Problem Set 7, Part I

Problem 1: Working with stacks and queues

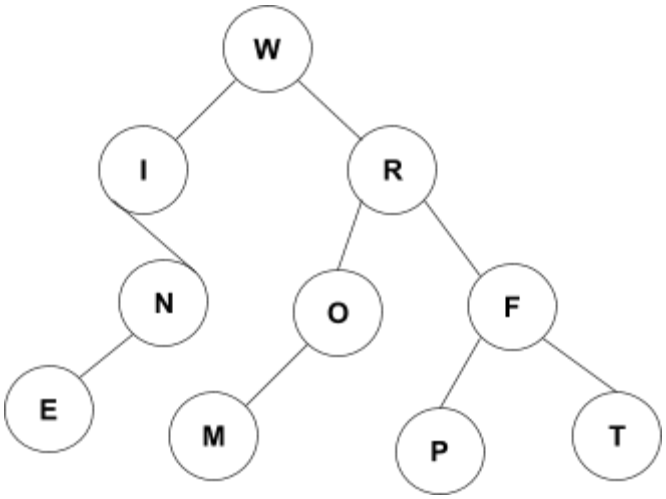
Problem 2: Using queues to implement a stack

Problem 3: Binary tree basics

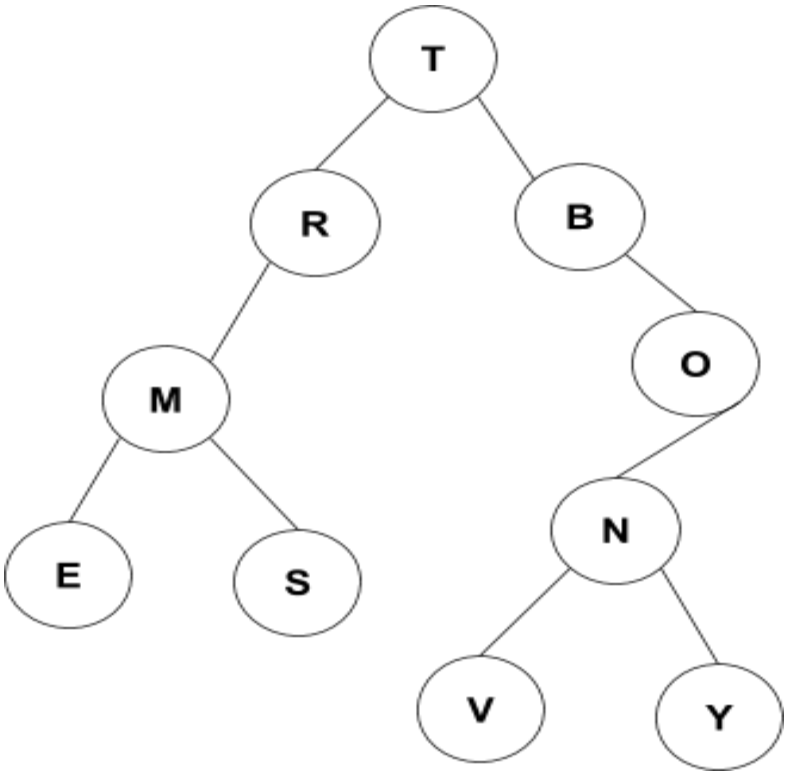
1) The height of the tree is 3
2) 4 leaf nodes and 5 interior nodes
3) 21 18 7 25 19 27 30 26 35
4) 19 7 25 18 26 35 30 27 21
5) 21 18 27 7 25 30 19 26 35
6) This is not a search tree because the node 25 which is a part of the left subtree is not less than 21.
7) This tree is balanced because there does not exist a node that is on its own level between each of the subtrees.

Problem 4: Tree traversal puzzles

4-1)

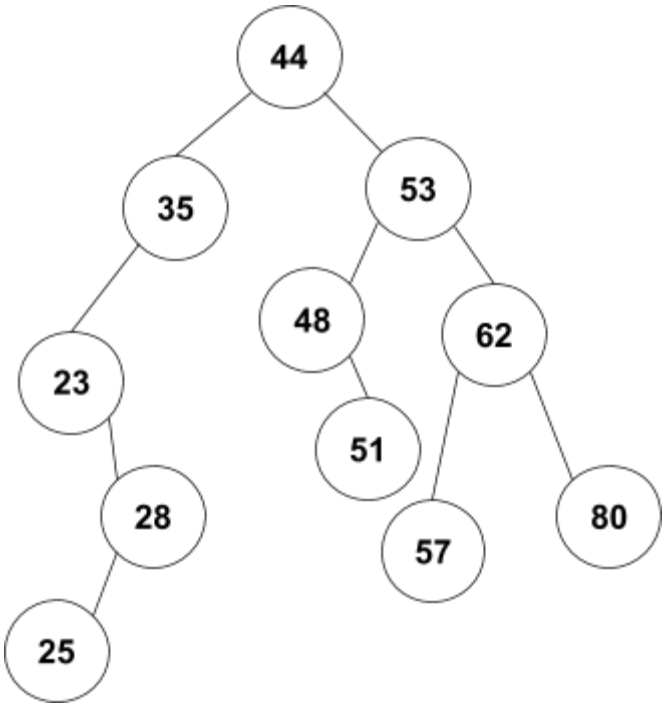


4-2)

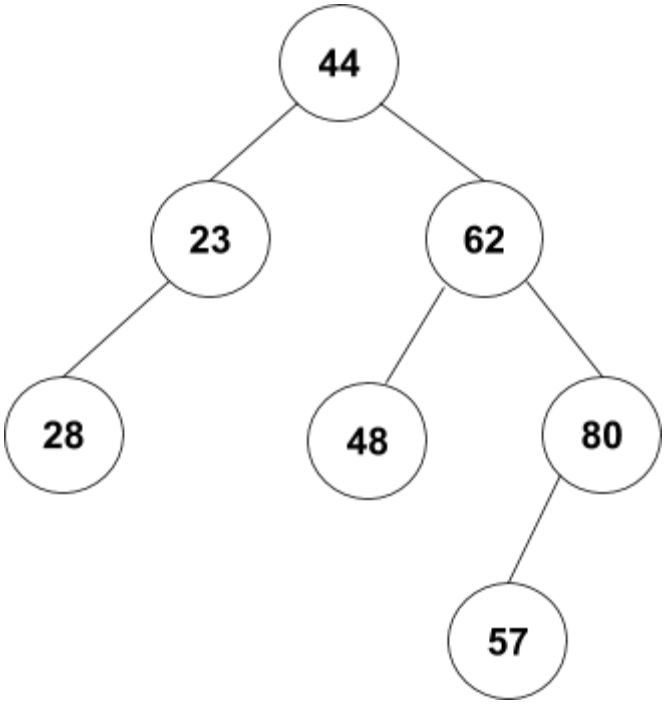


Problem 5: Binary search trees

5-1)



5-2)



Problem 6: Determining the depth of a node

1) In the best case, the time complexity of this function is O(1) which would occur if the depth of the given tree is 1. In the worst case, the time complexity of this function is O(n) as the method would have to pass through each node in order to find the depth. Whether the tree is balanced or not, the worst-case scenario is still O(n)

2)
```
public static int depthInTree(int key , Node root) {
      If (key ==root.key) {
            return 0;
      }
      If (root.left!=null && key<root.key) {
            int a = depthInTree(key,root.left);
            if(a!=-1) {
                  return a++;
            }
       }  else if(root.right!=null && key>root.key) {
            int b = depthInTree(key,root.right);
            if(b!=-1) {
                  return b++;
            }
       }
     return -1;
}
```

3) When dealing with a binary search tree, the time complexity becomes a lot smaller. We use the fact that the left subtree is all values smaller than the head node and the right subtree is greater than or equal to the head node. Therefore, we can simplify the time complexity to be O(the height of the tree).

Problem 7: 2-3 Trees and B-trees

7-1)