ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 335E

ANALYSIS OF ALGORITHMS
HOMEWORK 3

CRN : 15175

DATE : 11.01.2021

NAME : ALI KEREM YILDIZ

STUDENT ID : : 150170013

FALL 2020-2021

1. **Question 1**

- In order to obtain the asymptotic upper bounds for insertion and search operations, first we need to have an idea about height of the tree. Heuristically, by merging red nodes into its parent (parent must be black in order to satisfy red-black tree requirements). Now, all leaves have same depth. Their height is same as their black-height in red-black tree. We get leaves on same level beacuse it is a good thing to yield a pretty much balanced tree. Assume h' is height of node-merged tree and h is the height of red-black tree. We know that we have $n+1$ leaves (where n denotes internal node number.) In node-merged tree every node at least has 2 children. That means $2^{h'} \leq n+1$ and $h' \leq \lg(n+1)$
  We know that number of red nodes at most half of the lenght of the path from root to leaf. Therefore, $h \leq 2h'$, $h \leq 2\lg(n+1)$, O(lgn)

  For insertion operation, in the first step simple BST insert operation is performed. (O(lgn) because tree is balanced as we shown above.) Recoloring step is O(1) because it just sets a color value to a specific node. However, we migh encounter double-red situations which violates requirements. In worst case we emd up a double-red situation along the entire path. So the recoloring is O(lgn) in worst case. The restoration(rotation) operations takes O(lgn) too. The total time for insert and search is O(lgn) both average case and worst case

NOTE : You should compile the code as : g++ -std=c++11 main.cpp

2. Question2

- Red-Black Tree is not independent from Binary Search Tree. We can say Red-Black Tree is a binary search tree. Actually, Red-Black Tree is a special form of a binary search tree. Red-Black Tree is kind of balanced version. In implementation of both trees, nodes have some key values which is used for comparision with other nodes. Binary Search Tree is not always distributed uniformly. For example, if every node's key value greater than all nodes whose are already in tree, we evalaluate a tree which height is n (n is number of nodes.) In this kind of examples we lose the advantages of binary search tree which provide us to O(logn). Red-Black Tree satisfies all conditions to become a binary search tree and addition to all of these, in order to preserve its balance while doing operations such as inserting etc. we can use extra information for every node that are either red or black.

3. Question3

- Since we need to get i^{th} smallest, we may use inorder traversal beacuse in inorder traversal, the first node for the asked role that we encounter should be the smallest one. İf (node->role == "SF") and by counting the numbers of encountering asked role, (set a counter variable and increment it.) (if counter == i), if it is true we obtain the asked node. Then we should return it. (return node)

```
if (node == NULL)
    then return
inorder(node->left)
count = 0
if (role == "SF")
    then count++
if (count == i)
    then return node
    exit()
inorder(node->right)
```

NOTE : You should compile the code as : g++ -std=c++11 main.cpp