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

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 src, dest, weight;
} Edge;

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

void union1(int parent[], int rank[], int x, int y) {
    int xroot = find(parent, x);
    int yroot = find(parent, y);

    if (rank[xroot] < rank[yroot])
        parent[xroot] = yroot;
    else if (rank[xroot] > rank[yroot])
        parent[yroot] = xroot;
    else {
        parent[yroot] = xroot;
        rank[xroot]++;
    }
}

int compareEdges(const void *a, const void *b) {
    Edge *e1 = (Edge *)a;
    Edge *e2 = (Edge *)b;
    if (e1->weight != e2->weight)
        return e1->weight - e2->weight;
    if (e1->src != e2->src)
        return e1->src - e2->src;
    return e1->dest - e2->dest;
}


void kruskalMST(int **cost, int V) {
    Edge *edges = (Edge *)malloc(V * V * sizeof(Edge));
    int edgeCount = 0;

    for (int i = 0; i < V; i++) {
        for (int j = i + 1; j < V; j++) {
            if (cost[i][j] != INF) {
                edges[edgeCount].src = i;
                edges[edgeCount].dest = j;
                edges[edgeCount].weight = cost[i][j];
                edgeCount++;
            }
        }
   }
    qsort(edges, edgeCount, sizeof(Edge), compareEdges);

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

    int e = 0, i = 0, minCost = 0;

    while (e < V - 1 && i < edgeCount) {
        Edge next_edge = edges[i++];
        int x = find(parent, next_edge.src);
        int y = find(parent, next_edge.dest);

        if (x != y) {
            printf("Edge %d:(%d, %d) cost:%d\n", e, next_edge.src, next_edge.dest, ne
xt_edge.weight);
            minCost += next_edge.weight;
            union1(parent, rank, x, y);
            e++;
      }
   }

    printf("Minimum cost= %d\n", minCost);
    free(edges);
    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`