Practical – 10

AIM: Doubly Linked List

Implement below operations of doubly linked list.

(a) Insert a node at front

(b) Delete a node at last

(c) Delete all nodes of linked list

Note: Display content of linked list after each operation.

* Program

#include<bits/stdc++.h>

using namespace std;

struct node{

int data;

struct node\* prev, \*next; //Self refrencial Structure => Struct which point it self.

};

struct node\* L = NULL, \*R = NULL;

struct node\* create(int);

void insertNodeAtEnd(int);

void insertNodeAtFront(int);

void deleteNodeFromEnd();

void deleteNodeFromFront();

void insertBefore(int, int);

void insertAfter(int, int);

void find(int);

void display();

int main(){

int choice;

do{

cout << "1. Insert node at end." << endl;

cout << "2. Insert node at front." << endl;

cout << "3. Delete end" << endl;

cout << "4. Delete front" << endl;

cout << "5. Find the element" << endl;

cout << "6. Insert node before a specific element." << endl;

cout << "7. Insert node after a specific element." << endl;

cout << "8. Display." << endl;

cout << "9. Exit." << endl;

//cout << "3. Insert node at end." << endl;

cout << "Select the operation : ";

cin >> choice;

switch (choice)

{

case 1:

int a;

cout << "Enter a number you want to add : ";

cin >> a;

insertNodeAtEnd(a);

display();

break;

case 2:

int b;

cout << "Enter a number you want to add : ";

cin >> b;

insertNodeAtFront(b);

display();

break;

case 3:

deleteNodeFromEnd();

display();

break;

case 4:

deleteNodeFromFront();

display();

break;

case 5:

int c;

cout << "Enter a element you want to find : ";

cin >> c;

find(c);

break;

case 6:

int d, e;

cout << "Enter a element you want to add : ";

cin >> d;

cout << "Enter a element before you want to add " << d << " : ";

cin >> e;

insertBefore(d, e);

display();

break;

case 7:

int f, g;

cout << "Enter a element you want to add : ";

cin >> f;

cout << "Enter a element after you want to add " << f << " : ";

cin >> g;

insertAfter(f, g);

display();

break;

case 8:

display();

break;

case 9:

break;

default:

cout << "Please enter a valid number" << endl << endl;

break;

}

}while(choice!=9);

return 0;

}

struct node\* create(int x){

struct node\* temp;

temp = (struct node\*)malloc(sizeof(struct node));

temp->data = x;

temp->next = NULL;

temp->prev = NULL;

return temp;

}

void insertNodeAtEnd(int x){

struct node \*temp = create(x);

if(R == NULL){

L = temp;

R = temp;

}

else{

R->next = temp;

temp->prev = R;

R = temp;

}

}

void insertNodeAtFront(int x){

struct node \*temp = create(x);

if(R == NULL){

L = temp;

R = temp;

}

else{

temp->next = L;

L->prev = temp;

L = temp;

}

}

void deleteNodeFromFront(){

struct node\* temp = L;

if(temp==NULL){

cout << "Linklist is empty" << endl;

}

else if(temp->next == NULL){

free(temp);

L = NULL;

R = NULL;

}

else{

L = L->next;

L->prev = NULL;

free(temp);

}

}

void deleteNodeFromEnd(){

struct node\* temp = R;

if(temp==NULL){

cout << "Linklist is empty" << endl;

}

else if(temp->prev == NULL){

free(temp);

L = NULL;

R = NULL;

}

else{

R = R->prev;

R->next = NULL;

free(temp);

}

}

void find(int x){

struct node\* temp1 = L;

struct node\* temp2 = R;

int flag = 1;

if(L == NULL){

flag = 0;

}

while(temp1->data != x && temp2->data != x){

if((temp1==temp2) || (temp1->next == temp2 && temp2->prev == temp1)){

flag = 0;

break;

}

temp1 = temp1->next;

temp2 = temp2->prev;

}

if(flag == 1)

cout << endl << "Element founded" << endl << endl;

else

cout << endl << "Element not founded" << endl << endl;

}

struct node\* findLocation(int x){

struct node\* trav = L;

if(L == NULL)

return NULL;

while(trav->data != x){

trav = trav->next;

if(trav==NULL){

break;

}

}

return trav;

}

void insertBefore(int x, int y){

struct node\* trav = findLocation(y);

if(trav != NULL){

struct node\* temp = create(x);

temp->next = trav;

if(trav->prev != NULL){

trav->prev->next = temp;

temp->prev = trav->prev;

}

else{

L = temp;

}

trav->prev = temp;

}

else

cout << endl << "Element not founded" << endl;

}

void insertAfter(int x, int y){

struct node\* trav = findLocation(y);

if(trav != NULL){

struct node\* temp = create(x);

temp->prev = trav;

if(trav->next != NULL){

temp->next = trav->next;

trav->next->prev = temp;

}

else{

R = temp;

}

trav->next = temp;

}

else

cout << endl << "Element not founded" << endl;

}

void display(){

struct node\* trav = L;

cout << endl << "Elements : ";

while(trav != NULL){

cout << trav->data << " ";

trav = trav->next;

}

cout << endl << endl;

}

* Output

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Student Signature Faculty Signature Marks

Practical – 17

AIM: Creation of Binary Tree using Doubly Linked List

Implement below operations on Binary Tree

(a) Insert a node at Left

(b) Delete a node at Right

Note: First node will be the root node.

Display the content of Binary Tree as per Inorder, Preorder, Postorder and Level Order Traversal.

* Program

#include <bits/stdc++.h>

using namespace std;

struct TreeNode{

int val;

TreeNode\*lptr;

TreeNode\*rptr;

TreeNode\*Root;

};

TreeNode->lptr=NULL;

TreeNode->rptr=NULL;

TreeNode->Root=NULL;

struct TreeNode\* create(int val)

{

TreeNode\*temp=(struct TreeNode\*)malloc(sizeof(TreeNode));

TreeNode->lptr=NULL;

TreeNode->rptr=NULL;

TreeNode->val=val;

return temp;

};

class BinaryTree

{

public:

TreeNode\* root;

BinaryTree()

{

root = NULL;

}

void insertLeft (TreeNode\* parent, int val)

{

TreeNode\* new\_node = new TreeNode(val);

if (parent->left == NULL)

{

parent->left = new\_node;

}

else

{

new\_node->left = parent->left;

parent->left = new\_node;

}

}

void deleteRight (TreeNode\* parent)

{

if (parent->right)

{

delete parent->right;

parent->right = NULL;

}

}

void inOrderTraversal (TreeNode node)

{

if (node)

{

inOrderTraversal (node->left);

cout << node->val << " ";

inOrderTraversal (node->right);

}

}

void preorderTraversal (TreeNode\* node)

{

if (node)

{

cout << node->val << " ";

preOrderTraversal (node->left);

preOrderTraversal (node->right);

}

}

void postOrderTraversal (TreeNode\* node)

{

if

(node)

{

postOrderTraversal (node->left);

postOrderTraversal (node->right);

cout << node->val << " ";

}

}

void levelOrderTraversal (TreeNode node)

{

if (node == nullptr)

return;

queue<TreeNode\*> q;

q.push(node);

while (!q.empty())

{

TreeNode\* curr = q.front();

q.pop();

cout << curr->val <<< " ";

if (curr->left)

q.push(curr->left);

if (curr->right)

q.push(curr->right);

}

}

DoublyListNode\* convertToDoubly\_LinkedList(TreeNode\* node, DoublyListNode\* prev = nullptr)

{

if (node)

{

prev = convertToDoublyLinkedList(node->left, prev);

DoublyListNode\* new\_doubly\_node = new DoublyListNode(node->val);

if (prev)

{

prev->next = new doubly\_node;

new\_doubly\_node->prev = prev;

}

prev = new doubly\_node;

prev = convertToDoublyLinkedList (node->right, prev);

return new\_doubly\_node;

}

return prev;

}

};

int main()

{

BinaryTree tree;

tree.root = new TreeNode(1);

tree.insertLeft (tree.root, 2);

tree.insertLeft (tree.root, 3);

tree.insertLeft (tree.root->left, 4);

cout << "Inorder Traversal: ";

tree.inOrderTraversal (tree.root);

cout << endl;

cout << "Preorder Traversal: ";

tree.preorderTraversal (tree.root);

cout << endl;

cout << "Postorder Traversal: ";

tree.postorderTraversal (tree.root);

cout << endl;

cout << "Level Order Traversal: ";

tree.levelOrderTraversal (tree.root);

cout << endl;

tree.deleteRight (tree.root->left);

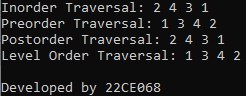
DoublyListNode\* doubly\_head, tree.convertToDoublyLinkedList(tree.root);

cout<<"\nDeveloped by 22CE068";

return 0;

}

* **Output**



* **Conclusion**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Student Signature Faculty Signature Marks

Practical – 23

AIM: Construct Binary Tree from Preorder and Inorder Traversal Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Example 1: Input: preorder = [3,9,20,15,7],

inorder = [9,3,15,20,7]

Output: [3,9,20,null,null,15,7]

* Program

class Solution {

public:

TreeNode\* buildTree(vector < int > & preorder, vector < int > & inorder) {

int preStart = 0, preEnd = preorder.size() - 1;

int inStart = 0, inEnd = inorder.size() - 1;

map < int, int > mp;

for (int i = inStart; i <= inEnd; i++) {

mp[inorder[i]] = i;

}

TreeNode\* root = constructTree(preorder, preStart, preEnd, inorder, inStart, inEnd, mp);

return root;

}

TreeNode\* constructTree(vector < int > & preorder, int preStart, int preEnd, vector < int > & inorder, int inStart, int inEnd, map < int, int > & mp) {

if (preStart > preEnd || inStart > inEnd) return NULL;

TreeNode\* root = new TreeNode(preorder[preStart]);

int elem = mp[root -> val];

int nElem = elem - inStart;

root -> left = constructTree(preorder, preStart + 1, preStart + nElem, inorder,

inStart, elem - 1, mp);

root -> right = constructTree(preorder, preStart + nElem + 1, preEnd, inorder,

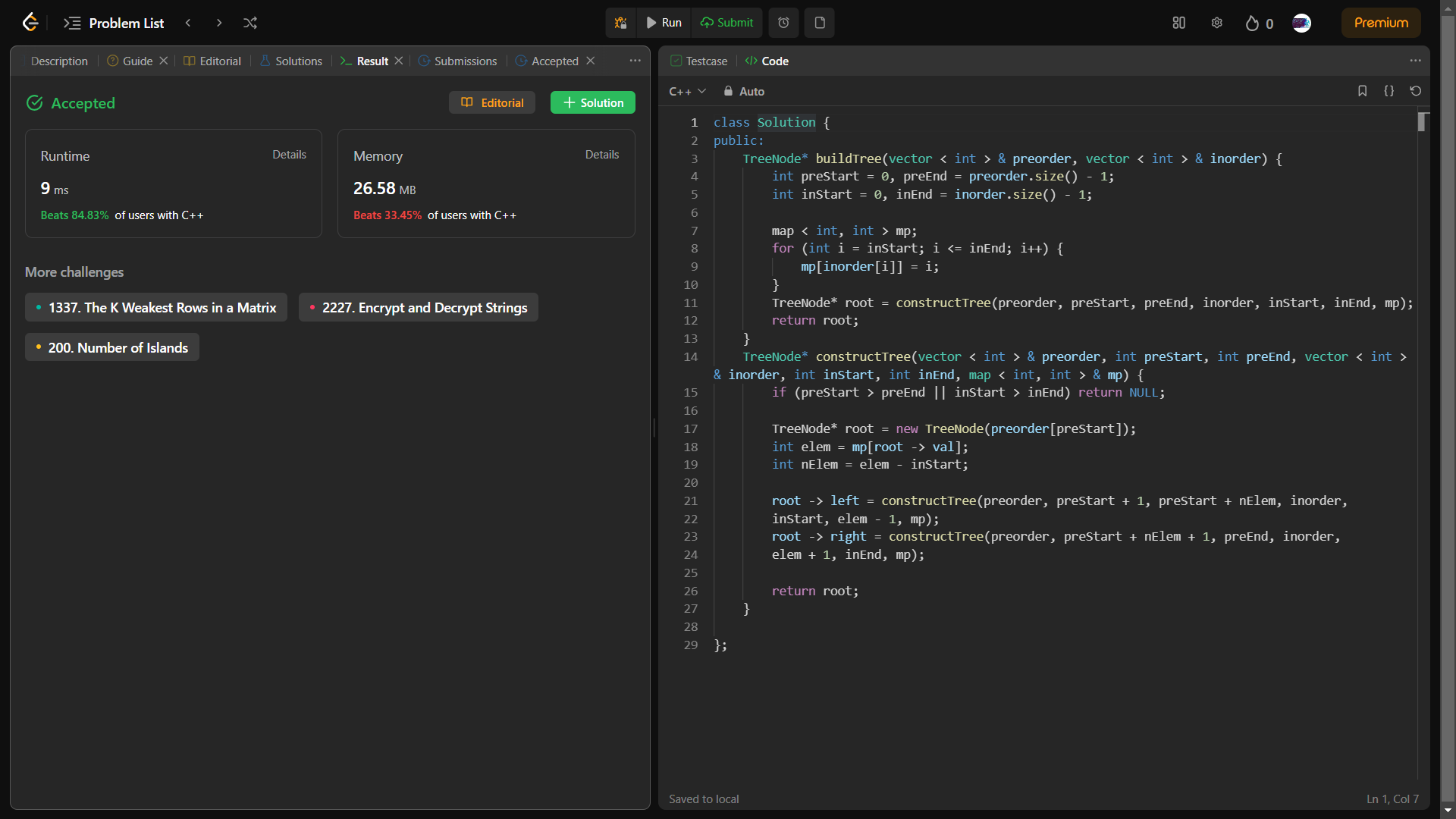
elem + 1, inEnd, mp);

return root;

}

};

* Output



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Student Signature Faculty Signature Marks

Practical – 24

AIM: Construct Binary Tree from Inorder and Postorder Traversal Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.

* Program

TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

unordered\_map<int, int> inToIndex;

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

inToIndex[inorder[i]] = i;

return build(inorder, 0, inorder.size() - 1, postorder, 0,

postorder.size() - 1, inToIndex);

}

TreeNode\* build(const vector<int>& inorder, int inStart, int inEnd,

const vector<int>& postorder, int postStart, int postEnd,

const unordered\_map<int, int>& inToIndex) {

if (inStart > inEnd)

return nullptr;

const int rootVal = postorder[postEnd];

const int rootInIndex = inToIndex.at(rootVal);

const int leftSize = rootInIndex - inStart;

TreeNode\* root = new TreeNode(rootVal);

root->left = build(inorder, inStart, rootInIndex - 1, postorder, postStart,

postStart + leftSize - 1, inToIndex);

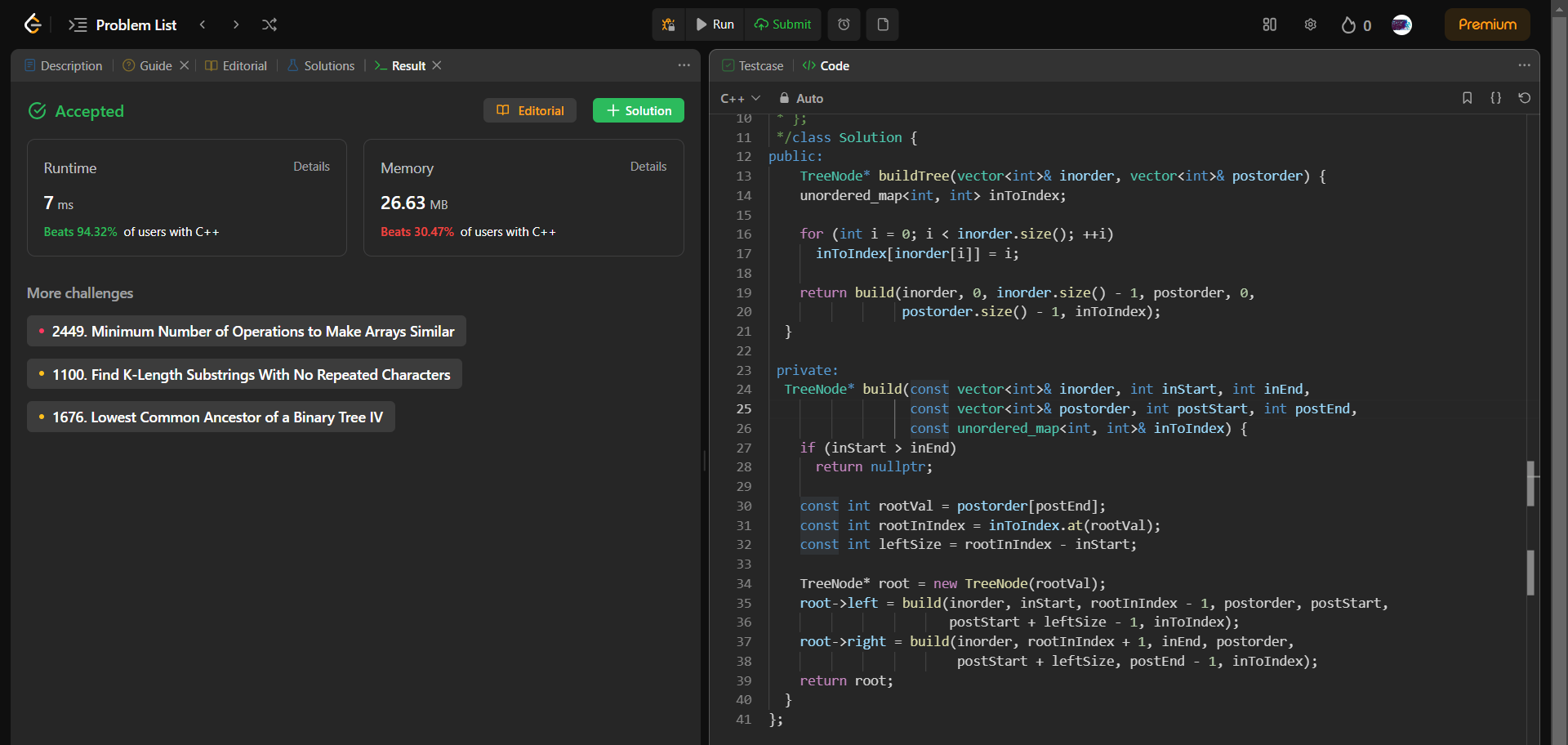
root->right = build(inorder, rootInIndex + 1, inEnd, postorder,

postStart + leftSize, postEnd - 1, inToIndex);

return root;

}

* Output

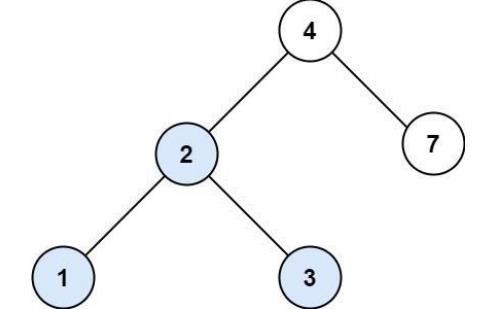


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Student Signature Faculty Signature Marks

Practical – 25

AIM: Search in a Binary Tree You are given the root of a binary search tree(BST) and an integer value. Find the node in the BST that node’s value equals value and return the subtree rooted with that node. If such a node does not exist, return null. Example 1: 0,4 1,2,4 9



* Program

TreeNode\* searchBST(TreeNode\* root, int val) {

while(root!=NULL && root->val != val)

{

if(root->val > val)

root = root->left;

else

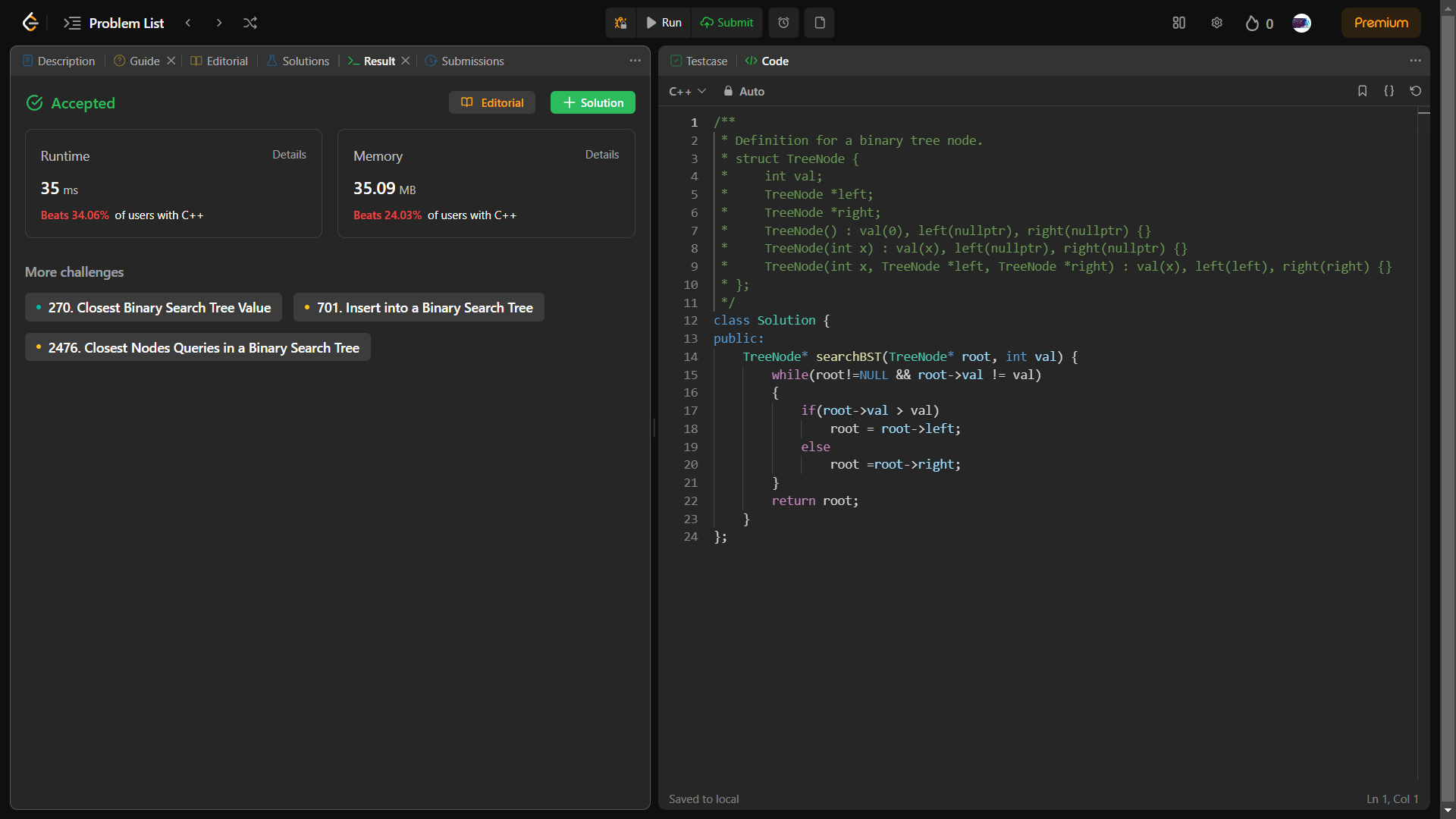
root =root->right;

}

return root;

}

* Output

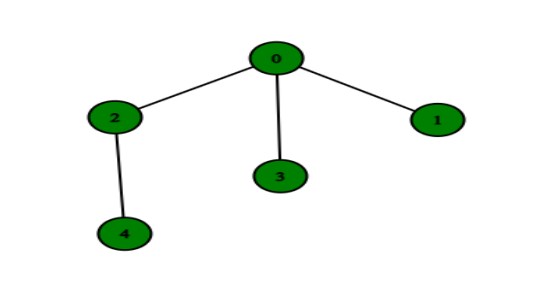


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

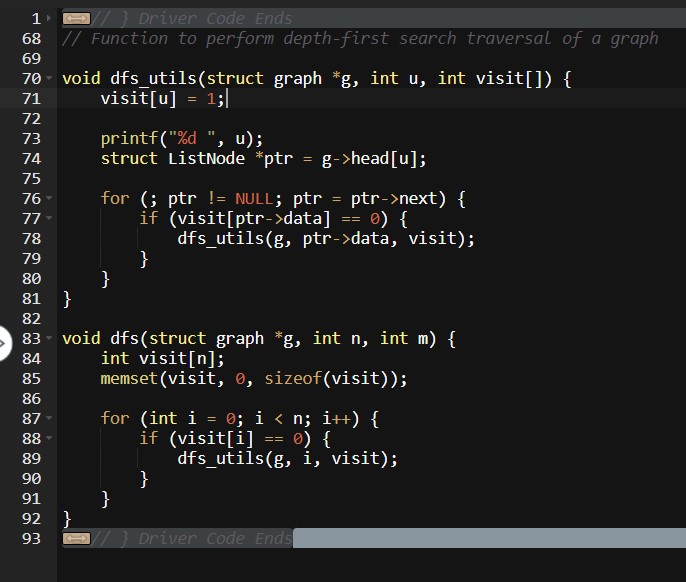
Student Signature Faculty Signature Marks

Practical – 26

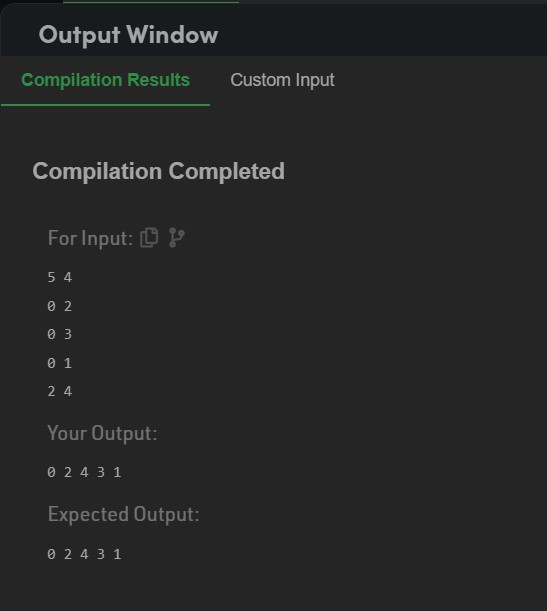
AIM: DFS of Graph You are given a connected undirected graph. Perform a Depth First Traversal of the graph. Note: Use a recursive approach to find the DFS traversal of the graph starting from the 0th vertex from left to right according to the graph.



* Program



* Output

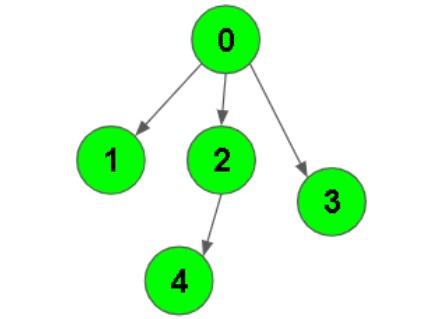


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

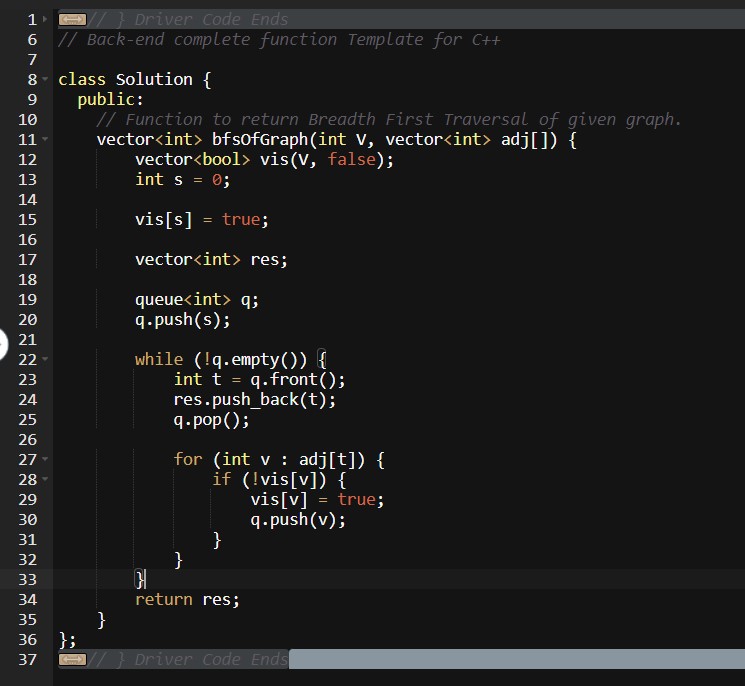
Student Signature Faculty Signature Marks

Practical – 27

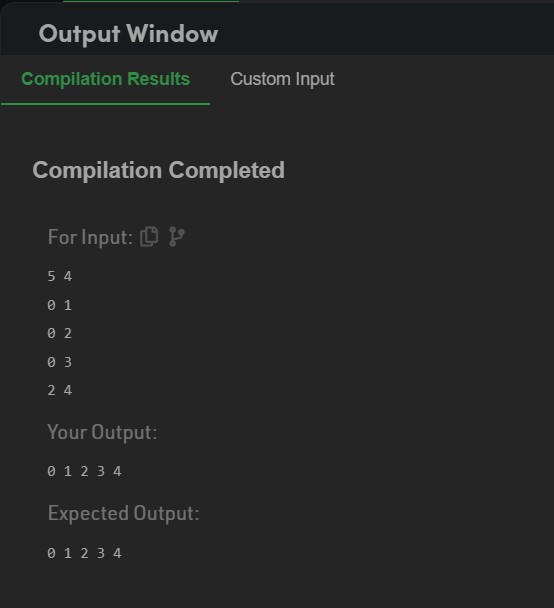
AIM: BFS of graph Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0. Note: One can move from node u to node v only if there's an edge from u to v and find the BFS traversal of the graph starting from the 0th vertex, from left to right according to the graph. Also, you should only take nodes directly or indirectly connected from Node 0 in consideration.



* Program



* Output



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Student Signature Faculty Signature Marks

Practical – 28

AIM: In an array of 20 elements, arrange 15 different values, which are generated randomly between 1,00,000 to 9,99,999. Use hash function to generate key using linear probing, quadratic probing and double hashing to avoid collision.

H(k) = 2k + 3 and m = 20.

Write a program to input and display the final values of array.

* Program

#include<bits/stdc++.h>

using namespace std;

vector<long int> hashArr(20, 0);

int noOfCollision = 0;

void insertData(long int, int);

int hashIndex(long int, int);

int main(){

int lb = 100000, ub = 999999;

for(int i = 0; i < 20; i++){

hashArr[i] = 0;

}

for(int i = 0; i < 15; i++){

long int randomNum = (rand() % (ub-lb+1)) + lb;

cout << i+1 << " : " << randomNum << endl;

insertData(randomNum, 0);

}

cout << endl << "After hashing " << endl;

for(int i = 0; i < 20; i++){

cout << i << " " << hashArr[i] << endl;

}

cout << "Number of collision : " << noOfCollision << endl;

}

void insertData(long int val, int i){

int index = 0;

do{

index = (hashIndex(val, i++)) % 20;

}while(hashArr[index] != 0);

hashArr[index] = val;

}

int hashIndex(long int val, int i){

if(i > 0){

noOfCollision++;

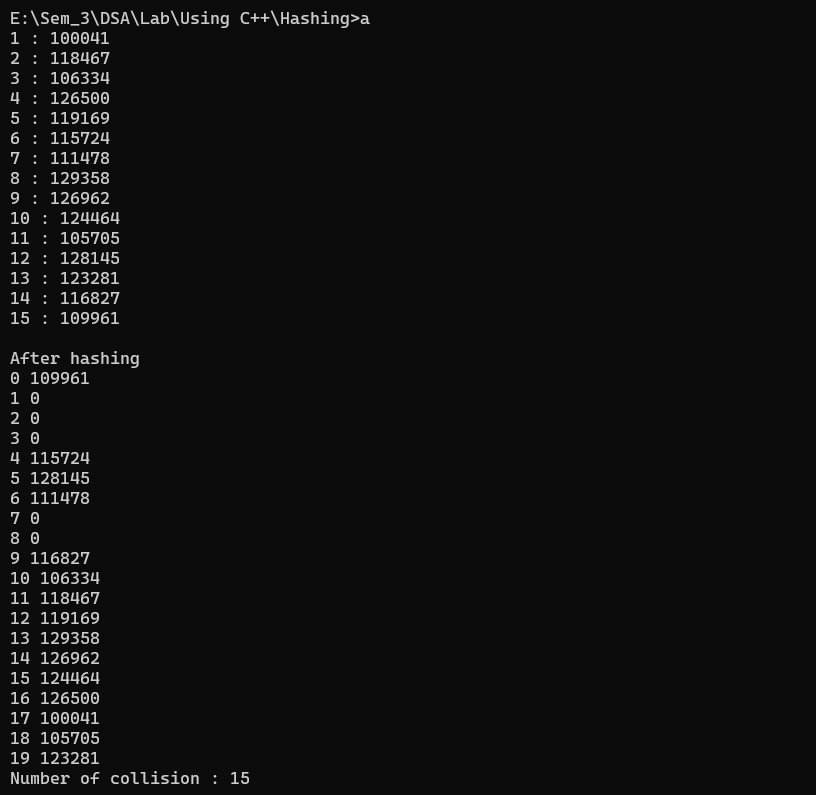
}

int h = (val % 18) + 2 + i;

return h;

}

* Output



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Student Signature Faculty Signature Marks