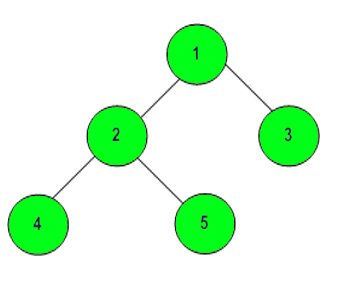
**Kth Ancestor in a Tree :-**

Medium Accuracy: 35.06% Submissions: 63K+ Points: 4

Given a binary tree of size  **N**, a **node,** and a positive integer **k**., Your task is to complete the function **kthAncestor()**, the function should return the **kth** ancestor of the given node in the binary tree. If there does not exist any such ancestor then return -1.  
**Note**: It is guaranteed that the **node** exists in the tree.

**Example 1:**



**Input:**

K = 2 Node = 4

**Output:** 1

**Explanation:**

Since, K is 2 and node is 4, so we

first need to locate the node and

look k times its ancestors.

Here in this Case node 4 has 1 as his

2nd Ancestor aka the Root of the tree.

**Example 2:**

**Input:**

k=1

node=3

1

/ \

2 3

**Output:**

1

**Explanation:**

K=1 and node=3 ,Kth ancestor of node 3 is 1.

**Your Task:**  
You are asked to complete the function **kthAncestor()** which accepts **root** of the tree, **k** and **node** as input parameters, and returns the **kth ancestor**of Node which contains node as its value.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(N)

**Constraints:**  
1<=N<=1051<= K <= 100  
1 <= Node.data <= N

**Code :-**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

};

Node\* newNode(int val)

{

Node\* temp = new Node;

temp->data = val;

temp->left = NULL;

temp->right = NULL;

return temp;

}

Node\* buildTree(string str)

{

// Corner Case

if(str.length() == 0 || str[0] == 'N')

return NULL;

// Creating vector of strings from input

// string after spliting by space

vector<string> ip;

istringstream iss(str);

for(string str; iss >> str; )

ip.push\_back(str);

// Create the root of the tree

Node\* root = newNode(stoi(ip[0]));

// Push the root to the queue

queue<Node\*> queue;

queue.push(root);

// Starting from the second element

int i = 1;

while(!queue.empty() && i < ip.size()) {

// Get and remove the front of the queue

Node\* currNode = queue.front();

queue.pop();

// Get the current node's value from the string

string currVal = ip[i];

// If the left child is not null

if(currVal != "N") {

// Create the left child for the current node

currNode->left = newNode(stoi(currVal));

// Push it to the queue

queue.push(currNode->left);

}

// For the right child

i++;

if(i >= ip.size())

break;

currVal = ip[i];

// If the right child is not null

if(currVal != "N") {

// Create the right child for the current node

currNode->right = newNode(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

int kthAncestor(Node \*root, int k, int node);

int main()

{

int t;

scanf("%d ",&t);

while(t--)

{

int k , node;

scanf("%d ",&k);

scanf("%d ",&node);

string s;

getline(cin,s);

Node\* root = buildTree(s);

cout<<kthAncestor(root,k,node)<<endl;

}

return 0;

}

// } Driver Code Ends

//User function Template for C++

void dfs(Node \*root, int k, int node, int &help, int &ans){

//base case

if(!root) return;

//calculations of pre-order traversal

if(root->data == node){

help--;

return;

}

//recursive calls

dfs(root->left, k, node, help, ans);

if(help < k){

help--;

if(help == -1)

ans = root->data;

return;

}

dfs(root->right, k, node, help, ans);

if(help < k){

help--;

if(help == -1)

ans = root->data;

}

return;

}

// your task is to complete this function

int kthAncestor(Node \*root, int k, int node){

int ans=-1;

dfs(root, k, node, k, ans);

return ans;

}

**T.C :- O(N)**

**S.C :- O(1) without recursive call stack**