**Week-10**

From a given vertex in a weighted connected graph, find shortest paths to other vertices using

Dijkstra’s algorithm.

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]&lt;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]&lt;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>  
#define MAX 100  
#define INF 9999  
  
void dijkstras(int c[MAX][MAX], int n, int src) {  
    int dist[MAX];  // To store the shortest distance from src to each vertex  
    int vis[MAX];   // To track if a vertex is visited  
    int count, min, u, i, j;  
  
    // Initialize distance and visited arrays  
    for (i = 1; i <= n; i++) {  
        dist[i] = c[src][i];  
        vis[i] = 0;  
    }  
  
    dist[src] = 0;  
    vis[src] = 1;  
    count = 1;  
  
    while (count != n) {  
        min = INF;  
  
        // Find the vertex with the minimum distance which is not yet visited  
        for (j = 1; j <= n; j++) {  
            if (dist[j] < min && vis[j] != 1) {  
                min = dist[j];  
                u = j;  
            }  
        }  
  
        vis[u] = 1;  
        count++;  
  
        // Update the distances of the adjacent vertices of the selected vertex  
        for (j = 1; j <= n; j++) {  
            if ((min + c[u][j] < dist[j]) && (vis[j] != 1)) {  
                dist[j] = min + c[u][j];  
            }  
        }  
    }  
  
    // Print the shortest distances from src to all vertices  
    printf("Shortest distances from node %d:\n", src);  
    for (i = 1; i <= n; i++) {  
        printf("%d -> %d : %d\n", src, i, dist[i]);  
    }  
}  
  
int main() {  
    int n, src;  
    int cost[MAX][MAX];  
  
    printf("Enter the number of vertices: ");  
    scanf("%d", &n);  
  
    printf("Enter the cost matrix:\n");  
    for (int i = 1; i <= n; i++) {  
        for (int j = 1; j <= n; j++) {  
            scanf("%d", &cost[i][j]);  
            if (cost[i][j] == 0 && i != j) {  
                cost[i][j] = INF;  
            }  
        }  
    }  
  
    printf("Enter the source node: ");  
    scanf("%d", &src);  
  
    dijkstras(cost, n, src);  
  
    return 0;  
}

|  |  |
| --- | --- |
| OUTPUT:  C:\Users\Dell\Downloads\image (10).png  Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm  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]&lt;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\_NODES 100 #define INF 9999  // Structure to represent an edge in the graph typedef struct {     int u, v, cost; } Edge;  // Function prototypes void kruskals(int c[MAX\_NODES][MAX\_NODES], int n); int find(int parent[], int i); void unionSets(int parent[], int x, int y);  int main() {     int n;     int cost[MAX\_NODES][MAX\_NODES];      printf("Enter the number of nodes (max %d): ", MAX\_NODES);     scanf("%d", &n);      // Input the cost matrix     printf("Enter the cost matrix (enter %d for no direct path):\n", INF);     for (int i = 0; i < n; i++) {         for (int j = 0; j < n; j++) {             scanf("%d", &cost[i][j]);             if (cost[i][j] == 0 && i != j) {                 cost[i][j] = INF;  // Convert 0 to INF for nodes that are not connected             }         }     }      kruskals(cost, n);      return 0; }  void kruskals(int c[MAX\_NODES][MAX\_NODES], int n) {     Edge edges[MAX\_NODES \* MAX\_NODES];  // Array to store all edges     int edgeCount = 0;      // Store all edges in the array     for (int i = 0; i < n; i++) {         for (int j = i + 1; j < n; j++) {             if (c[i][j] != INF) {                 edges[edgeCount].u = i;                 edges[edgeCount].v = j;                 edges[edgeCount].cost = c[i][j];                 edgeCount++;             }         }     }      // Sort edges based on their weights (costs)     // Using bubble sort for simplicity, can be replaced with more efficient algorithms like quicksort     for (int i = 0; i < edgeCount - 1; i++) {         for (int j = 0; j < edgeCount - i - 1; j++) {             if (edges[j].cost > edges[j + 1].cost) {                 Edge temp = edges[j];                 edges[j] = edges[j + 1];                 edges[j + 1] = temp;             }         }     }      // Initialize parent array for Union-Find     int parent[MAX\_NODES];     for (int i = 0; i < n; i++) {         parent[i] = 0;     }      int ne = 0;  // Number of edges in the MST     int mincost = 0;  // Minimum cost of the MST      printf("Edges in the Minimum Spanning Tree:\n");      // Process each edge in sorted order     for (int i = 0; i < edgeCount; i++) {         int u = edges[i].u;         int v = edges[i].v;         int cost = edges[i].cost;          // Find root of sets u and v         int root\_u = find(parent, u);         int root\_v = find(parent, v);          // If u and v are not in the same set, include this edge in the MST         if (root\_u != root\_v) {             printf("%d -> %d : %d\n", u, v, cost);             unionSets(parent, root\_u, root\_v);             mincost += cost;             ne++;         }          // If we have included n-1 edges, the MST is complete         if (ne == n - 1) {             break;         }     }      printf("Minimum Cost of Spanning Tree: %d\n", mincost); }  // Find function for Union-Find int find(int parent[], int i) {     while (parent[i] != 0) {         i = parent[i];     }     return i; }  // Union function for Union-Find void unionSets(int parent[], int x, int y) {     parent[y] = x; } |  |

**OUTPUT:**

