Trees : Level 3
Lesson 3

Preorder to Postorder

Given an array arr[] of N nodes representing preorder traversal of BST. The task is to print its postorder traversal.

Example 1:

```
Input:
N = 5
arr[]  = {40,30,35,80,100}
Output: 35 30 100 80 40
Explanation: PreOrder: 40 30 35 80 100
InOrder: 30 35 40 80 100
Therefore, the BST will be:
              40
           /      \
         30       80
           \        \
           35      100
Hence, the postOrder traversal will
be: 35 30 100 80 40
```

Example 2:

```
Input:
N = 8
arr[]  = {40,30,32,35,80,90,100,120}
Output: 35 32 30 120 100 90 80 40
```

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

1 <= N <= 103

1 <= arr[i] <= 104

```cpp
Node* create_node(int preorder[], vector<int> &in,int &k, int s, int e)
{
  if(s<=e)
  {
     Node *root=new Node();
     root->data=preorder[k];
     int i;
     for(i=s; i<=e && in[i]!=preorder[k]; i++);
     k++;
     root->left=create_node(preorder,in,k,s,i-1);
     root->right=create_node(preorder,in,k,i+1,e);
     return root;
  }
  return NULL;
}
//Function that constructs BST from its preorder traversal.
Node* constructTree(int preorder[], int n)
{
  //code here
  vector<int> in;
  for(int i=0; i<n; i++)
     in.push_back(preorder[i]);
  sort(in.begin(),in.end());
  int k=0;
  return create_node(preorder,in,k,0,n-1);

}
```

Pairs violating BST property

You are given a Binary tree. You are required to find the number of pairs violating the BST property. In BST every element in the left subtree must be less than every element in the right subtree. You are required to complete the function pairsViolatingBST(Node *root, int N).

Input:

The first line consists of an integer T denoting the number of test cases. Each test case consists of two lines. The first line of each test case consists of a single integer N, denoting the number of edges in the Binary tree. The next line contains the edges of the binary tree. Each edge consists of two integers and one character, the first of which is parent node, second is child node and character "L" representing Left child and "R" representing the right child.

Output:

You are required to complete the function pairsViolatingBST(Node *root, int N) which takes the root of the tree and the number of edges N as the arguments. The function returns the required number of pairs. As the results can be large, return your result modulo 10^9 + 7.

Constraints:

1 <= T <= 1000

1 <= N < 10^5

Example:

Input:

2

6

50 30 L 50 60 R 30 20 L 30 25 R 60 10 L 60 40 R

2

4 5 L 4 6 R

Output:

7

1

Explanation:

The binary tree for 1st test case is-

```
            50
        /          \
  30                 60
/       \          /      \
20      25        10       40
```

The pairs violating the BST property are: (20, 10), (25, 10), (30, 25), (30, 10), (50, 10), (50, 40), (60, 40). Hence, the result is 7.

The binary tree for 2nd test case is-

```
        4
  /          \
5              6
```

The pair violating the BST property is: (5, 4). Hence, the result is 1.

```
void chk(Node* root,vector<int> &c)
{
  if(root)
  {
     chk(root->left,c);
     c.push_back(root->data);
     chk(root->right, c);
  }
}
int pairsViolatingBST(Node *root,int n){
  vector <int> p;
  chk(root,p);
  int c=0;
  for(int k=0; k<p.size()-1; k++)
  {
     for(int j=k+1; j<p.size(); j++)
        if(p[k]>p[j])   c++;
  }
  return c;
}
```

Add all greater values to every node in a BST

Given a BST, modify it so that all greater values in the given BST are added to every node.

Example 1:

Input:
```
          50
        /    \
      30     70
     /   \   / \
   20   40 60  80
```
Output: 350 330 300 260 210 150 80
Explanation:The tree should be modified to following:
```
            260
          /       \
      330       150
     /   \   /     \
   350   300 210    80
```

Example 2:

Input:
```
         2
       /   \
      1      5
           /  \
          4    7
```
Output: 19 18 16 12 7

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(Height of the BST).

Constraints:

1<=N<=100000

```
void reverse_inorder(TreeNode* root, int &sum)
{
  if(root)
  {
     reverse_inorder(root->right,sum);
     sum+=root->val;
     root->val=sum;
     reverse_inorder(root->left,sum);
  }
}
class Solution {
public:
  TreeNode* convertBST(TreeNode* root) {
     int sum=0;
     reverse_inorder(root,sum);
     return root;
  }
};
```

Leftmost and rightmost nodes of binary tree

Given a Binary Tree of size N, Print the corner nodes ie- the node at the leftmost and rightmost of each level.

Example 1:

Input :

```
       1
     /  \
   2      3
  / \    / \
 4   5  6   7
```

Output: 1 2 3 4 7

Explanation:

Corners at level 0: 1

Corners at level 1: 2 3

Corners at level 2: 4 7

Example 2:

Input:

```
      10
    /    \
   20     30
  / \
 40  60
```

Output: 10 20 30 40 60

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(number of nodes in a level)

Constraints:

1 ≤ N ≤ 10^5

```
void printCorner(Node *root)
{

// Your code goes here
   if(root)
   {
      //cout<<root->data<<" ";
      queue<Node*>q;
      q.push(root);
      while(q.size()>0)
      {
         int c=q.size();
         vector <int> p;
         while(c--)
         {
            root=q.front();
            q.pop();
            p.push_back(root->data);
            if(root->left)  q.push(root->left);
            if(root->right) q.push(root->right);
         }
         cout<<p[0]<<" ";
         if(p.size()>=2)
            cout<<p[p.size()-1]<<" ";
      }
   }
}
```

Find the Closest Element in BST

Given a BST and an integer. Find the least absolute difference between any node value of the BST and the given integer.

Example 1:

Input:
```
       10
     /   \
    2    11
  /  \
 1    5
     /  \
    3    6
     \
      4
```
K = 13
Output: 2
Explanation: K=13. The node that has value nearest to K is 11. so the answer is 2
Example 2:

Input:
```
     8
   /   \
  1     9
   \     \
    4    10
   /
  3
```
K = 9
Output: 0
Explanation: K=9. The node that has the value nearest to K is 9. so the answer is 0.
Expected Time Complexity: O(Height of the BST).

Expected Auxiliary Space: O(Height of the BST).

Constraints:

1 <= Number of nodes <= 100000

```
int ans;
void preorder(Node* root, int K)
{
  if(root)
  {
    if(root->data==K)
    {
      ans=0;
      return;
    }
    ans=min(ans,abs(K-root->data));
    if(K>root->data)
      preorder(root->right,K);
    else
      preorder(root->left,K);
  }
}
class Solution
{
  public:
  int minDiff(Node *root, int K)
  {
    //Your code here
    ans=INT_MAX;
    preorder(root,K);
    return ans;
  }
};
```

Preorder Traversal and BST

Given an array arr[ ] of size N, write a program that returns 1 if the given array can represent preorder traversal of a possible BST, else returns 0.

Example 1:

Input:
N = 3
arr = {2, 4, 3}
Output: 1
Explanation: Given arr[] can represent preorder traversal of following BST:

```
    2
     \
      4
     /
    3
```

Example 2:

Input:
N = 3
Arr = {2, 4, 1}
Output: 0
Explanation: Given arr[] cannot represent preorder traversal of a BST.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(N)

Constraints:

1 ≤ N ≤ 10^5

1 ≤ arr[i] ≤ 10^5

```cpp
class Solution {
 public:
   int canRepresentBST(int arr[], int N) {
       // code here
       stack<int> s;
       set<int> chk;
       s.push(arr[0]);
       chk.insert(arr[0]);
       int k=INT_MIN;
       for(int i=1; i<N; i++)
       {
           if(chk.count(arr[i]))
               continue;
           if(arr[i]<s.top())
           {
               if(arr[i]<k)
                   return 0;
           }
           else
           {
               k=s.top();
               s.pop();
           }
           s.push(arr[i]);
       }
       return 1;
   }
};
```

Construct BST from Postorder

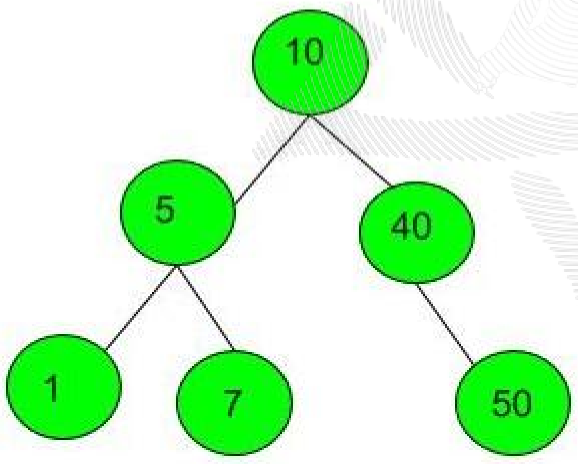
Given postorder traversal of a Binary Search Tree, you need to construct a BST from postorder traversal. The output will be inorder traversal of the constructed BST.

Example 1:

Input:
6
1 7 5 50 40 10

Output:
1 5 7 10 40 50

Explanation:
Testcase 1: The BST for the given post order traversal is:



Thus the inorder traversal of BST is: 1 5 7 10 40 50.

Expected Time Complexity: O(Height of the BST)

Expected Auxiliary Space: O(Height of the BST)

Constraints:

1 <= T <= 100

1 <= N <= 100

```cpp
Node* construct(int minm,int maxm,int post[],int &k)
{
  if(k>=0)
  {
     if(post[k]<maxm && post[k]>minm)
     {
        Node* root=new Node(post[k]);
        k--;
        root->right=construct(root->data,maxm,post,k);
        root->left=construct(minm,root->data,post,k);
        return root;
     }
  }
  return NULL;
}

Node *constructTree (int post[], int size)
{
//code here
  int minm=0,maxm=INT_MAX;
  int k=size-1;
  return construct(minm,maxm,post,k);
}
```