FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

DEPARTMENT OF COMMUNICATION TECHNOLOGY AND NETWORK

GROUP PROJECT

SEMESTER II 2023/2024

COURSE NAME : DESIGN AND ANALYSIS OF ALGORITHMS

COURSE CODE : CSC4202

GROUP : 6

PROJECT TITLE : STRATEGIC RELIEF SUPPLY IN DISASTER AREAS

LECTURER : DR. NUR ARZILAWATI BINTI MD YUNUS

NAME	MATRIC ID
1. MOHAMMAD FARHAN	211483
2. AFIQ FIKRI BIN MOHD ZAHIR	211323
3. MUHAMMAD ZIKRI BIN ZAKARIA	213118

Table of Content

1.0 Scenario Description

In the aftermath of a devastating earthquake in a city, the government needs to efficiently establish a network of routes to distribute relief supplies to various affected zones.

The city is divided into several zones, each needing supplies such as water, food, and medical kits. The relief supplies are stored in a central warehouse, and the goal is to quickly connect all zones with the minimal total travel time for the relief trucks.

To achieve this, the government aims to construct a network of roads that minimizes the total travel time while ensuring that every zone is reachable from the warehouse, either directly or indirectly.

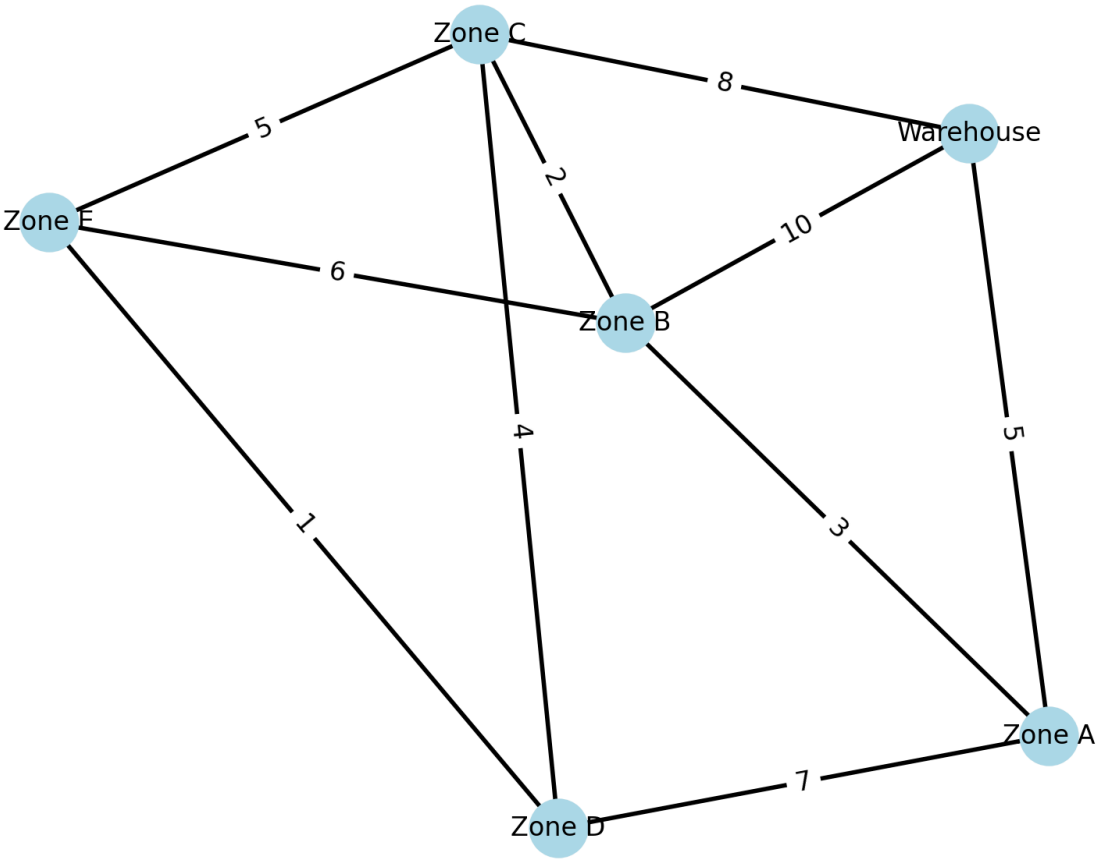


Figure 1: Map of the city and its delivery routes.

Here is the map of the city and its delivery routes:

- **Nodes** represent the central warehouse and the zones (Zone A, Zone B, Zone C, Zone D, Zone E).
- **Edges** represent the possible delivery routes between these points, with weights indicating the delivery times in minutes.

This visual representation helps to understand the layout of the city and plan the optimal distribution routes for relief supplies effectively.

2.0 Importance of Finding an Optimal Solution

Finding an optimal solution for this scenario is crucial for several reasons:

- **Timeliness**: Efficient distribution ensures that relief reaches the affected zones quickly, which is critical for saving lives and reducing suffering.

- **Resource Management**: Optimal use of available resources (trucks, supplies) ensures that no zone is left without aid while avoiding wastage or overstocking in any area.

- **Cost Efficiency**: Minimizing delivery times and optimizing routes can significantly reduce fuel and operational costs, allowing more resources to be allocated to relief efforts.

- **Coordination**: An optimal solution ensures better coordination among relief teams, avoiding duplication of efforts and ensuring comprehensive coverage of all affected zones.

3.0 Development of a Model for the Chosen Scenario

Data Type:

- **Graph Representation**: The city is represented as a graph where nodes are zones and edges are the roads between them with weights representing travel times.
- **Edge**: Each edge has a source zone, a destination zone, and a weight (travel time).

Objective Function:

- **Minimize Total Delivery Time**: The objective is to minimize the sum of the weights of the edges in the MST, ensuring all zones are connected with minimal total travel time.

Constraints:

- **Connectivity**: All zones must be connected to the central warehouse directly or indirectly.
- **Non-negative Weights**: Travel times (weights) are non-negative.
- **Graph is Connected**: There exists a path between any two zones in the city.

Examples and Requirements:

- **Zones**: A, B, C, D, E, Warehouse
- **Edges**: Representing roads between zones with weights (travel times)
- **Example Edge**: (Warehouse, A, 5), meaning 5 units of time to travel from the Warehouse to zone A.

4.0 Review of Greedy Algorithm as Solution Paradigm

Algorithm Comparison:

ALGORITHM	SUITABILITY	STRENGTHS	WEAKNESSES
Greedy Algorithms	Very High	● Simple and easy to implement ● Efficient in terms of time and space ● Directly applicable to pathfinding and shortest path problems	Non-optimal for some problems as it focuses on local optimization.
Dynamic Programming (DP)	Moderate	Suitable for problems with multiple optimal sub paths	Implementation can be complex
Divide and Conquer (DAC)	Low to Moderate	Useful for breaking down complex problems	May not handle interconnected paths efficiently
Sorting	Low	Simple and well understood	Does not address the pathfinding problem

Strengths:

1. Simplicity: Greedy algorithms are typically easy to understand and implement.
2. Efficiency: They usually run in polynomial time, making them suitable for large datasets.
3. Optimal for Certain Problems: For specific problems, such as minimum spanning tree or shortest path (Dijkstra's algorithm), greedy algorithms provide optimal solutions.
4. Local Optimization: Greedy algorithms make a series of choices, each of which looks best at the moment, often leading to a good solution quickly.

Weaknesses:

1. Non-Optimal for Some Problems: Greedy algorithms do not always produce the optimal solution, especially for problems requiring global optimization.
2. Local vs. Global: They focus on local optimization, which can sometimes lead to suboptimal global solutions.
3. Lack of Backtracking: Greedy algorithms do not backtrack or revise decisions, which might be necessary for optimal solutions in some cases.

Reason for Selection: It provides a straightforward and efficient approach to minimize delivery time and is relatively easy to implement and understand.

5.0 Algorithm Specification

The problem of efficiently distributing relief supplies to various affected zones in a city after a devastating earthquake is addressed using Prim's algorithm to find the Minimum Spanning Tree (MST). The goal is to minimize the total delivery time while ensuring that all zones receive the necessary supplies.

Selected Algorithm: Prim's Algorithm (Greedy Algorithm)

Rationale:

- Prim's Algorithm is chosen because it is well-suited for finding the Minimum Spanning Tree (MST) in a connected, undirected graph, ensuring that all zones are connected with the minimum total edge weight.

- Kruskal's Algorithm is another popular MST algorithm. However, Prim's algorithm is more efficient for dense graphs, as it can take advantage of a priority queue to efficiently find the next smallest edge.

- Dijkstra's Algorithm is useful for finding the shortest path from a single source to all other nodes but does not guarantee the minimal overall connection of all nodes (zones) like an MST algorithm.

- Bellman-Ford Algorithm is also for shortest paths and is not as efficient for MST problems, especially with non-negative weights.

6.0 Algorithm Design

Pseudocode for the Algorithm:

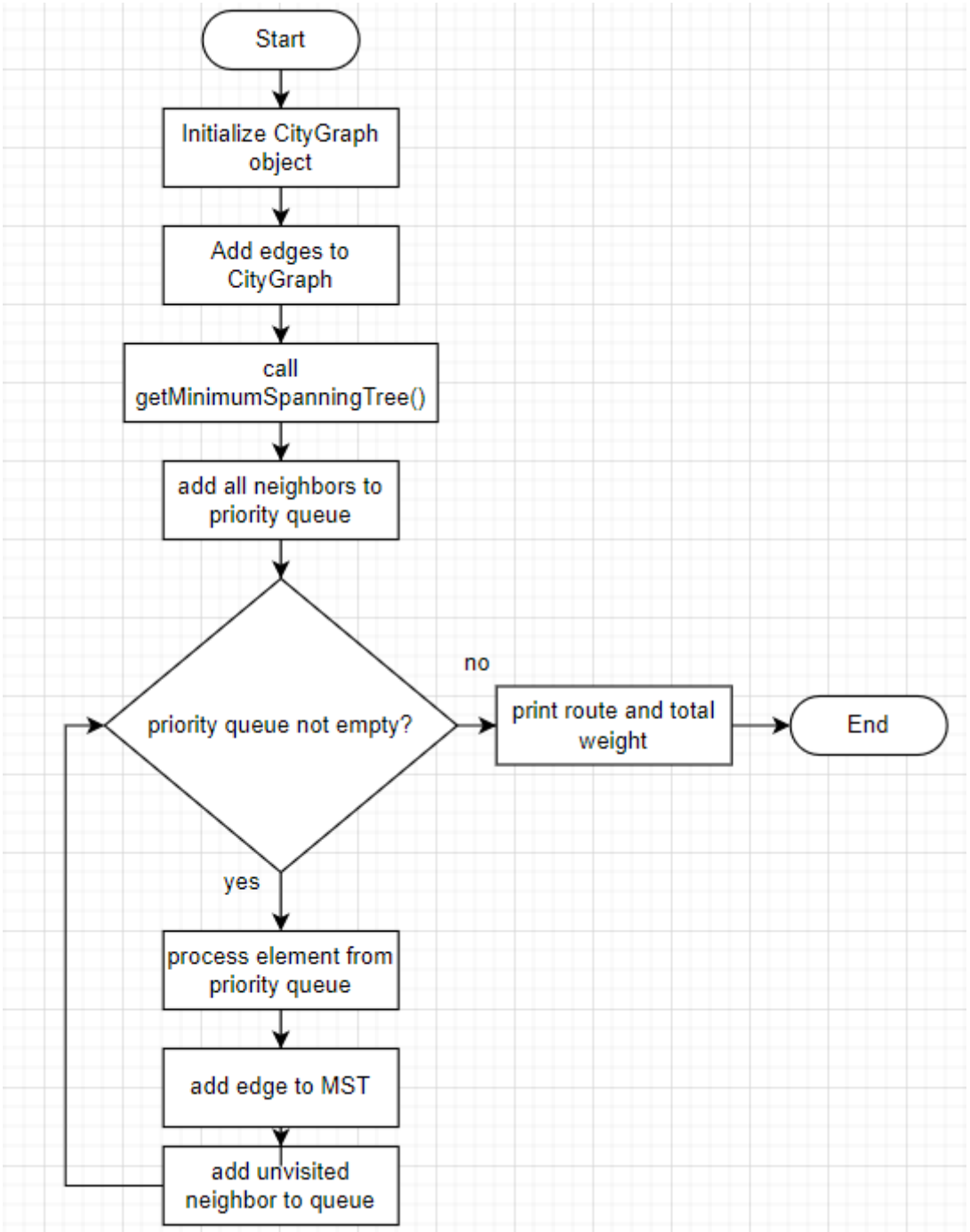
Initialization:

1. Start with an arbitrary node (in this case, the "Warehouse").
2. Initialize an empty list to store the Minimum Spanning Tree (MST) edges.
3. Use a priority queue (min-heap) to keep track of the edges with the smallest weights.
4. Maintain a set of visited nodes to avoid cycles.

Iterative Process:

1. Continuously select the edge with the smallest weight from the priority queue.
2. If the destination node of this edge has not been visited, add it to the MST and mark it as visited.
3. Add all edges from the newly visited node to the priority queue, provided the destination nodes have not been visited.
4. Repeat this process until all nodes are visited and the MST is complete.

Flowchart:


Start
Initialize CityGraph object
Add edges to CityGraph
call getMinimumSpanningTree()
add all neighbors to priority queue
priority queue not empty?
no
print route and total weight
End
yes
process element from priority queue
add edge to MST
add unvisited neighbor to queue

Function for Optimization:

1. **Priority Queue Management:**

   Optimizes the process of finding the smallest edge using a min-heap which ensures that each insertion and extraction operation is logarithmic in complexity (O(log E)), making the algorithm efficient.

2. **Iterative Approach:**

   The algorithm iteratively adds the smallest edge and updates the priority queue, ensuring optimal performance without the overhead of recursion.

Justification for the Lack of Recurrence:

Prim's algorithm does not require recursion because it is fundamentally an iterative algorithm. The process of continuously selecting the minimum weight edge and updating the priority queue lends itself well to an iterative approach.

1. **Priority Queue Management**:

   The priority queue is efficiently managed iteratively, where edges are added and removed based on their weights. This allows the algorithm to always pick the smallest edge without needing a recursive call stack to keep track of choices.

2. **Set of Visited Nodes**:

   The set of visited nodes ensures that each node is processed only once. Recursion would complicate this process by necessitating additional checks to ensure nodes are not revisited.

3. **Edge Selection Process**:

   Each iteration involves a simple loop that selects the next edge with the smallest weight. This loop can efficiently manage the selection process without the overhead of recursive calls.

7.0 Implementation in Java

Source Code:

```
import java.util.*;


public class ReliefDistribution {


    public static void main(String[] args) {
        CityGraph cityGraph = new CityGraph();
        cityGraph.addEdge("Warehouse", "A", 5);
        cityGraph.addEdge("Warehouse", "B", 10);
        cityGraph.addEdge("Warehouse", "C", 8);
        cityGraph.addEdge("A", "B", 3);
        cityGraph.addEdge("B", "C", 2);
        cityGraph.addEdge("C", "D", 4);
        cityGraph.addEdge("C", "E", 5);
        cityGraph.addEdge("D", "E", 1);


        String startNode = "Warehouse";
        List<Route> mst = getMinimumSpanningTree(cityGraph, startNode);


        System.out.println("Minimum Spanning Tree (MST) edges:");
        for (Route route : mst) {
            System.out.println(route.source + " - " + route.destination
+ " with weight " + route.weight);
        }


        int totalWeight = mst.stream().mapToInt(route ->
route.weight).sum();
        System.out.println("Total distance (time) to supply all zones:
" + totalWeight);
    }


    private static List<Route> getMinimumSpanningTree(CityGraph
cityGraph, String start) {
        List<Route> mst = new ArrayList<>();
```

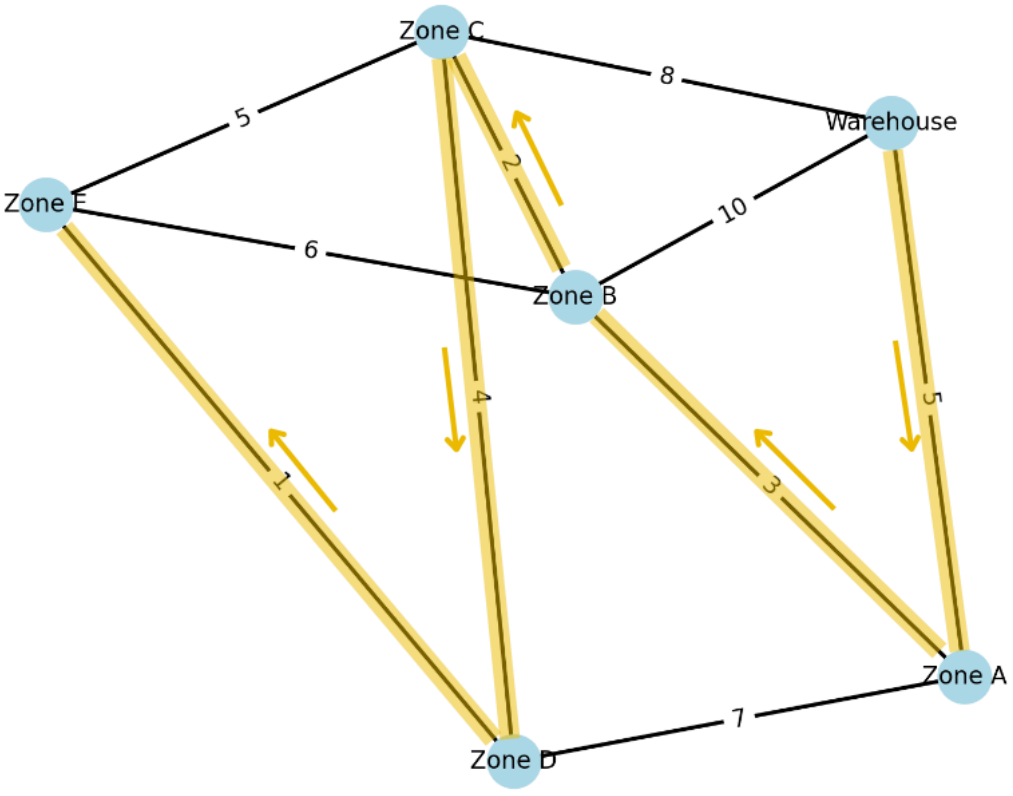
```
        Set<String> visited = new HashSet<>();
        PriorityQueue<Route> priorityQueue = new
PriorityQueue<>(Comparator.comparingInt(e -> e.weight));

        visited.add(start);
        priorityQueue.addAll(cityGraph.getNeighbors(start));

        while (!priorityQueue.isEmpty()) {
            Route route = priorityQueue.poll();
            if (visited.contains(route.destination)) {
                continue;
            }
            visited.add(route.destination);
            mst.add(route);

            for (Route neighbor :
cityGraph.getNeighbors(route.destination)) {
                if (!visited.contains(neighbor.destination)) {
                    priorityQueue.add(neighbor);
                }
            }
        }

        return mst;
    }
}

class CityGraph {
    private final Map<String, List<Route>> adjList = new HashMap<>();

    public void addEdge(String source, String destination, int weight)
{
        adjList.putIfAbsent(source, new ArrayList<>());
        adjList.putIfAbsent(destination, new ArrayList<>());
        adjList.get(source).add(new Route(source, destination,
weight));
```

```
        adjList.get(destination).add(new Route(destination, source,
weight)); // For undirected graph
    }

    public List<Route> getNeighbors(String zone) {
        return adjList.get(zone);
    }

    public Set<String> getZones() {
        return adjList.keySet();
    }
}

class Route {
    String source;
    String destination;
    int weight;

    Route(String source, String destination, int weight) {
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }
}
```

Sample Output:

```
\Users\ASUS\AppData\Roaming\Code\User\workspac
Minimum Spanning Tree (MST) edges:
Warehouse - A with weight 5
A - B with weight 3
B - C with weight 2
C - D with weight 4
D - E with weight 1
Total distance (time) to supply all zones: 15
```

Output Visualization:



8.0 Analysis of Algorithm Correctness and Time Complexity

Algorithm's Correctness Analysis:

To prove the correctness of Prim's algorithm, we need to demonstrate that the algorithm always produces a minimum spanning tree (MST) for any connected, undirected graph with non-negative weights.

Initialization:

- The algorithm begins with an arbitrary starting node (in this case, "Warehouse").
- It initializes the MST as an empty list and a priority queue to manage the edges based on their weights.

Maintaining Invariant:

- At each step, the algorithm maintains the invariant that the MST being built is always part of the global MST.
- The priority queue ensures that the smallest edge connecting the current MST to a new vertex is selected.
- By adding the smallest edge, the algorithm guarantees that no cycles are formed, maintaining the tree structure.

Greedy Choice Property:

- The greedy choice made at each step is optimal because adding the smallest edge available at each stage ensures that the MST grows in the most efficient manner.
- This is based on the cut property, which states that for any cut in the graph, the smallest edge crossing the cut is part of the MST.

Termination:

- The algorithm terminates when all vertices have been added to the MST.
- At this point, the MST contains exactly V−1 edges, where V is the number of vertices, ensuring that all vertices are connected with the minimum total edge weight.

Given these points, Prim's algorithm correctly produces an MST for any connected, undirected graph with non-negative edge weights.

Time Complexity Analysis:

To analyze the time complexity of our implementation, we need to consider the various operations performed during the execution of the algorithm. The main operations involve initializing the graph, managing the priority queue, and iterating through the edges to construct the Minimum Spanning Tree (MST).

Initialization of Graph

- **Adding Edges**: Each call to ‘addEdge” involves inserting an edge into the adjacency list. If there are E edges in the graph, this operation runs in O(E) time.

Priority Queue Operations

- **Inserting Edges**: Inserting an edge into a priority queue (min-heap) has a time complexity of O(logE).
- **Extracting Minimum Edge**: Extracting the minimum edge from a priority queue also has a time complexity of O(logE).

Iterative Edge Processing

- **Checking Visited Nodes**: Checking if a node has been visited is O(1) using a set.
- **Adding Edges to MST**: Adding an edge to the MST is O(1).
- **Adding Neighbors to Priority Queue**: Each insertion into the priority queue is O(logE).

Best Case Time Complexity: O(E log V)

- In the best-case scenario, the graph is already a minimum spanning tree (MST) or consists of disconnected components.
- Each edge added to the MST is the smallest among all available edges.
- The time complexity in this case is O(E log V), where E is the number of edges and V is the number of vertices.
- The algorithm selects the minimum-weight edge at each step, and the priority queue operations are optimized.

Average Case Time Complexity: O((V + E) log V)

- In the average case, the edges of the graph are randomly distributed, and the algorithm's performance depends on the density of edges.
- On average, each vertex has a constant number of edges adjacent to it.
- The time complexity in the average case is typically O((V + E) log V), where V is the number of vertices and E is the number of edges.
- Priority queue operations and updates (decrease-key operations) may vary, leading to an average logarithmic time complexity for each operation.

Worst Case Time Complexity: O((V + E) log V)

- In the worst-case scenario, the graph is densely connected, and each edge added to the MST results in multiple updates in the priority queue.
- The time complexity in the worst case is O((V + E) log V), where V is the number of vertices and E is the number of edges.
- This worst-case complexity arises when each edge insertion or update operation in the priority queue takes logarithmic time.

9.0 Conclusion

In conclusion, the application of Prim's algorithm for efficiently distributing relief supplies in a post-earthquake scenario proves to be a suitable choice. By leveraging a greedy approach, the algorithm ensures that all zones are connected through the Minimum Spanning Tree (MST) with minimal delivery times. This approach addresses critical needs such as timeliness, resource management, cost efficiency, and coordination, crucial in disaster relief operations. The algorithm's simplicity and efficiency make it well-suited for the task, providing a clear path to optimize relief efforts by minimizing total delivery time while maximizing coverage and resource utilization. Furthermore, the thorough analysis of Prim's algorithm confirms its correctness in constructing an MST and establishes its time complexity as manageable even for larger datasets, reinforcing its practical suitability for real-world applications in disaster response logistics.

References

Beheshtinia, M. A., Jozi, A., & Fathi, M. (2023). Optimizing disaster relief goods distribution and transportation: a mathematical model and metaheuristic algorithms. *Applied Mathematics in Science and Engineering*, *31*(1). https://doi.org/10.1080/27690911.2023.2252980.

Data Structures and Algorithms: Graph Algorithms. (2024). Umich.edu. https://www.eecs.umich.edu/courses/eecs380/ALG/prim.html#:~:text=The%20time%20complexity%20is%20O,O(E%20%2B%20logV).

GeeksforGeeks. (2024, February 7). *Greedy Algorithms*. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/greedy-algorithms/.

Greedy Algorithm. (2023). Programiz.com. https://www.programiz.com/dsa/greedy-algorithm.

Jochen Deuerlein, Gilbert, D., Abraham, E., & Piller, O. (2018). A Greedy Scheduling of Post-Disaster Response and Restoration using Pressure-Driven Models and Graph Segment Analysis. *Hal.science*, 14 p. https://hal.science/hal-01890932.

Wang, W.-C., Hsieh, M.-C., & Huang, C.-H. (2018). Applying Prim's Algorithm to Identify Isolated Areas for Natural Disaster Prevention and Protection. Engineering, 10(07), 417–431. https://doi.org/10.4236/eng.2018.107029.

Appendix

Initial Project Plan (week 10, submission date: 31 May 2024)

<table>
<tr><td>Group Name</td><td>Relief Squad</td></tr>
<tr><td>Members</td><td>
<table>
<tr><td>Name</td><td>Email</td><td>Phone number</td></tr>
<tr><td>MOHAMMAD FARHAN</td><td>211483@student.upm.edu.my</td><td>012-8037502</td></tr>
<tr><td>MUHAMMAD ZIKRI BIN ZAKARIA</td><td>213118@student.upm.edu.my</td><td>013-9709903</td></tr>
<tr><td>AFIQ FIKRI BIN MOHD ZAHIR</td><td>211323@student.upm.edu.my</td><td>011-109504449</td></tr>
</table>
</td></tr>
<tr><td>Problem scenario description</td><td>Distributing relief supplies from a central warehouse to multiple zones within a city that has been affected by a devastating earthquake.</td></tr>
<tr><td>Why it is important</td><td>1. Minimize Delivery Time
2. Ensure Comprehensive Coverage</td></tr>
<tr><td>Problem specification</td><td>1. Delivery Route Constraints
2. Immediate Supply Needs</td></tr>
<tr><td>Potential solutions</td><td>A solution that effectively plans and executes the distribution of relief supplies, supporting logistical strategies to achieve timely and effective relief operations across the affected city.</td></tr>
<tr><td>Sketch (framework, flow, interface)</td><td></td></tr>
</table>

Project Proposal Refinement (week 11, submission date: 7 June 2023)

<table>
<tr><td>Group Name</td><td>Relief Squad</td></tr>
<tr><td>Members</td><td>
<table>
<tr><th>Name</th><th>Role</th></tr>
<tr><td>Farhan</td><td>Create scenario</td></tr>
<tr><td>Zikri</td><td>Find suitable algorithm</td></tr>
<tr><td>Afiq</td><td>Identify problem</td></tr>
</table>
</td></tr>
<tr><td>Problem statement</td><td>Optimize delivery routes within a city graph, adhering strictly to available routes between nodes, while prioritizing critical supplies (water, medical kits) to zones with urgent needs.</td></tr>
<tr><td>Objectives</td><td>1. To find optimal path
2. To minimize time taken for delivery</td></tr>
<tr><td>Expected output</td><td>Determine the most efficient delivery routes.</td></tr>
<tr><td>Problem scenario description</td><td>Distributing relief supplies from a central warehouse to multiple zones within a city that has been affected by a devastating earthquake.</td></tr>
<tr><td>Why it is important</td><td>1. Minimize Delivery Time: Efficiently plan delivery routes to reduce the overall time required for supplies to reach all zones.
2. Ensure Comprehensive Coverage: Guarantee that every zone receives the exact quantities of supplies they need, tailored to their specific requirements.</td></tr>
<tr><td>Problem specification</td><td>1. Delivery Route Constraints: Deliveries must adhere to the available delivery routes between nodes (locations) in the city graph.
2. Immediate Supply Needs: Priority must be given to delivering critical supplies (e.g., water, medical kits) to zones with urgent needs.</td></tr>
<tr><td>Potential solutions</td><td>A solution that effectively plans and executes the distribution of relief supplies, supporting logistical strategies such Greedy Algorithm (Minimum Spanning Tree) to achieve timely and effective relief operations across the affected city.</td></tr>
<tr><td>Sketch (framework, flow, interface)</td><td></td></tr>
<tr><td>Methodology</td><td>
<table>
<tr><td>Milestone</td><td>Time</td></tr>
<tr><td><eg: scenario refinement></td><td>wk10</td></tr>
<tr><td><eg: find example solutions and suitable algorithms. Discuss in group why that solution and the example problems relate to the problem in the project></td><td>wk11</td></tr>
<tr><td><eg: edit the coding of the chosen problem and complete the coding. Debug></td><td>wk12</td></tr>
<tr><td><eg: conduct analysis of correctness and time complexity ></td><td>wk13</td></tr>
<tr><td><prepare online portfolio and presentation></td><td>wk14</td></tr>
</table>
</td></tr>
</table>

Project Progress (Week 10 – Week 14)

<table>
<tr><td>Milestone 1</td><td>Finish developing the program</td></tr>
<tr><td>Date (week)</td><td>14 JUNE 2024 (WEEK 12)</td></tr>
<tr><td>Description/ sketch</td><td>● Create suitable scenario
● Illustrate graph for the scenario
● Identify problem and challenges in the scenario created
● Develop java program</td></tr>
<tr><td>Role</td><td><table><tr><td>Farhan</td><td>Zikri</td><td>Afiq</td></tr><tr><td>create scenario and illustrate</td><td>develop program</td><td>identify problem and propose solution</td></tr></table></td></tr>
</table>

<table>
<tr><td>Milestone 2</td><td>Finalize the report</td></tr>
<tr><td>Date (Wk)</td><td>23 JUNE 2024 (WEEK 14)</td></tr>
<tr><td>Description/ sketch</td><td>● Analyze the output based on the experimental result
● Documenting and reporting</td></tr>
<tr><td>Role</td><td><table><tr><td>Farhan</td><td>Zikri</td><td>Afiq</td></tr><tr><td>Do reporting and analyze the program</td><td>Finishing program and documentation</td><td>Analyze output</td></tr></table></td></tr>
</table>