DAY-5

# 1.Shortest path prims in c programming:

#include<stdio.h>

int main()

{

int cost[10][10],i,j,n,source,target,visited[10]={0},min=999,dist[10],pre[10];

int start,m,d,path[10];

printf("Enter number of nodes\n ");

scanf("%d",&n);

printf("Enter weight of all the paths in adjacency matrix form\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("Enter the source\n");

scanf("%d",&source);

printf("Enter the target\n");

scanf("%d",&target);

start=source;

for(i=1;i<=n;i++)

{

dist[i]=999;

pre[i]=-1;

}

visited[source]=1;

dist[source]=0;

while(visited[target]==0)

{

min=999;

m=0;

for(i=1;i<=n;i++)

{

d=dist[start]+cost[start][i];

if(d<dist[i] && visited[i]==0)

{

dist[i]=d;

pre[i]=start;

}

if(min>dist[i] && visited[i]==0)

{

min=dist[i];

m=i;

}

}

start=m;

visited[m]=1;

}

start=target;

j=0;

while(start!=-1)

{

path[j++]=start;

start=pre[start];

}

for(i=j-1;i>=0;i--)

{

if(i!=j-1)

printf(" to ");

printf("%d",path[i]);

}

printf("\n shortest path is %d",dist[target]);

    return 0;

}

# 2.Minimum spanning tree*:*

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 5

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

int printMST(int parent[], int graph[V][V])

{

printf("Edge \tWeight\n");

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

printf("%d - %d \t%d \n", parent[i], i,

graph[i][parent[i]]);

}

void primMST(int graph[V][V])

{

int parent[V];

int key[V];

bool mstSet[V];

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

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

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

if (graph[u][v] && mstSet[v] == false

&& graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

// print the constructed MST

printMST(parent, graph);

}

// Driver's code

int main()

{

int graph[V][V] = { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

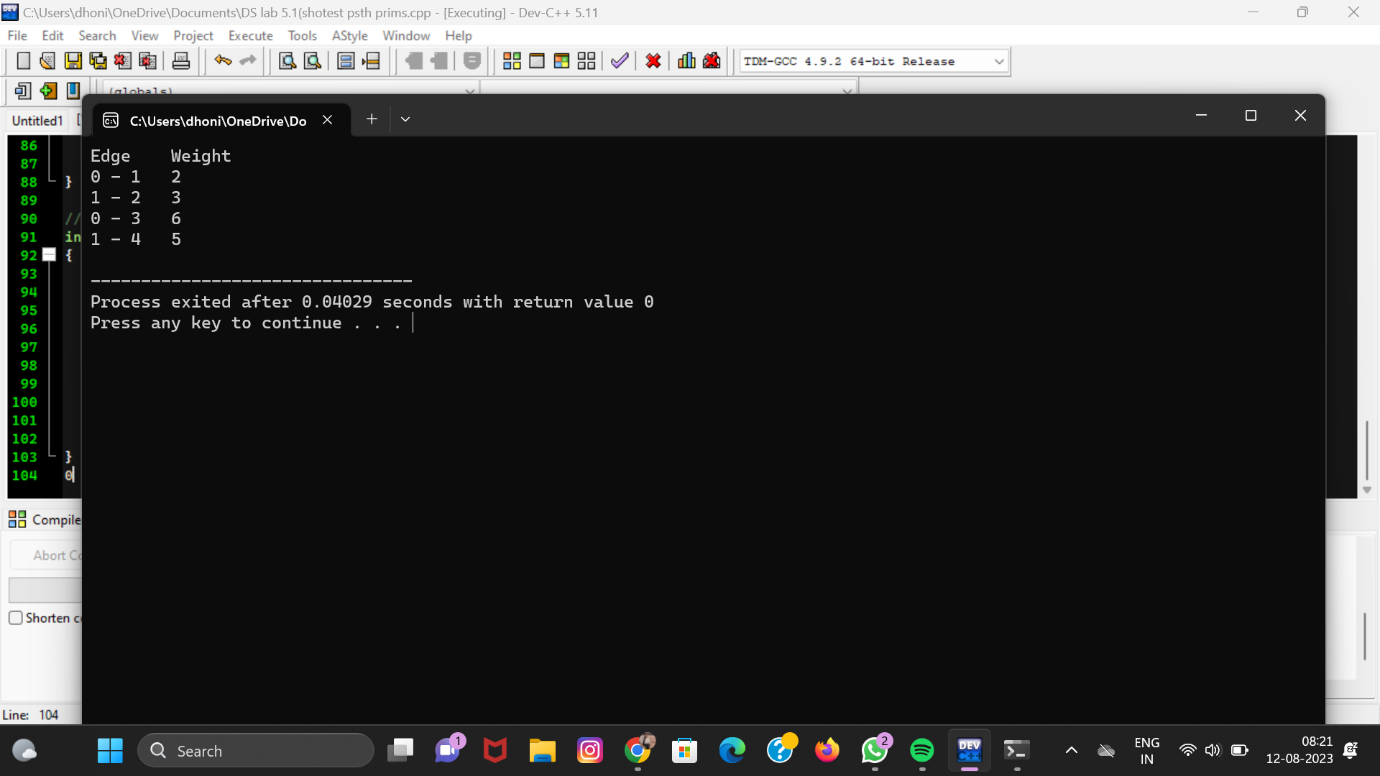
{ 0, 5, 7, 9, 0 } };

// Print the solution

primMST(graph);

return 0;

}



# 3.Graph transversal depth first search:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

bool visited[MAX\_VERTICES];

int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES];

int numVertices;

void initialize() {

for (int i = 0; i < MAX\_VERTICES; i++) {

visited[i] = false;

for (int j = 0; j < MAX\_VERTICES; j++) {

adjacencyMatrix[i][j] = 0;

}

}

}

void addEdge(int start, int end) {

adjacencyMatrix[start][end] = 1;

adjacencyMatrix[end][start] = 1;

}

void DFS(int vertex) {

visited[vertex] = true;

printf("%d ", vertex)

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

if (adjacencyMatrix[vertex][i] && !visited[i]) {

DFS(i);

}

}

}

int main() {

initialize();

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

int numEdges;

printf("Enter the number of edges: ");

scanf("%d", &numEdges)

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

int start, end;

printf("Enter edge %d (start end): ", i + 1);

scanf("%d %d", &start, &end);

addEdge(start, end);

}

int startVertex;

printf("Enter the starting vertex for DFS: ");

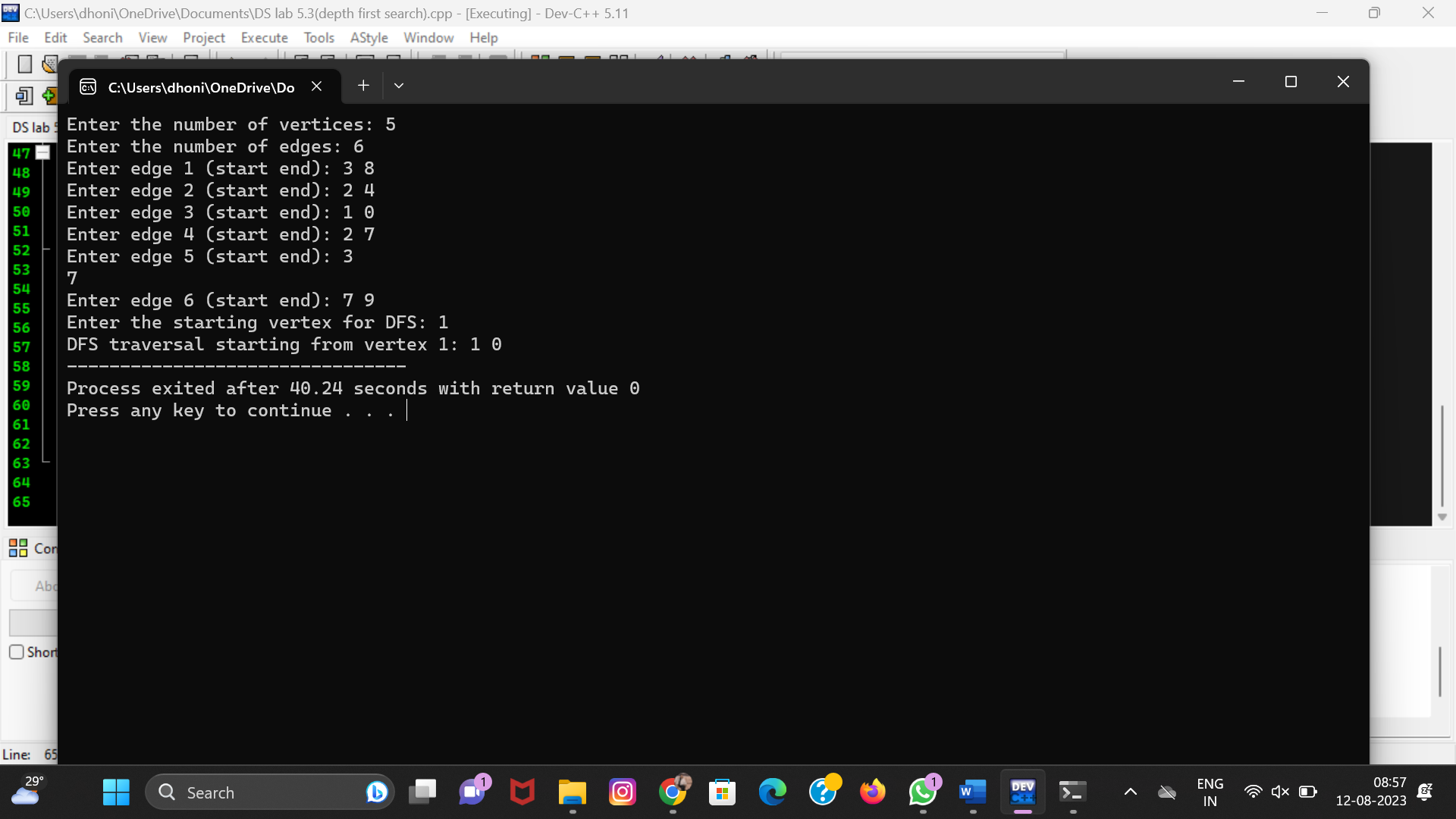
scanf("%d", &startVertex);

printf("DFS traversal starting from vertex %d: ", startVertex);

DFS(startVertex)

return 0;

}



# 4. Graph transversal [ BFS]:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

#define QUEUE\_SIZE 100

bool visited[MAX\_VERTICES];

int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES];

int numVertices;

int queue[QUEUE\_SIZE];

int front = -1, rear = -1;

void initialize() {

for (int i = 0; i < MAX\_VERTICES; i++) {

visited[i] = false;

for (int j = 0; j < MAX\_VERTICES; j++) {

adjacencyMatrix[i][j] = 0;

}

}

}

void addEdge(int start, int end) {

adjacencyMatrix[start][end] = 1;

adjacencyMatrix[end][start] = 1;

}

void enqueue(int vertex) {

if (rear == QUEUE\_SIZE - 1) {

printf("Queue is full.\n");

return;

}

if (front == -1)

front = 0;

rear++;

queue[rear] = vertex;

}

int dequeue() {

if (front == -1 || front > rear) {

printf("Queue is empty.\n");

return -1;

}

int vertex = queue[front];

front++;

return vertex;

}

void BFS(int startVertex) {

visited[startVertex] = true;

enqueue(startVertex);

while (front <= rear) {

int currentVertex = dequeue();

printf("%d ", currentVertex);

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

if (adjacencyMatrix[currentVertex][i] && !visited[i]) {

visited[i] = true;

enqueue(i);

}

}

}

}

int main() {

initialize();

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

int numEdges;

printf("Enter the number of edges: ");

scanf("%d", &numEdges);

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

int start, end;

printf("Enter edge %d (start end): ", i + 1);

scanf("%d %d", &start, &end);

addEdge(start, end);

int startVertex;

printf("Enter the starting vertex for BFS: ");

scanf("%d", &startVertex);

printf("BFS traversal starting from vertex %d: ", startVertex);

BFS(startVertex);

return 0;

}

