Day 16 / 100 :

Topic - Binary tree, Linked List

1 Problem statement: **Minimum depth of binary tree** (easy)

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.
Note: A leaf is a node with no children.

Example 1:

Input: root = [3,9,20,null,null,15,7]
Output: 2
Example 2:
Input: root = [2,null,3,null,4,null,5,null,6]
Output: 5

Solutions :

Approach 1 - DFS

Intuition:
The intuition is pretty straight forward we need to find the minimum distance so how to get the required answer?

Approach:
The approach that comes to mind is we need traversal because we cannot go from one place to another without travelling throught the road so we know that we need traversal but we have got two DFS BFS both will do the work.So can use any
I use DFS maintain a variable answer which intially contains the greatest possible integer value you know why? yes, because we need to calculate the minimum possible distance and we need something bigg to start with.
So just maintain a ans variable and a current sum variable that allows us to traverse further.
Go to each node and keep on incrementing the current sum variable.

Now when we reach our destination i.e the leaf node we calculate the min(answer,current Sum+1) +1 because we are also considering that leaf node.

Complexity:
Time complexity:
Same as Normal DFS O(Nodes);

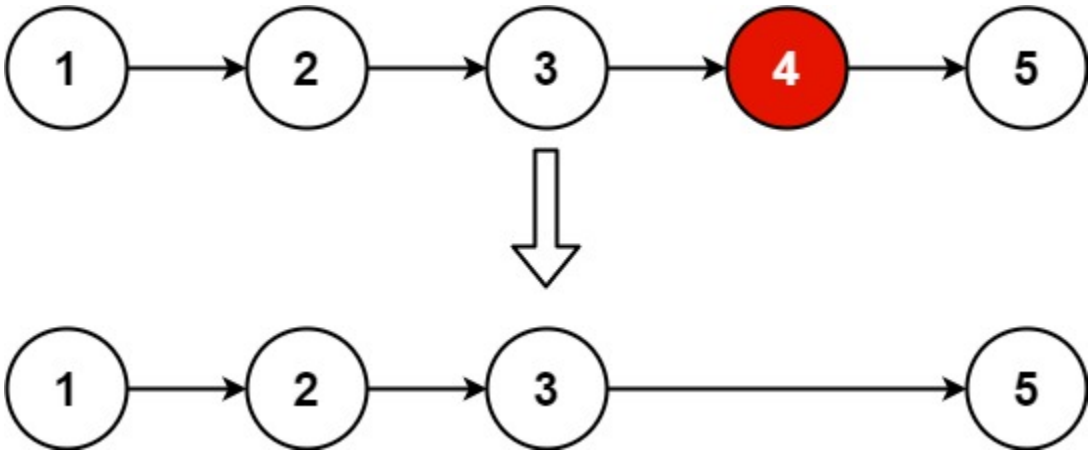
Space complexity:
O(1) we do not have any external container as such.

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
left(left), right(right) {}
 * };
 */
class Solution {
public:
    void dfs(TreeNode* node, int currS, int& ans){
        if(node->left==NULL and node->right==NULL){
            ans=min(ans,currS+1);
            return;
        }
        if(node->left)dfs(node->left,currS+1,ans);
        if(node->right)dfs(node->right,currS+1,ans);

    }
    int minDepth(TreeNode* root) {
        int ans=INT_MAX;
        if(root==NULL) return 0;
        dfs(root,0,ans);
        return ans;
    }};
```

2 Problem statement: Remove Nth node from end of list (Medium)

Given the head of a linked list, remove the nth node from the end of the list and return its head.
Example 1:



Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
Example 2:

Input: head = [1], n = 1
Output: []
Example 3:

Input: head = [1,2], n = 1
Output: [1]

Solutions :

Approach 1 - Traversing and Skipping

Intuition:
Find the element to be removed. Traverse till that element and skip that element.

Approach:

1. Traverse the linked list once to find the length of it.
2. Subtract n to get the index of the element to be removed.
3. Traverse the linked list again, reach that index and point it to it's next to next element.

Complexity:
Time complexity: O(n)
Space complexity: O(1)

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        // getting the length of the Linked List
        int len = 0;
        ListNode *temp = head;
        while(temp)
        {
            len++;
            temp = temp->next;
        }
        // getting the index of the element to be removed
        int index = len - n, i = 0;
        // edge case if the Linked List contain only 1 element
        if(index == 0)
            head = head->next;

        ListNode *t = head;
        while(t)
        {
            i++;
            // after reaching that index, skip that element
            if(index == i and t->next)
                t->next = t->next->next;
            t = t->next;
        }
        return head;
    }
};
```