CSA 0317 DATA STRUCTURES

PROGRAM 25

```
#include <stdio.h>
#include <stdlib.h>

#define V 5
#define E 7

// Structure to represent an edge
struct Edge {
   int src, dest, weight;
};

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

// Function to find set of an element i
int find(struct subset subsets[], int i) {
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}

// Function that does union of two sets
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++;
    }
}

// Compare function for sorting edges by weight
int compare(const void* a, const void* b) {
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}

// Kruskal's algorithm
void KruskalMST(struct Edge edges[]) {
    struct Edge result[V]; // Stores resultant MST
    int e = 0; // Index for result[]
    int i = 0; // Index for sorted edges

    // Sort all edges in non-decreasing order of weight
    qsort(edges, E, sizeof(edges[0]), compare);

    // Allocate memory for subsets
    struct subset* subsets = (struct subset*)malloc(V * sizeof(struct subset));

    // Create V subsets with single elements
```

```
for (int v = 0; v < V; v++) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
}

// Process all edges
while (e < V - 1 && i < E) {
    // Pick the smallest edge
    struct Edge next_edge = edges[i++];

    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);

    // If including this edge doesn't cause cycle, include it
    if (x != y) {
        result[e++] = next_edge;
        Union(subsets, x, y);
    }
}

// Print the MST
printf("Kruskal's Algorithm - Minimum Spanning Tree:\n");
printf("Edge \tWeight\n");
int totalWeight = 0;
for (i = 0; i < e; i++) {
    printf("%d - %d \t%d \n", result[i].src, result[i].dest, result[i].weight);
    totalWeight += result[i].weight;
}
printf("Total weight of MST: %d\n", totalWeight);

free(subsets);
```

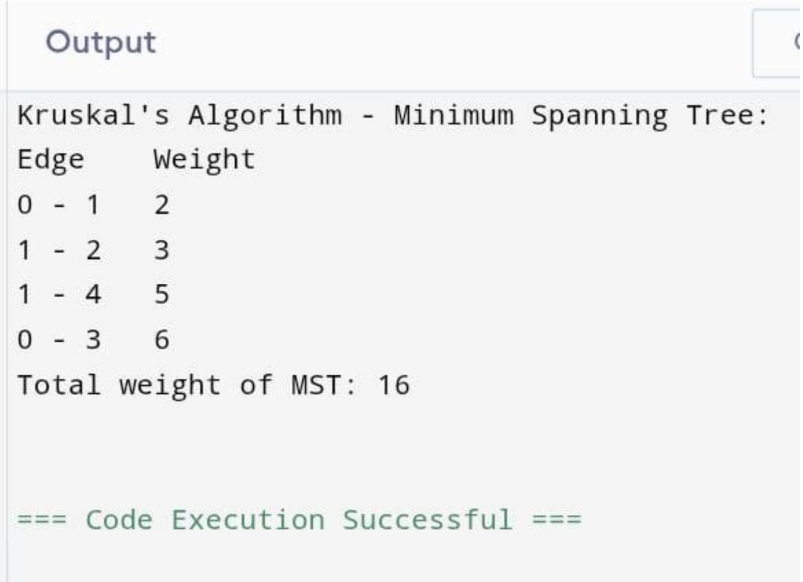
```
}

int main() {
    // Example graph represented as edges
    struct Edge edges[E] = {
        {0, 1, 2}, {0, 3, 6}, {1, 2, 3},
        {1, 3, 8}, {1, 4, 5}, {2, 4, 7},
        {3, 4, 9}
    };

    KruskalMST(edges);

    return 0;
}
```

Output:



```
Kruskal's Algorithm - Minimum Spanning Tree:
Edge    Weight
0 - 1   2
1 - 2   3
1 - 4   5
0 - 3   6
Total weight of MST: 16


=== Code Execution Successful ===
```