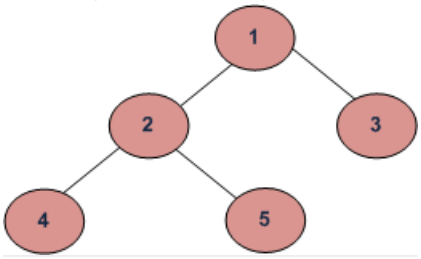
Find Minimum Depth of a Binary Tree

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;
```

```
    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
```