**M. Ali. Arslan**

**19F-0348**

**Lab-09**

**Task # 01**

#include<iostream>

using namespace std;

class Graph

{

private:

int Vertices;

int\* arr;

int\* input;

int edges;

void adjacencyMatrix(int u, int v, int cols)

{

\*(arr + u \* cols + v) = 1;

\*(arr + v \* cols + u) = 1;

};

void init()

{

//initialize by zero

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

{

for (int j = 0; j < Vertices; j++)

{

\*(arr + i \* Vertices + j) = 0;

}

}

}

public:

Graph(int\* e, int V, int edge)

{

edges = edge;

V++;

Vertices = V;

arr = new int[V \* V];

input = e;

init();

};

~Graph()

{

delete[] arr;

}

void driver()

{

int v1 = 0;

int v2 = 0;

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

{

v1 = \*(input + i \* 2 + 0);

v2 = \*(input + i \* 2 + 1);

//v1 = input[i][0];

//v2 = input[i][1];

adjacencyMatrix(v1, v2, Vertices);

};

}

void print2dArr()

{

for (int i = 1; i < Vertices; i++)

{

for (int j = 1; j < Vertices; j++)

{

cout << \*(arr + i \* Vertices + j) << " ";

}

cout << endl;

};

}

};

int main()

{

int N = 8;

int M = 7;

cout << "The Adjacency matrix of the graph is: " << endl;

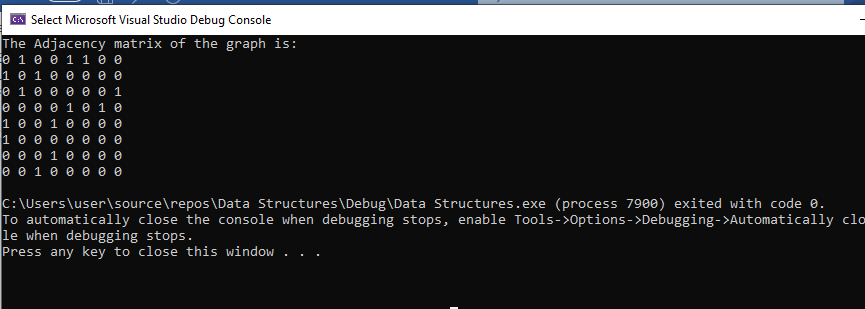
int input[7][2] = { {1, 2}, {2, 3}, {4, 5}, {1, 5}, {6, 1}, {7, 4}, {3, 8} };

Graph a((int\*)input, N, M);

a.driver();

a.print2dArr();

}



**Task # 02**

#include<iostream>

using namespace std;

class Graph

{

private:

int Vertices;

int\* arr;

int\* input;

int edges;

int colsEdges;

int start;

int\* edgesNum;

void adjacencyMatrix(int u, int v, int cols)

{

\*(arr + u \* cols + v) = 1;

\*(arr + v \* cols + u) = 1;

};

void init()

{

//initialize by zero

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

{

for (int j = 0; j < Vertices; j++)

{

\*(arr + i \* Vertices + j) = 0;

}

}

}

public:

Graph(int\* e, int V, int edge, int edgesCols)

{

start = 0;

colsEdges = edgesCols;

edges = edge;

if (start == 1)

V++;

Vertices = V;

arr = new int[V \* V];

input = e;

edgesNum = new int[V];

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

\*(edgesNum + i) = 0;

init();

};

~Graph()

{

delete[] arr;

delete[] edgesNum;

}

void matrixDriver()

{

int v1 = 0;

int v2 = 0;

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

{

v1 = \*(input + i \* colsEdges + 0);

v2 = \*(input + i \* colsEdges + 1);

//v1 = input[i][0];

//v2 = input[i][1];

adjacencyMatrix(v1, v2, Vertices);

};

}

void numOfEdgesCalculator()

{

int num = 0;

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

{

for (int j = 0; j < Vertices; j++)

{

if (\*(arr + i \* Vertices + j) == 1)

\*(edgesNum + i) += 1;

}

}

displayEdges();

}

void displayEdges()

{

cout << endl;

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

{

cout << "Vertex " << i << " Edges: ";

cout << \*(edgesNum + i);

cout << endl;

}

}

void printMatrix()

{

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

for (int i = start; i < Vertices; i++)

{

for (int j = start; j < Vertices; j++)

{

cout << \*(arr + i \* Vertices + j) << " ";

}

cout << endl;

};

}

};

int main()

{

int N = 10;

int M = 15;

int input[][2] =

{

{0, 1}, {0, 6}, {0, 8}, {1, 4}, {1, 6}, {1, 9}, {2, 4},

{2,6},{3,4},{3,5},{3,8},{4,5},{4,9},{7,8},{7,9}

};

int edgesCols = 2;

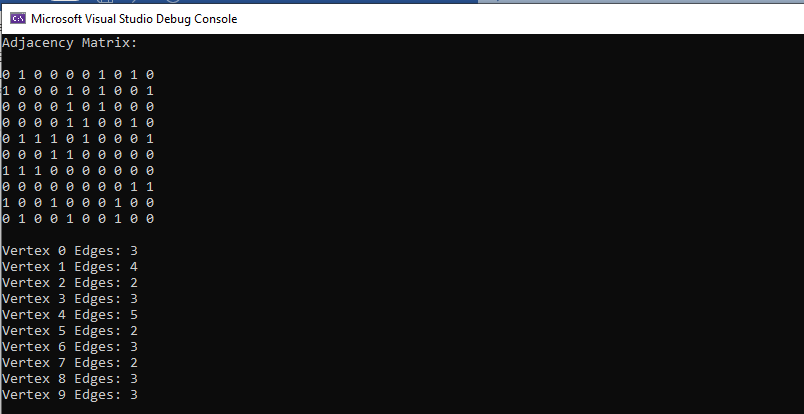
Graph a((int\*)input, N, M, edgesCols);

a.matrixDriver();

a.printMatrix();

a.numOfEdgesCalculator();

}



**Task # 03**

#include <iostream>

#include <iomanip>

using namespace std;

class Node

{

public:

Node\* link;

int data;

Node()

{

link = NULL;

data = NULL;

}

Node(int d)

:

data(d),

link(NULL)

{};

};

class LinkedList

{

Node\* head;

public:

LinkedList()

{

head = NULL;

}

void appendNode(int d) {

Node\* n = new Node();

n->data = d;

if (head == NULL) {

head = n;

}

else {

Node\* ptr = head;

while (ptr->link != NULL) {

ptr = ptr->link;

}

ptr->link = n;

}

}

void display()

{

Node\* temp = head;

while (temp != NULL)

{

cout << "( " << temp->data << " )-->";

temp = temp->link;

}

cout << endl << endl;

}

};

class Graph

{

private:

int\* arr;

int\* arr2;

int\* final;

int vertices;

int edges;

LinkedList\* l1;

public:

Graph(int e, int v)

{

edges = 3;

vertices = 4;

arr = new int[vertices \* vertices];

arr2 = new int[vertices \* vertices];

final = new int[vertices \* vertices];

l1 = new LinkedList[vertices];

init((int\*)arr, vertices, vertices);

fill();

};

~Graph()

{

delete[]arr;

delete[]arr2;

delete[] final;

delete[] l1;

}

void display(int\* arr, int rows, int cols)

{

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

{

for (int j = 0; j < cols; j++)

{

cout << \*(arr + i \* cols + j) << " ";

}

cout << endl;

}

}

void fill()

{

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

{

for (int j = 0; j < vertices; j++)

{

cout << "Index [" << i << "]" << "[" << j << "] : ";

cin >> \*(arr + i \* vertices + j);

cout << endl;

}

}

system("cls");

}

void init(int\* arr, int rows, int cols)

{

//initialize by zero

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

{

for (int j = 0; j < cols; j++)

{

\*(arr + i \* cols + j) = 0;

}

}

}

void transpose()

{

cout << "Orignal Matrix" << endl;

display((int\*)arr, vertices, vertices);

init((int\*)arr2, vertices, vertices);

//Temporary copying values

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

{

for (int j = 0; j < vertices; j++)

{

\*(arr2 + i \* vertices + j) = \*(arr + i \* vertices + j);

}

}

//Transpose

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

{

for (int j = 0; j < vertices; j++)

{

\*(arr + i \* vertices + j) = \*(arr2 + j \* vertices + i);

}

}

cout << "Transposed Matrix" << endl;

display((int\*)arr, vertices, vertices);

}

void mult()

{

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

{

for (int j = 0; j < vertices; j++)

{

\*(final + i \* vertices + j) = \*(arr + i \* vertices + j) \* \*(arr2 + i \* vertices + j);

}

}

cout << "\nMultiplied Matrix: \n";

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

{

for (int j = 0; j < vertices; j++)

{

cout << setw(3) << \*(final + i \* vertices + j) << " ";

}

cout << endl;

}

}

void adjacencyList()

{

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

{

for (int j = 0; j < vertices; j++)

{

if (\*(final + i \* vertices + j) != 0)

{

l1[i].appendNode(j);

};

}

}

}

void displayList()

{

cout << "Adjacency List: " << endl << endl;

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

{

cout << "Vertex: " << i << " ";

l1[i].display();

}

}

};

int main()

{

int e = 0;

int v = 0;

cout << "Enter Edges and Vertices: " << endl;

cin >> e >> v;

Graph a(e, v);

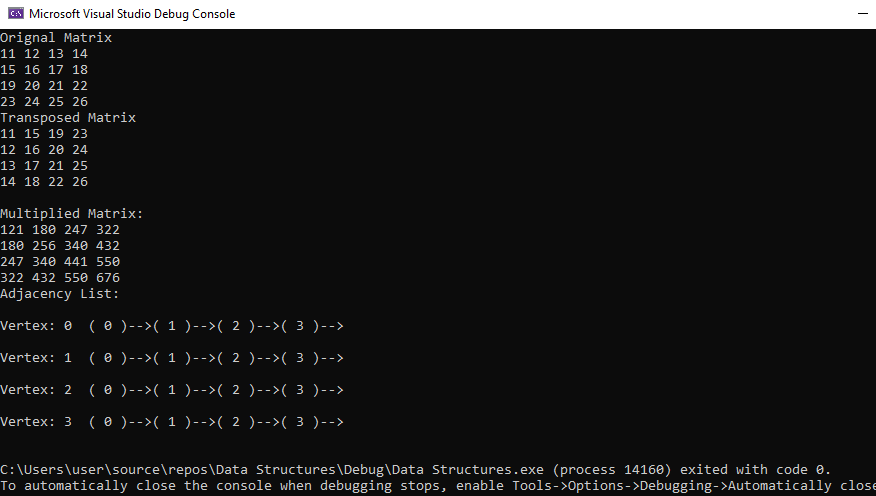
a.transpose();

a.mult();

a.adjacencyList();

a.displayList();

}



**Task # 04**

#include<iostream>

#include <vector>

#include <queue>

using namespace std;

int minEdgeBFS(vector <int> edges[], int u, int v, int n)

{

vector<bool> visited(n, 0);

vector<int> distance(n, 0);

queue <int> Q;

distance[u] = 0;

Q.push(u);

visited[u] = true;

while (!Q.empty())

{

int x = Q.front();

Q.pop();

for (int i = 0; i < edges[x].size(); i++)

{

if (visited[edges[x][i]])

continue;

distance[edges[x][i]] = distance[x] + 1;

Q.push(edges[x][i]);

visited[edges[x][i]] = 1;

}

}

return distance[v];

}

void addEdge(vector <int> edges[], int u, int v)

{

edges[u].push\_back(v);

edges[v].push\_back(u);

}

int main()

{

int n = 9;

cout << "The number of vertices which share the common edges between two vertices of the graph are: ";

vector <int> edges[9];

addEdge(edges, 0, 1);

addEdge(edges, 0, 7);

addEdge(edges, 1, 7);

addEdge(edges, 1, 2);

addEdge(edges, 2, 3);

addEdge(edges, 2, 5);

addEdge(edges, 2, 8);

addEdge(edges, 3, 4);

addEdge(edges, 3, 5);

addEdge(edges, 4, 5);

addEdge(edges, 5, 6);

addEdge(edges, 6, 7);

addEdge(edges, 7, 8);

int u = 0;

int v = 5;

cout << minEdgeBFS(edges, u, v, n);

return 0;

}

