 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

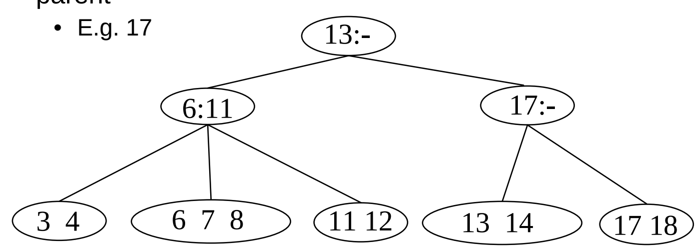
Inserting into B-Trees

- 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



Deleting From B-Trees

- Delete X : Do a find and remove from leaf
  - › Leaf underflows – borrow from a neighbor
    - E.g. 11
  - › Leaf underflows and can't borrow – merge nodes, delete parent
    - E.g. 17



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. But M is small compared to N.
  - Total time to find an item is O(depth*log M) = O(log N)

Storing Additional Data in a Tree Node

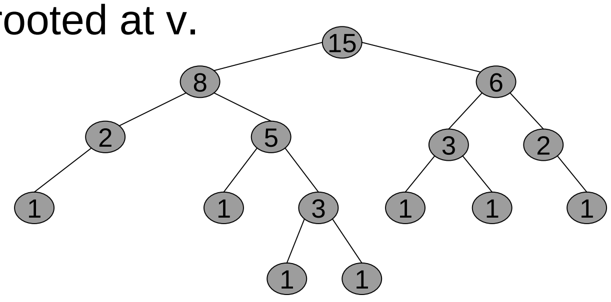
- In many cases, it is useful to store additional data in each node. This information can be used to implement efficiently additional operations.
- Definition: The rank of a value x in a set S is the location of x in a sorted list of the elements of S.
  - Example rank(5) in {8,2,7,5} is 2.
- Problem 1: rank(x) in O(log N)
- Problem 2: sum of elements smaller than x in O(log N).

Storing Additional Data in a Tree Node

- We will solve these problems by storing additional data in the tree nodes.
- First – we will see a solution in O(Tree depth)
- Next – we will see how to update and use the additional data in a balanced search tree.

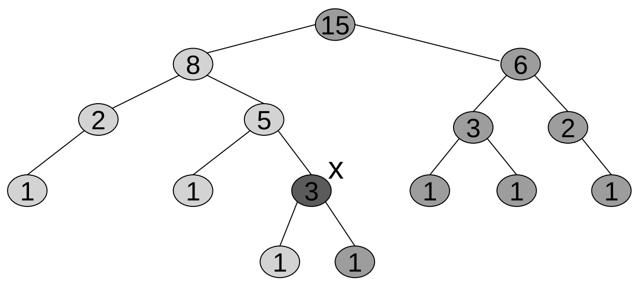
Rank Tree

- A rank tree is a tree in which we keep in each node v (in addition to all other data) the number, n(v), of nodes in the subtree rooted at v.



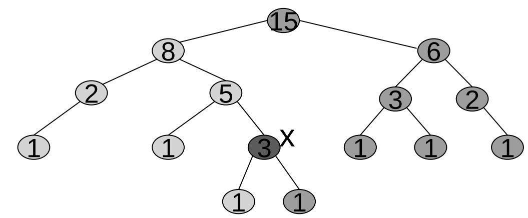
Rank Tree

- What is the rank of x?



Where in the tree are the values smaller than x?

Rank Tree



Consider the path, p, from the root to x.

rank(x) = $\sum_{w}$n(w) + |{v in p and v<x}| ,

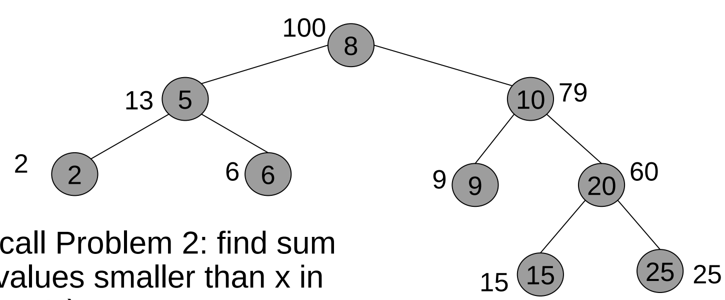
w is a left child of a node with value v≤x in p.

In the example, rank(x) = 2+1+1+ 2 = 6.

Time to calculate rank(x) is O(Tree depth)

Storing additional data

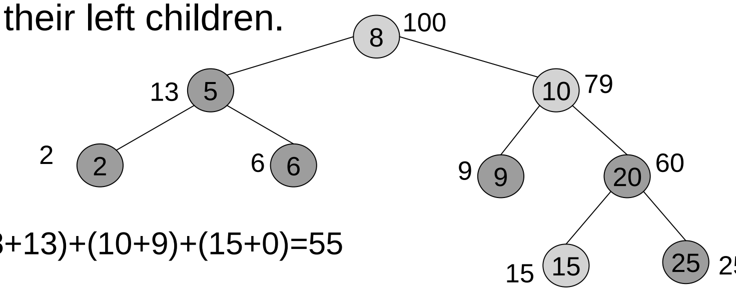
- Assume that we keep in each node v (in addition to all other data) the sum, s(v), of all the values in the subtree rooted at v.



Recall Problem 2: find sum of values smaller than x in O(log N).

Storing additional data

- Example: what is the sum of values smaller than 17?
- Sum the values along the search path for 17, that are smaller than 17 + the values stored in their left children.



(8+13)+(10+9)+(15+0)=55

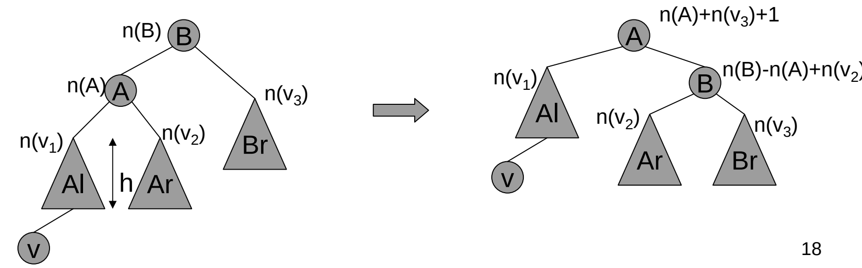
Balanced Rank Tree

- In order to implement rank(x) in O(log N) we update the value n(v) for each node for which this value is affected by insertions/deletions/rotations/any other balancing operations.
- The challenge – make these updates efficiently.
- We will see as example the way n(v) is updated in AVL trees.

Updating the additional data

- How do we update n(v)?

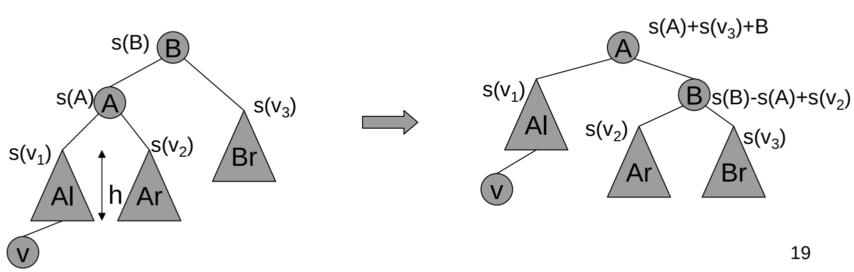
1. When inserting a node, increase by 1 all the values n(v) along the insertion path from the root to the leaf.
2. During rotations, update the values as needed: example: LL rotation



Updating the additional data

- How do we update s(v) [sum of values in subtree]?

1. When inserting a value x, increase by x all the values s(v) along the insertion path from the root to the leaf.
2. During rotations, update the values as needed: example: LL rotation



Exercise

Suggest a data structure that will support the following operations, each in O(log n), where n is the number of elements in the structure at the time of the operation.

The elements in the structure are integral number, each element is colored green or red.

•Insert(i) – add the number i to the structure (if it is not there yet). A new number is added as green.

•PaintRed(i) – if i is in the structure, and it is green, color it red.

•FindDiff(k) – return the difference between the sum of the green elements that are smaller than k, and the sum of the red elements that are smaller than k.

Summary of Search Trees

- Problem with Binary 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
- By storing additional data in tree nodes, many other operations can be supported in O(log n).