|  |
| --- |
| Experiment No. 7 |
| Kruskal’s Algorithm |
| Date of Performance: |
| Date of Submission: |

**Experiment No. 7**

**Title:** Kruskal’s Algorithm.

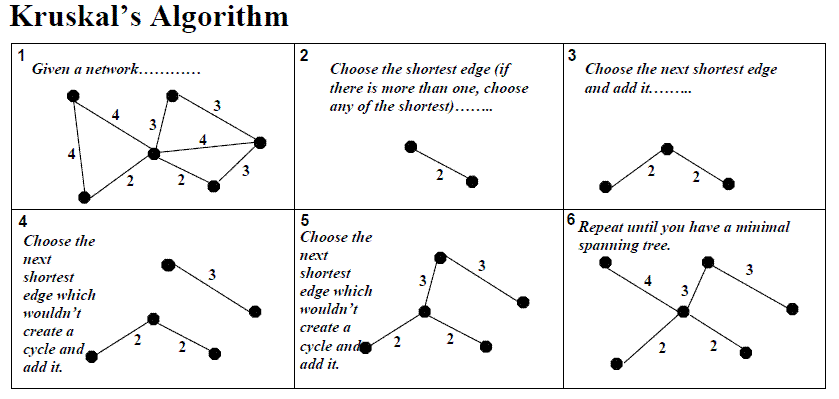
**Aim:** To study and implement Kruskal’s Minimum Cost Spanning Tree Algorithm.

**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:**

**A screenshot of a computer program

Description automatically generated**

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:**



**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.