

# **Experiment3**

StudentName:Om Ankur Prajapati

Branch:BE-CSE Semester:5<sup>TH</sup>

SubjectName:-AdvanceProgramminglab-2

UID:-22BCS15926 Group-602-A

DateofPerformance:23-1-25 SubjectCode:-22CSP-351

**Aim**:-Given therootofabinary tree, return*itsmaximumdepth*. Abinary tree's **maximumdepth**isthenumberofnodes along the longest path from the root node down to the farthest leaf node.

**Objective**:-Theobjectiveistodeterminethemaximum depthofabinarytree, whichrepresentsthelongest pathfrom therootto the deepest leaf node. This helps in understanding thetree's structure, height, and balance, which are essential in various applications like searching, sorting, and optimizing tree-based algorithms.

## **Apparatus Used:**

- 1. **Software**:-Leetcode
- 2. Hardware: Computer with 4GBRAM and keyboard.

## AlgorithmfortheTwoSumProblem:

- 1. **CheckBaseCondition**—If the root is null ptr, return 0 (empty tree has depth 0).
- 2. **RecursivelyComputeLeftDepth**—CallmaxDepth(root->left)tocomputethedepthoftheleftsubtree.
- 3. **RecursivelyComputeRightDepth**—CallmaxDepth(root->right)tocomputethedepthoftherightsubtree.
- 4. **CompareDepths**—Takethemaximumoftheleftandrightsubtreedepths.
- 5. **IncrementDepth**—Add1toincludethecurrentrootnodeinthedepthcount.
- 6. **Return Result**—Returnthecomputeddepthvalue.

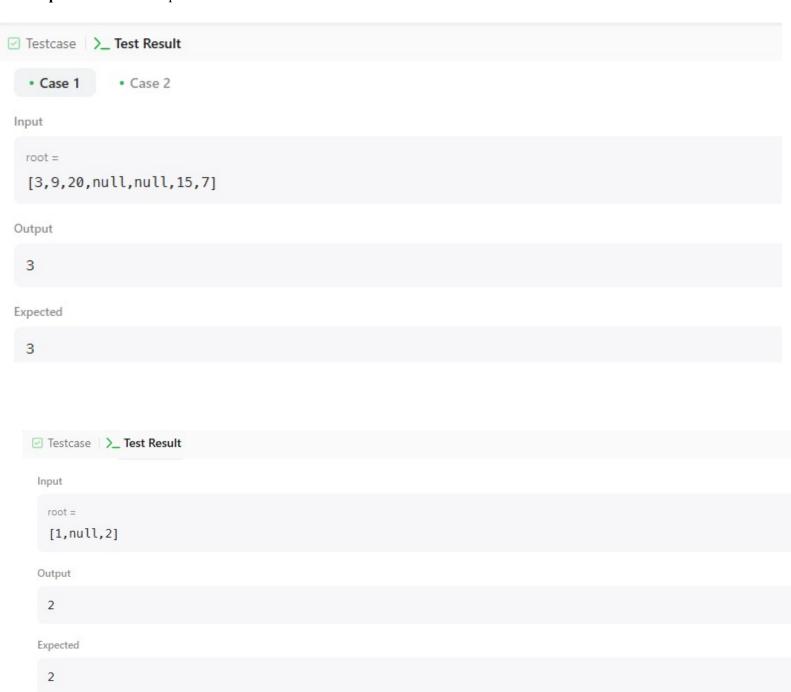
## **Code:**

```
classSolution{
public:
   intmaxDepth(TreeNode*root){ if
    (root == nullptr)
     return0;
   return1+max(maxDepth(root->left),maxDepth(root->right));
}
};
```

**TimeComplexity:**O(N)–Eachnodeisvisitedonce.

**SpaceComplexity:**O(N)–Worstcase(skewedtree),O(logN)–Bestcase(balancedtree).

## Output-Allthetestcases passed



## **Problem-2**

**Aim:**-Giventherootofabinary tree, return*itsmaximumdepth*. Abinarytree's maximumdepth isthenumber of nodes along the longest path from the root node down to the farthest leaf node.

**Objective-**The objective is to determine whether a given binary tree is a valid binary search tree (BST). Avalid BST ensures that foreachnode, itsleft subtree contains only smaller values, and its right subtree contains only larger values. This validation is crucial for efficient searching, sorting, and maintaining ordered data structures.

## Apparatus Used:

- 1. Software:-Leetcode
- 2. Hardware: Computer with 4GBRAM and keyboard.

## Algorithm to Checkifa Binary Tree is a Valid BST

- 1. InitializeFunction—Callvalid(root,LONG MIN,LONG MAX)tostartvalidation.
- 2. CheckBaseCondition-Ifthecurrentnodeisnullptr,returntrue(emptysubtreeis valid).
- 3. ValidateNodeValue-Ensurethenode'svalueiswithinthegivenrange(minimum<node->val<maximum).
- 4. **ReturnFalseifInvalid**—IfthenodeviolatestheBSTproperty,returnfalse.
- 5. **RecursivelyCheckLeftSubtree**—Callvalid(node->left,minimum,node->val).
- 6. RecursivelyCheckRightSubtree-Callvalid(node->right,node->val,maximum)andreturnthecombinedresult.

#### Code-

```
classSolution{
public:
    boolisValidBST(TreeNode*root){
    returnvalid(root,LONG_MIN,LONG_MAX);
}

private:
    boolvalid(TreeNode*node,longminimum,longmaximum){ if
        (!node) return true;
        if(!(node->val>minimum&&node->val<maximum))returnfalse;
        returnvalid(node->left,minimum,node->val)&&valid(node->right,node->val,maximum);
    }
};
```

TimeComplexity:O(N)-Eachnodeisvisited once.

SpaceComplexity: O(N)-Worst case (skewed tree), O(log N)-Best case (balanced tree).

# Result-Alltestcasespasses Accepted Runtime: 0 ms 0 • Case 1 • Case 2 Input root = [2,1,3]Output true Expected true Accepted Runtime: 0 ms 0 • Case 2 · Case 1 Input root =

[5,1,4,null,null,3,6]

Output

false

Expected

false

## **Problem-3**

**Aim-**Giventherootofabinarytree, *checkwhetheritisamirrorofitself* (i.e., symmetricarounditscenter).

**Objective**-Theobjectiveistodeterminewhetheragivenbinarytreeissymmetricarounditscenter. This involves checking if the left and right subtrees are mirror images of each other. The solution should efficiently compare corresponding nodes using recursion or iteration to verify structural and value-based symmetry.

## **Apparatus Used:**

- 1. Software:-Leetcode
- 2. Hardware: Computer with 4GBRAM and keyboard.

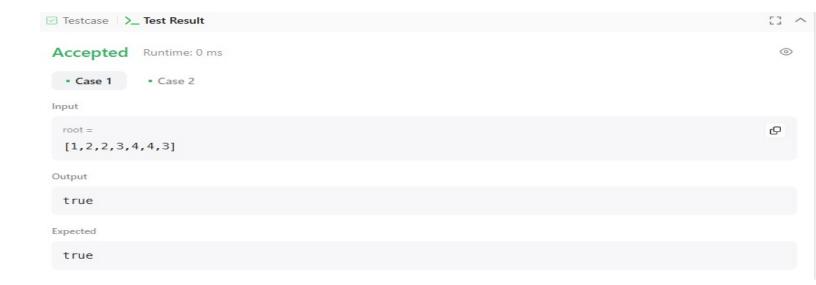
## AlgorithmtoCheckifaBinaryTreeisSymmetric

- 1. BaseCaseCheck:Iftherootisnullptr,returntrue(anemptytreeissymmetric).
- 2. CallHelperFunction:UseahelperfunctionisMirror()tocheckiftheleftandrightsubtreesaremirrorimages.
- 3. CheckNullNodes: Ifbothnodesarenullptr,returntrue.Ifonlyoneisnullptr,return false.
- 4. Compare Values: If the values of the two nodes do not match, return false.
- 5. RecursiveCheck:Recursivelycheckiftheleft subtreeofonetreematchestherightsubtreeoftheotherandviceversa.
- 6. ReturnResult:Returnthefinalresultafterallrecursivecomparisons.

#### Code-

```
classSolution{ public:
    boolisSymmetric(TreeNode*root){
        return isMirror(root->left, root->right);}    }
private: boolisMirror(TreeNode*n1,TreeNode*n2){
        if(n1==nullptr&&n2==nullptr){ return
            true; }
            if(n1==nullptr||n2==nullptr){ return
            false; }
            returnn1->val==n2->val&&isMirror(n1->left,n2->right)&&isMirror(n1->right,n2->left);
        }};
```

## Result-Alltestcases passes





## **LearningOutcomes**:

- 1. **UnderstandingTreeSymmetry:**Learnedhowtocheckifabinarytreeissymmetricbycomparingleftandrightsubtrees recursively.
- 2. **ValidatingBinarySearchTrees:**GainedknowledgeofverifyingwhetherabinarytreefollowsBSTpropertiesusing recursion and value constraints.
- 3. Computing TreeDepth: Explored recursive depth calculation to determine the height of a binary tree efficiently.
- 4. **RecursiveProblem-Solving:** Developedskillsinsolvingtree-relatedproblemsusingrecursionandunderstandingbase cases.
- 5. **HandlingEdgeCases:** Improvedunderstandingofhandling nullptrscenariostoensurerobustnessintreetraversal algorithms.