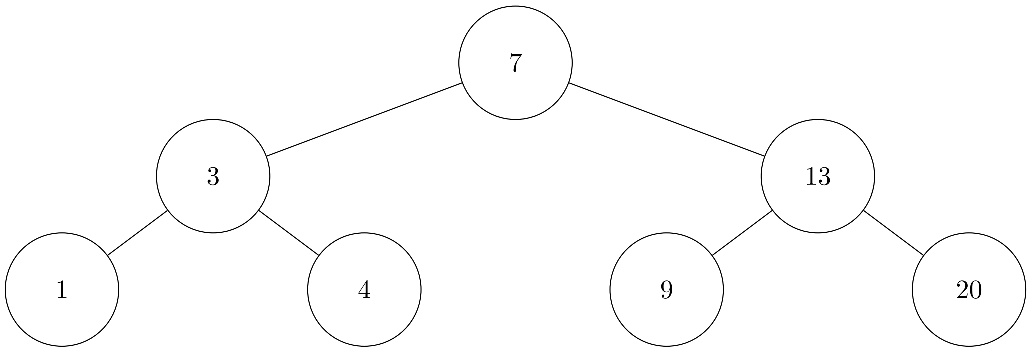
Binary Search Trees

Overview

Binary Search Trees (BST) are binary trees that are used specifically for storing values and efficiently retrieving desired values. Values are inserted in a specific way, so that finding a specific value in the BST is simply a flowchart asking the same questions over and over again.

Example



Here is an example BST. The basic structure of a BST is that the right child will always be **greater** than the parent and the left child will always be **less** than the parent. This means to find a specific element in a BST, we start at the root and just keep asking ourselves "is the current element bigger or small than the one we are looking for?" until we either find the element we want or don't find it.

If we try to insert an element that is the same as one already inside of the tree, it doesn't really matter how you resolve this as long as the solution is consistent. This means if you decide to put the equal element as the left child, then the next time you insert an equal element you should also put it as the left child. Flip flopping makes it confusing!

Bushy vs Spindly

It is important to understand that a BST can take on different shapes based on the insertion order of elements. Let's say we insert 1, 2, and 3 in that order in one tree and 2 before 1 or 3 in another tree. We will get trees that looks something like this

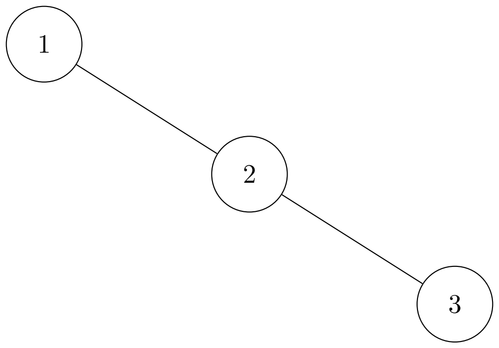


Figure 1: Spindly Tree (1, 2, 3)

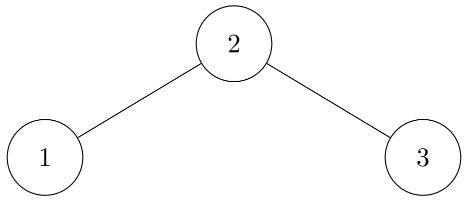


Figure 2: Bushy Tree (2, 1, 3)

Spindly Tree: A tree in which each level only has one element.
Bushy Tree: A tree in which each level is filled before moving onto the next level.

If we look at these two kinds of trees, we will see that the tree shape plays a role in how fast we can search for an element. Let's say we are searching for the number 3. In the spindly tree, we need to ask 2 questions to find it, whereas in the bushy tree we need to ask only 1 question to find 3. This observation is the key to why we prefer bushy tree than spindly trees.

Important Runtimes

When we consider the runtimes of BSTs, we also need to consider the height of them. A spindly tree of n elements will have a height of $\Theta(n)$, whereas a bushy tree of n elements will have a height of $\Theta(\log n)$.

Find
If we want to find a specific element in a spindly tree, we may need to search all of the elements, so we say that the runtime for a find function is $O(n)$. The same function for a bushy tree will take $O(\log n)$ because we will search at most the height of the bushy tree of which has a height of $\Theta(\log n)$.

Insertion
The runtime for insert for both spindly and bushy trees are the same as find because you may need to traverse the full height of the tree to insert a new element.

Deletion
There are three cases to consider when deleting a node from a BST, if the node is a leaf, if the node has 1 child, and if the node has 2 children.

1. Leaf
   If the node to be deleted has no children, then simply delete the node. This will have a runtime proportional to the height of the tree.
2. 1 Child
   If the node to be deleted has 1 child, then simply replace the deleted node with the child node. This will have a runtime proportional to the height of the tree.
3. 2 Children
   If the node to be deleted has 2 children, then we need to pick a side of the node to be deleted and find the smallest or largest value on that side, depending on the side. We will replace the node to be deleted with that node. If we want to replace the deleted node with a node from the right child side, then we need to look for the smallest value on that side and vice versa for the left side. The runtime of this operation will also be propotional to the height because we are at most traversing to the bottom of the tree once.

NOTE: Since we aren't guaranteed that the tree will be bushy, we say in general that BSTs have height of $O(n)$ even though we know it can have a height as good as $\Theta(\log n)$.

TL;DR

1. Smaller items to to left and larger items go to right
2. Spindly trees have height $\Theta(n)$
3. Bushy trees have height $\Theta(\log n)$
4. Find operation runtime directly proportional to height
5. Add operation runtime directly proportional to height
6. Delete Operation has 3 cases
   (a) **Leaf**: Simply delete node (runtime proportional to height)
   (b) **1 Child**: Replace deleted node with child node (runtime proportional to height)
   (c) **2 Children**: Find node in child subtree closest to the to be deleted node and replace it with that one (runtime proportional to height)