***TASK 1: (Dijkstra Algorithm)***

#include<iostream>

#define V 9

using namespace std;

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

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

if (sptSet[v] == false && dist[v] <= min)

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

return min\_index;

}

void printSolution(int dist[])

{

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

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

cout << i << " " << dist[i] << endl;

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

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

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

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

{

int u = minDistance(dist, sptSet);

sptSet[u] = true;

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

{

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v])

{

dist[v] = dist[u] + graph[u][v];

}

}

}

printSolution(dist);

}

int main()

{

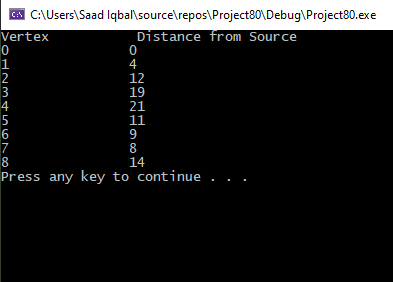
int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

system("pause");

return 0;

}

******

***TASK 2(Prims)***

#include<iostream>

#define V 5

using namespace std;

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;

}

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

{

cout << "Edge \tWeight\n";

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

{

cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << endl;

}

}

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];

}

}

printMST(parent, graph);

}

int main()

{

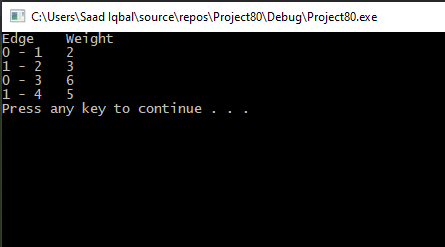
int graph[V][V] = { { 0, 2, 0, 6, 0 },{ 2, 0, 3, 8, 5 },{ 0, 3, 0, 0, 7 },{ 6, 8, 0, 0, 9 },{ 0, 5, 7, 9, 0 } };

primMST(graph);

system("pause");

return 0;

}

******

***TASK 3 (Bellman Ford)***

#include<iostream>

using namespace std;

struct edge

{

int src;

int dest;

int wt;

};

void bellman\_ford(int nv, edge e[], int src\_graph, int ne)

{

int u, v, weight, i, j = 0;

int dis[10];

for (i = 0; i < nv; i++)

{

dis[i] = 999;

}

dis[src\_graph] = 0;

for (i = 0; i < nv - 1; i++)

{

for (j = 0; j < ne; j++)

{

u = e[j].src;

v = e[j].dest;

weight = e[j].wt;

if (dis[u] != 999 && dis[u] + weight < dis[v])

{

dis[v] = dis[u] + weight;

}

}

}

for (j = 0; j < ne; j++)

{

u = e[j].src;

v = e[j].dest;

weight = e[j].wt;

if (dis[u] + weight < dis[v])

{

cout << "NEGATIVE CYCLE PRESENT ";

return;

}

}

cout << "Vertex" << " Distance from source";

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

{

cout << "\n" << i << "\t" << dis[i];

}

}

int main()

{

int vert, n, src;

edge obj[10];

cout << "Enter the number of vertices: ";

cin >> vert;

cout << "enter source vertex:";

cin >> src;

cout << "Enter no of edges: ";

cin >> n;

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

{

cout << "For edge " << i + 1 << "=>";

cout << "Enter source vertex :";

cin >> obj[i].src;

cout << "Enter destination vertex :";

cin >> obj[i].dest;

cout << "Enter weight :";

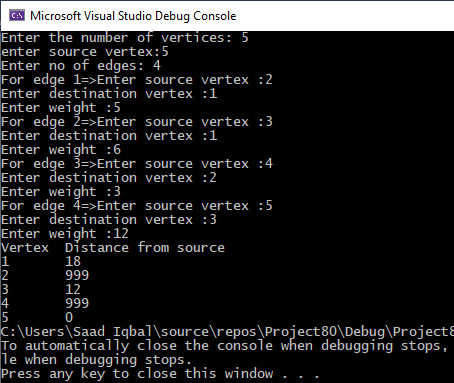
cin >> obj[i].wt;

}

bellman\_ford(vert, obj, src, n);

return 0;

}

******