CSM 61B Trees, BSTs, and Balanced BSTs Crib Sheet

Spring 2018

March 11, 2018

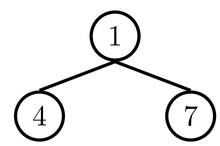
1 Overview

Trees represent recursively defined, hierarchical objects with a root node and subtrees of children with a parent node. **Binary trees** have at most two children per node, while **binary search trees** and **balanced search trees** are specialized trees used for searching and sorting.

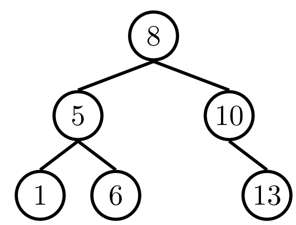
2 Definitions

2.1 Data Structures

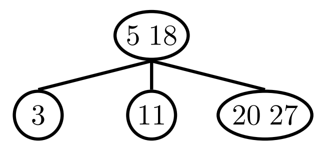
- Tree: A set of linked nodes, each of which has a label value and one or more child nodes, such that no node descends (directly or indirectly) from itself.



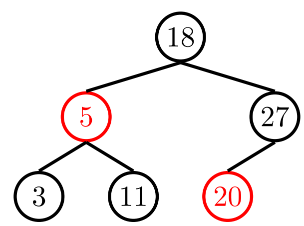
- Binary search tree (BST): Every node has either 0, 1, or 2 children. For every node X in the tree, every key in the left subtree is less than X's key and every key in the right subtree is greater than X's key.



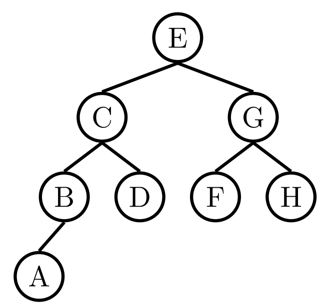
- Balanced search tree (B-tree): A self-balancing tree data structure that maintains near logarithmic height. For a 2-3 tree, each node has either 1 value and 0 or 2 children, or 2 values and 0 or 3 children. For a 2-3-4, or 2-4, tree, each node has at most 3 values, and all non-leaf nodes have 2, 3, or 4 children.



- Left leaning red black tree (LLRB): A self-balancing BST in which no node has two red links touching it, red links lean left, and every path from the root to a leaf has the same number of black links. For a corresponding 2-3 B-tree, red links glue node values together, while black links connect nodes.



2.2 Depth First Traversals



- Pre-order: Visit each node, then traverse its children. ECBADGFH
- Post-order: Traverse both children, then visit the node. ABDCFHGE
- In-order (binary trees only): Traverse the left child, visit a node, then traverse the right child. Yields elements in sorted order on a BST. ABCDEFGH

3 Special Operations

3.1 Deletion in BSTs with 2 children (Hibbard deletion)

- Delete a node x by replacing it with its successor, the leftmost node of the right subtree, to preserve the order of the BST.

3.2 Insertion into balanced search trees

- Insert an element x into its appropriate node. If that node is now overstuffed, or past its max capacity, push the left middle element up to its parent by one level. Repeat the process until no nodes are overstuffed.
- Observation: splitting trees have perfect balance. If we split the root, every node gets pushed down by exactly one level. If we split a leaf or internal node, the height remains the same.

4 Runtime Analysis

Data Structure	Insertion	Deletion	Search	Height
BST	$O(n)$	$O(n)$	$O(n)$	$\Omega(\log n), O(n)$
B-tree	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$
LLRB tree	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$

Note: Operations on "bushy" BSTs are logarithmic runtime while "spindly" trees have linear performance.