Week 5 - Binary Search Trees

AD 325 - 2022

Contents

Reading & Videos

- Carrano & Henry: Chapters 26, 27
- https://www.coursera.org/learn/algorithms-part1/home/week/4

Reference

- https://algs4.cs.princeton.edu/30searching/ (3.1 - 3.3)
- https://www.geeksforgeeks.org/binary-tree-data-structure/ (Sets 1-3)

Reference

- Binary Search Tree structure (BST)
- Searching a BST
- Adding & Removing BST entries
- Maintaining Heap order

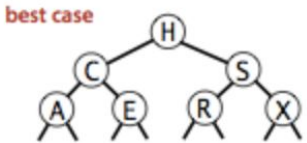
BST Structure

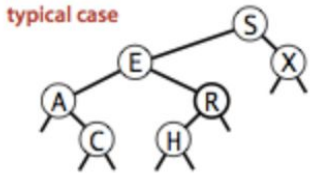
A BST is a ***binary tree*** that meets these conditions*:*

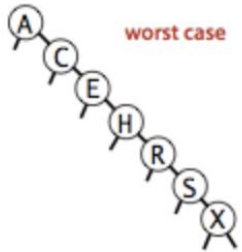
- each node contains a **comparable** object
- a node's data is greater than any data in the node's left subtree
- a node's data is less than any data in the node's right subtree (for unique values)
- a node's data is less than or equal to any data in the node's right subtree (for duplicate values)

The tree's mallest value will be the left-most node and the largest value will be the right-most node.

A BST may be **balanced** or **unbalanced**. Adding items to an empty tree in **random** order will result in a tree that is approximately balanced.







BST Operations

BST's can be traversed recursively **in-order** to visit entries in ascending order. BST's support basic database operations such as:

- **Add** - adds an entry to the tree. In case of unique values, overwrites any existing value.
- **Retrieve** - returns the requested entry, or entries in case of duplicate values.
- **Remove** - removes and returns the requested entry, or entries in case of duplicate values. Code needs to account for scenarios where removed node has a) one child or b) two children.
- **Search** - returns a boolean value indicating whether the requested entry is in the tree

If a tree supports duplicate entries, retrieve and remove operations should determine whether to affect the first occurence of an entry or all occurrences.

Searches begin at the root and the max number of comparisons for each operation is directly proportional to tree height.

Time complexity to search, add, & delete are **O(h)** for an unbalanced tree and for a height-balanced tree is **O(log n).**