Twisted Rotations

locked

by maruthisarat

The goal is to maintain a Binary Search Tree (BST) on integers along with the following operations. You may assume that the BST contains distinct integers.

1. **INSERT k** : insert value k into BST. This operation is already implemented for you.
2. **LR k** : left rotate node containing the value k. This is applicable to a node which has a right child. You can assume that the value k is already present in the BST.
3. **RR k** : right rotate node containing the value k. This is applicable to a node which has a left child. You can assume that the value k is already present in the BST.
4. **PRE-ORDER** : print BST in preorder format. However, we need to print additional data. For each node in the preorder traversal print the following triplet in **constant time.** (key<space>left_subtree_size<space>right_subtree_size)

Input Format

First line contains N, denotes number of operatons. Next N lines, each line contains one of four operations.

Constraints

- 1 <= N <= 10^4.
- 1 <= k <= 10^6.
- all keys are unique in tree at any time.

Output Format

For "preorder" operation, print triplet (key<space>left_subtree_size<space>right_subtree_size) in preorder of BST.

Sample Input 0

```
5
INSERT 2
INSERT 1
INSERT 3
RR 2
PRE-ORDER
```

Sample Output 0

```
(1 0 2) (2 0 1) (3 0 0)
```

Sample Input 1

```
8
INSERT 1
INSERT 2
INSERT 5
INSERT 4
LR 2
INSERT 6
INSERT 10
PRE-ORDER
```

Sample Output 1

```
(1 0 5) (5 2 2) (2 0 1) (4 0 0) (6 0 1) (10 0 0)
```

Submissions: 67
Max Score: 60
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++

```cpp
#include <bits/stdc++.h>
using namespace std;

struct node{
    int key;
    node *left, *right;
};

class BST {
private:
   node* root;
public:
    node* new_node(int key);
    void insert(int key);
    void insert(node **root, int key);
    BST() {
        root = NULL;
    }
};

node* BST::new_node(int key){
    node* temp = new node;
    temp->key = key;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

void BST::insert(int key){
   insert(&root, key);
}

void BST::insert(node **root, int key){
   if(*root == NULL){
      *root = new_node(key);
   }
   else{
      if((*root) -> key < key){
         insert(&((*root) -> right), key);
      }
      else if((*root) -> key > key){
         insert(&((*root) -> left), key);
      }
   }
}

int main() {
    int T, N;
    string s;
    cin >> T;
    BST bst;
    for(int t=0; t<T; t++){
        cin >> s;
        if(s == "INSERT"){
            cin >> N;
            bst.insert(N);
        }
         // continue from here
```

```
    }
    return 0;
}

```

Line: 1 Col: 1

Upload Code as File

Run Code Submit Code