GeeksforGeeks

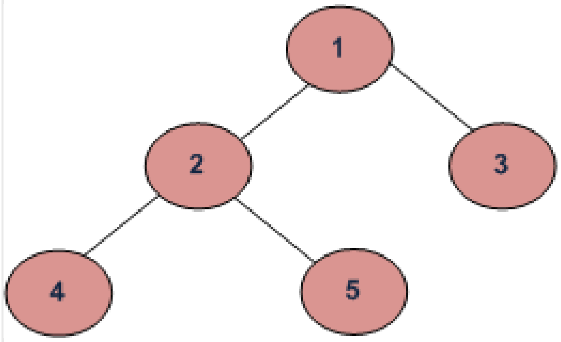
A computer science portal for geeks

Placements Practice GATE CS IDE Q&A
GeeksQuiz

Find Minimum Depth of a Binary Tree

Given a binary tree, find its minimum depth. The minimum depth is the number of nodes along the shortest path from root node down to the nearest leaf node.

For example, minimum height of below Binary Tree is 2.



Note that the path must end on a leaf node. For example, minimum height of below Binary Tree is also 2.

```
          10
        /    
      5
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to traverse the given Binary Tree. For every node, check if it is a leaf node. If yes, then return 1. If not leaf node then if left subtree is NULL, then recur for right subtree. And if right subtree is NULL, then recur for left subtree. If both left and right subtrees are not NULL, then take the minimum of two heights.

Below is implementation of the above idea.

C++

```
// C++ program to find minimum depth of a given Binary Tree
#include<bits/stdc++.h>
using namespace std;

// A BT Node
```

```
struct Node
{
    int data;
    struct Node* left, *right;
};

int minDepth(Node *root)
{
    // Corner case. Should never be hit unless the code is
    // called on root = NULL
    if (root == NULL)
        return 0;

    // Base case : Leaf Node. This accounts for height = 1.
    if (root->left == NULL && root->right == NULL)
       return 1;

    // If left subtree is NULL, recur for right subtree
    if (!root->left)
       return minDepth(root->right) + 1;

    // If right subtree is NULL, recur for right subtree
    if (!root->right)
       return minDepth(root->left) + 1;

    return min(minDepth(root->left), minDepth(root->right)) + 1;
}

// Utility function to create new Node
Node *newNode(int data)
{
    Node *temp = new Node;
    temp->data  = data;
    temp->left  = temp->right = NULL;
    return (temp);
}

// Driver program
int main()
{
    // Let us construct the Tree shown in the above figure
    Node *root        = newNode(1);
    root->left        = newNode(2);
    root->right       = newNode(3);
    root->left->left  = newNode(4);
    root->left->right = newNode(5);
    cout << minDepth(root);
    return 0;
}
```

Java

```
//Java implementation to find minimum depth of a given Binary tree

/* Class containing left and right child of current
     node and key value*/
class Node
{
    int data;
    Node left, right;
       public Node(int item)
    {
        data = item;
        left = right = null;
    }
}
public class BinaryTree
{
    //Root of the Binary Tree
    Node root;
     int minimumDepth()
     {
         return minimumDepth(root);
     }

    /* Function to calculate the minimum depth of the tree */
    int minimumDepth(Node root)
    {
     // Corner case. Should never be hit unless the code is
    // called on root = NULL
    if (root == null)
        return 0;

    // Base case : Leaf Node. This accounts for height = 1.
    if (root.left == null && root.right == null)
       return 1;

    // If left subtree is NULL, recur for right subtree
    if (root.left != null)
       return minimumDepth(root.right) + 1;

    // If right subtree is NULL, recur for right subtree
    if (root.right != null)
       return minimumDepth(root.left) + 1;

    return Math.min(minimumDepth(root.left), minimumDepth(root.right)) + 1;
    }
    /* Driver program to test above functions */
    public static void main(String args[])
    {
       BinaryTree tree= new BinaryTree();
       tree.root= new Node(1);
       tree.root.left= new Node(2);
       tree.root.right= new Node(3);
       tree.root.left.left= new Node(4);
       tree.root.left.right= new Node(5);

       System.out.println("The minimum depth of binary tree is : " + tree.minimumDepth());
    }
}
```

Python

```
# Python program to find minimum depth of a given Binary Tree

# Tree node
class Node:
    def __init__(self , key):
        self.data = key
        self.left = None
        self.right = None

def minDepth(root):
    # Corner Case.Should never be hit unless the code is
    # called on root = NULL
    if root is None:
        return 0

    # Base Case : Leaf node.This acoounts for height = 1
    if root.left is None and root.right is None:
        return 1

    # If left subtree is Null, recur for right subtree
    if root.left is None:
        return minDepth(root.right)+1

    # If right subtree is Null , recur for left subtree
    if root.right is None:
        return minDepth(root.left) +1

    return min(minDepth(root.left), minDepth(root.right))+1

# Driver Program
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print minDepth(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
2
```

Time complexity of above solution is O(n) as it traverses the tree only once.
Thanks to Gaurav Ahirwar for providing above solution.

The above method may end up with complete traversal of Binary Tree even when the topmost leaf is close to root. A **Better Solution** is to do Level Order Traversal. While doing traversal, returns depth of the first encountered leaf node. Below is implementation of this solution.

C

```
// C++ program to find minimum depth of a given Binary Tree
#include<bits/stdc++.h>
using namespace std;

// A Binary Tree Node
struct Node
{
    int data;
    struct Node *left, *right;
};

// A queue item (Stores pointer to node and an integer)
struct qItem
{
   Node *node;
   int depth;
};

// Iterative method to find minimum depth of Bianry Tree
int minDepth(Node *root)
{
    // Corner Case
    if (root == NULL)
        return 0;
    // Create an empty queue for level order tarversal
    queue<qItem> q;

    // Enqueue Root and initialize depth as 1
    qItem qi = {root, 1};
    q.push(qi);

    // Do level order traversal
    while (q.empty() == false)
    {
       // Remove the front queue item
       qi = q.front();
       q.pop();

       // Get details of the remove item
       Node *node = qi.node;
       int depth = qi.depth;

       // If this  is the first leaf node seen so far
       // Then return its depth as answer
       if (node->left == NULL && node->right == NULL)
          return depth;

       // If left subtree is not NULL, add it to queue
       if (node->left != NULL)
       {
          qi.node  = node->left;
          qi.depth = depth + 1;
          q.push(qi);
       }

       // If right subtree is not NULL, add it to queue
       if (node->right != NULL)
       {
          qi.node  = node->right;
          qi.depth = depth+1;
          q.push(qi);
       }
    }
    return 0;
}

// Utility function to create a new tree Node
Node* newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
```

```
// Driver program to test above functions
int main()
{
    // Let us create binary tree shown in above diagram
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    cout << minDepth(root);
    return 0;
}
```

Python

```
# Python program to find minimum depth of a given Binary Tree

# A Binary Tree node
class Node:
    # Utility to create new node
    def __init__(self , data):
        self.data = data
        self.left = None
        self.right = None

def minDepth(root):
    # Corner Case
    if root is None:
         return 0

    # Create an empty queue for level order traversal
    q = []

    # Enqueue root and initialize depth as 1
    q.append({'node': root , 'depth' : 1})

    # Do level order traversal
    while(len(q)>0):
        # Remove the front queue item
        queueItem = q.pop(0)

        # Get details of the removed item
        node = queueItem['node']
        depth = queueItem['depth']
        # If this is the first leaf node seen so far
        # then return its depth as answer
        if node.left is None and node.right is None:
            return depth

        # If left subtree is not None, add it to queue
        if node.left is not None:
            q.append({'node' : node.left , 'depth' : depth+1})

        # if right subtree is not None, add it to queue
        if node.right is not None:
            q.append({'node': node.right , 'depth' : depth+1})

# Driver program to test above function
# Lets construct a binary tree shown in above diagram
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print minDepth(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
2
```

Thanks to Manish Chauhan for suggesting above idea and Ravi for providing implementation.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

27 Comments Category: Trees

Related Posts:

- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?
- Closest leaf to a given node in Binary Tree

(Login to Rate and Mark)

2.2 Average Difficulty : **2.2/5.0**
Based on **13** vote(s)

□ Add to TODO List

□ Mark as DONE

Like Share 25 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

27 Comments **GeeksforGeeks** 1 Login

Recommend Share Sort by Newest

Join the discussion…

Akshat Misra • 19 days ago

```
// If left subtree is NULL, recur for right subtree

if (root.left != null)

return minimumDepth(root.right) + 1;

// If right subtree is NULL, recur for right subtree

if (root.right != null)

return minimumDepth(root.left) + 1;
```

The above section in first Java solution is wrong

It should be modified to :

```
// If left subtree is NULL, recur for right subtree

if (root.left == null)

return minimumDepth(root.right) + 1;

// If right subtree is NULL, recur for right subtree

if (root.right == null)

return minimumDepth(root.left) + 1;
```

1 ^ | ⌄ • Reply • Share ›

Vyacheslav • 23 days ago

Hi!

I guess both the Algorithm will work even without these extra if checks, when you call only one NOT NULL branch..

^ | ⌄ • Reply • Share ›

Vyacheslav ➦ Vyacheslav • 23 days ago

Ah, right, They are needed. Thank you Ayan

^ | ⌄ • Reply • Share ›

Ayushrazz Choudhary • 2 months ago

```
int mindeft(struct node *root)
{
if(root==NULL)
return INT_MAX;
if(!root->lchild && !root->rchild)
return 1;
return 1+min(mindeft(root->lchild),mindeft(root->rchild));
}
```

^ | ⌄ • Reply • Share ›

surbhijain93 • 2 months ago

```
int minDepth(node *root,int *min,int count)


{


if(root==NULL)


return 0;
if(root->left==NULL && root->right==NULL)

{

if(*min>count)

*min=count;

}

minDepth(root->left,min,count+1);

minDepth(root->right,min,count+1);

return *min;

}
```

• Reply • Share ›

r2t2 • 2 months ago

Here's my code in Java. It compiles fine on my machine but the webIDE complains about a missing ">".

http://code.geeksforgeeks.org/...

• Reply • Share ›

m3ghd007 • 4 months ago

```
int minDepth(Node *root)
{
return (root == NULL) ? 0 : min(minDepth(root->left), minDepth(root->right)) + 1;
}
```

Only this is sufficient.
Intermediate 3 checks are redundant.

• Reply • Share ›

> **r2t2** ➦ m3ghd007 • 2 months ago
>
> Actually, this violates the rule that the path of minDepth terminates in a leaf node. It was my first solution as well but realized the error.
>
> • Reply • Share ›

Dev • 5 months ago

Level order traversal using Java

http://code.geeksforgeeks.org/...

• Reply • Share ›

Subi114 • 5 months ago

Won't this simply work?

```
private static int MinHeightFinder(TreeNode root) {

if(root == null)
return 0;

return Math.min(MinHeightFinder(root.getLeft()), MinHeightFinder(root.getRight())) + 1;

}
```

• Reply • Share ›

Ayan Subi114 • 4 months ago

That would be wrong. Consider the following example:

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

The min height of this tree is 3. However, your function would return 2 as the min. Since, node 3 doesn't have a left child, and would return min(0,1)+1 which is 1 and is wrong. Node 3 should actually return 2. A similar Does that make sense?

• Reply • Share ›

Hiccup Subi114 • 5 months ago

This solution will work but we care about optimization. Think that tree is pretty deep and we are going deeper than looking near to root. This is not good in practice.

Any way in your approach we end up using system stack but suggested approach uses queue which is better. As soon as we get leaf we stop as our goal is to find minimum depth.
This is level order/ BFS traversal.

Hope you now clear.

1 • Reply • Share ›

r2t2 Hiccup • 2 months ago

@Hiccup, if the question asked for maxDepth instead of minDepth, would DFS be better? or is there anything better than DFS?

• Reply • Share ›

Sahil • 5 months ago

Why can't we simply store the level being traversed in a variable rather than storing depth along with each node? We can simply return the level being traversed + 1 on finding a leaf node.

• Reply • Share ›

Hiccup ➦ Sahil • 5 months ago

You can do that with extra work. You need a marker element in a queue. This will seperate levels. As soon as you see maker after getting front from queue you increase level. This will work.

But above solution is also good as int space is not huge for each item..and we push item in queue on demand basic and do not allocate for all nodes up front

Hope now you clear...

• Reply • Share ›

Tyrion • 5 months ago

In recursive solution, except the first if condition remaining if conditions are redundant.

1 • Reply • Share ›

vinit • 5 months ago

why do we need this extra code ? Why can't we have a check for NULL in the function itself (ex: if (root == NULL) return 0, and call 1+ min(left, right)directly ?

```
// If left subtree is NULL, recur for right subtree
if (!root->left)
return minDepth(root->right) + 1;

// If right subtree is NULL, recur for right subtree
if (!root->right)
return minDepth(root->left) + 1;
```

• Reply • Share ›

Dev • 5 months ago

If you are using recursion, would this following snippet not work?

```
public int minLevel(Node node) {
if(node == null) return 0;
int left = maxLevel(node.left);
int right = maxLevel(node.right);
return Math.min(left, right)+1;
}
```

˄ | ˅ • Reply • Share ›

Luis Gomez ➦ Dev • 5 months ago

your approach is good and will work, but using BFS instead of DFS works better for this case.

use BFS and break the loop when you find a node that does not add a any node to the queue (either left or right)

˄ | ˅ • Reply • Share ›

Dev ➦ Luis Gomez • 5 months ago

Thanks for the reply. I tried that as well Here is the code for it:

http://code.geeksforgeeks.org/...

˄ | ˅ • Reply • Share ›

nitin • 5 months ago

Another approach using recursion -
http://ideone.com/oJMH2D
Time Complexity - O(n)

˄ | ˅ • Reply • Share ›

Tomer Ben David • 5 months ago

How about this scala impl? https://github.com/tomer-ben-d...

˄ | ˅ • Reply • Share ›

code down • 5 months ago

For first solution, it might be a good idea to have corner case in main function and other code in a util function, so that we don't execute the corner case in every recursive call.

˄ | ˅ • Reply • Share ›

Manish Chauhan • 5 months ago

Use level order traversal. In worst case it will be "n" but will find the solution much faster for unbalanced trees.

4 ˄ | ˅ • Reply • Share ›

Ravi ➦ Manish Chauhan • 5 months ago

C++ code http://code.geeksforgeeks.org/...

2 ˄ | ˅ • Reply • Share ›

GeeksforGeeks Mod ➦ Manish Chauhan • 5 months ago

Thanks for suggesting this solution. We will soon be adding it to the original post.

1 ˄ | ˅ • Reply • Share ›

Mysterious Mind ➦ GeeksforGeeks • 5 months ago

@GeeksforGeeks improved DFS (Backtracking)

http://code.geeksforgeeks.org/...

Worst case o(n) space o(1) without considering recursion execution;

^ | ˅ • Reply • Share ›