**Question#1**

**Code:**

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

class Graph {

int V;

string id[100];

bool mat[100][100];

bool vis[100];

bool hasCycleUtil(int cur, int parent) {

vis[cur] = true;

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

if (mat[cur][i]) {

if (!vis[i]) {

if (hasCycleUtil(i, cur)) return true;

}

else if (i != parent) {

return true;

}

}

}

return false;

}

void dfsHelper(int v) {

vis[v] = true;

cout << id[v] << " ";

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

if (mat[v][i] && !vis[i])

dfsHelper(i);

}

}

public:

Graph(int n) : V(n) {

fill\_n(vis, 100, false);

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

id[i] = "";

fill(mat[i], mat[i] + V, false);

}

}

void setNames(string names[]) {

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

id[i] = names[i];

}

}

void addEdge(int x, int y) {

mat[x][y] = mat[y][x] = true;

}

bool hasCycle() {

fill(vis, vis + V, false);

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

if (!vis[i]) {

if (hasCycleUtil(i, -1)) return true;

}

}

return false;

}

void displayDegrees() {

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

int degree = count(mat[i], mat[i] + V, true);

cout << id[i][0] << " -> in: " << degree << ", out: " << degree << endl;

}

}

void dfs(string start) {

fill(vis, vis + V, false);

int startIndex = -1;

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

if (id[i][0] == start[0]) {

startIndex = i;

break;

}

}

if (startIndex != -1) {

cout << "DFS from " << start << ": ";

dfsHelper(startIndex);

cout << endl;

}

else {

cout << "Start vertex not found." << endl;

}

}

void displayMatrix() {

cout << "Adjacency Matrix:" << endl << " ";

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

cout << id[i][0] << ' ';

cout << endl;

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

cout << id[i][0] << ' ';

for (int j = 0; j < V; j++) {

cout << mat[i][j] << ' ';

}

cout << endl;

}

}

};

int main() {

Graph g(5);

string names[5] = { "a", "b", "c", "d", "e" };

g.setNames(names);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(2, 3);

g.addEdge(3, 4);

g.displayMatrix();

cout << endl;

if (g.hasCycle() == true) {

cout << "Cycle : Yes" << endl;

}

else {

cout << "Cycle : No" << endl;

}

cout << "Degrees:" << endl;

g.displayDegrees();

cout << endl;

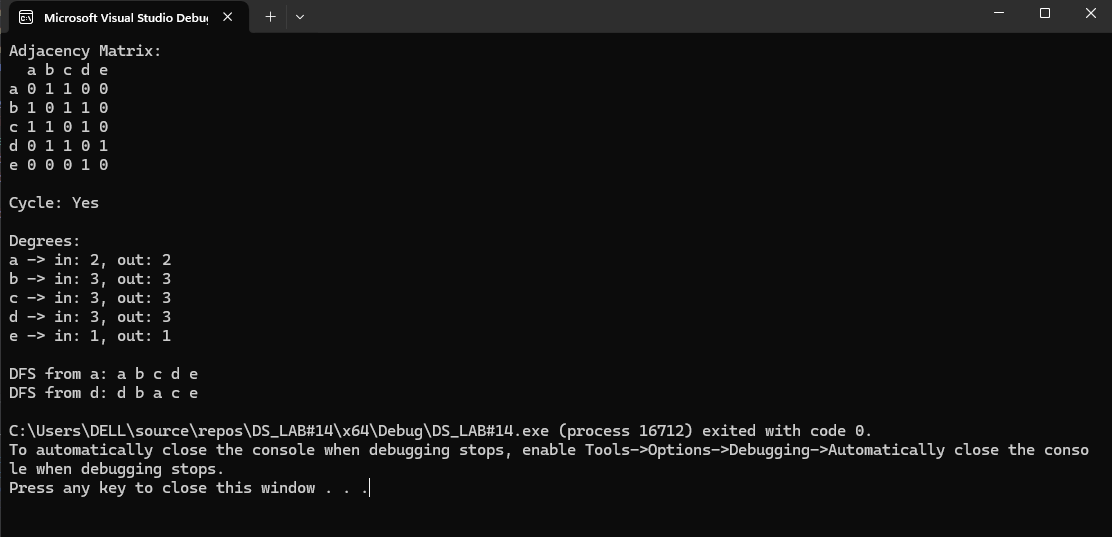
g.dfs("a");

g.dfs("d");

return 0;

}

**Output:**

****

**Question#2**

**Code:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node() : data(0), next(nullptr) {}

};

class list {

private:

Node\* head;

Node\* current;

public:

list() : head(nullptr), current(nullptr) {}

void insert(int val) {

Node\* newNode = new Node;

newNode->data = val;

if (head == nullptr) {

head = newNode;

}

else {

current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

}

void print() {

current = head;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

};

struct AdjNode {

int data;

AdjNode\* next;

list obj;

AdjNode() : data(0), next(nullptr) {}

};

class adjList {

private:

AdjNode\* head;

AdjNode\* current;

int vertexCount;

public:

adjList() : head(nullptr), current(nullptr) , vertexCount(0) {}

void insertVertex() {

AdjNode\* newVertex = new AdjNode;

newVertex->data = vertexCount++;

if (head == nullptr) {

head = newVertex;

}

else {

current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newVertex;

}

}

void insertAtVertex(int vertex, int val) {

current = head;

while (current != nullptr && current->data != vertex) {

current = current->next;

}

if (current == nullptr) {

cout << "Vertex# " << vertex << " Not Found!" << endl;

return;

}

current->obj.insert(val);

}

void print() {

current = head;

while (current != nullptr) {

cout << current->data << " : ";

current->obj.print();

current = current->next;

}

}

};

int main() {

adjList obj;

obj.insertVertex();

obj.insertVertex();

obj.insertVertex();

obj.insertVertex();

obj.insertVertex();

obj.insertVertex();

obj.insertVertex();

obj.insertAtVertex(0, 1);

obj.insertAtVertex(0, 4);

obj.insertAtVertex(1, 5);

obj.insertAtVertex(1, 0);

obj.insertAtVertex(1, 3);

obj.insertAtVertex(1, 6);

obj.insertAtVertex(2, 6);

obj.insertAtVertex(2, 5);

obj.insertAtVertex(3, 4);

obj.insertAtVertex(3, 1);

obj.insertAtVertex(3, 5);

obj.insertAtVertex(4, 3);

obj.insertAtVertex(4, 6);

obj.insertAtVertex(4, 0);

obj.insertAtVertex(5, 1);

obj.insertAtVertex(5, 2);

obj.insertAtVertex(5, 3);

obj.insertAtVertex(6, 4);

obj.insertAtVertex(6, 2);

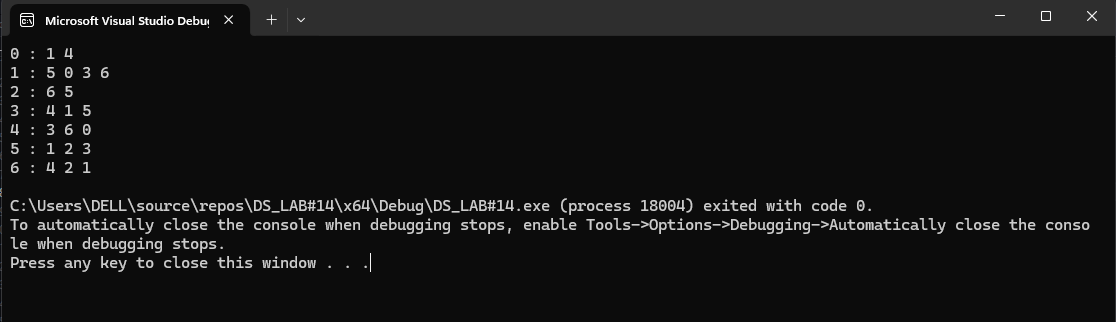
obj.insertAtVertex(6, 1);

obj.print();

return 0;

}

**Output:**

****

**Question#3**

**Code:**

#include<iostream>

using namespace std;

class Graph {

private:

int\*\* Data;

int Size;

public:

Graph(int size) : Size(size) {

Data = new int\* [Size];

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

Data[i] = new int[Size];

}

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

for (int j = 0; j < Size; j++) {

Data[i][j] = 0;

}

}

}

void insert\_edge(int index1, int index2, int weight) {

// Check for valid index ranges

if (index1 < 0 || index1 >= Size || index2 < 0 || index2 >= Size) {

cout << "Wrong Index" << endl;

return;

}

if (index1 != index2) {

Data[index1][index2] = weight;

Data[index2][index1] = weight;

}

}

int smallest\_weight() {

int min = 10000;

int index1 = -1, index2 = -1;

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

for (int j = 0; j < Size; j++) {

if (Data[i][j] != 0 && Data[i][j] < min) {

min = Data[i][j];

index1 = i;

index2 = j;

}

}

}

if (index1 != -1 && index2 != -1) {

cout << "Edges With Smallest Weight are: (" << index1 << ", " << index2 << ") with weight " << min << endl;

}

else {

cout << "No edges found." << endl;

}

return min;

}

};

int main() {

int size;

cout << "Enter Size of The Graph: "; cin >> size;

Graph obj1(size);

int edge1, edge2, weight;

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

cout << "Enter Edge#1: "; cin >> edge1;

cout << "Enter Edge#2: "; cin >> edge2;

cout << "Enter Weight of The Edge: "; cin >> weight;

obj1.insert\_edge(edge1, edge2, weight);

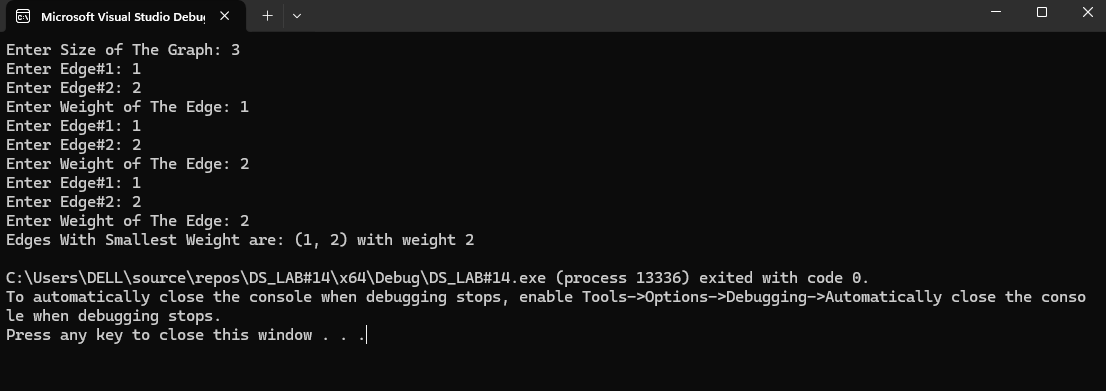
}

obj1.smallest\_weight();

return 0;

}

**Output:**

****