Problem Set 4, Part I

Problem 1: Uninformed state-space search
1-1) a, b, c, d, e, f

1-2) a, b, d, h, e, i

1-3) a, a, b, c, a, b, d, e

Problem 2: Determining the depth of a node
2-1)
In the best case, the time complexity of depthInTree is O(1). This reflects the time complexity when the root key is equal to the key being searched for. In the worst case, the time complexity is O(n), whether the tree is balanced or unbalanced. The algorithm performs an uninformed search, so it is necessary to visit every node in the worst case when the key is not in the tree. Thus, the time complexity is linear, or O(n).

2-2)

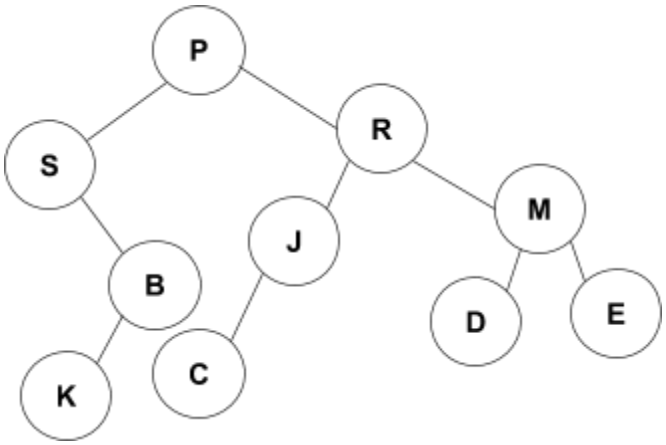
```
private static int depthInTree(int key, Node root) {
    if (key == root.key){   // key found, return
        return 0;
    }
    if (key < root.key) {  // only need to search left subtree
        if (root.left == null){  // the key is not found
            return -1;
        }
        int depthInSubtree = depthInTree(key, root.left);
        if (depthInSubtree == -1){ // if not found in recursive call,
                                   // we want the -1 to propagate up
            return -1;
        }
        return 1 + depthInSubtree;

    }
    // key is greater than root.key so we need to search right subtree
    if (root.right == null) {  // not found
        return -1;
    }
    int depthInSubtree = depthInTree(key, root.right);
    if (depthInSubtree == -1){
            return -1;
    }
    return 1 + depthInSubtree;

}
```

2-3)

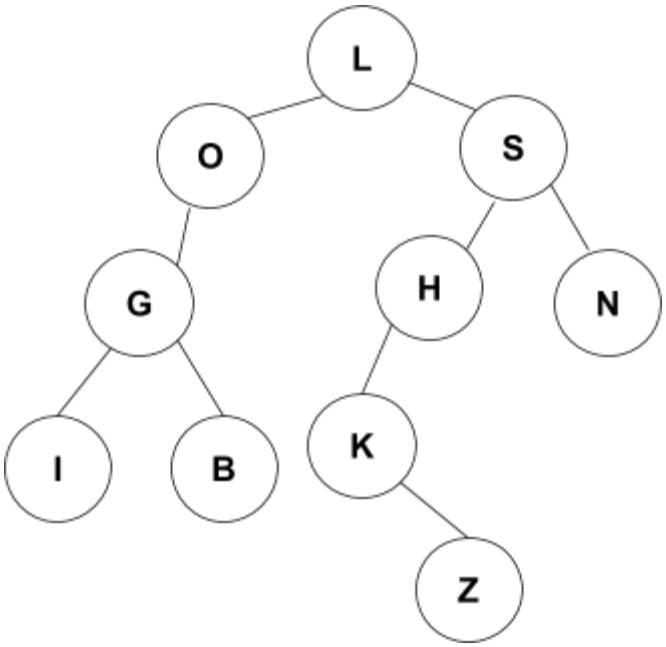
In the best case the time complexity of the revised algorithm is still O(1), when the key is at the root. In the worst case when the tree is balanced, the time complexity is O(logn). This is because the height of a balanced search tree is O(logn), and in the worst case we must search one path from top to bottom. If the tree is not balanced, the time complexity is O(n). In this worst case, the key is not in the tree and each node has one child, so it is necessary to visit every node in the tree, which is O(n).

Problem 3: Tree traversal puzzles

3-1)

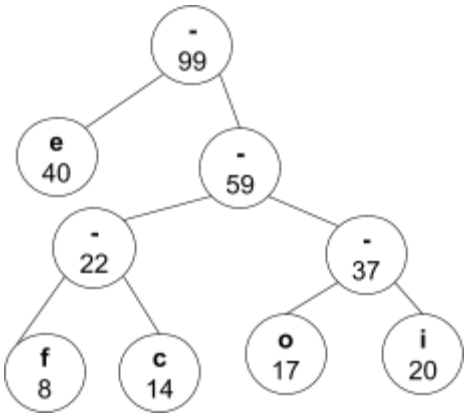


3-2)



Problem 4: Huffman encoding

4-1)



4-2)

1101001001111010

Problem 5: Binary search trees

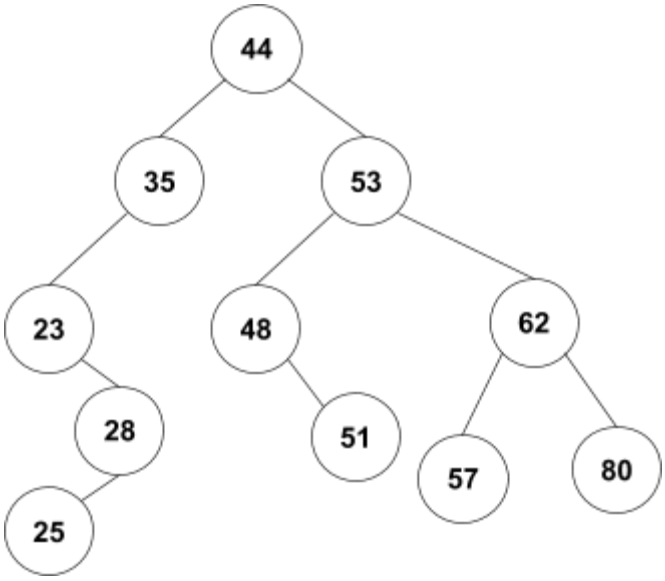
5-1)

44 35 23 28 53 48 62 57 80

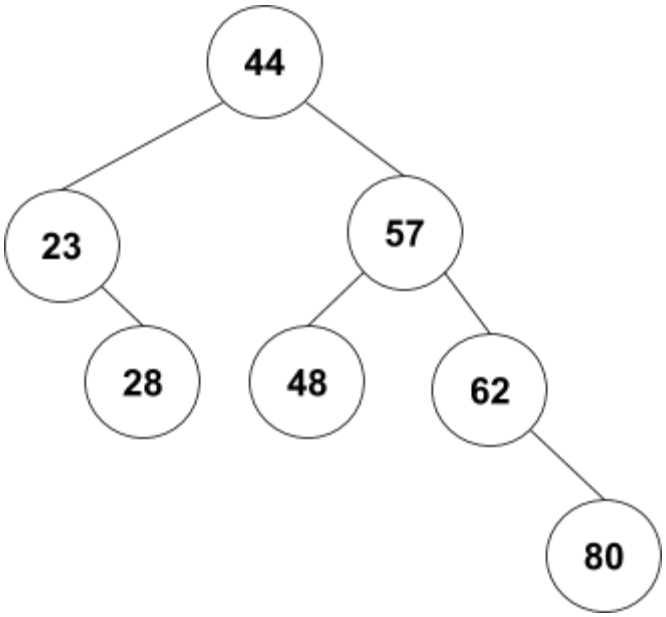
5-2)

28 23 35 48 57 80 62 53 44

5-3)



5-4)

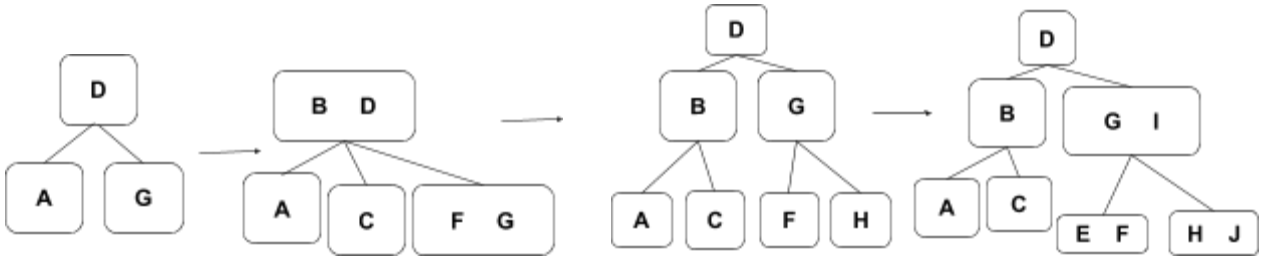


5-5)

The original tree is not balanced. The node with key 35 has a left subtree with depth 2 and no right subtree, so height of zero. Therefore, the tree is not balanced.

Problem 6: 2-3 Trees and B-trees

6-1)



6-2)