Lab Assignment: Implementing Kruskal's Algorithm

Course: CS 2511(the Fundies 2 Lab)

Topic:

Graph Algorithms - Kruskal's Minimum Spanning Tree Algorithm

Purpose:

The purpose of this lab assignment is to familiarize students with the fundamental graph algorithms, specifically Kruskal's Algorithm for finding the Minimum Spanning Tree (MST) of a connected, undirected graph with weighted edges. Students will implement the Union-Find data structure to optimize the algorithm's performance.

Objective:

1. To understand and implement Kruskal's Algorithm using either an adjacency matrix or an adjacency list representation of a graph.
2. To analyze the complexity and efficiency of Kruskal's Algorithm.

Instructions:

1. Implement the Union-Find data structure with the operations Find and Union.
2. Choose either an adjacency matrix or list to represent the graph.
3. Ensure that your implementation correctly identifies the MST of the input graph.

Your implementation can consider below hint:

1. Test the corner case, especially overflow or IO Exception.
2. Sort functions need to be written on your own, avoid using Java library for sort.
3. If you use UnionFind: make sure you define a class which implements the Union-Find data structure with find and union methods.
4. Use the main class, do basic tests and then implement at least 5 unit tests.

5. Write comments for some functions or complex operations in your code.
6. Dry run your idea with peers and draw the design structure for the whole problem.
7. Write well-structured technology documentation.
8. Debug trajectory(print, write small test for some part of function…adjust the simple, verify the type of variables…)
9. Time Complexity and space complexity theory before and after implementation..
10. Re-define the class structure if use IDE to do this problem(maintenance)

Evaluation:

Submissions will be assessed based on:

1. Correctness (40%): The algorithm must accurately find the MST for any provided graph.
2. Code Quality (30%): The code must be clear, commented, and adhere to standard naming conventions.
3. Efficiency (10%): The algorithm should perform with the optimal complexity of O(E log E).
4. Testing (10%): At least five JUnit tests must be passed, addressing a range of cases and potential edge conditions.
5. Documentation (10%): A README file must be included to explain design choices and any assumptions.

Submission Requirements:

1. All class source code files, including JUnit test files, should be submitted to a GitHub repository or Canvas by one group(3-4 people per group).
2. A README file documenting your approach and any assumptions.
3. A PDF file containing a graph representation (in matrix or list form) that was used for testing.

Deadline:

Note to Students:

Consider using version control systems like Git to manage and document the evolution of your code. This practice can help in maintaining a clear history of changes and assists in collaborative projects.