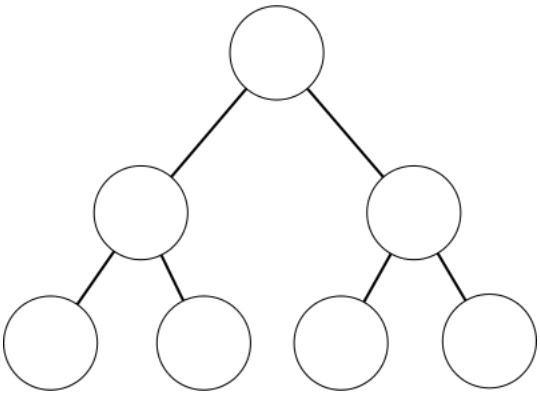
Binary Data Structures

Neal Bayya

November 2017

1 Introduction

Binary data structures are non-linear, and are a means by which we can store data. Each node stores a value and has 0, 1, or 2 children. The nodes at the very bottom of the tree are called leaves, and the top node is called the root.



We usually use binary data structures when we have a dynamic collection of of items (high number of insertions, deletions, and searches).

2 Binary Search Tree

2.1 Description

A binary search tree is a tree such that the its left node is less than it, and its right node is greater than it. Binary search trees used to 'divide and conquer' and problem into parts of two recursively. Ideally, we want to keep the search tree 'balanced', which means that the tree does not have a disproportional amount of nodes on one side as opposed to the other. We want to keep the tree balanced because that would allow a O(log n) run time for a root->leaf search on the tree. If the tree is imbalanced, this run time can degenerate to O(n).

2.2 Implementation Guide

```
class BSTNode {
    BSTNode left; //Pointer to left node
    BSTNode right; //Pointer to right node
    int v; //value at node
    public BSTNode (int nodeValue, BSTNode l, BSTNode r) {
        v = nodeValue;
```

```
        left = l;
        right = r;
    }
    //methods for insertion, deletion, search(element), etc.
}
```

The left and right nodes of a leaf should be kept as **null**, giving you a base case upon which you can terminate a search.

2.3 Relevant Data Structures in Java

The Java **TreeSet** stores a set of elements, but can order elements because of its internal BST data structure. The Java **TreeMap** stores a mapping between a key and value. It also keeps an ordering of its elements by its keys. Both of these structures use optimization for keeping the tree balanced, which is perfect for competitions.

3 Heap

3.1 Description

There are two types of heaps: **min heaps and max heaps**. In a min heap, each node is smaller than its two children. Likewise, each node is greater than its two children in a max heap.
Heaps are a subset of **complete** binary trees. This means that every level, except possibly the last, is completely filled. If the last level is only partially filled, then it must be filled from left to right. Additionally, heaps are expandable, meaning that you can remove the top element and restructure it, or add a new element.

3.2 Representation

We utilize the property of completeness in the representation of a heap. Instead of implementing a new data structure, we can easily represent it as an array. If we make the 0 index of the array null and fill the array from index 1, an interesting property arises:
The parent of a node at index i is located at index $\lfloor \frac{i}{2} \rfloor$.
The children of a node at index i are at $2 \times i$ and $2 \times i + 1$

3.3 Usage

We usually use heaps when we care about the greatest or lowest element in a set, and not about the ordering of the other elements. Heaps are slightly meticulous to implement, because of the restructuring property. However, The Java **PriorityQueue** is a max heap itself and can often be used as a substitute. If you need a min heap specifically, then you can override the compareTo method and reverse the priority function.

4 Problems

- Draw a BST with the string: 'LOVEICT', adding elements in the order they appear.
- Determine whether the following array is a heap: [null, 1, 3, 5, 10, 12, 6, 7, 11].
- Pre-order is an ordering of tree nodes in the form of node, right. In-order is left, node, right. Post-order is node, left, right. Write functions that traverses a tree in pre-order, in-order, and post-order.
- Write a function to determine if a binary tree is a BST.