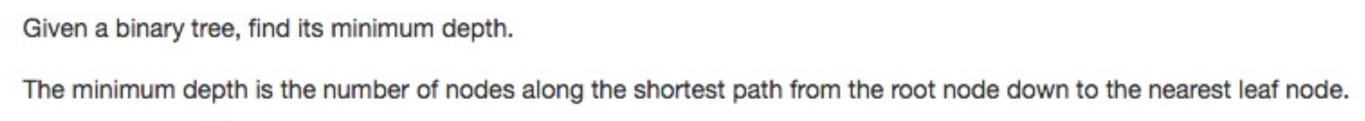
111. Minimum Depth of Binary Tree

111 Minimum Depth of Binary Tree

- **Depth-first Search** + Tree
- **Breath-first Search** + Tree + Queue

Description



Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

1. Thought line

- The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.
- The key is to find the LEAF node.

2. Depth-first Search + Tree

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
private:
    void depthFirstSearchMinDepth(TreeNode* node, int& res, int depOfNode){
        if (node == nullptr) return;
        ++depOfNode;
        if (node->left==nullptr && node->right==nullptr && res>depOfNode) res = depOfNode;
        depthFirstSearchMinDepth(node->left, res, depOfNode);
        depthFirstSearchMinDepth(node->right, res, depOfNode);

    }
public:
    int minDepth(TreeNode* root) {
        if (root == nullptr) return 0;
        int res = INT_MAX;
        depthFirstSearchMinDepth(root, res, 0);
        return res;
    }
}
```

3. Breadth-first Search + Tree + Queue

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int minDepth(TreeNode* root) {
        if (root == nullptr) return 0;

        int curDep = 1; // When the root is not null, the minimal depth is 1.
        queue<TreeNode*> que;
        que.emplace(root);

        while(!que.empty()){
            // detect if the next lay has both children
            queue<TreeNode*> nextLay;
            while (!que.empty()){
                // find the leaf node
                if (que.front()->left==nullptr && que.front()->right==nullptr)
                    return curDep;
                if (que.front()->left) nextLay.push(que.front()->left);
                if (que.front()->right) nextLay.push(que.front()->right);
                que.pop();
            }
            ++curDep;
            que.swap(nextLay);
        }
        return curDep;
    }
};
```