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.

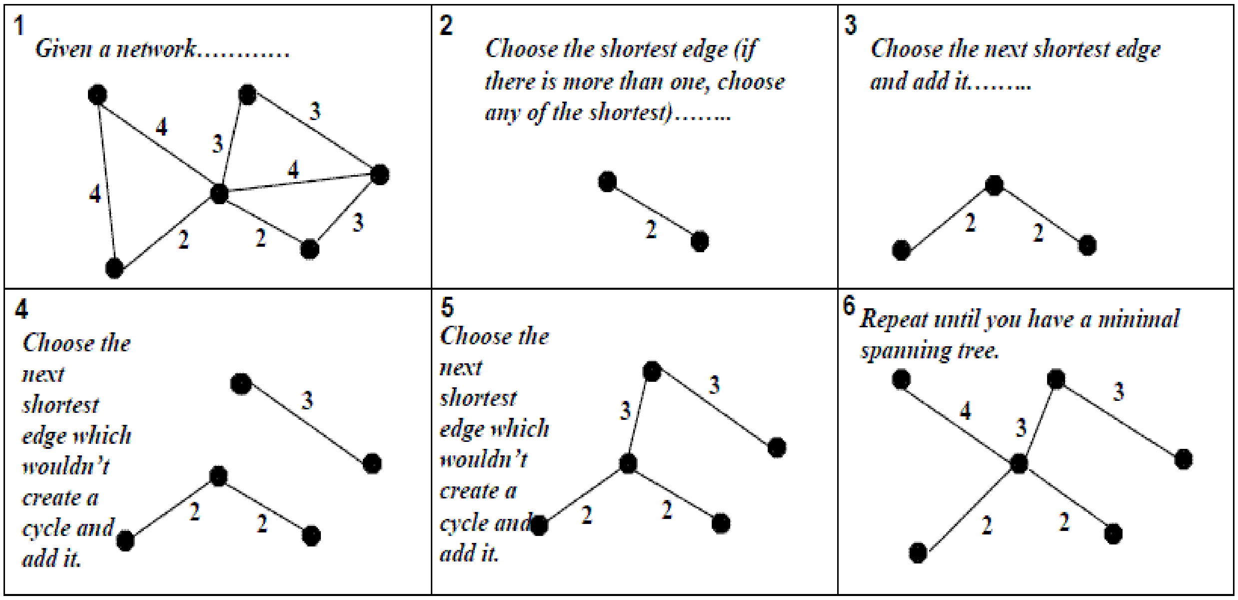
Objective: To introduce Greedy based algorithms

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:

Kruskal's Algorithm



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 ≠ k) then
        {
            i := i + 1;
            t[i, 1] := u; t[i, 2] := v;
            mincost := mincost + cost[u, v];
            Union(j, k);
        }
    }
    if (i ≠ n - 1) then write ("No spanning tree");
    else return mincost;
}
```

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

Implementation:

Code:

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

int comparator(const void* p1, const void* p2)
{
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;

    return (*x)[2] - (*y)[2];
```

```
}

void makeSet(int parent[], int rank[], int n)
{
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}

int findParent(int parent[], int component)
{
    if (parent[component] == component)
        return component;

    return parent[component]
        = findParent(parent, parent[component]);
}

void unionSet(int u, int v, int parent[], int rank[], int n)
{
    u = findParent(parent, u);
    v = findParent(parent, v);

    if (rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if (rank[u] > rank[v]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;
        rank[u]++;
    }
}

void kruskalAlgo(int n, int edge[n][3])
{
    qsort(edge, n, sizeof(edge[0]), comparator);

    int parent[n];
```

```
    int rank[n];
    makeSet(parent, rank, n);
    int minCost = 0;

    printf("Following are the edges in the constructed MST\n");
    for (int i = 0; i < n; i++) {
        int v1 = findParent(parent, edge[i][0]);
        int v2 = findParent(parent, edge[i][1]);
        int wt = edge[i][2];

        if (v1 != v2) {
            unionSet(v1, v2, parent, rank, n);
            minCost += wt;
            printf("%d -- %d == %d\n", edge[i][0],
                edge[i][1], wt);
        }
    }

    printf("Minimum Cost Spanning Tree: %d\n", minCost);
}
int main()
{
    int edge[5][3] = { { 0, 1, 10 },
                       { 0, 2, 6 },
                       { 0, 3, 5 },
                       { 1, 3, 15 },
                       { 2, 3, 4 } };

    kruskalAlgo(5, edge);

    return 0;
}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SEARCH ERROR

PS C:\Users\Lenovo\Downloads\AOA Experiments> cd "c:\Users\Lenovo\Downloads\AOA Experiments\" ;
.c -o kruskal } ; if ($?) { .\kruskal }
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
PS C:\Users\Lenovo\Downloads\AOA Experiments>
```

Conclusion:

In conclusion, the Kruskal's algorithm efficiently finds the Minimum Spanning Tree (MST) of a connected, undirected graph. The MST is a subgraph that includes all the vertices of the original graph with the minimum possible total edge weight, ensuring that there are no cycles and the graph remains connected.

Kruskal's algorithm achieves this by iteratively adding the shortest edge that does not form a cycle in the current set of selected edges. It maintains a forest of disjoint sets, initially containing single vertices. Edges are sorted by their weights in ascending order, and then each edge is added to the MST if it connects two vertices from different sets, effectively merging the sets.

This implementation of Kruskal's algorithm utilizes the disjoint-set data structure, which efficiently keeps track of the connected components and performs union-find operations. It sorts the edges by their weights and then iterates through them, adding edges to the MST if they do not create a cycle.

The time complexity of Kruskal's algorithm is dominated by the sorting step, which takes O(E log E), where E is the number of edges in the graph. The union-find operations take nearly O(E) time in total, resulting in a total time complexity of O(E log E).