LAB-10

Question :

1. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.
2. Program on Kruskal's algorithm.

1.SOURCE 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;
        }
```

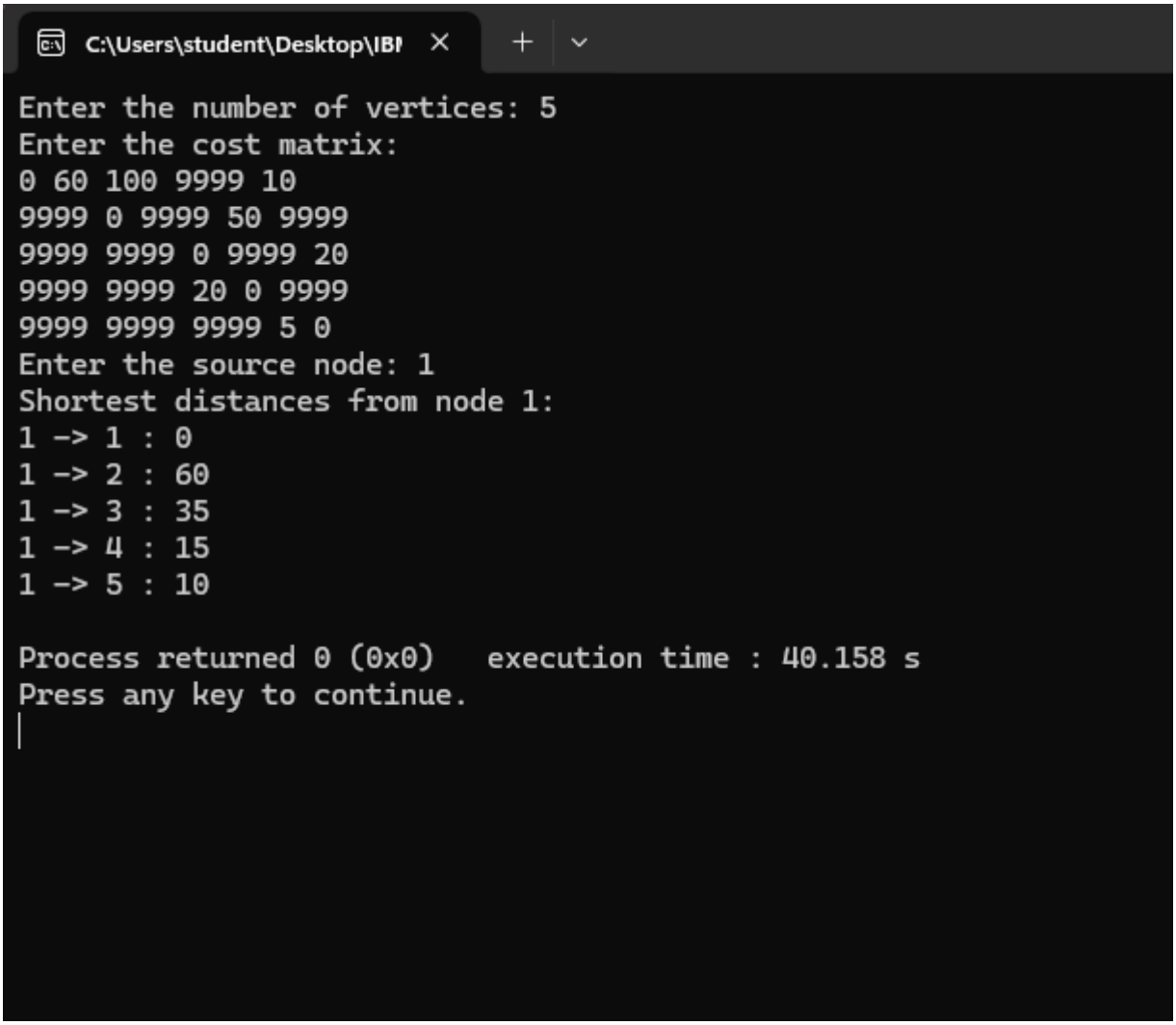
```
    }
  }

  printf("Enter the source node: ");
  scanf("%d", &src);

  dijkstras(cost, n, src);

  return 0;
}
```

RESULT:



2.SOURCE CODE:

```
#include <stdio.h>
#define MAX 100
#define INF 9999

// Structure to represent an edge in the graph
struct Edge {
   int u, v, weight;
};

// Structure to represent a subset for union-find
struct Subset {
   int parent;
   int rank;
};

int find(struct Subset subsets[], int i) {
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}

void Union(struct Subset subsets[], int x, int y) {
   int xroot = find(subsets, x);
   int yroot = 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 kruskals(int c[MAX][MAX], int n) {
   struct Edge edges[MAX * MAX];  // Array to store all edges
   struct Subset subsets[MAX];
   int ne = 0;  // Number of edges in minimum spanning tree
   int mincost = 0;  // Cost of minimum spanning tree
   int i, j, k;

   // Initialize subsets for union-find
   for (i = 1; i <= n; i++) {
      subsets[i].parent = i;
      subsets[i].rank = 0;
   }

   // Store all edges in the graph in the edges array
   k = 0;
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         if (c[i][j] != INF) {
            edges[k].u = i;
            edges[k].v = j;
            edges[k].weight = c[i][j];
            k++;
         }
      }
   }

   // Sort edges based on their weights
   for (i = 0; i < k - 1; i++) {
      for (j = 0; j < k - i - 1; j++) {
         if (edges[j].weight > edges[j + 1].weight) {
            struct Edge temp = edges[j];
            edges[j] = edges[j + 1];
            edges[j + 1] = temp;
         }
```

```
    }
  }

  // Iterate through all sorted edges
  for (i = 0; i < k; i++) {
    int u = edges[i].u;
    int v = edges[i].v;

    int set_u = find(subsets, u);
    int set_v = find(subsets, v);

    if (set_u != set_v) {
      // Include this edge in the minimum spanning tree
      printf("%d -> %d : %d\n", u, v, edges[i].weight);
      Union(subsets, set_u, set_v);
      mincost += edges[i].weight;
      ne++;
    }
  }

  printf("Minimum cost of spanning tree: %d\n", mincost);
}

int main() {
  int n;
  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;
        }
      }
    }

    kruskals(cost, n);

    return 0;
}
```

RESULT: