S.No: 6	Exp. Name: ***kruskals Algorithm***	Date:2025-09-24

ID: 24070521121

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
#define MAX 100

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

int parent[MAX];

int find(int i) {
// Write your code here...
   if(parent[i] == i)
      return i;
   return parent[i] = find(parent[i]);
}


int uni(int i, int j) {
   int r1 = find(i);
   int r2 = find(j);
   if(r1 != r2) {
      parent[r2] = r1;
      return 1;
   }
   return 0;
   // Write your code here...
}
```

```
void kruskalMST(int **cost, int V) {

   // Write your code here...
   Edge edges[MAX * MAX];
   int edgeCount = 0;

   for(int i = 0; i < V; i++) {
      for(int j = i+1; j < V; j++) {
         if(cost[i][j] != 0 && cost[i][j] != INF) {
            edges[edgeCount].u = i;
            edges[edgeCount].v = j;
            edges[edgeCount].cost = cost[i][j];
            edgeCount++;
         }
      }
   }

   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;
         }
      }
   }
   for(int i = 0; i < V; i++)
      parent[i] = i;
   int totalCost = 0, count = 0, edgeNum = 0;
   for(int i=0; i < edgeCount; i++) {
      int uSet = edges[i].u;
      int vSet = edges[i].v;
      if(uni(uSet,vSet)) {
         printf("Edge %d:(%d, %d) cost:%d\n", edgeNum++, edges[i].u, edges[i].v, edge
s[i].cost);
         totalCost += edges[i].cost;
         count++;
         if(count == V-1)
            break;
      }
   }
   printf("Minimum cost= %d\n", totalCost);
}


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