TED ÜNİVERSİTESİ

CMPE 224

PROGRAMMING ASSIGMENT 3

REPORT

Author:Melisa SUBAŞI

id: 22829169256

PROBLEM STATEMENT AND CODE DESIGN for Q1:

a)Problem Statement: *The task is to design a Java program for a new electricity distribution company. The program reads city coordinates from an input text file, constructs a graph representing the connections between cities, and then finds the paths that require the least number of electricity poles to cover all cities. The program calculates the Euclidean distance between cities and outputs the paths in ascending order of length, starting with the alphabetically first city. The program should use a graph data structure to represent the connections between cities and employ proper sorting techniques for the output.*

b)Top-down, stepwise refinement of the problem solution:

Graph Initialization:*The program initializes a graph to represent connections between cities, where each city is a node, and edges represent paths between cities.*

File Reading and City Representation:*City coordinates are read from the input text file, and City objects are created to store the name and coordinates (x, y) of each city.*

Euclidean Distance Calculation:*The program calculates the Euclidean distance between two cities using their coordinates.*

Path Finding:*The program iterates through all pairs of cities, calculates the distances, and stores the paths along with their lengths in a list.*

Path Sorting:*The paths are sorted first by length and then alphabetically, as per the specified requirements.*

Duplicate Path Removal:*Duplicate paths with the same length are removed to provide a unique set of paths.*

Output:*The program prints the paths in ascending order of length, starting with the alphabetically first city.*

c)Implementation and Functionality:

City Class:*The City class represents a city with a name and coordinates (x, y).*

Graph Construction:*The program constructs a graph using a list of City objects, where each city is connected to every other city.*

Distance Calculation:*Euclidean distance is calculated using the coordinates of two cities.*

Path Finding and Sorting:*The program finds all paths, calculates their lengths, and sorts them based on the specified criteria.*

Duplicate Path Removal:*Duplicate paths with the same length are removed to ensure a unique set of paths.*

Output Formatting*:The program prints the paths in the required format.*

d)Code Assessment:

The code effectively reads input from a file, constructs a graph, and finds paths based on the specified criteria.The usage of a graph data structure is appropriate for modeling city connections.The program successfully calculates distances, sorts paths, and removes duplicate paths.The code follows good coding practices, such as using proper variable names, encapsulation, and modularity.There is room for improvement in terms of code comments to enhance readability and understanding.The program can be further optimized for large datasets, as the current implementation has a time complexity of O(n^2).

Overall, the code provides a solid foundation for solving the given problem, with potential enhancements in terms of code documentation and optimization.

PROBLEM STATEMENT AND CODE DESIGN for Q2:

a)Problem Statement *The task is to design a Java program that finds the shortest route between a source city and a destination city while including specific cities to visit in between. The program reads a map of Turkey from an input text file, constructs an undirected graph with weights representing the distances between cities, and then determines the shortest route considering the order of cities to visit. The program should handle cases where the source or destination city is not present and provide meaningful output about the routes.*

b)Top-down, stepwise refinement of the problem solution:

Graph Initialization:*The program initializes an undirected graph to represent connections between cities, where each city is a node, and edges represent paths between cities with associated weights.*

File Reading and Graph Construction:*City connections and distances are read from the input text file, and the program constructs an undirected graph based on this information.*

Shortest Route Finding:*The program employs a modified Dijkstra's algorithm, utilizing a priority queue and considering the order of cities to visit. It iterates through the graph to find the shortest route from the source city to the destination city, including the specified cities in between.*

Output Formatting:*The program prints the routes in the format "City1 -> City2" and displays the total length of the route.*

c)Implementation and Functionality:

Graph Construction:*The CityGraph class represents an undirected graph with weighted edges between cities.*

File Reading:*The program reads city connections and distances from an input text file, constructing the graph accordingly.*

Shortest Route Algorithm:*The modified Dijkstra's algorithm finds the shortest route, considering the order of cities to visit.*

Output Formatting:*The program prints the routes in a clear and readable format, including the total length of the route.*

d)Code Assessment:

Code Quality:*The code is well-structured, readable, and follows good coding practices.Adequate comments have been added to enhance code understanding.Input validation is implemented to handle potential errors.*

Error Handling:*The program checks for cases where the source or destination city is not present, providing appropriate feedback.Input validation is performed to ensure the correctness of user inputs.*

Efficiency:*The program utilizes a priority queue and appropriate data structures for efficient route finding.Dijkstra's algorithm is employed for finding the shortest route, ensuring efficiency in larger datasets.*

Overall Assessment:

The code provides an effective solution to the problem of finding the shortest route in a graph with specified cities to visit. It is well-organized and demonstrates a solid understanding of graph algorithms. The implemented error handling and input validation contribute to the robustness of the program. Further enhancements could focus on additional optimization techniques for large datasets and providing more detailed information in error messages.