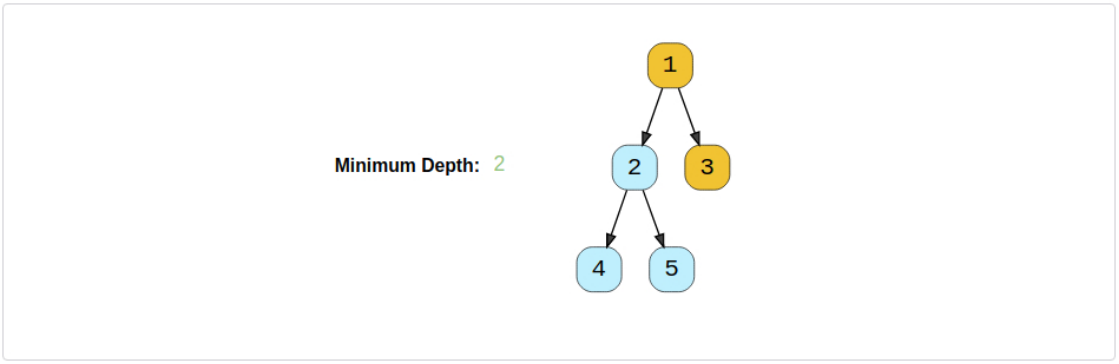
Minimum Depth of a Binary Tree (easy)

We'll cover the following

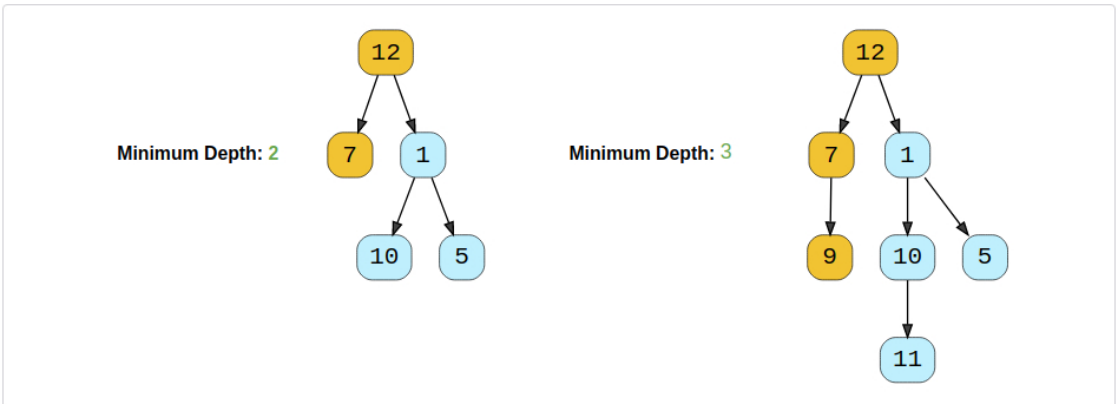
Problem Statement

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:

```
  def __init__(self, val):
    self.val = val
    self.left, self.right = None, None


def find_minimum_depth(root):
  # TODO: Write your code here
  return -1


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  print("Tree Minimum Depth: " + str(find_minimum_depth(root)))
  root.left.left = TreeNode(9)
  root.right.left.left = TreeNode(11)
  print("Tree Minimum Depth: " + str(find_minimum_depth(root)))


main()
```

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

```
from collections import deque


class TreeNode:
  def __init__(self, val):
    self.val = val
    self.left, self.right = None, None


def find_minimum_depth(root):
  if root is None:
    return 0

  queue = deque()
  queue.append(root)
  minimumTreeDepth = 0
  while queue:
    minimumTreeDepth += 1
    levelSize = len(queue)
    for _ in range(levelSize):
      currentNode = queue.popleft()

      # check if this is a leaf node
      if not currentNode.left and not currentNode.right:
        return minimumTreeDepth

      # insert the children of current node in the queue
      if currentNode.left:
        queue.append(currentNode.left)
      if currentNode.right:
        queue.append(currentNode.right)


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
```

```
  root.right = TreeNode(1)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  print("Tree Minimum Depth: " + str(find_minimum_depth(root)))
  root.left.left = TreeNode(9)
  root.right.left.left = TreeNode(11)
  print("Tree Minimum Depth: " + str(find_minimum_depth(root)))


main()

```

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

```python
from collections import deque


class TreeNode:
  def __init__(self, val):
    self.val = val
    self.left, self.right = None, None


def find_maximum_depth(root):
  if root is None:
    return 0

  queue = deque()
  queue.append(root)
  maximumTreeDepth = 0
  while queue:
    maximumTreeDepth += 1
    levelSize = len(queue)
    for _ in range(levelSize):
      currentNode = queue.popleft()

      # insert the children of current node in the queue
      if currentNode.left:
        queue.append(currentNode.left)
      if currentNode.right:
        queue.append(currentNode.right)

  return maximumTreeDepth


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  print("Tree Maximum Depth: " + str(find_maximum_depth(root)))
  root.left.left = TreeNode(9)
  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
Level Averages in a Binary Tree (easy)

Next
Level Order Successor (easy)

Completed