TED ÜNİVERSİTESİ

CMPE 224

PROGRAMMING ASSIGMENT 2

REPORT

Author:Melisa SUBAŞI

id: 22829169256

PROBLEM STATEMENT AND CODE DESIGN for Q1:

a)Problem Statement: *The objective of the program is to represent a flight network as a directed graph and perform operations to find cities reachable from a given city within a certain number of stops (hops). The program reads flight network data from a text file, constructs the graph, and then utilizes a depth-first search (DFS) method to identify reachable cities while considering the specified number of hops. However, the provided code does not produce the expected output for certain test cases.*

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

Graph Initialization:

The program initializes a directed graph representing the flight network using a HashMap to store city nodes and their corresponding destinations.

DFS with Hops:

The DFS method attempts to find cities reachable from a given source city within a specified number of hops. It uses recursion to explore neighboring cities, maintaining a set of visited cities to avoid revisiting.

User Input Handling and File Reading:

The program reads flight network data from a text file, constructs the graph, and determines the source city and number of hops based on the first line of the file.

c)Implementation and Functionality:

The question1 class contains methods for adding routes to the flight network graph (addRoute) and finding cities with a specified number of hops (findCitiesWithHops).

The findCitiesWithHops method uses a helper method (findCitiesWithHopsHelper) for recursive DFS traversal. The current implementation, however, fails to produce the correct results for certain test cases.

The main method initializes the flight network, reads data from the file, and determines the source city and number of hops.

d)Code Assessment:

The code that is given aims to address the issue of locating reachable cities in a predetermined number of hops. For several test instances, it presently yields inaccurate results, though. More work has to be done on the DFS implementation to guarantee precise traversal and reachable city identification.

The primary areas that want improvement are making sure that the source city and hop count are properly initialized, as well as verifying that the DFS traversal has the relevant criteria.

The usage of a graph representation is acceptable for the task, and the program's structure is evident. To solve the problems found, the DFS traversal's logic must be examined and modified.

PROBLEM STATEMENT AND CODE DESIGN for Q2:

a)Problem Statement: *The program analyzes a maze file to identify treasures and outputs the number of treasures along with sorted paths to each. The maze consists of walls ('+', '-', '|'), available paths (lowercase letters), and treasures ('E'). The goal is to find all reachable treasures and list paths to each.*

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

User Input Handling: *The program reads a maze file and extracts dimensions and maze structure.*

Maze Initialization: *The maze is initialized based on the file content.*

Treasure Identification*: The program uses Depth-First Search (DFS) to identify treasures and store paths.*

Output Formatting*: The program prints the number of treasures found and sorted paths.*

Error Handling*: The code checks for invalid maze file formats and handles errors appropriately.*

c)Implementation and Functionality:

Main Class: *Handles user input, initializes the maze, and performs DFS to find treasures.*

DFS Method: *Recursively explores the maze, identifies treasures, and stores paths.*

Output Format: *Prints the number of treasures and sorted paths as specified.*

Error Handling: *Properly handles invalid maze file formats and other potential errors.*

d)Code Assessment:

The code demonstrates an effective solution to the problem, successfully identifying treasures within a maze and generating sorted paths to each. Its structure is well-organized, employing clear classes and methods. The implementation handles user input accurately and incorporates error handling for potential issues, such as invalid maze file formats. The code has been rigorously tested with sample input, consistently producing the expected output. Notably, recent enhancements have been made to improve error handling, allowing for a more graceful response when no treasures are found. Additionally, the Depth-First Search (DFS) algorithm has been optimized to handle maze walls and out-of-bounds conditions more efficiently. Overall, the code showcases a reliable and well-structured solution to the specified problem, meeting the criteria of effective implementation and thorough testing.