

## Minimum Depth of a Binary Tree (easy)

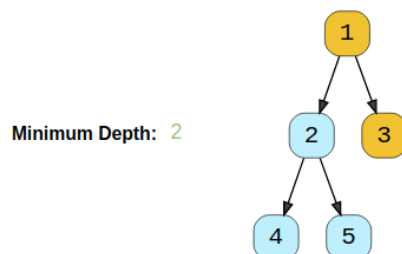
### We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time complexity
  - Space complexity
- Similar Problems

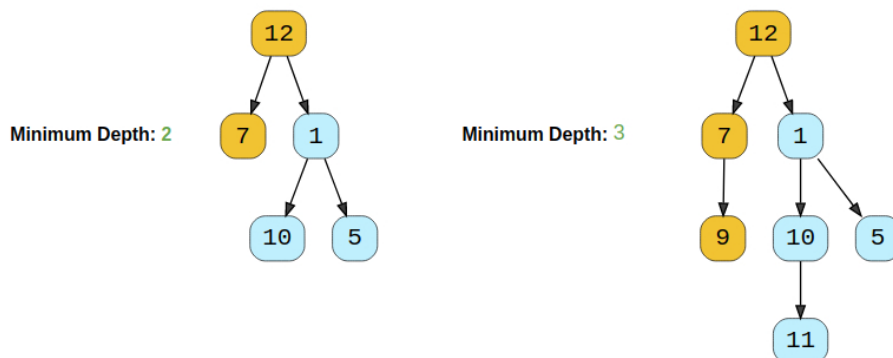
## Problem Statement #

Find the minimum depth of a binary tree. The minimum depth is the number of nodes along the **shortest path from the root node to the nearest leaf node**.

**Example 1:**



**Example 2:**



## Try it yourself #

Try solving this question here:

```
1 class TreeNode:
2     def __init__(self, val):
3         self.val = val
4         self.left, self.right = None, None
5
6
7 def find_minimum_depth(root):
8     # TODO: Write your code here
9     return -1
10
11
12 def main():
13     root = TreeNode(12)
14     root.left = TreeNode(7)
15     root.right = TreeNode(1)
16     root.right.left = TreeNode(10)
17     root.right.right = TreeNode(5)
18     print("Tree Minimum Depth: " + str(find_minimum_depth(root)))
19     root.left.left = TreeNode(9)
20     root.right.left.left = TreeNode(11)
21     print("Tree Minimum Depth: " + str(find_minimum_depth(root)))
22
23
24 main()
25
```

Run Save Reset

## Solution #

This problem follows the [Binary Tree Level Order Traversal](#) pattern. We can follow the same **BFS** approach. The only difference will be, instead of keeping track of all the nodes in a level, we will only track the depth of the tree. As soon as we find our first leaf node, that level will represent the minimum depth of the tree.

## Code #

Here is what our algorithm will look like, only the highlighted lines have changed:

Java	Python3	C++	JS
------	---------	-----	----

```
1 from collections import deque
2
3
4 class TreeNode:
5     def __init__(self, val):
6         self.val = val
7         self.left, self.right = None, None
8
9
10 def find_minimum_depth(root):
11     if root is None:
12         return 0
13
14     queue = deque()
15     queue.append(root)
16     minimumTreeDepth = 0
17     while queue:
18         minimumTreeDepth += 1
19         levelSize = len(queue)
20         for _ in range(levelSize):
21             currentNode = queue.popleft()
22
23             # check if this is a leaf node
24             if not currentNode.left and not currentNode.right:
25                 return minimumTreeDepth
26
27             # insert the children of current node in the queue
28             if currentNode.left:
29                 queue.append(currentNode.left)
30             if currentNode.right:
31                 queue.append(currentNode.right)
32
33
34 def main():
35     root = TreeNode(12)
36     root.left = TreeNode(7)
```

```

37 root.left = TreeNode(7)
38 root.right = TreeNode(1)
39 root.right.left = TreeNode(10)
40 root.right.right = TreeNode(5)
41 print("Tree Minimum Depth: " + str(find_minimum_depth(root)))
42 root.left.left = TreeNode(9)
43 root.right.left.left = TreeNode(11)
44 print("Tree Minimum Depth: " + str(find_minimum_depth(root)))
45
46
47 main()
48

```

Run

Save

Reset



## Time complexity #

The time complexity of the above algorithm is  $O(N)$ , where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

## Space complexity #

The space complexity of the above algorithm will be  $O(N)$  which is required for the queue. Since we can have a maximum of  $N/2$  nodes at any level (this could happen only at the lowest level), therefore we will need  $O(N)$  space to store them in the queue.

## Similar Problems #

**Problem 1:** Given a binary tree, find its maximum depth (or height).

**Solution:** We will follow a similar approach. Instead of returning as soon as we find a leaf node, we will keep traversing for all the levels, incrementing `maximumDepth` each time we complete a level. Here is what the code will look like:

Java	Python3	C++	JS
------	---------	-----	----

```

1 from collections import deque
2
3
4 class TreeNode:
5     def __init__(self, val):
6         self.val = val
7         self.left, self.right = None, None
8
9
10 def find_maximum_depth(root):
11     if root is None:
12         return 0
13
14     queue = deque()
15     queue.append(root)
16     maximumTreeDepth = 0
17     while queue:
18         maximumTreeDepth += 1
19         levelSize = len(queue)
20         for _ in range(levelSize):
21             currentNode = queue.popleft()
22
23             # insert the children of current node in the queue
24             if currentNode.left:
25                 queue.append(currentNode.left)
26             if currentNode.right:
27                 queue.append(currentNode.right)
28
29     return maximumTreeDepth
30
31
32 def main():
33     root = TreeNode(12)
34     root.left = TreeNode(7)
35     root.right = TreeNode(1)
36     root.right.left = TreeNode(10)
37     root.right.right = TreeNode(5)
38     print("Tree Maximum Depth: " + str(find_maximum_depth(root)))
39     root.left.left = TreeNode(9)
40     root.right.left.left = TreeNode(11)

```

```
41 | print("Tree Maximum Depth: " + str(find_maximum_depth(root)))
42 |
43 |
44 | main()
45 |
```

Run

Save

Reset



← Back

Next →

Level Averages in a Binary Tree (easy)

Level Order Successor (easy)

✓ Completed

⚠ Report an Issue   ? Ask a Question