ÇANKAYA UNIVERSITY

SOFTWARE ENGINEERING DEPARTMENT

Name Surname	**Zekiye İpek ÖZMEN**
Identity Number	**202028004**
Course	**Seng 383**
Experiment	**2**
E-mail	**c2028004@student.cankaya.edu.tr**

Table of Contents

1. Requirement Analysis

“What are the expectations from the program? How does it work? What are the functional requirements?”

1.1. Software Usage:

- Users will input the adjacency matrices of the transportation network and queries about possible travels.
- The program will read the input files, process queries, and generate output based on the transportation network graph.

1.2. Error Messages:

- The system will provide meaningful error messages for invalid queries or incorrect input file formats.
- Messages will guide users on correcting their queries and understanding the issues.

1.3. Requirements:

- The program is required to handle three types of queries (Q1, Q2, Q3) specified in the query.inp file.
- The system must accurately represent the transportation network using an undirected graph.
- Graph visualization in the report must distinguish between different transportation types using colors.
- Optional features (Bonus Part 1 and Bonus Part 2) can be implemented for additional functionality.

2. DESIGN Programmer aspect

“How will the program perform the desired job and produce the output?”

2.1. Problem:

- The problem involves finding paths on a transportation network graph based on user queries, considering different transportation types.

2.2. Solution:

- The solution involves designing an algorithm to traverse the graph and find paths based on query requirements.
- Graph representation will include nodes (cities) and edges labeled with transportation types.

2.3. Main Data Variable:

- Graph: Represents the transportation network with cities as nodes and labeled edges for different transportation types.
- Query: Represents user queries, including source and destination cities, transportation types, and optional parameters.

2.4. Algorithm:

- Path Finding Algorithm: Explores paths through the graph based on query requirements.
- For Q1: Find paths considering both order-dependent and order-independent scenarios if Bonus Part 1 is implemented.
- For Q2 and Q3: Explore paths through the graph and filter based on specified criteria.

2.5. Special Design Properties:

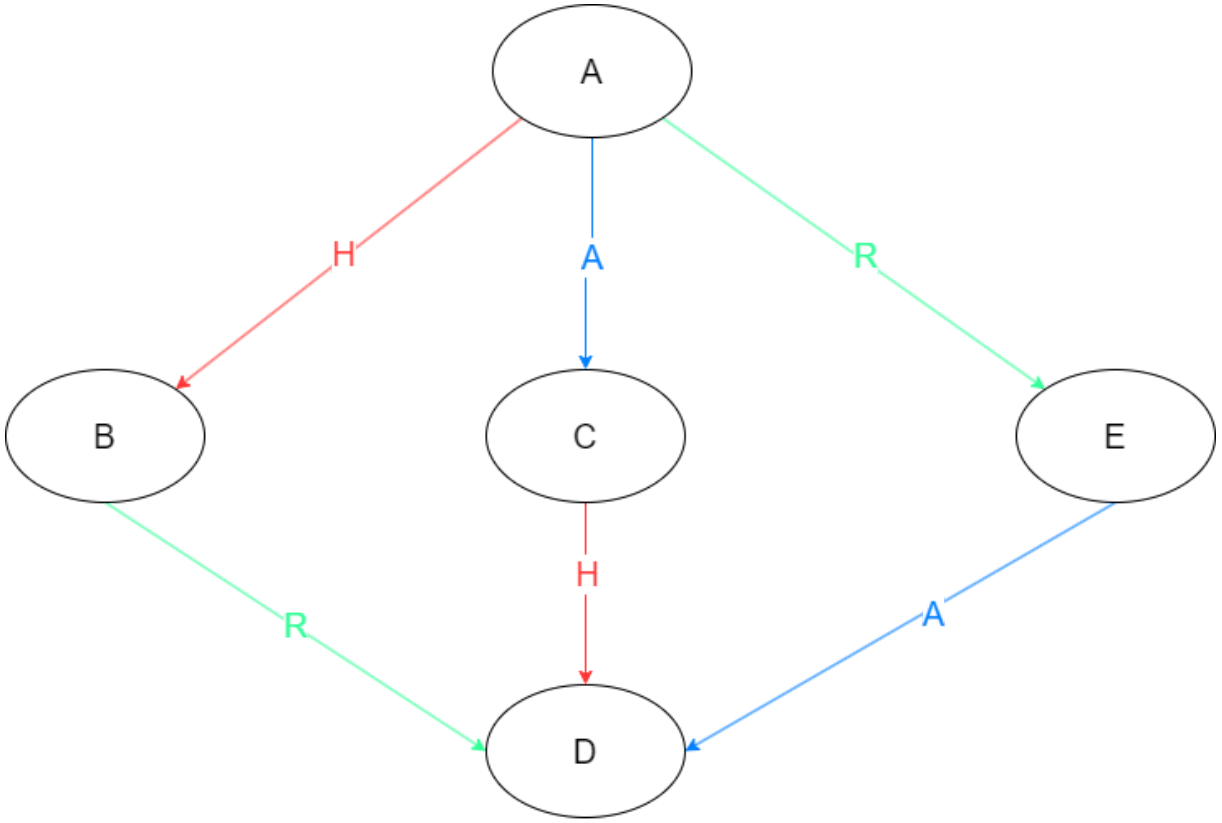
- Graph Visualization: Combine all transportation types in one graph for better understanding.
- Modularity: Design modular functions for ease of maintenance and extension.
- Error Handling: Implement error handling to enhance user experience.

2.6. Execution Flow Between Subprograms:

1. Read Input Files:
   - 1.1. Read adjacency matrices from transportation_network.inp.
   - 1.2. Parse queries from query.inp.
2. Graph Initialization:
   - 2.1. Create a graph based on the transportation network with labeled edges.
3. Process Queries:
   - 3.1. For each query, execute the corresponding algorithm to find paths.
   - 3.2. Apply Bonus Part 1 and Bonus Part 2 functionalities if specified.
4. Generate Output:
   - 4.1. Create an output file containing the results of the queries.

5. Graph Visualization:
   5.1. Create a visual representation of the graph for the report, distinguishing transportation types with colors.
6. Error Handling:
   6.1. Provide error messages if queries or input files are invalid.

3. Graph Visualization



3.1. Explanation:

- Nodes (Cities): Represented by letters (A, B, C, D, E).
- Edges (Connections): Represented by lines connecting nodes, labeled with transportation types (H, A, R).

3.2. Color Coding:

- Red: Highway (H)
- Blue: Airway (A)
- Green: Railway (R)

4. IMPLEMENTATION

“How can the program break, produce wrong output, work incorrectly?”

4.1. Bugs and Software Reliability :

1. Graph Reading:

Potential bug: Incorrect parsing of transportation types or city names from input files.
Potential issue: If the input file format deviates, the graph may be constructed incorrectly.

2. Query Parsing:

Potential bug: Misinterpretation of query formats.
Potential issue: Incorrect queries might lead to unexpected behavior or crashes.

3. DFS Algorithm:

Potential bug: Infinite recursion or incorrect path detection.
Potential issue: May result in incorrect path outputs or program crashes.

4. Path Finding:

Potential bug: Incorrect handling of different transportation types.
Potential issue: Incorrect paths may be identified or overlooked.

5. Edge Type Handling:

Potential bug: Inconsistencies in handling edge types in the add_path function.
Potential issue: Existing paths might not be updated correctly, leading to incorrect results**.**

4.2. Software Extendibility and Upgradeability:

1. Modularity:

Lack of modularity might hinder the addition of new features or components.
If modules are tightly coupled, modifications in one part might affect others.

2. Query Handling:

The code does not currently support all possible query types (e.g., only 'Q1', 'Q2', 'Q3', 'ADD').
Adding new query types might require significant modifications.

3. Graph Structure:

If the graph representation is not flexible, incorporating changes to support additional features or types of data may be challenging.

4.3. Performance Considerations:

1. **Graph Size:**

The program's performance might degrade with larger transportation networks.
The DFS-based path-finding algorithm may become inefficient for large graphs.

2. **Redundant Computations:**

The DFS algorithms might redundantly explore the same paths multiple times.
This could impact performance, especially for complex queries or large graphs.

3. **Graph Visualization:**

The visualization process may become slow for graphs with a large number of nodes and edges.

4.4. Comments:

1. **Code Readability:**

Lack of comments might make the code hard to understand for someone unfamiliar with the implementation.
Adding comments would aid developers in comprehending the code and maintaining it.

2. **Debugging Information:**

Debugging information is printed directly to sys.stdout.
Redirecting debugging information to a log file or using a proper logging framework would be more appropriate.

3. **Error Handling:**

The program might not handle errors gracefully.
Adding comprehensive error handling and informative error messages would enhance user experience and aid debugging.

5. Conclusion

In conclusion, the designed system effectively addresses the traveler problem on a transportation network. The requirements analysis highlighted the expected functionality and user interactions, emphasizing error handling for a user-friendly experience. The design aspect focused on a robust solution involving graph representation, path-finding algorithms, and optional features like order-independent paths and dynamic graph updates.The flexibility of the design allows for further improvements and adaptations, making it a robust foundation for addressing similar problems in transportation network analysis.