**M. Ali. Arslan**

**19F-0348**

**Lab-11**

**Task # 01**

#include<iostream>

using namespace std;

struct edge {

int src;

int dest;

int weight;

};

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

int min = 999999, min\_index;

for (int v = 0; v < 9; v++)

if (sptSet[v] == false &&

dist[v] <= min)

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

return min\_index;

}

void printPath(int parent[], int j) {

if (parent[j] == -1) {

return;

}

printPath(parent, parent[j]);

cout << j;

}

void dijkstra(int graph[9][9], int src) {

int dist[9];

int parent[9];

bool sptSet[9];

int index = 0;

while (index < 9) {

parent[0] = -1;

dist[index] = INT\_MAX;

sptSet[index] = false;

index++;

}

dist[src] = 0;

int count = 0;

while (count < 8) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

int vIndex = 0;

while (vIndex < 9) {

if (!sptSet[vIndex] && graph[u][vIndex] && dist[u] + graph[u][vIndex] < dist[vIndex]) {

parent[vIndex] = u;

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

}

vIndex++;

}

count++;

}

//print path

src = 0;

cout << "\n\n\n\n..........................................................";

cout << "\n\n\nVertex\t Distance\tPath";

for (int i = 1; i < 9; i++) {

cout << "\n" << src << "->" << i << "\t\t " << dist[i] << "\t\t" << src;

printPath(parent, i);

}

}

// Driver Code

int main() {

int graph[9][9];

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) {

cout << "Enter weight of " << i << " - " << j << " : ";

cin >> graph[i][j];

}

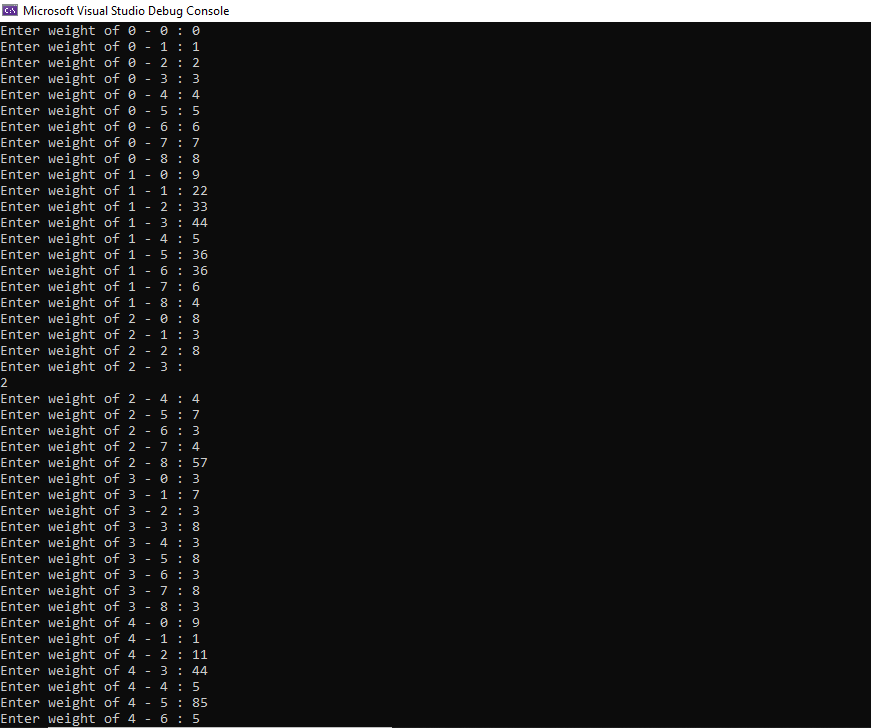
}

dijkstra(graph, 0);

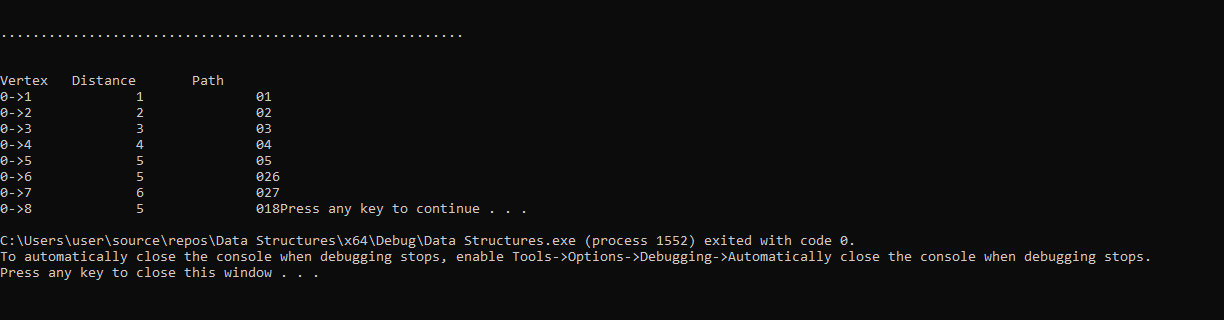
system("pause");

return 0;

}







**Task # 02**

#include<iostream>

using namespace std;

struct edge {

int src;

int dest;

int weight;

};

void bellman\_ford(int numberOfVector, edge e[], int src\_graph, int edgesN) {

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

int dis[10];

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

{// put by default a large distance in array

dis[i] = 9999;

}

/\* distance of source vertex from source vertex is o \*/

dis[src\_graph] = 0;

/\* relaxing all the edges nv - 1 times \*/

for (i = 0; i < numberOfVector - 1; i++) {

for (j = 0; j < edgesN; j++) {

u = e[j].src;

v = e[j].dest;

weight = e[j].weight;

if (dis[u] != 9999 && dis[u] + weight < dis[v]) {

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

}

}

}

int index = 0;

while (index < edgesN) {

u = e[index].src;

v = e[index].dest;

weight = e[index].weight;

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

return;

}

index++;

}

cout << "\nVertex" << " Distance from source";

for (i = 1; i <= numberOfVector; i++) {

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

}

}

// Driver Code

int main() {

int verticesN = 0;

int edgesN = 0;

int src\_graph;

edge Edge[10];

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

cin >> verticesN;

cout << "Enter the source vertex :";

cin >> src\_graph;

cout << "Enter number of edges: ";

cin >> edgesN;

int index = 0;

while (index < verticesN)

{

cout << "\nFor edge " << index + 1 << "=>";

cout << "\nEnter source vertex :";

cin >> Edge[index].src;

cout << "Enter destination vertex :";

cin >> Edge[index].dest;

cout << "Enter weight :";

cin >> Edge[index].weight;

index++;

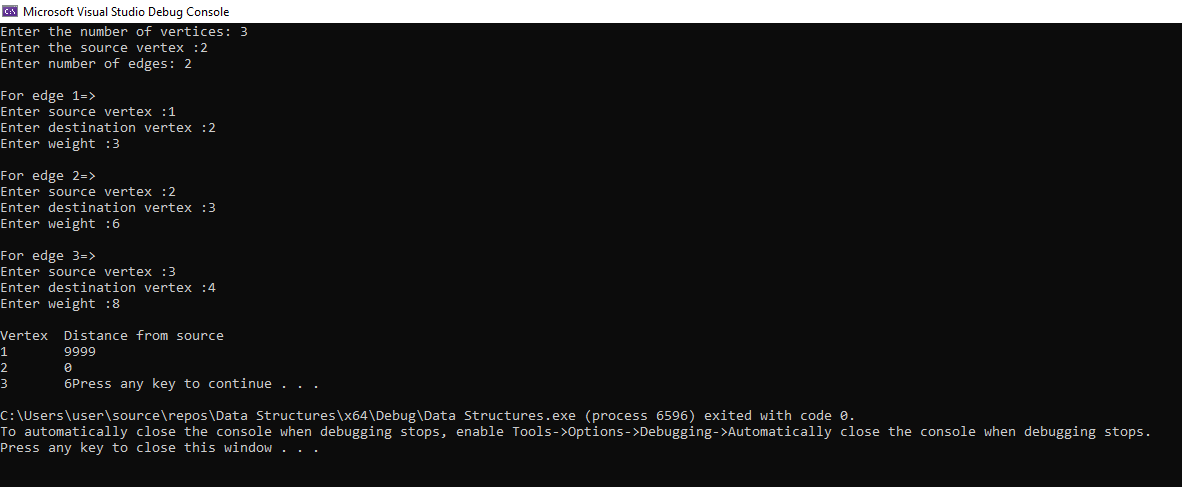
}

bellman\_ford(verticesN, Edge, src\_graph, edgesN);

system("pause");

return 0;

}



**Task # 03**

#include<iostream>

#include<algorithm>

using namespace std;

struct PRIMS {

int parent[5];

int key[5];

bool mstSet[5];

PRIMS() {

int index = 0;

while (index < 5) { key[index] = 99999; mstSet[index] = false; index++; }

key[0] = 0;

parent[0] = -1;

}

};

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

int min = 99999;

int min\_index;

for (int v = 0; v < 5; v++) { if (mstSet[v] == false && key[v] < min) { min = key[v]; min\_index = v; } }

return min\_index;

}

class Edge {

public:

int src;

int dest;

int weight;

Edge() {

src = 0;

dest = 0;

weight = 0;

}

Edge(const Edge& RHS) {

this->dest = RHS.dest;

this->src = RHS.src;

this->weight = RHS.weight;

}

int myComp(const void\* a, const void\* b) {

Edge\* a1 = (Edge\*)a;

Edge\* b1 = (Edge\*)b;

return a1->weight > b1->weight;

}

};

int main() {

int graph[5][5];

cout << "\nEnter Graph : \n";

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

cout << "Enter weight of " << i << " - " << j << " : ";

cin >> graph[i][j];

}

}

PRIMS prim;

int count = 0;

while (count < 4) {

int min = minKey(prim.key, prim.mstSet); prim.mstSet[min] = true; int index = 0;

while (index < 5) {

if (graph[min][index] && !prim.mstSet[index] && graph[min][index] < prim.key[index]) { prim.parent[index] = min; prim.key[index] = graph[min][index]; } index++;

}

count++;

}

// print the constructed MST

cout << "\n.........................................................";

cout << "\n\nEdge \tWeight\n";

int index = 1;

while (index < 5) { cout << prim.parent[index] << " - " << index << " \t" << graph[index][prim.parent[index]] << " \n"; index++; }

return 0;

}



**Task # 04**

#include<iostream>

#include<algorithm>

using namespace std;

class Edge {

public:

int src;

int dest;

int weight;

Edge() {

src = 0;

dest = 0;

weight = 0;

}

Edge(const Edge& RHS) {

this->dest = RHS.dest;

this->src = RHS.src;

this->weight = RHS.weight;

}

int myComp(const void\* a, const void\* b) {

Edge\* a1 = (Edge\*)a;

Edge\* b1 = (Edge\*)b;

return a1->weight > b1->weight;

}

};

class kruskals {

public:

int NumberOfVertices;

int edg;

Edge\* edge;

kruskals() {

NumberOfVertices = 0;

edg = 0;

}

kruskals(kruskals const& RHS) {

this->NumberOfVertices = RHS.NumberOfVertices;

this->edg = RHS.edg;

this->edge->dest = RHS.edge->dest;

this->edge->src = RHS.edge->src;

this->edge->weight = RHS.edge->weight;

}

};

struct subset {

int parent;

int rank;

subset() {

parent = 0;

rank = 0;

}

subset(subset const& RHS) {

this->parent = RHS.parent;

this->rank = RHS.rank;

}

int find(subset subsets[], int i) {

if (subsets[i].parent != i) subsets[i].parent = find(subsets, subsets[i].parent);

else { return subsets[i].parent; }

}

};

void Union(subset subsets[], int x, int y) {

int xroot = subsets->find(subsets, x);

int yroot = subsets->find(subsets, y);

if (subsets[xroot].rank < subsets[yroot].rank) { subsets[xroot].parent = yroot; }

else if (subsets[xroot].rank > subsets[yroot].rank) { subsets[yroot].parent = xroot; }

else { subsets[yroot].parent = xroot; subsets[xroot].rank++; }

}

void Sort(Edge\* ptr) {

for (int i = 0; i < 4; i++) { for (int j = i + 1; j < 4; j++) { if (ptr[i].weight > ptr[j].weight) { int temp = ptr[i].weight; ptr[i].weight = ptr[j].weight; ptr[j].weight = temp; } } }

}

void KruskalMST(kruskals\* graph) {

int NumberOfVertices = graph->NumberOfVertices;

Edge result[5];

int edg = 0;

int index = 0;

Sort(graph->edge);

subset\* subsets = new subset[(NumberOfVertices \* sizeof(subset))];

for (int v = 0; v < NumberOfVertices; ++v) { subsets[v].parent = v; subsets[v].rank = 0; }

while (edg < NumberOfVertices - 1 && index < graph->edg) {

Edge next\_edge = graph->edge[index++];

int x = subsets->find(subsets, next\_edge.src); int y = subsets->find(subsets, next\_edge.dest);

if (x != y) { result[edg++] = next\_edge; Union(subsets, x, y); }

}

cout << "Following are the edges in the constructed " "MST\n";

int minimumCost = 0;

for (index = 0; index < edg; ++index) { cout << result[index].src << " -- " << result[index].dest << " == " << result[index].weight << endl; minimumCost = minimumCost + result[index].weight; }

}

int main() {

int vertices = 4;

int edges = 5;

kruskals\* kruskalsGraph = new kruskals;

kruskalsGraph->NumberOfVertices = vertices;

kruskalsGraph->edg = edges;

kruskalsGraph->edge = new Edge[edges];

//input form user

for (int i = 0; i < 5; i++) {

cout << "\nEnter src : ";

cin >> kruskalsGraph->edge[0].src;

cout << "\nEnter destination : ";

cin >> kruskalsGraph->edge[0].dest;

cout << "\nEnter weight : ";

cin >> kruskalsGraph->edge[0].weight;

}

KruskalMST(kruskalsGraph);

return 0;

}

