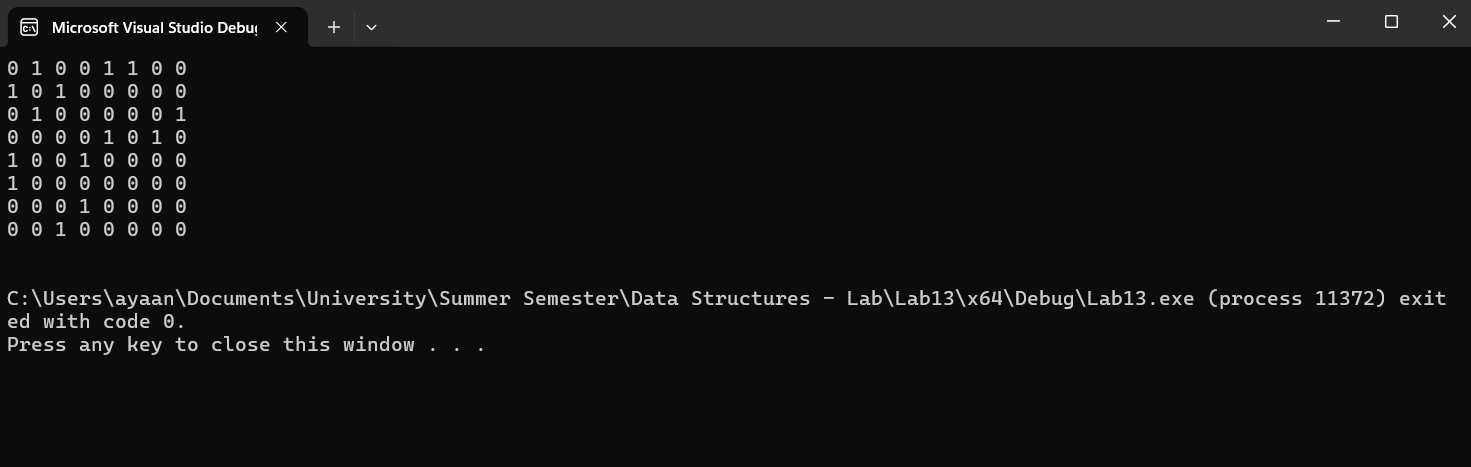
Task 01:



#pragma once

#include <iostream>

using namespace std;

class graph

{

private:

bool\*\* adjMatrix;

int numVertices;

int count;

int numEdges;

int M;

public:

graph()

{

numVertices = 0;

adjMatrix = NULL;

count = 0;

M = 7;

numEdges = 0;

}

graph(int x)

{

//x--;

numVertices = x;

M = 7;

numEdges = 0;

adjMatrix = new bool\* [numVertices];

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

{

adjMatrix[i] = new bool[numVertices];

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

{

adjMatrix[i][j] = 0;

}

}

count = 0;

}

graph(int x, int y)

{

//x--;

numVertices = x;

M = y;

numEdges = 0;

adjMatrix = new bool\* [numVertices];

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

{

adjMatrix[i] = new bool[numVertices];

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

{

adjMatrix[i][j] = 0;

}

}

count = 0;

}

bool isEdge(int x, int y)

{

if (x >= 0 && x < numVertices && y >= 0 && y < numVertices)

{

return adjMatrix[x][y];

}

else

{

cout << "Invalid vertices\n";

return 0; // Return false for invalid vertices

}

}

void addEdge(int x, int y)

{

x--;

y--;

if (x >= 0 && x < numVertices && y >= 0 && y < numVertices)

{

if (numEdges >= M)

{

cout << "Edges exceeding limit\n";

return;

}

adjMatrix[x][y] = 1;

adjMatrix[y][x] = 1;

numEdges++;

}

else

{

cout << "Invalid vertices\n";

}

}

void removeEdge(int x, int y)

{

x--;

y--;

if (x >= 0 && x < numVertices && y >= 0 && y < numVertices)

{

adjMatrix[x][y] = 0;

adjMatrix[y][x] = 0;

numEdges--;

}

else

{

cout << "Invalid vertices\n";

}

}

void printGraph()

{

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

{

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

{

cout << adjMatrix[i][j] << " ";

}

cout << "\n";

}

cout << "\n";

}

void display()

{

printGraph();

}

~graph()

{

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

{

delete[] adjMatrix[i];

}

delete[] adjMatrix;

}

};

int main()

{

graph g1(8, 7);

g1.addEdge(1, 2);

g1.addEdge(2, 3);

g1.addEdge(4, 5);

g1.addEdge(1, 5);

g1.addEdge(6, 1);

g1.addEdge(7, 4);

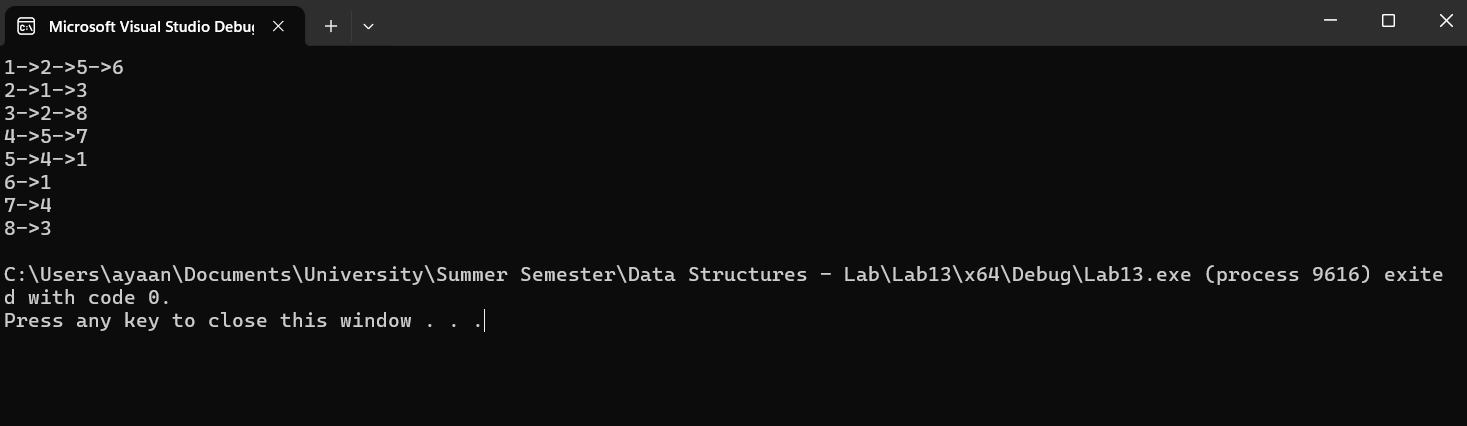
g1.addEdge(3, 8);

g1.display();

return 0;

}

Task 02:



#include<iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node()

{

data = -1;

next = NULL;

}

Node(int id)

{

data = id;

next = NULL;

}

};

class AdjList

{

public:

Node\* head;

AdjList()

{

head = NULL;

}

void Insert(Node\* ptr)

{

ptr->next = head;

head = ptr;

}

void Display()

{

Node\* temp = head;

while (temp != nullptr)

{

cout << temp->data;

temp = temp->next;

}

//cout << std::endl;

}

};

class Graph

{

public:

int maxVertices;

int noOfVertices;

AdjList\* list;

int count;

Graph()

{

maxVertices = 0;

noOfVertices = 0;

list = NULL;

count = 0;

}

Graph(int x)

{

maxVertices = x;

noOfVertices = 0;

list = new AdjList[maxVertices];

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

{

list[i].head = new Node(i + 1);

}

count = 0;

}

~Graph()

{

delete[] list;

}

Node\* createVertex(int data)

{

if (noOfVertices < maxVertices)

{

Node\* ptr = new Node(data);

noOfVertices++;

return ptr;

}

else

{

return NULL;

}

}

void insertEdge(int x, int y)

{

Node\* temp = list[x - 1].head;

while (temp->next)

{

temp = temp->next;

}

temp->next = new Node(y);

Node\* temp2 = list[y - 1].head;

while (temp2->next)

{

temp2 = temp2->next;

}

temp2->next = new Node(x);

}

void showGraphStructure()

{

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

{

Node\* temp = list[i].head;

while (temp->next)

{

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

temp = temp->next;

}

cout << temp->data << endl;

}

}

void Graphdisplay()

{

showGraphStructure();

}

};

int main()

{

Graph g1(8);

g1.insertEdge(1, 2);

g1.insertEdge(2, 3);

g1.insertEdge(4, 5);

g1.insertEdge(1, 5);

g1.insertEdge(6, 1);

g1.insertEdge(7, 4);

g1.insertEdge(3, 8);

g1.Graphdisplay();

return 0;

}

Task 03:



void dijkstra(int x)

{

int\* dist = new int[maxVertices];

bool\* visited = new bool[maxVertices];

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

{

dist[i] = INT\_MAX;

visited[i] = false;

}

dist[x - 1] = 0;

queue<int> q;

q.push(x);

while (!q.empty())

{

int u = q.front();

q.pop();

if (visited[u - 1])

continue;

visited[u - 1] = true;

Node\* temp = list[u - 1].head;

while (temp != nullptr)

{

int v = temp->data;

int weight = temp->weight;

if (!visited[v - 1] && dist[u - 1] != INT\_MAX && dist[u - 1] + weight < dist[v - 1]) {

dist[v - 1] = dist[u - 1] + weight;

q.push(v);

}

temp = temp->next;

}

}

cout << "Vertex \t Distance from Source" << endl;

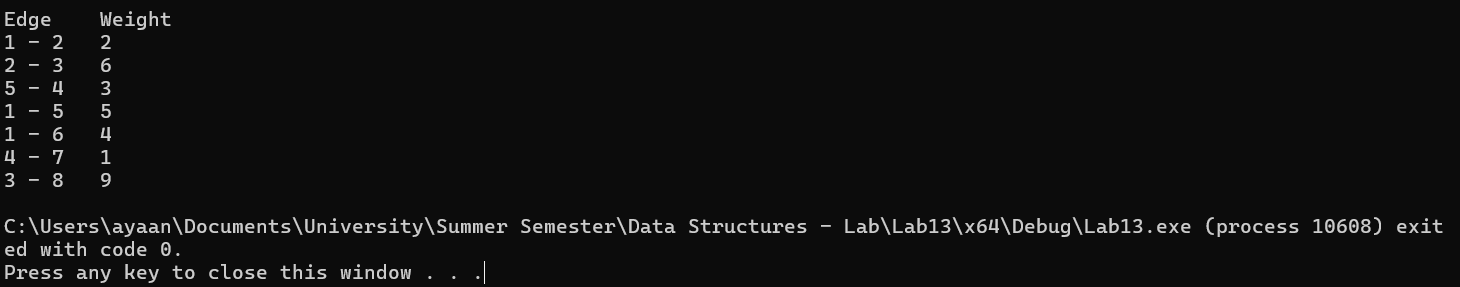
for (int i = 0; i < maxVertices; i++) {

cout << i + 1 << " \t\t " << dist[i] << endl;

}

}

Task 04:



void primMST()

{

int\* parent = new int[maxVertices];

int\* key = new int[maxVertices];

bool\* inMST = new bool[maxVertices];

queue<int> q;

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

{

key[i] = INT\_MAX;

inMST[i] = false;

}

key[0] = 0;

parent[0] = -1;

q.push(0);

while (!q.empty())

{

int u = q.front();

q.pop();

if (inMST[u])

continue;

inMST[u] = true;

Node\* temp = list[u].head;

while (temp != nullptr)

{

int v = temp->data - 1;

int weight = temp->weight;

if (!inMST[v] && weight < key[v])

{

key[v] = weight;

parent[v] = u;

q.push(v);

}

temp = temp->next;

}

}

cout << "Edge \tWeight\n";

for (int i = 1; i < maxVertices; i++) {

cout << parent[i] + 1 << " - " << i + 1 << " \t" << key[i] << " \n";

}

}