Homework 5

aryanrao

PART A

1. **Algorithm:**

- Start with the original MST T.
- Remove the edge e from T.
- Add the updated edge e with the increased weight δ to T.
- Run a minimum spanning tree algorithm (e.g., Prim's algorithm or Kruskal's algorithm) on the modified graph G with the updated MST T.
- Return the new MST obtained from the minimum spanning tree algorithm.

Pseudocode:

```
UpdateMST(G, T, e, δ):
    // Step 1: Start with the original MST T
    newT = T

    // Step 2: Remove edge e from newT
    newT.remove(e)

    // Step 3: Add the updated edge e with increased weight δ to newT
    newE = Edge(e.u, e.v, e.weight + δ)
    newT.add(newE)

    // Step 4: Run minimum spanning tree algorithm on modified graph G with updated MST newT
    if G is an adjacency matrix:
        modifiedGraph = Copy(G)
        modifiedGraph[e.u][e.v] = e.weight + δ
        modifiedGraph[e.v][e.u] = e.weight + δ
        return MinimumSpanningTree(modifiedGraph)
    else if G is an edge list:
        modifiedEdges = Copy(G.edges)
        modifiedEdges.remove(e)
        modifiedEdges.add(newE)
        return MinimumSpanningTree(modifiedEdges)
```

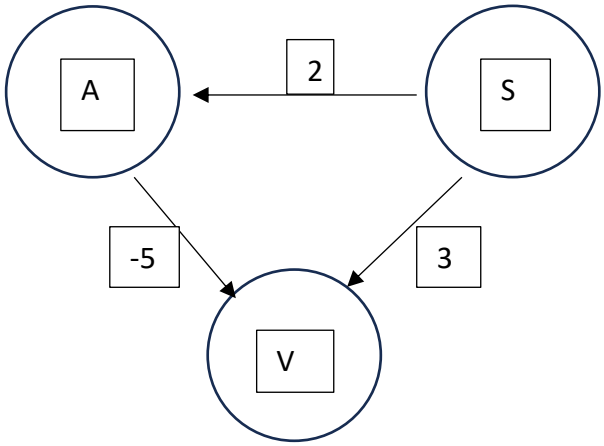
```
// Step 5: Return the new MST
return new MST obtained from the minimum spanning tree algorithm
```

- Again, the algorithm starts with the original MST T and removes edge e from it.
- It then adds the updated edge e with the increased weight δ to T.
- By doing this, the algorithm ensures that the new MST includes the updated edge e, and thus accounts for the increased weight.
- Running a minimum spanning tree algorithm on the modified graph with the updated MST will find a new MST that satisfies the minimum weight criterion.

Runtime:

- The runtime of the algorithm depends on the runtime of the minimum spanning tree algorithm used.
- If the minimum spanning tree algorithm has a runtime of O(V^2) when the graph is represented as an adjacency matrix, the overall runtime will be O(V^2).
- If the minimum spanning tree algorithm has a runtime of O(E log E) when the graph is represented as an edge list, the overall runtime will be O(E log E), where E is the number of edges in the graph.

2.



Pseudocode:

```
function Dijkstra(Graph, s):

        dist[s] = 0
        for each vertex v in Graph:

        if v ≠ s

        dist[v] = infinity

        previous[v] = undefined

        add v to Q
        while Q is not empty:

        u = vertex in Q with min dist[u]

        remove u from Q
        for each neighbor v of u:

        alt = dist[u] + length(u, v)

        if alt < dist[v]

        dist[v] = alt

        previous[v] = u

return dist[], previous[]
```

Algorithm:

Let's say we have a graph with nodes S, A, and V. Let S serve as the source node. The graph's edges are:

S to A with a weight of 2
S to V with a weight of 3
A to V with a weight of -5

Here, Dijkstra's algorithm incorrectly determines the shortest distance from S to V as 3, not -3.

This occurs because the method assumes the shortest path to a node (in this case, V) has been found after visiting it once, and it does not do so again. Due to its greedy approach, Dijkstra's method would not locate the shorter path from S to V, which is S to A to V with a total weight of 2 - 5 = -3.

Because there is a negative edge weight in this case, Dijkstra's method returns the erroneous shortest path from S to V.