**ADS LAB ASSIGNMENT**

**Experiment-11: Trees**

**Q 1.** Create a binary tree using an array/linked List.

**Ans 1. CODE: -**

#include <iostream>

using namespace std;

struct nod {

nod \*l, \*r;

int d;

}\*r = NULL, \*p = NULL, \*np = NULL, \*q;

void create() {

int v,c = 0;

while (c < 6) {

if (r == NULL) {

r = new nod;

cout<<"enter value of root node\n";

cin>>r->d;

r->r = NULL;

r->l = NULL;

} else {

p = r;

cout<<"enter value of node\n";

cin>>v;

while(true) {

if (v< p->d) {

if (p->l == NULL) {

p->l = new nod;

p = p->l;

p->d = v;

p->l = NULL;

p->r = NULL;

cout<<"value entered in left\n";

break;

} else if (p->l != NULL) {

p = p->l;

}

} else if (v >p->d) {

if (p->r == NULL) {

p->r = new nod;

p = p->r;

p->d = v;

p->l = NULL;

p->r = NULL;

cout<<"value entered in right\n";

break;

} else if (p->r != NULL) {

p = p->r;

}

}

}

}

c++;

}

}

void inorder(nod \*p) {

if (p != NULL) {

inorder(p->l);

cout<<p->d<<endl;

inorder(p->r);

}

}

void preorder(nod \*p) {

if (p != NULL) {

cout<<p->d<<endl;

preorder(p->l);

preorder(p->r);

}

}

void postorder(nod \*p) {

if (p != NULL) {

postorder(p->l);

postorder(p->r);

cout<<p->d<<endl;

}

}

int main() {

create();

cout<<" traversal in inorder\n";

inorder(r);

cout<<" traversal in preorder\n";

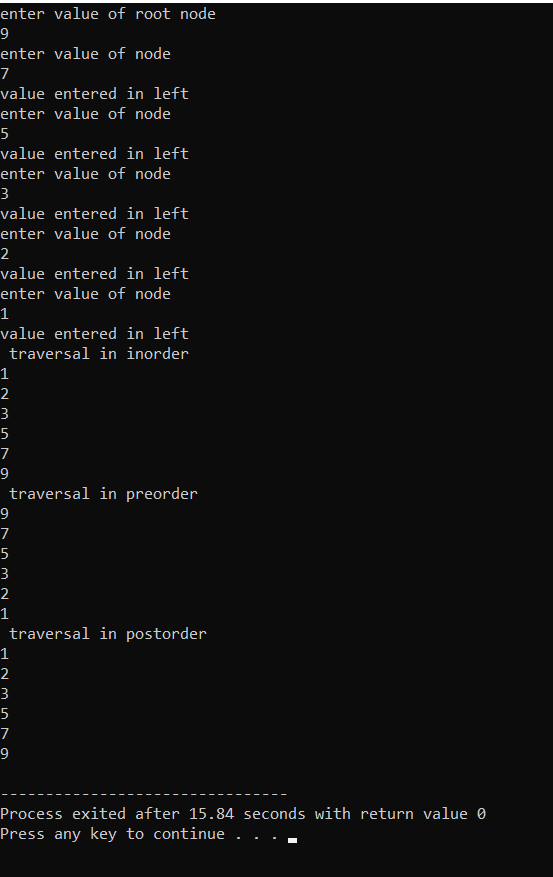
preorder(r);

cout<<" traversal in postorder\n";

postorder(r);

}

**OUTPUT: -**

****

**Q 2.** Construct a Binary Tree and perform Inorder, Preorder and Postorder Traversal.

**Ans 2. CODE: -**

#include <iostream>

using namespace std;

struct Node

{

int data;

struct Node\* left, \*right;

Node(int data)

{

this->data = data;

left = right = NULL;

}

};

void printPostorder(struct Node\* node)

{

if (node == NULL)

return;

printPostorder(node->left);

printPostorder(node->right);

cout << node->data << " ";

}

void printInorder(struct Node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

cout << node->data << " ";

printInorder(node->right);

}

void printPreorder(struct Node\* node)

{

if (node == NULL)

return;

cout << node->data << " ";

printPreorder(node->left);

printPreorder(node->right);

}

int main()

{

struct Node \*root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

cout << "\nPreorder traversal of binary tree is \n";

printPreorder(root);

cout << "\nInorder traversal of binary tree is \n";

printInorder(root);

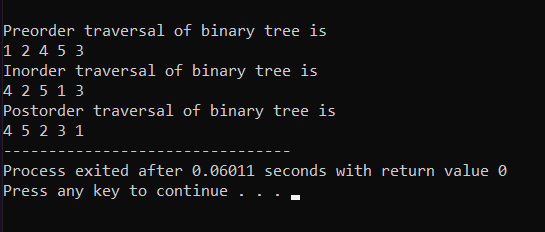
cout << "\nPostorder traversal of binary tree is \n";

printPostorder(root);

return 0;

}

**OUTPUT: -**

****

**Q 3.** Implement Heap Sort.

**Ans 3. CODE: -**

#include <iostream>

using namespace std;

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2\*i + 1; // left = 2\*i + 1

int r = 2\*i + 2; // right = 2\*i + 2

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i)

{

swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

}

void heapSort(int arr[], int n)

{

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i=n-1; i>0; i--)

{

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

for (int i=0; i<n; ++i)

cout << arr[i] << " ";

cout << "\n";

}

int main()

{

int arr[] = {12, 11, 13, 5, 6, 7};

int n = sizeof(arr)/sizeof(arr[0]);

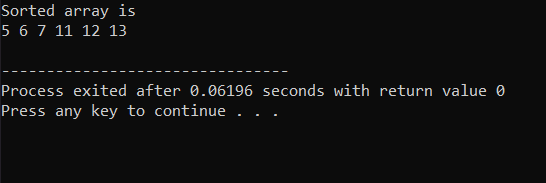
heapSort(arr, n);

cout << "Sorted array is \n";

printArray(arr, n);

}

**OUTPUT: -**

****

**Experiment-12: Graphs**

**Q 1.** Accept the vertices and edges for a graph and stores it as an adjacency matrix. Implement functions to print indegree, outdegree and to display the adjacency matrix.

**Ans 1. CODE: -**

#include<iostream>

using namespace std;

void print\_adjacency\_matrix(int \*\*matrix, int matrix\_size){

printf("\nAdjacency Matrix:\n ");

for(int i{}; i<matrix\_size; i++) printf("%d ", i+1);

cout<<endl;

for(int y{}; y<matrix\_size; y++){

printf("%d ", y+1);

for(int x{}; x<matrix\_size; x++){

printf("%d ", matrix[y][x]);

}

cout<<endl;

}

cout<<endl;

}

void add\_connection(int \*\*matrix, int size, int source, int destination){

matrix[source-1][destination-1]= 1;

}

void print\_indegree(int \*\*matrix, int size){

printf("\nPrinting Indegree of all nodes:\n");

for(int src{}; src<size; src++){

int count{};

for(int dest{}; dest<size; dest++){

if( matrix[dest][src] ) count++;

}

printf("Node %d: Indegree: %d\n", src+1, count);

}

}

void print\_outdegree(int \*\*matrix, int size){

printf("\nPrinting Outdegree of all nodes:\n");

for(int dest{}; dest<size; dest++){

int count{};

for(int src{}; src<size; src++){

if( matrix[dest][src] ) count++;

}

printf("Node %d: Outdegree: %d\n", dest+1, count);

}

}

int main(){

// Getting Vertices

int matrix\_size{};

printf("Enter number of nodes:");

cin>>matrix\_size;

int \*\*adjacency\_matrix= new int\*[matrix\_size];

for(int y{}; y<matrix\_size; y++){

adjacency\_matrix[y]= new int[matrix\_size];

for(int x{}; x<matrix\_size; x++){

adjacency\_matrix[y][x]= 0;

}

}

int connections{};

printf("Enter number of connections:");

cin>>connections;

// Getting Edges for the graph

int source{}, destination;

for(int i{}; i<connections; i++){

printf("Enter source and destination:");

cin>>source;

cin>>destination;

add\_connection(adjacency\_matrix, matrix\_size, source, destination);

}

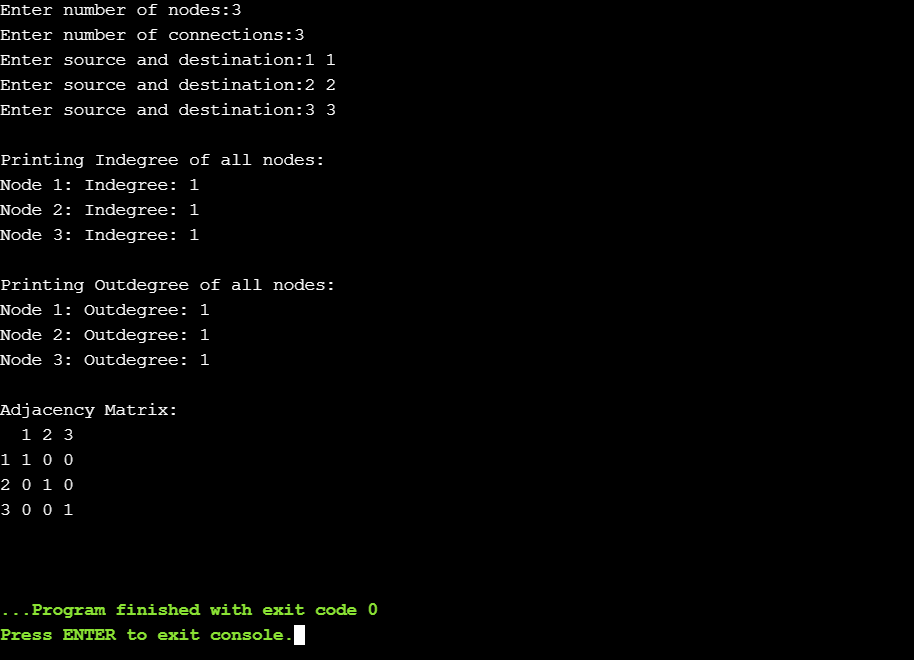
print\_indegree(adjacency\_matrix, matrix\_size);

print\_outdegree(adjacency\_matrix, matrix\_size);

print\_adjacency\_matrix(adjacency\_matrix, matrix\_size);

return 0;

}

**OUTPUT: -**

**Q 2.** Accept the vertices and edges for a graph and stores it as an adjacency list. Implement functions to print outdegree of any vertex i.

**Ans 2. CODE: -**

#include<iostream>

using namespace std;

void print\_adjacency\_matrix(int \*\*matrix, int matrix\_size){

printf("\nAdjacency Matrix:\n ");

for(int i{}; i<matrix\_size; i++) printf("%d ", i+1);

cout<<endl;

for(int y{}; y<matrix\_size; y++){

printf("%d ", y+1);

for(int x{}; x<matrix\_size; x++){

printf("%d ", matrix[y][x]);

}

cout<<endl;

}

cout<<endl;

}

void add\_connection(int \*\*matrix, int size, int source, int destination){

matrix[source-1][destination-1]= 1;

}

void print\_outdegree(int \*\*matrix, int size, int src){

int count{};

for(int dest{}; dest<size; dest++){

if( matrix[src-1][dest] )

count++;

}

printf("\nOutdegree for Node-%d is %d\n", src, count);

}

int main(){

// Getting Vertices

int matrix\_size{};

printf("Enter number of nodes:");

cin>>matrix\_size;

int \*\*adjacency\_matrix= new int\*[matrix\_size];

for(int y{}; y<matrix\_size; y++){

adjacency\_matrix[y]= new int[matrix\_size];

for(int x{}; x<matrix\_size; x++){

adjacency\_matrix[y][x]= 0;

}

}

int connections{};

printf("Enter number of connections:");

cin>>connections;

// Getting Edges for the graph

int source{}, destination;

for(int i{}; i<connections; i++){

printf("Enter source and destination:");

cin>>source;

cin>>destination;

add\_connection(adjacency\_matrix, matrix\_size, source, destination);

}

print\_adjacency\_matrix(adjacency\_matrix, matrix\_size);

int src{};

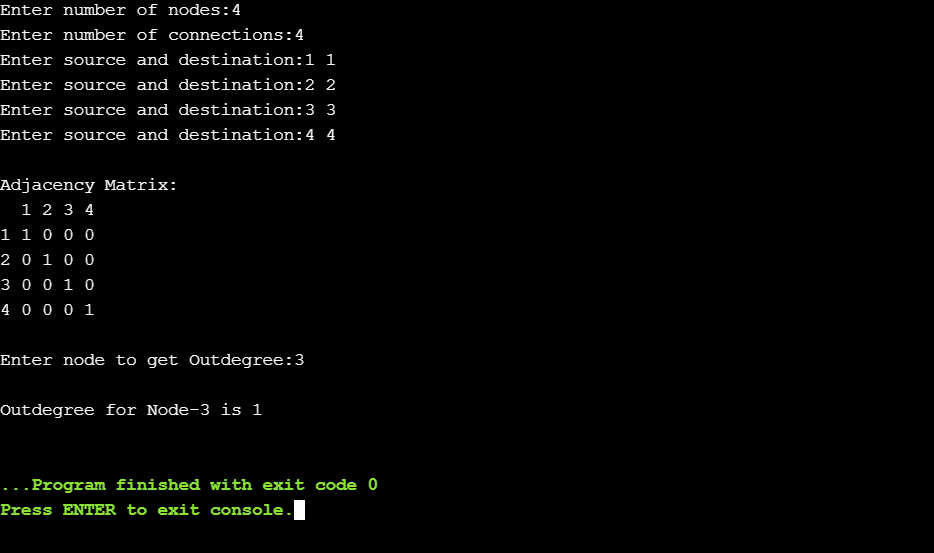
printf("Enter node to get Outdegree:");

cin>>src;

print\_outdegree(adjacency\_matrix, matrix\_size, src);

return 0;

}

**OUTPUT: -**

**Q 3.** Accept the graph as an adjacency matrix and checks if the graph is undirected. The matrix for undirected graph is symmetric.

**Ans 3. CODE: -**

#include<iostream>

using namespace std;

void print\_adjacency\_matrix(int \*\*matrix, int matrix\_size){

printf("\nAdjacency Matrix:\n ");

for(int i{}; i<matrix\_size; i++) printf("%d ", i+1);

cout<<endl;

for(int y{}; y<matrix\_size; y++){

printf("%d ", y+1);

for(int x{}; x<matrix\_size; x++){

printf("%d ", matrix[y][x]);

}

cout<<endl;

}

cout<<endl;

}

bool is\_undirected\_graph(int \*\*matrix, int size){

for(int y{}; y<size; y++){

for(int x{}; x<size; x++){

if( matrix[y][x] != matrix[x][y] )

return false;

}

}

return true;

}

int main(){

// Getting Vertices

int matrix\_size{};

printf("Enter number of nodes:");

cin>>matrix\_size;

// Getting Adjacency Matrix

printf("Enter %d values for Adjacency Matrix:\n", matrix\_size\*matrix\_size);

int \*\*adjacency\_matrix= new int\*[matrix\_size];

for(int y{}; y<matrix\_size; y++){

printf("Enter %d values for row %d: ", matrix\_size, y+1);

adjacency\_matrix[y]= new int[matrix\_size];

for(int x{}; x<matrix\_size; x++){

cin>>adjacency\_matrix[y][x];

}

}

print\_adjacency\_matrix(adjacency\_matrix, matrix\_size);

bool is\_undirected= is\_undirected\_graph(adjacency\_matrix, matrix\_size);

if(is\_undirected) printf("Given Graph is a Undirected Graph");

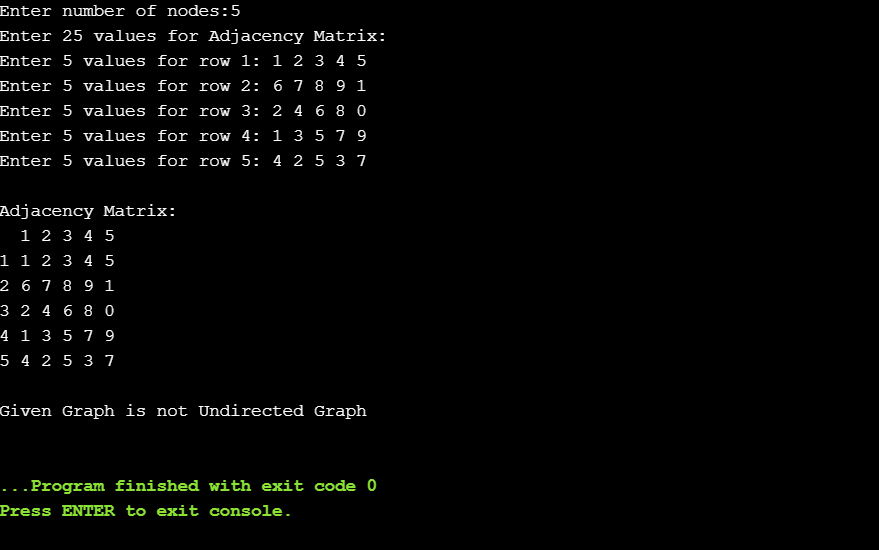
else printf("Given Graph is not Undirected Graph");

cout<<endl;

return 0;

}

**OUTPUT: -**

****

**NAME -> ROHAN NYATI**

**SAP ID -> 500075940**

**ROLL NO. -> R177219148**

**BRANCH -> AI&ML B5**