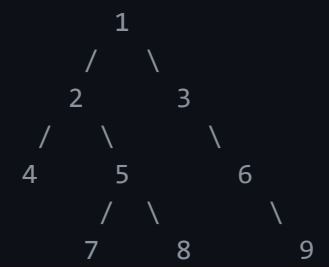
Burn a Binary Tree Full Explanation

By : Luvkush Sharma

Btech CS(Hons.)

```
/*



Given a binary tree and a node called target. Find the minimum time required to burn the complete binary tree if the
target
is set on fire. It is known that in 1 second all nodes connected to a given node get burned. That is its left child,
right child, and parent.

Rules fpr burning the nodes :

1. Fire will spread constantly to the connected nodes only.

2. Every node takes the same time to burn.

3. A nose burns only once.
```

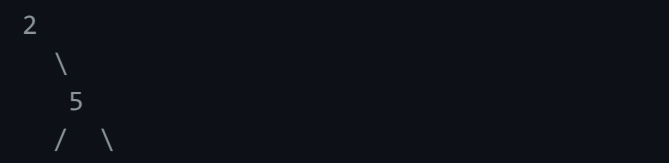


```
                                                \
                                                10

                    Let's say Target Node : 6

                    in 1 sec 6 burns it's neighbouring nodes i.e 3 and 9
                    in next 1 sec 3 burns it's neighbouring node i.e 1 and 9 burns it's neighbouring node i.e 10
                    in another 1 sec  1 burns it's neighbouring node i.e 2
                    in next 1 sec 2 burns it's neighbouring nodes i.e 4 and 5
                    in another next 1 sec 5 burns it's neighbouring nodes i.e 7 and 8.

                    And the whole tree burns in 5 sec.

                    Target Node : 8
                    Output : 7

                    Explanation : If leaf with the value 8 is set on fire.
                    After 1 sec: 5 is set on fire.
                    After 2 sec: neighbouring nodes of 5 i.e 2,7 is set on fire.
                    After 3 sec: neighbouring node of 2 i.e 4,1 is set on fire.
                    After 4 sec: 3 is set on fire.
                    After 5 sec: 6 is set on fire.
                    After 6 sec: 9 is set on fire.
                    After 7 sec: 10 is set on fire.

                    It takes 7s to burn the complete tree.


*/

/*
```

Approach :

Step - 1:

we have the neighbouring nodes of a node as it's left and right child
i.e



neighbouring nodes of 5 are 7 and 2.
We can easily go to the 7 as it is the left child of 5 but going on to the node having data as 2 is difficult

So, we will store the parent node also.
Since for a node we have 3 things to burn
1. It's left child i.e 7 here
2. It's right child i.e 8 already burnt.
3. And it's parent i.e 2

```
-----------------------------------------------------
So, Create mapping of each node with it''s parent. |
                                                   |
map < node*, node* > NodeToParent;                 | <---- Time complexity is O(n)
-----------------------------------------------------
```

Step - 2:

```
------------------
Find target node. |  <---- Time complexity is O(Height)
```

```
-------------------


Step - 3:

We will create a map which will store that node that we are going to burnt is visited or not.
If it is visited it means we had burnt that node already.

map < node* , bool > visited;

bool will tell true --> if node is visited
or false if node is not visited.

In the starting assign 8 as true.

queue <node*> ---> Traversal using target node.

initially we will store taregt in the queue.

-----------------------
8                      ---->  queue.
-----------------------

then 5 will be inserted.

-----------------------
pop(8) |  5   |                     ---->  queue.
-----------------------

Checking Left child, Right child and parent node of a node.
```

Like for 7 ---> no left child and right child is there it has only it's parent node.
So, now store that parent node into the queue.

And initialize time = 0;

Step - 4:

if there was any addition in queue.
Than check is it visited or not.
If not i.e true then time++.

If false then no increamenting in the time.

```
-------------------------
pop(8) |  pop(5)   |  7                    ---->  queue.
-------------------------
```

And map will loke like :

```
-----------------------------------------------------------------------------------------
8 : true | 5 : true | 7 : true | 2 : false |  3 : false |  4 : false |  5 : false | 6 : false |.....
-----------------------------------------------------------------------------------------
```

Code In C++ :-

```cpp
#include<bits/stdc++.h>
using namespace std;

class TreeNode{


    public:
      int data;
      TreeNode* left;
      TreeNode* right;

      // Constructor
      TreeNode(int val){
        this -> data = val;
        this -> left = NULL;
        this -> right = NULL;
      }
};

TreeNode* Buildtree(TreeNode* root){

     cout << "Enter the data : " << endl;
     int data;
```

```
    cin >> data;
    root = new TreeNode(data); // Node creates as constructor calls

    if(data == -1){
       return NULL;

    }

    cout << "------- Enter data for inserting in left of " << data << " -------- " << endl;
    root -> left = Buildtree(root -> left);


    cout << "------- Enter data for inserting in right of " << data << " -------- " << endl;
    root -> right = Buildtree(root -> right);

    return root;

}

// Create mapping.
// and returns target.
TreeNode* CreateParentMapping(TreeNode* root , int target , map<TreeNode*, TreeNode*> &nodeToParent ){

      TreeNode* res =  NULL;

      queue<TreeNode*> q;
      q.push(root);

      // Since root node has no parent i.e root node is mapped with NULL
      nodeToParent[root] = NULL;
```

```
        while(! q.empty()){
            TreeNode* frontNode = q.front();
            q.pop();

            if(frontNode -> data == target){

                  // Lage haat target node bhi search ho gayi.
                  // res will store the target node.
                  res = frontNode;
            }

            if(frontNode -> left != NULL){

                 //Mapping frontNode -> left with it's parent
                 nodeToParent[frontNode -> left] = frontNode;
                 q.push(frontNode -> left);
            }

            if(frontNode -> right != NULL){

                //Mapping frontNode -> right with it's parent
                 nodeToParent[frontNode -> right] = frontNode;
                 q.push(frontNode -> right);
            }

        }
           return res;
}

int burnTree(TreeNode* TargetNode , map <TreeNode* , TreeNode*> &nodeToParent){
```

```cpp
    map< TreeNode*, bool > visited;
    queue <TreeNode*> q;

    q.push(TargetNode);

    // Initially TargetNode is visited i.e store true means visited.
    visited[TargetNode] = true;

    // Time to burn the tree
    int ans = 0;

    while(! q.empty()){

         bool flag = false; // If flag changes to 1 it means node is burnt then increament the time.
         // Even, if a single node goes to the queue then it means that node is burnt and we do flag = true for that
         int Size = q.size();

         // We will run the loop till Size
         // We will burn all the neighbours of a node using for loop

         for(int i = 0; i < Size ; i++){

            TreeNode* frontNode = q.front();
            q.pop();

            // Burning frontNode -> left
            if(frontNode -> left != NULL && !visited[frontNode -> left]){
               flag = true;
               q.push(frontNode -> left);
               visited[frontNode -> left] = true; // Marking frontNode -> left as true.
```

```
            }

            // Burning frontNode -> right
            if(frontNode -> right != NULL && !visited[frontNode -> right]){
               flag = true;
               q.push(frontNode -> right);
               visited[frontNode -> right] = true; // Marking frontNode -> right as true.

            }

           // Burning frontNode's parent also
            if(nodeToParent[frontNode] && !visited[nodeToParent[frontNode]]){ // nodeToParent[frontNode] ---> Gives us
Parent of frontNode
                flag = true;
                q.push(nodeToParent[frontNode]);
                visited[nodeToParent[frontNode]] = true; // Marking frontNode's parent as visited
            }

            }

         if(flag == true){
            ans++;
         }

     }

      return ans;
}

int minTime(TreeNode* root, int target){
```

```
    // algo:
    // Step 1 : Create nodeToParent mapping
    // Step 2: Find Target node
    // Step 3: Burn the tree in min time.

    map <TreeNode* , TreeNode*> nodeToParent;

     // T.C --> for this is O(n)

    // This will create the mapping between node and it's parent node
    TreeNode* TargetNode = CreateParentMapping(root , target , nodeToParent);

     // T.C --> for this is O(n)
    int ans = burnTree(TargetNode , nodeToParent);

     // Total time complexity is O(n)
     // Space complexity is O(n)

     return ans;
}


int main(){

   TreeNode* root = NULL;

   root = Buildtree(root);

   int target;
   cout << "Enter the target : " << endl;
   cin >> target;
```

```
    int time = minTime(root , target);
    cout << "Minimum time to burn the whole tree is : ";
    cout << time << endl;

    // 1 2 4 -1 -1 5 7 -1 -1 8 -1 -1 3 -1 6 -1 9 -1 10 -1 -1
    // 8

}
```

END OF DOCUMENT