DATA STRUCTURES

Division	**CS(AIML) -A**
Batch	**2**
GR-no	**12311493**
Roll no	**54**
Name	**Atharva Kangralkar**

Assignment 13:

Generate Minimum Spanning Tree Using Kruskals Algorithm when Graph is Represented using A. Adjacency Matrix. B. Adjacency Lists.

Code:-

1. Adjacency Matrix

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

#define MAX 100

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

int parent[MAX];

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

void union_set(int u, int v) {
   parent[find(u)] = find(v);
}

void kruskal(int adj[MAX][MAX], int n) {
   Edge edges[MAX * MAX];
   int edge_count = 0;

   // Build edge list from adjacency matrix
   for (int i = 0; i < n; i++)
      for (int j = i + 1; j < n; j++)
         if (adj[i][j] != 0) {
            edges[edge_count++] = (Edge){i, j, adj[i][j]};
         }

   // Sort edges by weight (Bubble Sort)
   for (int i = 0; i < edge_count - 1; i++)
      for (int j = 0; j < edge_count - i - 1; j++)
         if (edges[j].weight > edges[j + 1].weight) {
            Edge temp = edges[j];
```

```
            edges[j] = edges[j + 1];
            edges[j + 1] = temp;
        }

    for (int i = 0; i < n; i++)
        parent[i] = i;

    int total = 0;
    printf("\nEdges in MST:\n");
    for (int i = 0; i < edge_count; i++) {
        int u = edges[i].u;
        int v = edges[i].v;

        if (find(u) != find(v)) {
            printf("(%d, %d) = %d\n", u, v, edges[i].weight);
            total += edges[i].weight;
            union_set(u, v);
        }
    }
    printf("Total weight = %d\n", total);
}

int main() {
    int n, adj[MAX][MAX];
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (0 for no edge):\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);

    kruskal(adj, n);
    return 0;
}
```

2. Adjacency List

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

#define MAX 100
```

```
typedef struct {
   int u, v, weight;
} Edge;

typedef struct Node {
   int vertex, weight;
   struct Node* next;
} Node;

Node* adjList[MAX];
int parent[MAX];

void addEdge(int u, int v, int weight) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->vertex = v;
   newNode->weight = weight;
   newNode->next = adjList[u];
   adjList[u] = newNode;
}

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

void union_set(int u, int v) {
   parent[find(u)] = find(v);
}

void kruskalFromAdjList(int n) {
   Edge edges[MAX * MAX];
   int edge_count = 0;

   for (int u = 0; u < n; u++) {
      Node* temp = adjList[u];
      while (temp != NULL) {
         int v = temp->vertex;
         if (u < v) // Avoid duplicate edges
            edges[edge_count++] = (Edge){u, v, temp->weight};
```

```
            temp = temp->next;
        }
    }

    for (int i = 0; i < edge_count - 1; i++)
        for (int j = 0; j < edge_count - i - 1; j++)
            if (edges[j].weight > edges[j + 1].weight) {
                Edge temp = edges[j];
                edges[j] = edges[j + 1];
                edges[j + 1] = temp;
            }

    for (int i = 0; i < n; i++)
        parent[i] = i;

    int total = 0;
    printf("\nEdges in MST:\n");
    for (int i = 0; i < edge_count; i++) {
        int u = edges[i].u;
        int v = edges[i].v;

        if (find(u) != find(v)) {
            printf("(%d, %d) = %d\n", u, v, edges[i].weight);
            total += edges[i].weight;
            union_set(u, v);
        }
    }
    printf("Total weight = %d\n", total);
}

int main() {
    int n, e, u, v, w;

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        adjList[i] = NULL;

    printf("Enter number of edges: ");
    scanf("%d", &e);
```

```
    printf("Enter edges (u v weight):\n");
    for (int i = 0; i < e; i++) {
        scanf("%d %d %d", &u, &v, &w);
        addEdge(u, v, w);
        addEdge(v, u, w); // For undirected graph
    }

    kruskalFromAdjList(n);
    return 0;
}
```

Output:-

1. **Adjacency matrix:-**

```
Enter number of vertices: 4
Enter adjacency matrix (0 for no edge):
0 5 0 7
5 0 6 0
0 6 0 0
7 0 0 0

Edges in MST:
(0, 1) = 5
(1, 2) = 6
(0, 3) = 7
Total weight = 18
```

2. **Adjacency list:-**

```
Enter number of vertices: 4
Enter number of edges: 3
Enter edges (u v weight):
0 1 5
0 3 7
1 2 6

Edges in MST:
(0, 1) = 5
(1, 2) = 6
(0, 3) = 7
Total weight = 18
```