Experiment3

StudentName:Om Ankur Prajapati **UID:-22BCS15926**
Branch:BE-CSE **Group-602-A**
Semester:6TH **DateofPerformance:23-1-25**
SubjectName:-AdvanceProgramminglab-2 **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**:Computerwith4GBRAMand keyboard.

AlgorithmfortheTwoSumProblem:

1. **CheckBaseCondition**–Iftherootisnullptr,return0(emptytreehasdepth0).
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

Testcase | Test Result

• Case 1 • Case 2

Input

root =

```
[3,9,20,null,null,15,7]
```

Output

```
3
```

Expected

```
3
```

Testcase | Test Result

Input

root =

```
[1,null,2]
```

Output

```
2
```

Expected

```
2
```

Problem-2

Aim:-Giventherootofabinary tree, return*itsmaximumdepth*.Abinarytree's maximumdepth isthenumber ofnodes 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 subtreecontainsonlysmallervalues,anditsright subtreecontainsonlylargervalues.Thisvalidation is crucial for efficient searching, sorting, and maintaining ordered data structures.

Apparatus Used:

1. **Software**:-Leetcode
2. **Hardware**:Computerwith4GBRAMand keyboard.

AlgorithmtoCheckifaBinaryTreeisaValidBST

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)–Worstcase(skewedtree),O(logN)–Bestcase(balancedtree).

Result-Alltestcasespasses

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root =
[2,1,3]

Output

true

Expected

true

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

Output

false

Expected

false

Problem-3

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

Objective-Theobjectiveistodeterminewhetheragivenbinarytreeissymmetricarounditscenter.Thisinvolvescheckingifthe 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**:Computerwith4GBRAMand keyboard.

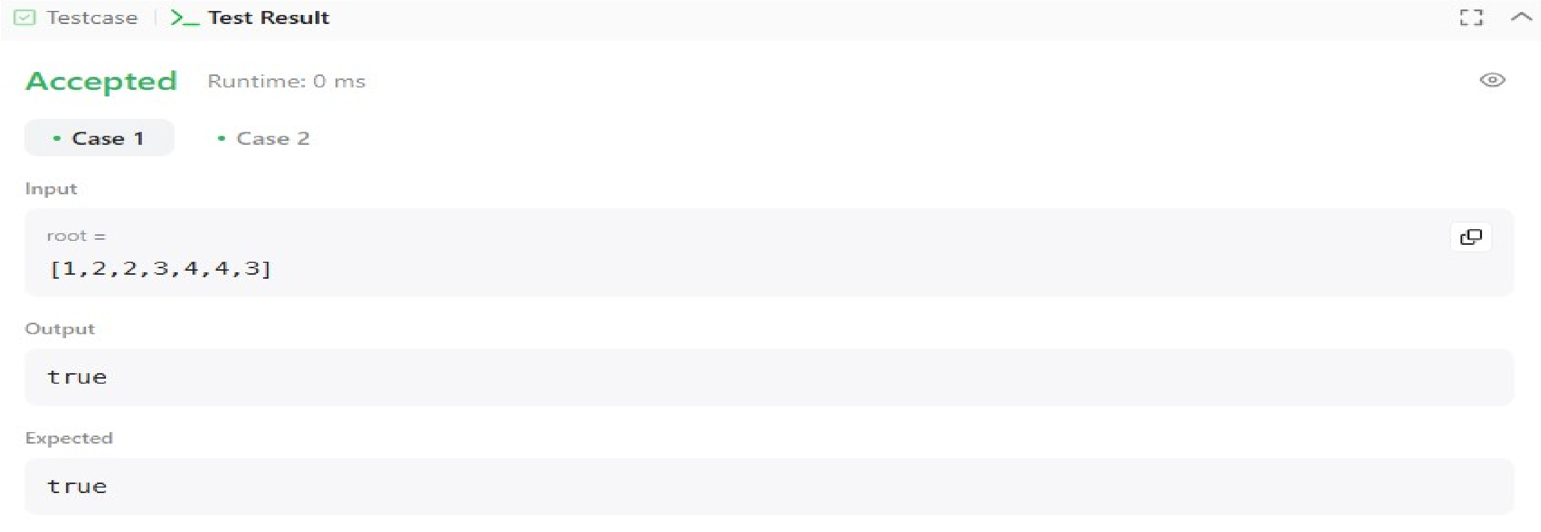
AlgorithmtoCheckifaBinaryTreeisSymmetric

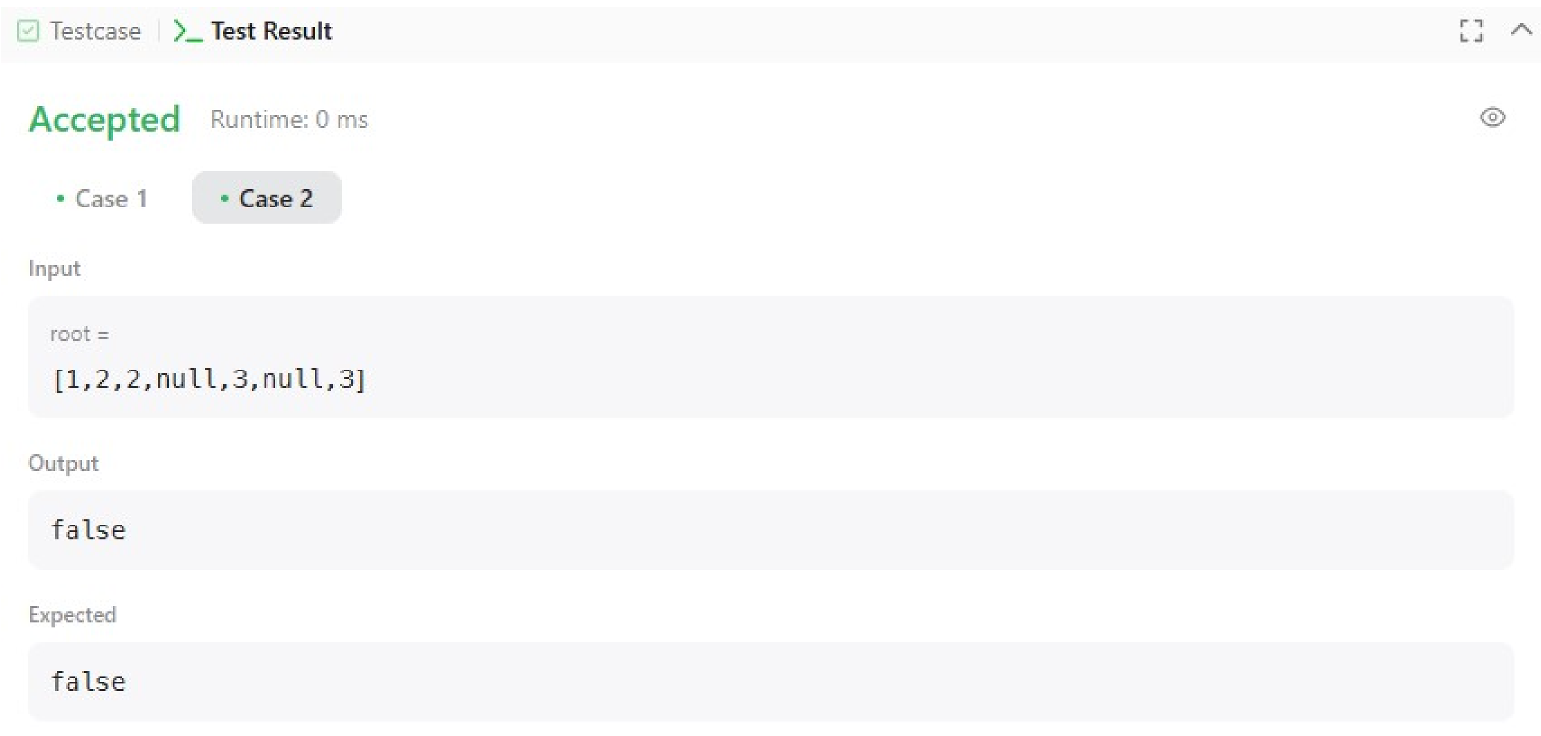
1. BaseCaseCheck:Iftherootisnullptr,returntrue(anemptytreeissymmetric).
2. CallHelperFunction:UseahelperfunctionisMirror()tocheckiftheleftandrightsubtreesaremirrorimages.
3. CheckNullNodes: Ifbothnodesarenullptr,returntrue.Ifonlyoneisnullptr,return false.
4. CompareValues:Ifthevaluesofthetwonodesdonotmatch,returnfalse.
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. **ComputingTreeDepth:**Exploredrecursivedepthcalculationtodeterminetheheightofabinarytreeefficiently.

4. **RecursiveProblem-Solving:**Developedskillsinsolvingtree-relatedproblemsusingrecursionandunderstandingbase cases.

5. **HandlingEdgeCases:** Improvedunderstandingofhandling nullptrscenariostoensurerobustnessintreetraversal algorithms.