**Nodes at given distance in binary tree: -**

**Hard** Accuracy: **32.36%** Submissions: **65K+** Points: **8**

Given a binary tree, a target node in the binary tree, and an integer value k, find all the nodes that are at distance k from the given target node. No parent pointers are available.  
**Note**:

* You have to return the list in sorted order.
* The tree will **not**contain **duplicate**values.

**Example 1:**

**Input:**

20

/ \

8 22

/ \

4 12

/ \

10 14

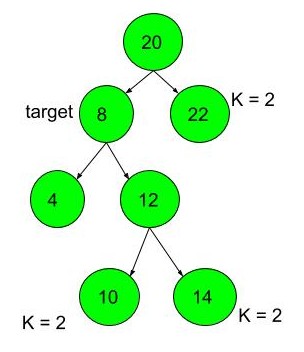
Target Node = 8

K = 2

**Output:** 10 14 22

**Explanation:** The three nodes at distance 2

from node 8 are 10, 14, 22.



**Example 2:**

**Input:**

20

/ \

7 24

/ \

4 3

/

1

Target Node = 7

K = 2

**Output:** 1 24

**Your Task:**  
You don't need to read input or print anything. Complete the function**KDistanceNodes()** which takes the root of the tree, target, and K as input parameters and returns a list of nodes at k distance from target in a sorted order.

**Expected Time Complexity:**O(N\*logN)  
**Expected Auxiliary Space:**O(Height of tree)

**Constraints:**  
1 ≤ N ≤ 105  
1 ≤ data of node ≤ 109  
1 ≤ target ≤ 109  
1 ≤ k ≤ 20

**Code: -**

//{ Driver Code Starts

#include<bits/stdc++.h>

using namespace std;

// Tree Node

struct Node

{

int data;

Node\* left;

Node\* right;

};

// Utility function to create a new Tree Node

Node\* newNode(int val)

{

Node\* temp = new Node;

temp->data = val;

temp->left = NULL;

temp->right = NULL;

return temp;

}

// Function to Build Tree

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;

}

// } Driver Code Ends

/\* A binary Tree node

struct Node

{

int data;

struct Node \*left, \*right;

};

\*/

class Solution

{

private:

void dfsfindTg(Node \*nod, int k, vector<int>& ans) {

if (!nod) return;

if (k == 0) {

ans.push\_back(nod->data);

return;

} else {

dfsfindTg(nod->left, k-1, ans);

dfsfindTg(nod->right, k-1, ans);

}

}

int findTg(Node \*nod, int t, int k,

vector<int>& ans, unordered\_map<Node \*, pair<bool, int>>& mp) {

if (!nod) return -1;

if (nod->data == t) {

dfsfindTg(nod, k, ans);

return 1;

}

int lfnd = findTg(nod->left, t, k, ans, mp);

if (lfnd >= 1) {

mp[nod] = { 0, lfnd };

return 1+lfnd;

}

int rfnd = findTg(nod->right, t, k, ans, mp);

if (rfnd >= 1) {

mp[nod] = { 1, rfnd };

return 1+rfnd;

}

return -1;

}

public:

vector<int> KDistanceNodes(Node\* root, int target , int k) {

vector<int> ans;

unordered\_map<Node\*, pair<bool, int>> mp;

findTg(root, target, k, ans, mp);

for(auto &p : mp){

Node \*node = p.first;

bool dir = p.second.first;

int disfound = p.second.second;

// cout << "(" << node->data << "," << dir << "," << disfound << ")";

int searchdis = k - disfound;

if(searchdis == 0){

ans.push\_back(node->data);

}

else if(searchdis > 0){

// found on left side, so search in right side

if(dir == 0)

dfsfindTg(node->right, searchdis-1, ans);

// found on right side, so search in left side

else

dfsfindTg(node->left, searchdis-1, ans);

}

}

sort(ans.begin(), ans.end());

return ans;

}

};

//{ Driver Code Starts.

int main()

{

int t;

cin>>t;

getchar();

Solution x = Solution();

while(t--)

{

string s;

getline(cin,s);

Node\* head = buildTree(s);

int target, k;

cin>> target >> k;

getchar();

vector <int> res = x.KDistanceNodes(head, target, k);

for( int i=0; i<res.size(); i++ )

cout<< res[i] << " ";

cout<<endl;

}

return 0;

}

// } Driver Code Ends

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

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