Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Name:	SHREYA GANESH WANKHEDE
Roll No:	62
Class/Sem:	SE/IV
Experiment No.:	7
Title:	Kruskal's Algorithm.
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	

Experiment No. 7

Title: Kruskal's Algorithm.

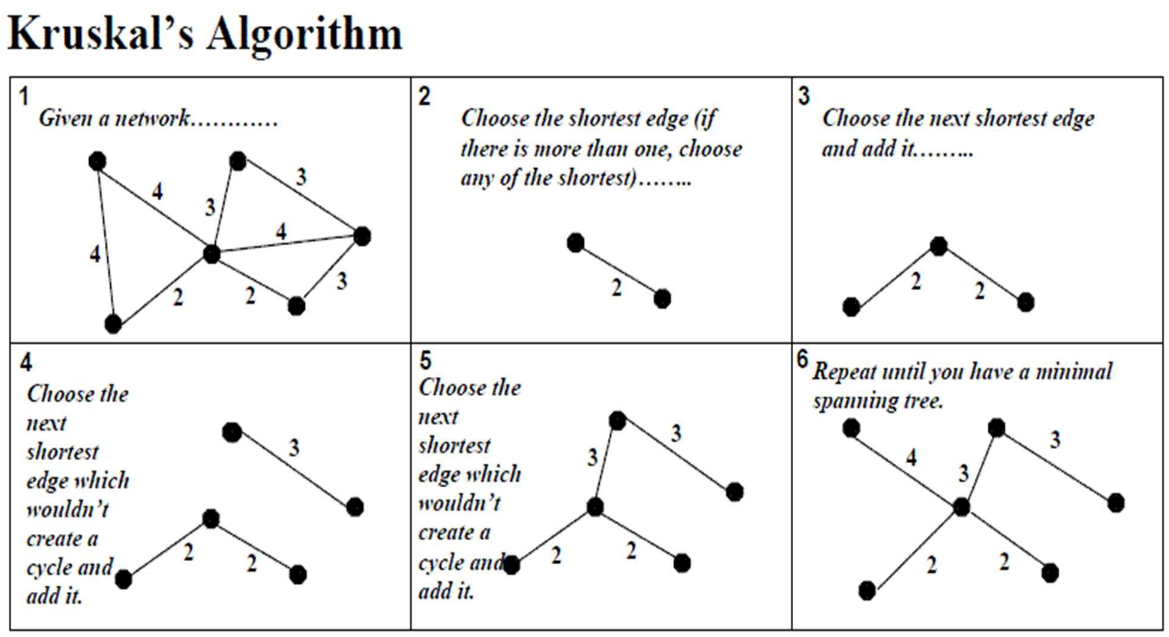
Aim: To study and implement Kruskal's Minimum Cost Spanning Tree Algorithm.

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:


Kruskal's Algorithm
1 Given a network…………
2 Choose the shortest edge (if there is more than one, choose any of the shortest)……..
3 Choose the next shortest edge and add it……..
4 Choose the next shortest edge which wouldn't create a cycle and add it.
5 Choose the next shortest edge which wouldn't create a cycle and add it.
6 Repeat until you have a minimal spanning tree.

Algorithm and Complexity:

```
Algorithm Kruskal($E, cost, n, t$)
// $E$ is the set of edges in $G$. $G$ has $n$ vertices. $cost[u, v]$ is the
// cost of edge $(u, v)$. $t$ is the set of edges in the minimum-cost
// spanning tree. The final cost is returned.
{
    Construct a heap out of the edge costs using Heapify;
    for $i := 1$ to $n$ do $parent[i] := -1$;
    // Each vertex is in a different set.
    $i := 0$; $mincost := 0.0$;
    while (($i < n - 1$) and (heap not empty)) do
    {
        Delete a minimum cost edge $(u, v)$ from the heap
        and reheapify using Adjust;
        $j :=$ Find($u$); $k :=$ Find($v$);
        if ($j \neq k$) then
        {
            $i := i + 1$;
            $t[i, 1] := u$; $t[i, 2] := v$;
            $mincost := mincost + cost[u, v]$;
            Union($j, k$);
        }
    }
    if ($i \neq n - 1$) then write ("No spanning tree");
    else return $mincost$;
}
```

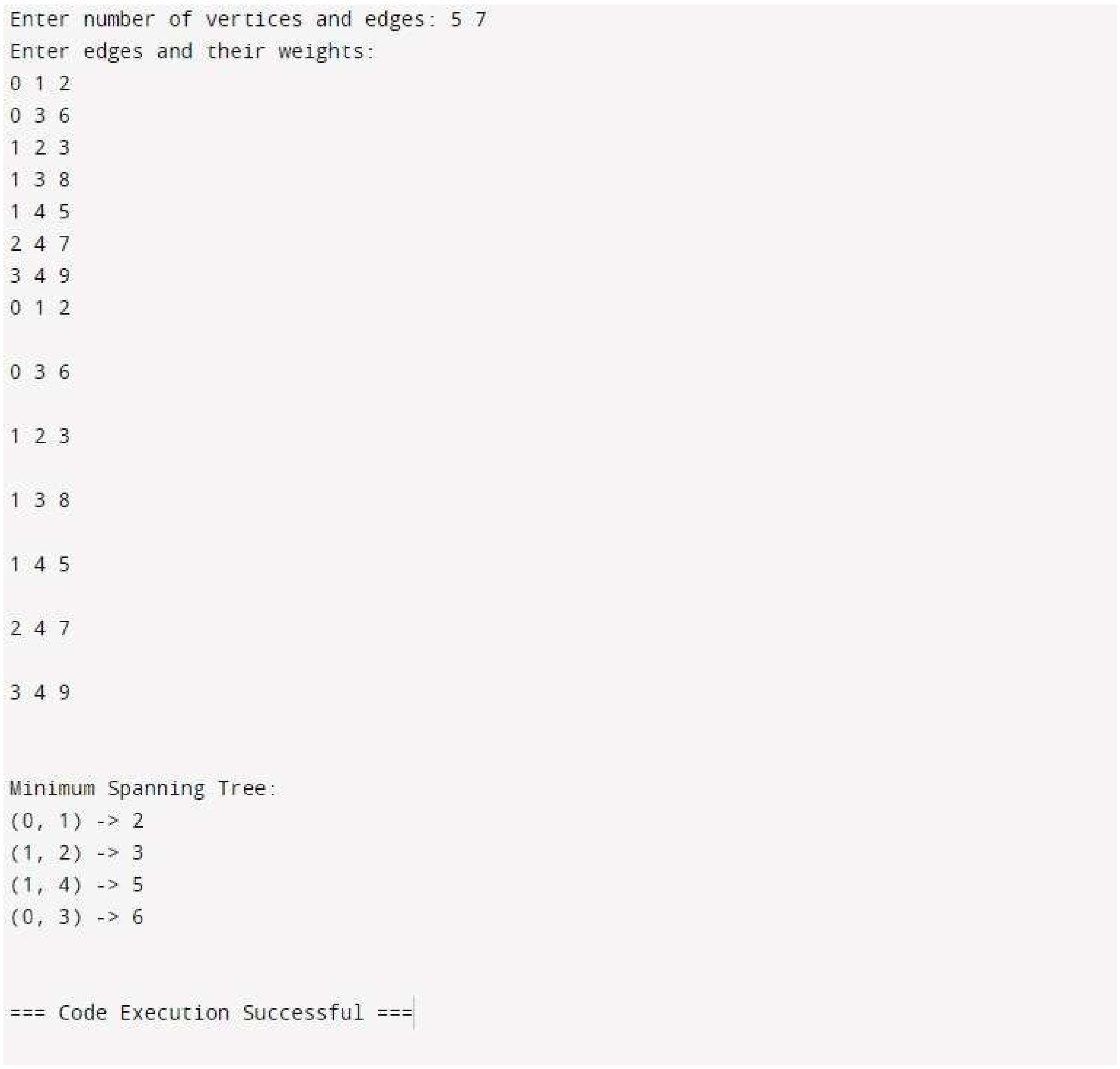
Time Complexity is O(nlog n), Where, n = number of Edges

Implemenation:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_EDGES 1000

typedef struct Edge {

   int src, dest, weight;

} Edge;

typedef struct Graph {

   int V, E;

   Edge edges[MAX_EDGES];
} Graph;
typedef struct Subset {
    int parent, rank;
} Subset;
Graph* createGraph(int V, int E) {
    Graph* graph = (Graph*) malloc(sizeof(Graph));
    graph->V = V;
    graph->E = E;
    return graph;
}
int find(Subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}
void Union(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++;
    }
}
```

```
int compare(const void* a, const void* b) {
    Edge* a_edge = (Edge*) a;
    Edge* b_edge = (Edge*) b;
    return a_edge->weight - b_edge->weight;
}
void kruskalMST(Graph* graph) {
    Edge mst[graph->V];
    int e = 0, i = 0;
    qsort(graph->edges, graph->E, sizeof(Edge), compare);
    Subset* subsets = (Subset*) malloc(graph->V * sizeof(Subset));
    for (int v = 0; v < graph->V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    while (e < graph->V - 1 && i < graph->E) {
        Edge next_edge = graph->edges[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);
        if (x != y) {
            mst[e++] = next_edge;
            Union(subsets, x, y);
        }
    }
    printf("Minimum Spanning Tree:\n");
    for (i = 0; i < e; ++i) {
        printf("(%d, %d) -> %d\n", mst[i].src, mst[i].dest, mst[i].weight);
}
}
int main() {
    int V, E;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &V, &E);
    Graph* graph = createGraph(V, E);
    printf("Enter edges and their weights:\n");
    for (int i = 0; i < E; ++i) {
        scanf("%d %d %d", &graph->edges[i].src, &graph->edges[i].dest, &graph-
>edges[i].weight);
    }
    kruskalMST(graph);
    return 0;
}
```

Output:



```
Enter number of vertices and edges: 5 7
Enter edges and their weights:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
0 1 2

0 3 6

1 2 3

1 3 8

1 4 5

2 4 7

3 4 9


Minimum Spanning Tree:
(0, 1) -> 2
(1, 2) -> 3
(1, 4) -> 5
(0, 3) -> 6


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

Conclusion: Implementing Kruskal's algorithm proved effective in finding the minimum spanning tree of a given graph. Its simplicity and efficiency make it a valuable tool for solving graph optimization problems, demonstrating its practical applicability in various domains.