Assignment No-03(Group A)

Title: - Implement Greedy search algorithm for Dijkstra's Minimal Spanning Tree Algorithm

Objectives:-

1. Understand the concept of Greedy search algorithm.
2. Understand the implementation of Dijkstra's Minimal Spanning Tree Algorithm

Problem Statement:-

Implement Greedy search algorithm for any of the following application:

I. Selection Sort
II. Minimum Spanning Tree
III. Single-Source Shortest Path Problem
IV. Job Scheduling Problem
V. Prim's Minimal Spanning Tree Algorithm
VI. Kruskal's Minimal Spanning Tree Algorithm
VII. Dijkstra's Minimal Spanning Tree Algorithm

Objective

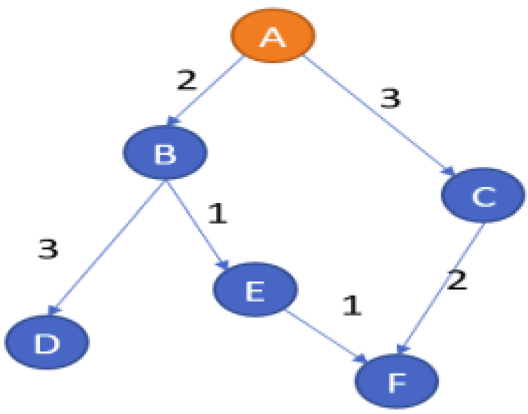
Student should be able to implement Dijkstra's algorithm

Software and Hardware requirements:-

1. **Operating system:** Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10,
2. **RAM-** 2GB RAM (4GB preferable)
3. You have to install **Python3** or higher version

Theory-

Consider the following graph given in figure below.



The state space representation of the above graph is done using dictionary data structure as follows.

graph={ 'A':{'B':2,'C':3}, 'B':{'D':3,'E':1}, 'C':{'F':2}, 'D':{}, 'E':{'F':1}, 'F':{}}

The above graph representation is a python dictionary wherein a node and its immediate successors or child nodes are specifies, for example, Node ‘A‘ has node ‘B‘ and node ‘C‘ as successors. The path cost from node ‘A‘ to node ‘B‘ is 2 and that of path ‘A‘ to ‘C‘ is 3.

Dijkstra‘s algorithm:

```
function Dijkstra(Graph, source):
    dist[source] ← 0 // the start node has distance 0
    create vertex priority queue Q //Use priority Queue here
    for each vertex v in Graph:
        if v ≠ source
            dist[v] ← INFINITY //Assign the weight
        prev[v] ← UNDEFINED //predecessor of current node v

    Q.add_with_priority(v, dist[v]) //
    while Q is not empty:
        u ← Q.extract_min() // Remove and return best vertex
    for each neighbor v of u still in Q: // only v that are still in Q
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u
            Q.decrease_priority(v, alt)
    return dist, prev
```

Application:

The algorithm finds the Minimal Spanning Tree for given graph when a source is specified so it is used in Google Map and other similar application to find the shortest route between the source and destination.

Input:

State Space representation of the given Graph or the problem.

Output:

The algorithm finds the Minimal Spanning Tree for given graph when a source is specified.

Conclusion:

The Dijkstra‘s Algorithm finds the Single source and multiple destinations with shortest distance between the two nodes in the given graph thereby finding the shortest distance between a given source and the destination.