From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra’s algorithm**.

Handwrite  the Algorithm

- Handwrite the Program

- Pasting of the printout of the Output or handwrite the Output

**ALGORITHM**: dijkstras(c[1….n,1….n],src)

//To compute shortest distance  from given source node to all nodes of a weighted undirected graph

//Input: An nXn cost matrix c[1…n,1….n] with source node *src*

//Output: The length *dist[j]* of a shortest path from *src* to *j*

**for** j🡨1 to n **do**

     dist[j]🡨c[src,[j]

**end for**

**for** j🡨1 to n **do**

      vis[j]🡨0

**end for**

dist[src]🡨0

vis[src]🡨1

count🡨1

**while** count!=n **do**

         min🡨9999

**for** j🡨1 to n **do**

**if** dist[j]<min **and** vis[j]!=1

                  min🡨dist[j]

                  u🡨j

**end if**

**end for**

          vis[u]🡨1

          count🡨count+1

**for** j🡨1 to n **do**

**if** min+c[u,j]<dist[j] **and** vis[j]!=1

                   dist[j]🡨min+c[u,j]

**end if**

**end for**

**end while**

write ‘shortest distance is’

**for** j🡨1 to n **do**

     write src,j,dist[j]

**end for**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_NODES 100

#define INF 9999

void dijkstra(int n, int src, int cost[MAX\_NODES][MAX\_NODES]);

int main() {

int n;

int cost[MAX\_NODES][MAX\_NODES];

int src;

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

scanf("%d", &n);

printf("Enter the cost adjacency matrix (use -1 for infinity):\n");

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

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

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

if (cost[i][j] == -1 && i != j) {

cost[i][j] = INF;

}

}

}

printf("Enter the source node: ");

scanf("%d", &src);

dijkstra(n, src, cost);

return 0;

}

void dijkstra(int n, int src, int cost[MAX\_NODES][MAX\_NODES]) {

int dist[MAX\_NODES];

int vis[MAX\_NODES];

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

dist[j] = cost[src][j];

vis[j] = 0;

}

dist[src] = 0;

vis[src] = 1;

int count = 1;

while (count != n) {

int min = INF;

int u = -1;

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

if (!vis[j] && dist[j] < min) {

min = dist[j];

u = j;

}

}

if (u == -1) break;

vis[u] = 1;

count++;

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

if (!vis[j] && cost[u][j] != INF && dist[u] + cost[u][j] < dist[j]) {

dist[j] = dist[u] + cost[u][j];

}

}

}

printf("Shortest distances from source node %d:\n", src);

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

if (dist[j] == INF) {

printf("To %d: Infinity\n", j);

} else {

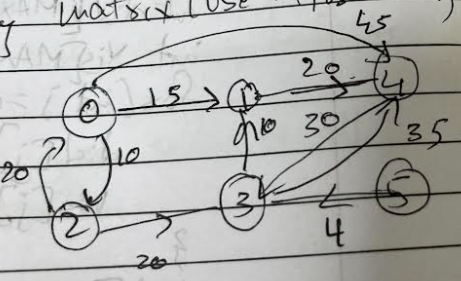
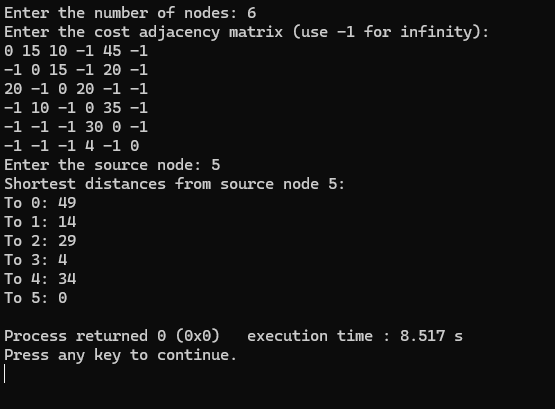
printf("To %d: %d\n", j, dist[j]);

}

}

}

**Output:**



Find Minimum Cost Spanning Tree of a given undirected graph using **Kruskals algorithm**.

- Handwrite  the Algorithm

- Handwrite the Program

- Pasting of the printout of the Output or handwrite the Output

**ALGORITHM:** kruskals(c[1…n,1…n])

//To compute the minimum spanning tree of a given weighted undirected graph using Kruskal’s

// algorithm

//Input: An nXn cost matrix c[1…n,1….n]

//Output: minimum cost of spanning tree of given undirected graph

ne🡨0

mincost🡨0

**for** i🡨1 to n **do**

     parent[i]🡨0

**end for**

**while** ne!=n-1 **do**

     min🡨9999

**for** i🡨1 to n **do**

**for** j🡨1 to n **do**

**if** c[i,j]<min

                   min🡨c[i,j]

                   u🡨i

                   a🡨i

                v🡨j

                b🡨j

**end if**

**end for**

**end for**

**while** parent[u]!=0 **do**

             u🡨parent[u]

**end while**

**while** parent[v]!=0 **do**

             v🡨parent[v]

**end while**

**if** u!= v

       write a,b,min

       parent[v]🡨u

       ne🡨ne+1

       mincost🡨mincost+min

**end if**

    c[a,b]🡨9999

    c[b,a]🡨9999

**end while**

write mincost

**return**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

#define INF 9999

struct Edge {

int u, v, weight;

};

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

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

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

return a1->weight - b1->weight;

}

int find(int parent[], int i) {

if (parent[i] == 0)

return i;

return find(parent, parent[i]);

}

void unionSets(int parent[], int u, int v) {

parent[v] = u;

}

void kruskals(int cost\_matrix[][MAX], int n) {

struct Edge edges[MAX \* MAX];

int edge\_count = 0;

int parent[MAX] = {0};

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

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

if (cost\_matrix[i][j] != INF) {

edges[edge\_count++] = (struct Edge){i, j, cost\_matrix[i][j]};

}

}

}

qsort(edges, edge\_count, sizeof(edges[0]), compare);

int mincost = 0;

int ne = 0;

printf("Edges in the Minimum Cost Spanning Tree:\n");

for (int i = 0; i < edge\_count; i++) {

int u = find(parent, edges[i].u);

int v = find(parent, edges[i].v);

if (u != v) {

printf("%d - %d : %d\n", edges[i].u, edges[i].v, edges[i].weight);

unionSets(parent, u, v);

mincost += edges[i].weight;

ne++;

}

if (ne == n - 1)

break;

}

printf("Minimum Cost of Spanning Tree: %d\n", mincost);

}

int main() {

int n;

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

scanf("%d", &n);

int cost\_matrix[MAX][MAX];

printf("Enter the cost matrix (n x n):\n");

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

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

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

if (cost\_matrix[i][j] == 0 || cost\_matrix[i][j] == -1)

cost\_matrix[i][j] = INF;

}

}

kruskals(cost\_matrix, n);

return 0;

**Output:**

}