

TUTORIAL 4

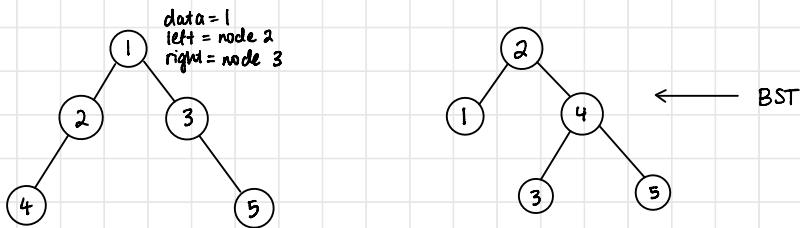
Recap: binary search trees (BSTs)

The motivation for BSTs comes from sorted arrays. If we have an arbitrary array with n elements, then searching for an element in the array is a worst-case $O(n)$ operation. But if the array is sorted, we can use binary search to do this search in worst-case $O(\log n)$ time:

Search for 6 in $\{3, 6, 11, 7, 8, 1, 4\} \Rightarrow O(n)$
 $\{1, 3, 4, 6, 7, 8, 11\} \Rightarrow O(\log n)$

Keeping an array sorted is hard and expensive though. (More on this later in the term)
⇒ think hierarchically instead of laterally!

A binary tree is a data structure consisting of nodes, where each node has 0, 1 or 2 children.



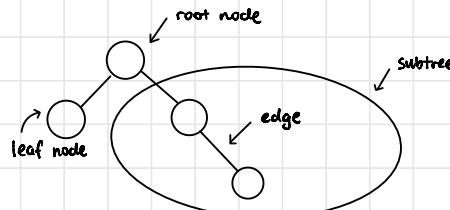
Each node consists of some data and references to its left and right child node.

If the nodes of a binary tree have the property that

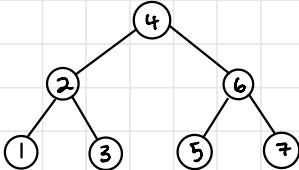
left data < node data < right data

then we call it a binary search tree. BSTs allow us to insert and search for a node in $O(\log n)$ time if done properly. In this way, we can think about them as generalisations of the idea of binary search.

Some terminology:



1. Insert 4 2 6 5 1 7 3

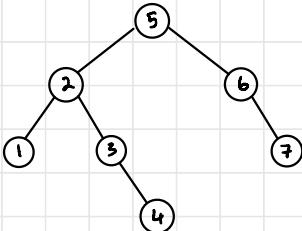


Balanced tree: all subtrees have equal

of nodes on left and right

Benefits: $O(\log n)$ insert + search!

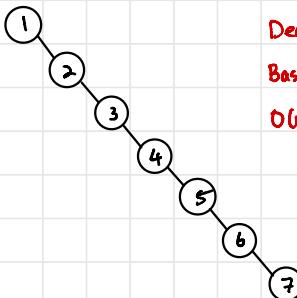
Insert 5 6 2 3 4 7 1



Reasonably balanced

Most trees on average probably look similar to this
close enough to $O(\log n)$ insert + search

Insert 1 2 3 4 5 6 7

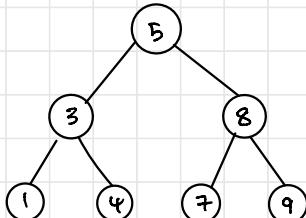


Degenerate tree: maximal possible imbalance

Basically just a linked list

$O(n)$ insert + search

2.



Tree traversals:

Infix: 1 3 4 5 7 8 9 \Rightarrow Left, then parent, then right. LNR

Prefix: 5 3 1 4 8 7 9 \Rightarrow Parent, then left, then right NLR

Postfix: 1 4 3 7 9 8 5 \Rightarrow Left, then right, then parent. LRN

Level: 5 3 8 1 4 7 9

Infix same as postfix: Right degenerate trees (see: last tree on prev. page)

All traversals the same: Empty tree, single node trees.