**Assignment- I**

**Name: Soumyadip Ghosh**

**Stream: CSE -A**

**Roll Number: 1951007**

**Subject: Data Analysis and Algorithms Lab**

**Problem Statement:**

To Implement Dijkstra Algorithm using C,  
1. Take all the input through keyboard (number of vertices, graph, source vertex)  
2. Convert the entered graph into adjacency list for Cost Calculation (you may take the graph input directly as adjacency list to save time)  
3. Print the minimum cost from source to destination and also print the path between source and destination vertex

**C Code Implementation:**

#include <stdio.h>

#include <stdlib.h>

#define INF 9999

struct node {

int dest;

struct node\* next;

};

struct graph {

int n;

struct node\*\* adjlist;

};

struct node\* createnode(int v);

struct graph\* creategraph(int vertices);

void addEdge(struct graph\* graph, int s, int d, int w);

void printgraph(struct graph\* graph);

void pathfunc(int path[], int v);

void dijkstra(int n,int cost[n][n],int s);

int main() {

int n,e,i,j,src,dest,weight;

printf("Enter no. of vertices: ");

scanf("%d",&n);

printf("Enter no. of edges: ");

scanf("%d",&e);

struct graph\* graph = creategraph(n);

int cost[n][n];

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

{

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

{

cost[i][j]= INF;

}

}

printf("Enter source node, destination node and weight of edges:\n");

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

{

scanf("%d %d %d",&src, &dest, &weight);

addEdge(graph, src, dest, weight);

cost[src][dest]=weight;

cost[dest][src]=weight;

}

printf("\nEnter source vertex: ");

scanf("%d",&src);

printf("\nAdjacency List:\n");

printgraph(graph);

dijkstra(n,cost,src);

return 0;

}

struct node\* createnode(int v) {

struct node\* temp = (struct node\*)malloc(sizeof(struct node));

temp->dest = v;

temp->next = NULL;

return temp;

}

struct graph\* creategraph(int vertices) {

struct graph\* graph = malloc(sizeof(struct graph));

graph->n = vertices;

graph->adjlist = malloc(vertices \* sizeof(struct node\*));

int i;

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

graph->adjlist[i] = NULL;

return graph;

}

void addEdge(struct graph\* graph, int s, int d, int w) {

struct node\* temp = createnode(d);

temp->next = graph->adjlist[s];

graph->adjlist[s] = temp;

temp = createnode(s);

temp->next = graph->adjlist[d];

graph->adjlist[d] = temp;

}

void printgraph(struct graph\* graph) {

int i;

for (i=0; i<graph->n; i++) {

struct node\* temp = graph->adjlist[i];

printf("\n%d-> ", i);

printf("%d", temp->dest);

temp = temp->next;

while (temp) {

printf("-> %d ", temp->dest);

temp = temp->next;

}

printf("\n");

}

}

void pathfunc(int path[], int v)

{

if (path[v] == - 1)

return;

pathfunc(path, path[v]);

printf("%d--> ", v);

}

void dijkstra(int n,int cost[][n],int s)

{

int dist[n], p[n], v[n], count, min, path[n];

int i,k;

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

{

dist[i] = cost[s][i];

p[i] = s;

v[i] = 0;

}

dist[s] = 0;

v[s] = 1;

count = 1;

p[s] = -1;

path[0] = s;

while (count < n-1)

{

min = INF;

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

{

if(dist[i] < min && !v[i])

{

min = dist[i];

k = i;

}

}

v[k] = 1;

path[count] = k;

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

{

if(!v[i] && ((min + cost[k][i]) < dist[i]))

{

dist[i] = min + cost[k][i];

p[i] = k;

}

}

count++;

}

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

{

if (i!=s)

{

printf("Distance from %d to %d: %d\n",s,i, dist[i]);

}

}

printf("\n");

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

{

if (i!=s)

{

printf("\nPath from %d to %d: ",s,i);

printf("%d-->",s);

pathfunc(p, i);

}

}

}

**Output:**

