Time to burn tree

Problem statement Send feedback

You have a binary tree of **'N'** unique nodes and a ***Start*** node from where the tree will start to burn. Given that the Start node will always exist in the tree, your task is to print the time (in minutes) that it will take to burn the whole tree.

It is given that it takes 1 minute for the fire to travel from the burning node to its adjacent node and burn down the adjacent node.

For Example :

For the given binary tree: [1, 2, 3, -1, -1, 4, 5, -1, -1, -1, -1]
Start Node: 3

```
  1
 /\
2  3
  /\
  4  5
```

Output: 2

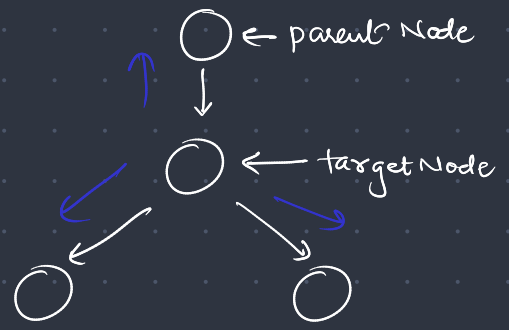
Explanation :
In the zeroth minute, Node 3 will start to burn.

After one minute, Nodes (1, 4, 5) that are adjacent to 3 will burn completely.

After two minutes, the only remaining Node 2 will be burnt and there will be no nodes remaining in the binary tree.

So, the whole tree will burn in 2 minutes.

→ Approach :-



→ We need to delete the left, right & parent node.

↓

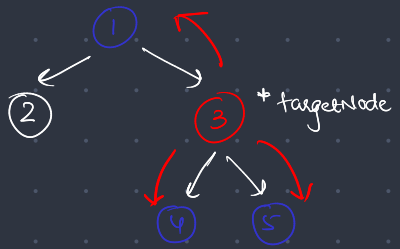
We need to create a mapping for node to parent.

Step I : Create mapping

5	3
4	3
3	1
2	1
1	NULL

Step II : Find the targetNode. Get the targetNode. OR you can get the targetNode while creating the mapping only.

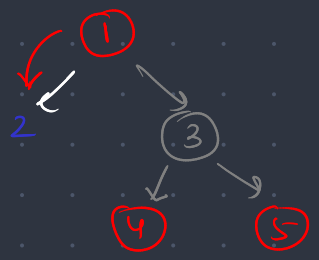
Step III :



3 4 5 1

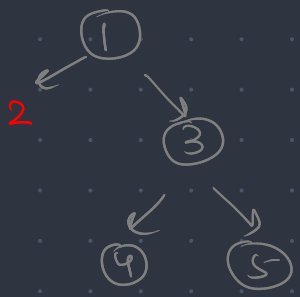
Increament time by 1.

After 1s:



3 4 5 1 2

After 2s:



Whole tree burnt in 2 seconds

→ Approach 2 :-

Using Distance of farthest Node.

```cpp
#include<bits/stdc++.h>

BinaryTreeNode<int>* createParentMapping(BinaryTreeNode<int>* root, int target, map<BinaryTreeNode<int>*, BinaryTreeNode<int>*> &nodeToParent) {
    BinaryTreeNode<int>* res = NULL;

    queue<BinaryTreeNode<int>*>q;
    q.push(root);
    nodeToParent[root] = NULL;

    while(!q.empty()) {
        BinaryTreeNode<int>* front = q.front();
        q.pop();
        if(front->data == target) {
            res = front;
        }
        if(front->left) {
            nodeToParent[front->left] = front;
            q.push(front->left);
        }
        if(front->right) {
            nodeToParent[front->right] = front;
            q.push(front->right);
        }
    }
    return res;
}

int timeToBurnTree(BinaryTreeNode<int>* root, int start) {
    if (root == nullptr)
       return 0;

    map<BinaryTreeNode<int>*, bool> visited;
    map<BinaryTreeNode<int>*, BinaryTreeNode<int>*> nodeToParent;

    BinaryTreeNode<int>* targetNode = createParentMapping(root, start, nodeToParent);

    int timeToBurn = 0;

    queue<BinaryTreeNode<int>*> q;
    q.push(targetNode);
    visited[targetNode] = true;

    while (!q.empty()) {
        int size = q.size();
        bool isPushed = false;

        while (size--) {
            BinaryTreeNode<int>* front = q.front();
            q.pop();

            if (front->left && !visited[front->left]) {
                q.push(front->left);
                visited[front->left] = true;
                isPushed = true;
            }

            if (front->right && !visited[front->right]) {
                q.push(front->right);
                visited[front->right] = true;
                isPushed = true;
            }

            if (nodeToParent[front] && !visited[nodeToParent[front]]) {
                q.push(nodeToParent[front]);
                visited[nodeToParent[front]] = true;
                isPushed = true;
            }
        }
        if (isPushed) {
            timeToBurn++;
        }
    }

    return timeToBurn;
}
```

Time Complexity : O(N)

Space Complexity : O(N)