S.No: 6	Exp. Name: ***kruskals Algorithm***	Date:2025-08-28

Aim:

Write a C program to implement Kruskal’s algorithm for finding the Minimum Cost Spanning Tree (MCST) and the total minimum cost of travel for a given undirected graph. The graph will be represented by an adjacency matrix.

Input Format:

- The first input should be an integer, representing the number of vertices in the graph.
- The next input should be an adjacency matrix representing the weighted graph.
- If there is no edge between two vertices, the weight should be given as 9999 (representing infinity).

Output Format:

- The program should print the edges selected in the Minimum Spanning Tree (MST) along with their weights.

Note:

- Refer to the visible test cases to strictly match the input and output layout.

Source Code:

minCostFinding.c

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define INF 9999

typedef struct{
int u,v, cost;
}Edge;

int find(int parent[], int i) {
   while(parent[i] != i)
      i = parent[i];
   return i;
}

void union1(int parent[], int rank[], int i, int j) {
   int root1 = find(parent,i);
   int root2 = find(parent,j);

   if(rank[root1] < rank[root2])
      parent[root1] = root2;
   else if (rank[root1] > rank[root2])
      parent[root2] = root1;
   else{
      parent[root2] = root1;
      rank[root1]++;
   }
}

int compare(const void *a, const void *b){
   Edge *e1 = (Edge*)a;
   Edge *e2 = (Edge*)b;
   if(e1 -> cost != e2 -> cost)
      return e1 -> cost - e2 -> cost;
   if(e1 -> u != e2 -> u)
      return e1 -> u - e2 -> u;

   return e1 -> v - e2 -> v;
}
void kruskalMST(int **cost, int V) {
   Edge edges[V*V];
   int edgeCount = 0;

   for(int i =  0; i < V; i++){
      for(int j = i; j< V; j++)
         {
            if(cost[i][j] != INF)
            {
               edges[edgeCount].u = i;
               edges[edgeCount].v = j;
               edges[edgeCount].cost = cost[i][j];
               edgeCount++;
            }
         }
      }


qsort(edges, edgeCount, sizeof(Edge), compare);

   int *parent =  (int*)malloc(V*sizeof(int));
   int *rank = (int*)calloc(V, sizeof(int));

   for(int i = 0 ; i< V; i++){
      parent[i]  =i;
   }

   int numEdges = 0, minCost = 0;

   for(int i = 0; i< edgeCount && numEdges < V -1 ; i++)
      {
         int u = edges[i].u;
         int v = edges[i].v;
         int costEdge = edges[i].cost;

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

         if(set_u != set_v){
            printf("Edge %d:(%d, %d) cost:%d\n",numEdges,u,v, costEdge);
            union1(parent, rank, set_u, set_v);
            minCost += costEdge;
            numEdges ++;
         }
      }
   printf("Minimum cost= %d\n",minCost);
   free(parent);
   free(rank);
}

int main() {
    int V;
    printf("No of vertices: ");
    scanf("%d", &V);

    int **cost = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; i++)
        cost[i] = (int *)malloc(V * sizeof(int));

    printf("Adjacency matrix:\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &cost[i][j]);

    kruskalMST(cost, V);

    for (int i = 0; i < V; i++)
        free(cost[i]);
    free(cost);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
No of vertices: 5
Adjacency matrix: 9999 2 9999 9999 5
2 9999 3 9999 9999
9999 3 9999 4 9999
9999 9999 4 9999 9999
5 9999 9999 9999 9999
Edge 0:(0, 1) cost:2
Edge 1:(1, 2) cost:3
Edge 2:(2, 3) cost:4
Edge 3:(0, 4) cost:5
Minimum cost= 14

Test Case - 2
User Output
No of vertices: 4
Adjacency matrix: 9999 3 6 3
3 9999 5 2
6 5 9999 4
3 2 4 9999
Edge 0:(1, 3) cost:2

```
Edge 1:(0, 1) cost:3
Edge 2:(2, 3) cost:4
Minimum cost= 9
```