Second Largest Element in Tree

Problem Description:

You are given a generic binary tree; you have to print the second largest element present in the given tree.

How to approach:

Approach 1:

We will try to find the maximum element present in the tree. After this we will replace this element by minus infinity and then we will try to find the maximum again in the updated tree.

But there is a problem with this approach that is, if we have multiple maximum nodes then we have to replace each node with minus infinity, if any node is left then the solution will not be correct. Also, we are changing the given input itself hence this approach is not considered as a good approach.

Approach 2:

In this approach, we will try to find the maximum element in the tree. After finding the maximum element we will proceed with finding an element just smaller than the maximum element, this will give you the 2nd largest element present in the tree.

The given function is:

```
TreeNode <int>* secondLargest(TreeNode<int> *root) {
TreeNode<int> *n1=maxN(root);  //function to find the node with maximum value in the tree
 return nextsmaller(root,n1->data); //finding the element just smaller than the maximum
}
```

//Function to find maximum value node(recursively):

```
TreeNode <int>* maxN(TreeNode<int> *root){
    if(root==NULL){
     return NULL;
  }
  TreeNode<int>*max=root;
  for(int i=0;i<root->children.size();i++){
    TreeNode<int> *n=maxN(root->children[i]);
    if(max->data<n->data){
       max=n;
    }
  }
```

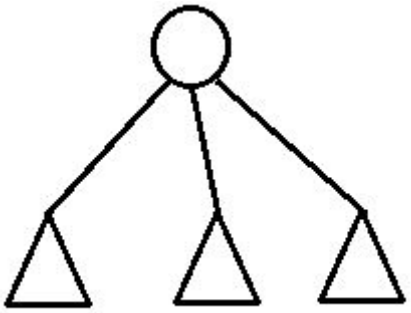
```
    return max;
  }
```

//Function to return next smaller value than max node i.e. just smaller(recursively)

```
TreeNode <int>* nextsmaller(TreeNode<int>* root,int a)
{
 TreeNode<int>* small=NULL;
  if(root->data<a)
  {
     small=root;
  }
  for(int i=0;i<root->children.size();i++)
  {
   TreeNode<int>* b = nextsmaller(root->children[i],a);
   if(small==NULL&&b!=NULL)
   {
      small=b;
   }
   else if(small!=NULL&&b!=NULL)
   {
   if(small->data<b->data)
   {
       small=b;
   }
   }
  }
  return small;
}
```

Approach 3:

Visualize the tree as:



Now, recursively ask each node to return it's max and 2nd max of children. Compare it with the answer max and 2nd max and update its value by comparing.

During comparison a lot of cases arises:

Case 1: If the max of the children > max of the answer

Case 1a: - If the 2nd max of children is NULL.

Let us say we have answer: max=10

2nd max=5

Children: max=20

2nd max=NULL

Replace max ans by children max

Now the 2nd max of ans will be previous max of ans.

Case 1b: Children 2nd max > ans max

Let us say we have answer: max=10

2nd max=5

Children: max=20

2nd max=15

Then, replace ans max ascchildren max

And 2nd max of answer as 2nd max of children.

Case 1c: Children 2nd max < ans max

Let us say we have ans: max=10

2nd max=5

Children: max=20

2^{nd} max=9

Then, replace max of ans=max of children

And keep the 2^{nd} max of answer as previos max of ans.

Case 2: If the max of ans = max of children and 2^{nd} max of children is not NULL.
Let us say we have ans: max=10

2^{nd} max=5

Children: max=10

2^{nd} max=9

Then, 2^{nd} max of ans will be replaced by 2^{nd} max of children.

Case 3: If the max of ans is not equal to max of children and (2^{nd} max of ans is NULL or max of children > 2^{nd} max of ans)

Let us say we have ans: max=20

2^{nd} max=NULL

Children: max=10

2^{nd} max=9

Then, replace 2^{nd} max of ans with max of children.

After taking into consideration all these cases, we can return the 2^{nd} max of the final ans, this will give you the second largest element of the tree.