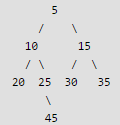
**Binary Tree Operations**

# Find distance from root to given node in a binary tree

Distance means number of edges between two nodes.

Given root of a binary tree and a key x in it, find distance of the given key from root.



// CPP

#include <iostream>

using namespace std;

struct tNode {

int data;

struct tNode \* left;

struct tNode \* right;

};

struct tNode \* newNode(int arg\_val);

int find\_distance\_from\_root(struct tNode \* arg\_root, int arg\_key);

struct tNode \* newNode(int arg\_val) {

struct tNode \* node = new struct tNode;

node -> data = arg\_val;

node -> left = NULL;

node -> right = NULL;

return node;

}

int find\_distance\_from\_root(struct tNode \* arg\_root, int arg\_key) {

int distance = -1;

if(NULL == arg\_root)

return distance;

if(arg\_key == (arg\_root -> data)) {

return (distance + 1);

}

else if((distance = find\_distance\_from\_root((arg\_root -> left), arg\_key)) >= 0) {

return (distance + 1);

}

else if((distance=find\_distance\_from\_root((arg\_root -> right), arg\_key)) >= 0) {

return (distance + 1);

}

else {

// do nothing

}

return distance;

}

int main() {

struct tNode \* root = NULL;

int distance = 0;

root = newNode(5);

root -> left = newNode(10);

root -> right = newNode(15);

(root -> left) -> left = newNode(20);

(root -> left) -> right = newNode(25);

(root -> right) -> left = newNode(30);

(root -> right) -> right = newNode(35);

((root -> left) -> right) -> right = newNode(45);

distance = find\_distance\_from\_root(root, 45);

cout << "Distance from root is : " << distance << endl;

return 0;

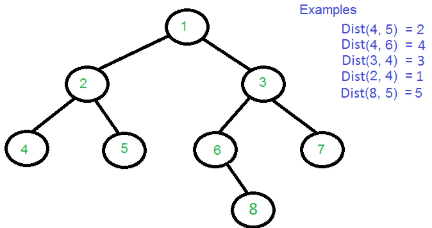
}

Output:

Distance from root is : 3

# Distance between two nodes of a Binary Tree

Distance between two nodes is the minimum number of edges to be traversed to reach one node from other.



Find the distance between two keys in a binary tree, no parent pointers are given.

## Formula in terms of LCA and distance from root

**Dist(n1, n2) = Dist(root, n1) + Dist(root, n2) - 2\*Dist(root, lca)**

'n1' and 'n2' are the two given keys

'root' is root of given Binary Tree.

'lca' is lowest common ancestor of n1 and n2

Dist(n1, n2) is the distance between n1 and n2

#include <iostream>

#include <vector>

using namespace std;

// A Bianry Tree node

struct Node {

int key;

struct Node \*left, \*right;

};

Node \* newNode(int k);

Node \*findLCA(Node \*root, int n1, int n2);

int findDistance(Node \*root, int n1, int n2);

int find\_distance\_from\_root(Node \* arg\_root, int arg\_key);

// Driver program to test above functions

int main() {

// Let us create the Binary Tree shown in above diagram.

Node \* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

root->right->left = newNode(6);

root->right->right = newNode(7);

root->right->left->right = newNode(8);

cout << "Distance(4, 5) = " << findDistance(root, 4, 5) << endl;

cout << "Distance(4, 6) = " << findDistance(root, 4, 6) << endl;

cout << "Distance(3, 4) = " << findDistance(root, 3, 4) << endl;

cout << "Distance(2, 4) = " << findDistance(root, 2, 4) << endl;

cout << "Distance(8, 5) = " << findDistance(root, 8, 5) << endl;

return 0;

}

// Utility function creates a new binary tree node with given key

Node \* newNode(int k) {

Node \*temp = new Node;

temp->key = k;

temp->left = temp->right = NULL;

return temp;

}

int findDistance(Node \*root, int n1, int n2) {

Node \* lca = NULL;

int dis\_n1 = 0;

int dis\_n2 = 0;

int dis\_lca = 0;

lca = findLCA(root, n1, n2);

dis\_n1 = find\_distance\_from\_root(root, n1);

dis\_n2 = find\_distance\_from\_root(root, n2);

if(NULL != lca)

dis\_lca = find\_distance\_from\_root(root, lca -> key);

else {

cout << "LCA is null";

return -1;

}

return ((dis\_n1 + dis\_n2) - 2\*dis\_lca);

}

Node \*findLCA(Node \*root, int n1, int n2) {

Node\* left\_lca;

Node\* right\_lca;

if (root == NULL)

return root;

if (root->key == n1 || root->key == n2)

return root;

left\_lca = findLCA(root->left, n1, n2);

right\_lca = findLCA(root->right, n1, n2);

if (left\_lca && right\_lca)

return root;

if (left\_lca != NULL)

return findLCA(root->left, n1, n2);

return findLCA(root->right, n1, n2);

}

int find\_distance\_from\_root(Node \* arg\_root, int arg\_key) {

int distance = -1;

if(NULL == arg\_root)

return distance;

if(arg\_key == (arg\_root -> key)) {

return (distance + 1);

}

else if((distance = find\_distance\_from\_root((arg\_root -> left), arg\_key)) >= 0) {

return (distance + 1);

}

else if((distance=find\_distance\_from\_root((arg\_root -> right), arg\_key)) >= 0) {

return (distance + 1);

}

else {

// do nothing

}

return distance;

}

Output:

Distance(4, 5) = 2

Distance(4, 6) = 4

Distance(3, 4) = 3

Distance(2, 4) = 1

Distance(8, 5) = 5

## Distance from LCA (Better Solution)

We first find LCA of two nodes. Then we find distance from LCA to two nodes.

#include <iostream>

#include <vector>

using namespace std;

// A Bianry Tree node

struct Node {

int key;

struct Node \*left, \*right;

};

Node \* newNode(int k);

Node \*findLCA(Node \*root, int n1, int n2);

int findDistance(Node \*root, int n1, int n2);

int findLevel(Node \*root, int k, int level);

// Driver program to test above functions

int main() {

// Let us create the Binary Tree shown in above diagram.

Node \* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

root->right->left = newNode(6);

root->right->right = newNode(7);

root->right->left->right = newNode(8);

cout << "Distance(4, 5) = " << findDistance(root, 4, 5) << endl;

cout << "Distance(4, 6) = " << findDistance(root, 4, 6) << endl;

cout << "Distance(3, 4) = " << findDistance(root, 3, 4) << endl;

cout << "Distance(2, 4) = " << findDistance(root, 2, 4) << endl;

cout << "Distance(8, 5) = " << findDistance(root, 8, 5) << endl;

return 0;

}

// Utility function creates a new binary tree node with given key

Node \* newNode(int k) {

Node \*temp = new Node;

temp->key = k;

temp->left = temp->right = NULL;

return temp;

}

int findDistance(Node \*root, int n1, int n2) {

Node \* lca = NULL;

int dis\_n1 = 0;

int dis\_n2 = 0;

lca = findLCA(root, n1, n2);

dis\_n1 = findLevel(lca, n1, 0);

dis\_n2 = findLevel(lca, n2, 0);

return (dis\_n1 + dis\_n2);

}

Node \*findLCA(Node \*root, int n1, int n2) {

Node\* left\_lca;

Node\* right\_lca;

if (root == NULL)

return root;

if (root->key == n1 || root->key == n2)

return root;

left\_lca = findLCA(root->left, n1, n2);

right\_lca = findLCA(root->right, n1, n2);

if (left\_lca && right\_lca)

return root;

if (left\_lca != NULL)

return findLCA(root->left, n1, n2);

return findLCA(root->right, n1, n2);

}

// Returns level of key k if it is present in

// tree, otherwise returns -1

int findLevel(Node \*root, int k, int level) {

if(root == NULL) return -1;

if(root->key == k) return level;

int left\_level = findLevel(root->left, k, level+1);

if (left\_level == -1)

return findLevel(root->right, k, level+1);

return left\_level;

}

Output:

Distance(4, 5) = 2

Distance(4, 6) = 4

Distance(3, 4) = 3

Distance(2, 4) = 1

Distance(8, 5) = 5

# END