Week 6 Notes

Prof Bill - Apr 2018

In Week 6, we’ll cover:

A. Binary Trees and BST

B. Priority Queue

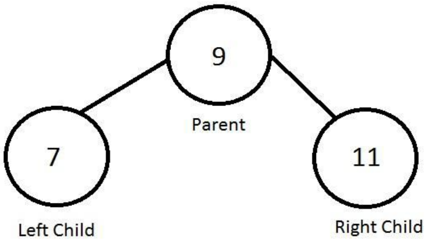
thanks… yow, bill

A. Binary Trees, Binary Search Tree (BST)

** Book: **Muganda 22.1-22.2**
** Online: **Princeton**, algs4.cs.princeton.edu/32bst
** Online: This **animation** is great: www.cs.usfca.edu/~galles/visualization/BST.html
** Online: A nice lecture: www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html

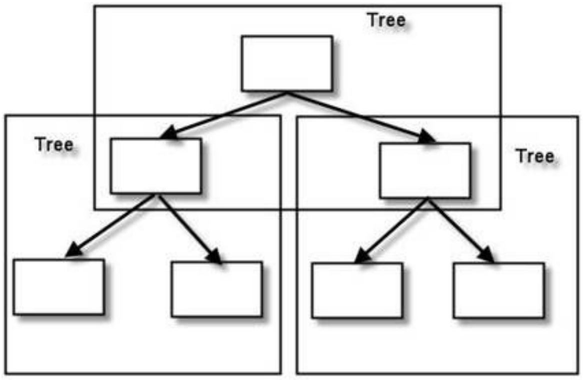
22.1 Binary Trees

Binary tree has nodes like a linked list. Each **node** has data (key, value) and then left and right node pointers. The **root** is the first node in the tree.



Source: cppbetterexplained.com/binary-search-trees/

Binary trees are recursive structures because nodes have nodes in them.
Also, subtrees behave just like the overall tree. Makes for easy recursive methods.



Binary tree consisting of 3 binary trees

Source: www.sqa.org.uk/e-learning/LinkedDS04CD/page_30.htm

Traversal:

- ➢ Preorder: root, left, right
- ➢ Inorder: left, root, right   // sorted order in a BST
- ➢ Postorder: left, right, root

/* memory helper: 1) root determines pre, in, or post and 2) left always before right */

Pseudocode… start process with call: inorder(root):

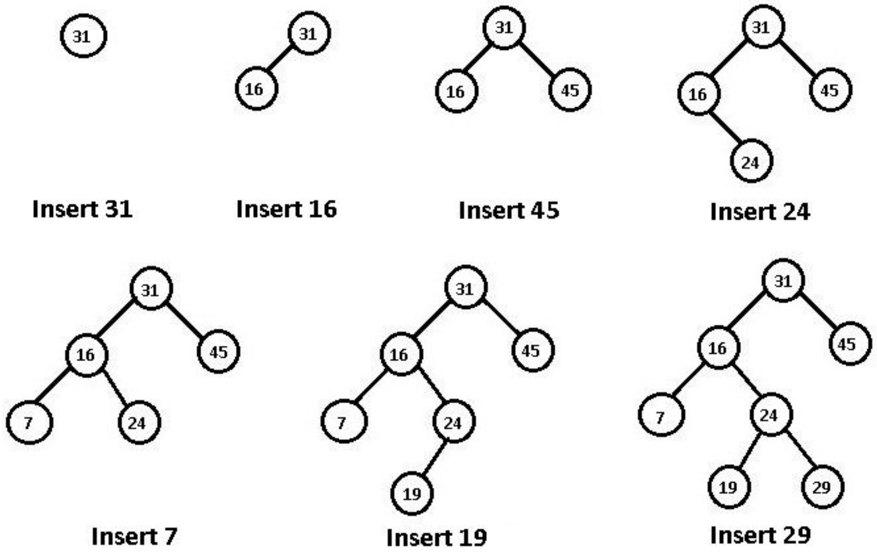
```
// inorder traversal to print binary tree
void inorder( Node n)
  if n == null then return
  inorder( n.left)
  print n
  inorder( n.right)
```

22.2 Binary Search Trees (BST)

A binary tree + this magic… for every node:

left child is less than (<) node
right child is greater than (>) node

That’s it. Let’s build one. An empty BST is root = null (not shown below).
Below: the root is the first node added; in this case 31.



.

Source: csegeek.com/csegeek/view/tutorials/algorithms/trees/tree_part2.php

Notice in our example:

- The root doesn’t change when adding to the tree
- Every new node is added as a leaf

Performance for BST magic,

- Average performance is O(log n), problem cut in half with each subtree
- Worst-case performance is O(n), unbalanced tree turns into a linked list (dop!)

There are 3 important methods in the BST ADT:

1. put(K key, V value) - we just did this
2. V get(K key)
3. V remove(K key)

See **Muganda Code 22-8, 22-9** for Java code.

With **put()** - Often, we just show the keys. The value is there or it's just keys (like a set). Use same left/right algorithm as get() below. New node is always a leaf!

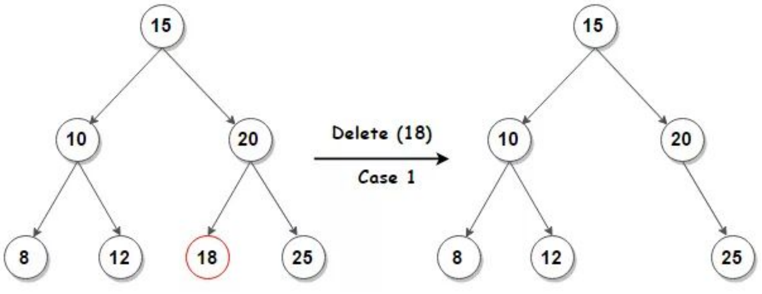
Here's **get()** pseudocode… it's a recursive search:

```
get( K key) {
   return getNode( root, key)              // start at root
}

V getNode( Node n, K key) {
   if n == null then return null           // NOT found

   if key == node.key return node.value    // FOUND

   if key < node.key
        return getNode( node.left, key) // look LEFT
   else
        return getNode( node.right, key)    // look RIGHT
}
```

Remove is a little tougher.

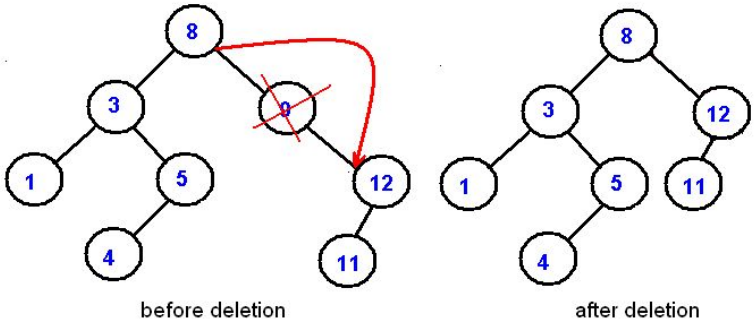
Three cases, removing a node with no children (leaf), one child, and two children:

- ➔ No children (leaf) - null out the parent’s link to the node (easy)



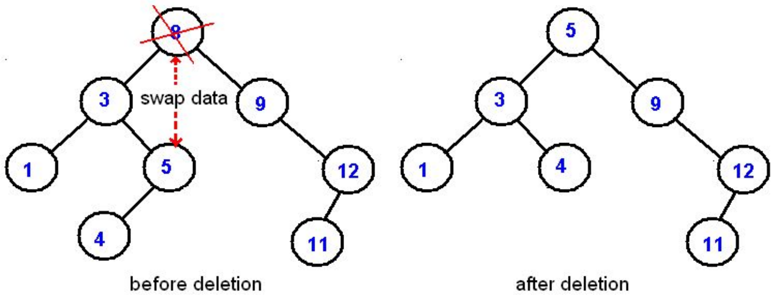
Source: www.techiedelight.com/deletion-from-bst/

- ➔ One child - replace node with its child (pretty easy)



Source: www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/pix/del01.bmp

- ➔ Two children - replace node with predecessor (largest node in left subtree, tougher)



About two children - Successor is ok too, smallest node in right subtree; Pred/Succ are always a leaf or one-child node. Yes?

B. Priority Queue

** Book: **Muganda 22.4**
** Online: great animation: www.cs.usfca.edu/~galles/visualization/Heap.html

22.4 Priority Queue

Priority Queue ADT - Insert based on a user-specified priority rather than order of insertion like a regular, old queue. Operations include:

```
insert( item)
item removeMin()
boolean isEmpty()
```

Heap property - each node is smaller than its children
Pseudocode:

```
// insert item into the heap
insert( item)
    add item as next leaf node
    while heap property is not met
        swap node with parent

// remove and return the smallest item from the heap
item removeMin()
    minItem = root
    put last leaf as root         // restore heap
    while head property is not met
        swap node with smallest child
    return minItem
```

JCF **PriorityQueue** holds **Comparable** objects. You can use a **Comparator** as well.
docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html

Heapsort - add items to heap, then removeMin() them for sorted order.

Terms: complete binary tree, binary tree depth

Heap performance:

- insert is O(log n) because the tree depth is log n.

- removeMin is O(log n)... the min is right there, O(1), but you have to swap and restore the heap, which is O(log n)

BTW - PQ and Heap can be flipped to max if you like… parent greater than children and removeMax().

Heap as an array

The BIG $$$ for heap = removeMin() and storing the heap as an array

- works because it is a complete binary tree

The visualization shows you the array representation right next to the graphical tree.
www.cs.usfca.edu/~galles/visualization/Heap.html

Equations for array storage of a heap:

root of tree = A[0]
parent of node A[k] = A[(k-1)/2]
left child of node A[k] = A[2k+1]
right child of node A[k] = A[2k + 2]   // left child + 1