CS 202 - Spring 2022

Homework 3

Heaps, Priority Queues and AVL Trees

Alperen CAN

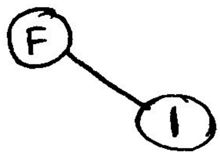
-

21601740

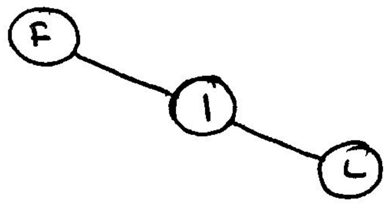
Question 1

a) → Insert 'F'

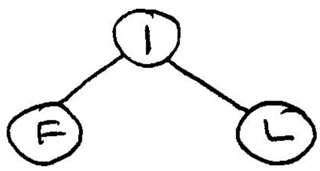
→ Insert 'I'



→ Insert 'L'

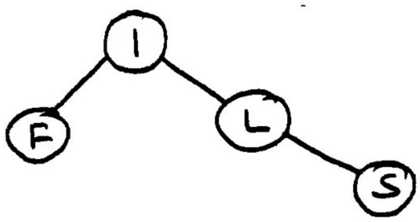


→ Single Left Rotation

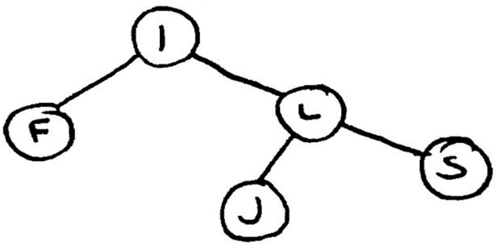


1

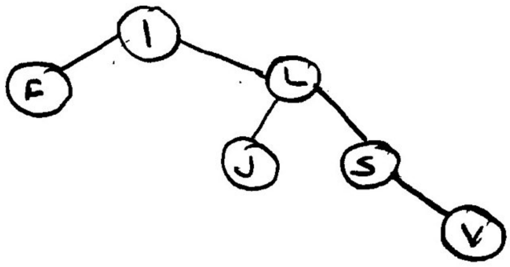
→ Insert 'S'



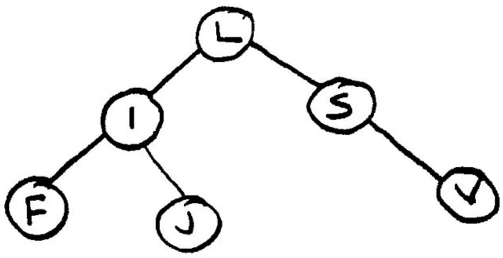
→ Insert 'J'



→ Insert 'V'

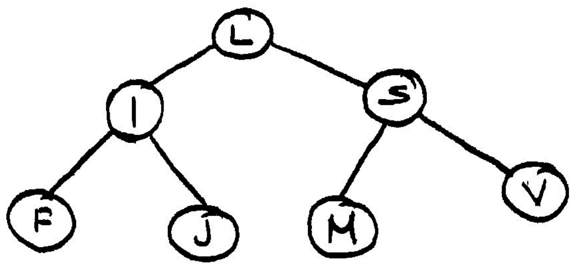


→ Single Left Rotation

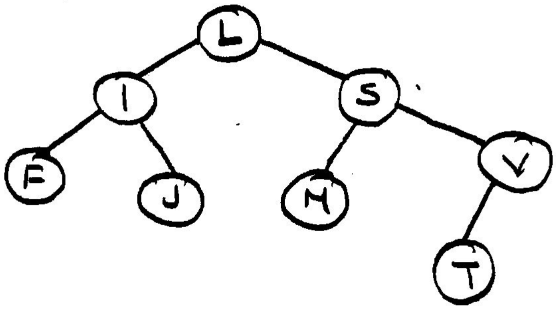


2

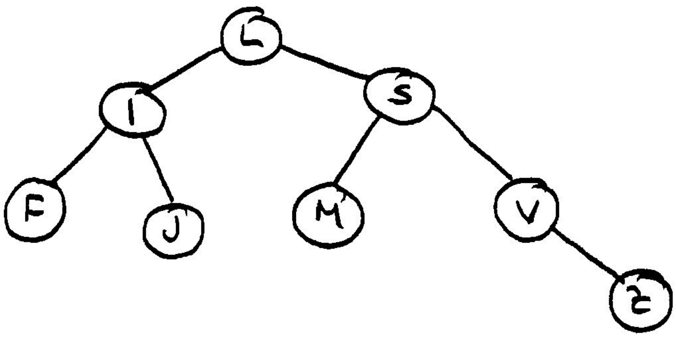
→ Insert 'M'



→ Insert 'T'

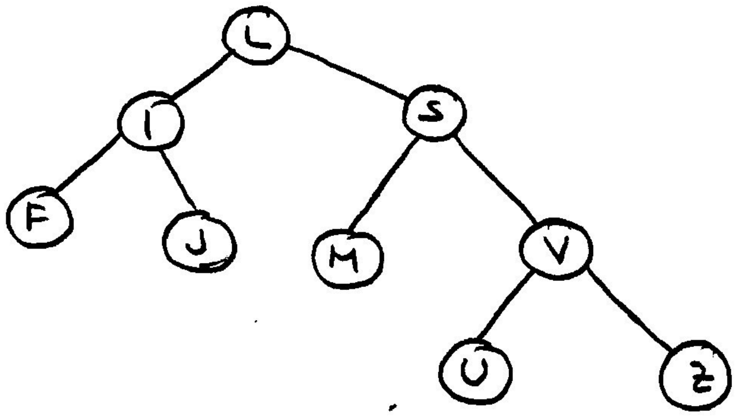


→ Insert 'Z'

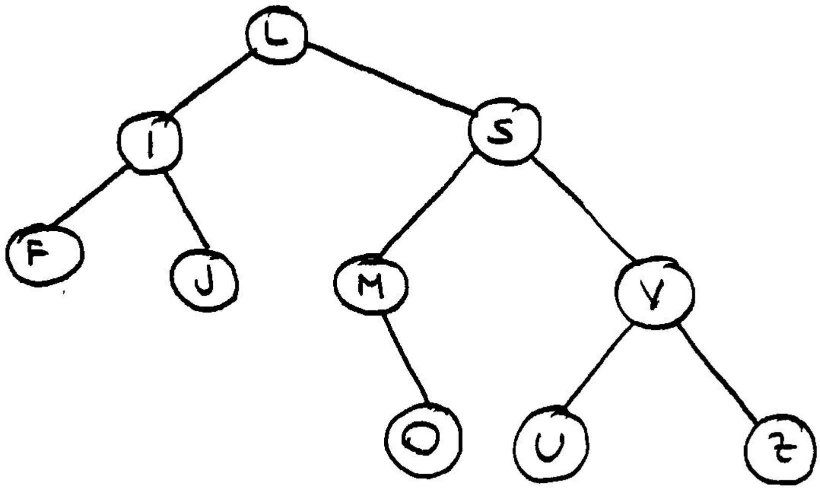


3

→ Insert 'U'



→ Insert 'O'



(Resulting Tree)

(4)

b) The node structure of the tree holds an extra 'size' property which holds the size of the subtree where the given node is root.

Algorithm:

```
computeMedian (root) {
    treeSize = root -> size
    medianIndex = treeSize / 2
    if (treeSize % 2 = 1) {
        medianIndex ++
    }
    node = root
    while (node != NULL) {
        if (medianIndex = 1) {
            return node -> item
        }
        if (node -> left = NULL) {
            medianIndex --
            node = node -> right
        }
        ...
```

(S)

```
else if(node->left->size > medianIndex){
   node = node->left
   medianIndex --
}
else if (node->left->size < medianIndex){
   medianIndex -= node->left->size
   if(medianIndex = 1){
      return root->item
   }
   else{
      medianIndex --
      node = node-> right
   }
}
else{
   return node->left->item
}
```

Time Complexity: O(log N)

Logic: The algorithm first calculates the index of the median. Then, it looks for left subtree size. If left subtree size is smaller then median index, we decrease median index by left subtree

16

size and continue with right subtree.
Otherwise, if it is larger than median index, we decrease median index by 1, because we eliminated root, we continue the search from left subtree.

c) Algorithm:

```
checkAVL(root) {
    isAVL = true;
    checkAVLHeight(root, isAVL)
    return isAVL;
}
checkHeight(node, &isAVL) {
    if (node == NULL)
        return 0;
    leftHeight = checkHeight(node->left, isAVL);
    rightHeight = checkHeight(node->right, isAVL);
    if (Math.abs(leftHeight - rightHeight) > 1) {
        isAVL = false;
        return;
    }
    :
```

7

```
return max(leftHeight, rightHeight);
}
```

Time Complexity: In worst case, all nodes will be visited. Therefore, $\underline{O(n)}$.

Logic: While calculating height of a node, algorithm also checks for if the node creates any imbalance on the tree or not.

8

Question 3

For the simulator, minimum computer count can be 1, at minimum and as many as request count, at maximum. Using these numbers, we can start a binary search to find optimum count.

9