Complexity Analysis of Algorithms (US13, US17, US18)

US13

Description: Implementation of Kruskal's algorithm.

Complexity Analysis

Primitive Operations in Kruskal’s algorithm

- Initialization:
  - parent: Initialization of an array of size n
  - rank: Initialization of an array of size n
  - Total: $2n$
- Main Loop:
  - find: Recursive call
  - Total: Variable, dependent on recursive calls
  - Union:
    - Comparison of rank: 1 operation
    - Total: 1
  - Total: $2n +$ Variable, dependent on recursive calls

Complexity Table

Line	Operations	Operation Count
1	A	$1A$
2	A	$V \times 1A$
3	A	$V \times 1A$
4	I	$2V$
5	Variable	$Variable$
6	R	$1R$
Total	-	$Variable$

US17

Description: Finds the shortest path from a starting point to an assembly point (AP) using Dijkstra's algorithm.

Pseudocode

```
Dijkstra(graph, start):
    dist = array of size graph.length initialized to ∞
    prev = array of size graph.length initialized to -1
    dist[start] = 0
    queue = priority queue initialized with start

    while queue is not empty:
        u = queue.extract_min()

        for each neighbor v of u:
            if graph[u][v] != 0 and dist[u] + graph[u][v] <
dist[v]:
                dist[v] = dist[u] + graph[u][v]
                prev[v] = u
                queue.insert(v)

    return dist, prev

getShortestPath(prev, target):
    path = empty list
    while target != -1:
        path.add(target)
        target = prev[target]
    reverse(path)

    return path
```

Complexity Analysis

Complexity Table

Line	Operations	Operations Count
1	A	$1A$
2	A	$V \times 1A$
3	A	$V \times 1A$
4	I, C	$V \times (I + C)$
5	I, C, Op	$E \times (I + C + Op)$
6	R	$1R$
Total	-	$O((V + E)\log(V))$

US18

Description: Finds the shortest path from a point to the nearest assembly point (AP) using Dijkstra's algorithm. For all points, calculates the path to the nearest AP.

Pseudocode

```
findNearestAP(graph, startPoint):
    dijkstra = Dijkstra(graph, startPoint)
    minCost = ∞
    nearestAP = -1
    for each ap in assemblyPoints:
        if dijkstra.dist[ap] < minCost:
            minCost = dijkstra.dist[ap]
            nearestAP = ap
    return nearestAP

Dijkstra(graph, start):
    dist = array of size graph.length initialized to ∞
    prev = array of size graph.length initialized to -1
    dist[start] = 0
    queue = priority queue initialized with start

    while queue is not empty:
        u = queue.extract_min()

        for each neighbor v of u:
            if graph[u][v] != 0 and dist[u] + graph[u][v] <
dist[v]:
                dist[v] = dist[u] + graph[u][v]
                prev[v] = u
                queue.insert(v)

    return dist, prev

computeAllShortestPaths(graph):
    for each point in graph:
        if not point is assemblyPoint:
            nearestAP = findNearestAP(graph, point)
            Dijkstra(graph, nearestAP)
            path = getShortestPath(dijkstra.prev, point)
            printPath(path, dijkstra.dist[point])
```

Complexity Analysis

Complexity Table

Line	Operations	Operations Count
1	A	$1A$
2	A, Op	$A \times (A + Op)$
3	A	$1A$
4	I, C	$A \times (I + C)$
5	I, C, Op	$A \times (I + C + Op)$
6	R	$1R$
Total	-	$O(A \cdot (V + E) log\,(V))$

Conclusion

In this analysis, we examined three user stories (US) related to graph manipulation and algorithms for shortest paths and MST (Minimum Spanning Tree). Here are some conclusions about each:

- **US13 (Kruskal)**
    - Kruskal's algorithm to find the Minimum Spanning Tree (MST) in a weighted graph was implemented.
    - The algorithm's complexity depends on the specific implementation, with the main part being the sorting of edges and cycle checking.
    - The complexity analysis indicated an average complexity of $O(Elog\,(E))$, where E is the number of edges.

- **US17 (Dijkstra)**
    - US17 describes the implementation of Dijkstra's algorithm to find the shortest path from a starting point to an assembly point (AP) in a weighted graph.
    - The complexity analysis revealed an average complexity of $O((V+E)log(V))$, where V is the number of vertices and EEE is the number of edges in the graph.
- **US18**
    - In this US, Dijkstra's algorithm is used to find the shortest path from a point to the nearest assembly point (AP), and this is done for all points in the graph.
    - The complexity analysis showed an average complexity of $O(A \cdot (V+E)log(V))$, where A is the number of assembly points (APs).

In summary, each of the user stories implements fundamental algorithms for graph manipulation, such as finding the MST and calculating shortest paths. The provided complexity analyses are crucial for understanding the performance of these algorithms in different scenarios and help in decision-making related to the design and optimization of systems that use them.