20) CHECK WHETHER THE BST CONTAINS DEAD END OR NOT:

METHOD 1:
The main crux of the question is that the leaf nodes only decides whether the given BST contains dead end or not.
The main logic is that the BST must not contain the nodes with value of leaf_node+1 and leaf_node-1. If it contains both the element then the leaf node becomes dead end.
So we use two unordered map, one which stores the leaf node and the other which stores all the nodes. Then we traverse through the map of leaf nodes and check whether there in leaf_node+1 and leaf_node-1 waala element in the tree using the unordered map of all the elements.
Space Complexity : O(N)
Time Complexity : O(N)

METHOD 2:(efficient)
In this method, we use the range method to solve the sum.
If for any leaf node. The upper and lower bounds of the range for any leaf node becomes equal then the function returns true else return false.
Time Complexity : O(N)
Space Complexity : O(1)

LINK OF EXPLANATION : Check whether BST contains Dead End | BST | Love Babbar DSA …

CODE FOR METHOD 1:

```
void InOrderTraversal(Node *root,unordered_map<int,int> &leaf,unordered_map<int,int>
&all_nodes){
   if(root!=NULL){
      InOrderTraversal(root->left,leaf,all_nodes);
      all_nodes[root->data]=1;
      if(root->left==NULL && root->right==NULL){
         leaf[root->data]=1;
      }
      InOrderTraversal(root->right,leaf,all_nodes);
   }
}

bool isDeadEnd(Node *root)
{
   unordered_map<int,int> leaf;
   unordered_map<int,int> all_nodes;
   InOrderTraversal(root,leaf,all_nodes);
   for(auto it=leaf.begin();it!=leaf.end();it++){
      if(it->first==1 && all_nodes[it->first+1]==1){
         return true;
      }
      else if(all_nodes[it->first+1]==1 && all_nodes[it->first-1]==1){
         return true;
      }
```

```
    }
    return false;
}
```

CODE FOR METHOD 2:

```
bool fun(Node *root,int low,int high){
    if(root!=NULL){
        bool left=fun(root->left,low,root->data-1);
        if(root->left==NULL && root->right==NULL){
            if(low==high){
                return true;
            }
            else{
                return false;
            }
        }
        bool right=fun(root->right,root->data+1,high);
        return left || right;
    }
}

bool isDeadEnd(Node *root)
{
    bool answer=fun(root,1,INT_MAX);
    return answer;
}
```