Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL401**	**Course Name:**	**Analysis of Algorithm Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**7**
Title of the Experiment:	**Kruskal's Algorithm**
Date of Performance:	**20/02/2025**
Date of Submission:	**06/03/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mrs. Sneha Yadav

Signature :

Date:

Experiment No. 7

Title: Kruskal's Algorithm.

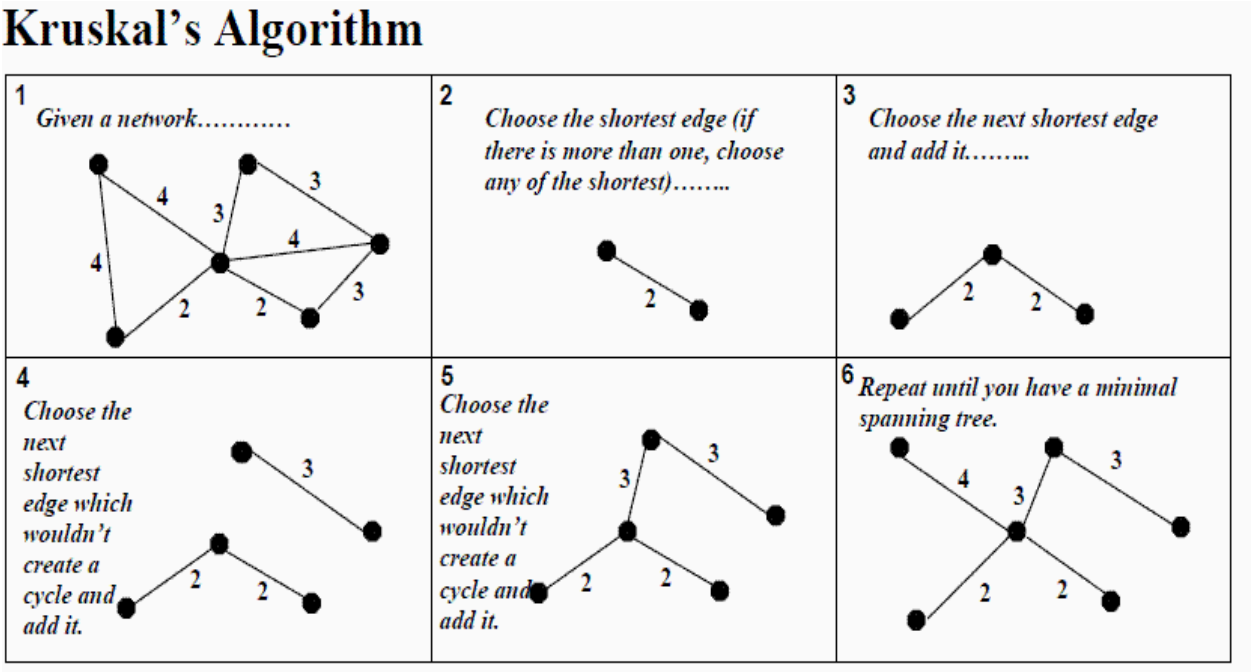
Aim: To study and implement Kruskal's Minimum Cost Spanning Tree Algorithm.

Objective: To introduce Greedy based algorithms

Theory:

Kruskal's algorithm finds a minimum spanning forest of an undirected edge-weighted graph. If the graph is connected, it finds a minimum spanning tree. (A minimum spanning tree of a connected graph is a subset of the edges that forms a tree that includes every vertex, where the sum of the weights of all the edges in the tree is minimized. For a disconnected graph, a minimum spanning forest is composed of a minimum spanning tree for each connected component.) It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest.

Example:



Algorithm and Complexity:

```
1   Algorithm Kruskal(E, cost, n, t)
2   // E is the set of edges in G. G has n vertices. cost[u, v] is the
3   // cost of edge (u, v). t is the set of edges in the minimum-cost
4   // spanning tree. The final cost is returned.
5   {
6       Construct a heap out of the edge costs using Heapify;
7       for i := 1 to n do parent[i] := -1;
8       // Each vertex is in a different set.
9       i := 0; mincost := 0.0;
10      while ((i < n - 1)  and (heap not empty)) do
11      {
12          Delete a minimum cost edge (u, v) from the heap
13          and reheapify using Adjust;
14          j := Find(u); k := Find(v);
15          if (j ≠ k) then
16          {
17              i := i + 1;
18              t[i, 1] := u; t[i, 2] := v;
19              mincost := mincost + cost[u, v];
20              Union(j, k);
21          }
22      }
23      if (i ≠ n - 1) then write ("No spanning tree");
24      else return mincost;
25  }
```

Time Complexity is O(nlog n), Where, n = number of Edges

Implementation:

```
#include <stdio.h>

struct Edge {
   int src, dest, weight;
};

int n, m, i, j;
struct Edge edges[20], result[20];
int parent[20];

void kruskalMST();
int find(int);
void unionSets(int, int);

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

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

```
    printf("Enter edges (source, destination, weight):\n");
    for (i = 0; i < m; i++) {
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }

    kruskalMST();
}

// Function to find the parent of a node
int find(int v) {
    if (parent[v] == v)
        return v;
    return find(parent[v]);
}

// Function to union two sets
void unionSets(int u, int v) {
    parent[v] = u;
}

// Function to implement Kruskal's Algorithm
void kruskalMST() {
    struct Edge temp;
    int cost = 0, edgeCount = 0;

    // Sort edges in ascending order of weight (Simple Bubble Sort)
    for (i = 0; i < m - 1; i++) {
        for (j = 0; j < m - i - 1; j++) {
            if (edges[j].weight > edges[j + 1].weight) {
                temp = edges[j];
                edges[j] = edges[j + 1];
                edges[j + 1] = temp;
            }
        }
    }

    // Initialize disjoint sets
    for (i = 0; i < n; i++)
        parent[i] = i;

    printf("\nMinimum Spanning Tree (MST):\n");
    printf("Edge \tWeight\n");

    // Pick edges one by one in sorted order
    for (i = 0; i < m && edgeCount < n - 1; i++) {
        int u = find(edges[i].src);
        int v = find(edges[i].dest);
```

```
            if (u != v) {
                result[edgeCount++] = edges[i];
                unionSets(u, v);
                cost += edges[i].weight;
                printf("%d - %d\t%d\n", edges[i].src, edges[i].dest, edges[i].weight);
            }
        }

        printf("Total Minimum Cost: %d\n", cost);
    }
```

```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter edges (source, destination, weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4

Minimum Spanning Tree (MST):
Edge    Weight
2 - 3   4
0 - 3   5
0 - 1   10
Total Minimum Cost: 19
```

Conclusion:

Kruskal's algorithm is another fundamental greedy approach used in graph theory to find a minimum spanning tree (MST) or a minimum spanning forest in the case of disconnected graphs. The algorithm works by sorting all the edges in ascending order of weight and then adding them one by one to the growing MST, provided they do not form a cycle. This process continues until all vertices are connected (in the case of a connected graph) or all components are individually spanned (in a disconnected graph). Kruskal's greedy nature—always picking the lowest weight edge available—ensures that the resulting spanning tree has the minimal total edge weight.