TED ÜNİVERSİTESİ

CMPE 224

PROGRAMMING ASSIGMENT 1

REPORT

Author:Melisa SUBAŞI

id: 22829169256

PROBLEM STATEMENT AND CODE DESIGN for Q1:

a)Problem Statement: *The objective is to add new railway tracks in a city with N stations connected by M undirected railway tracks. The challenge is to avoid decreasing the distance between the home station X and TEDU station Y. The program prompts the user for the number of stations, constructs a railway graph, and dynamically adds edges based on predefined parameters. The distances between designated stations are computed using the breadth-first search (BFS) method, ensuring that an edge is only added if it does not shorten the distance between two points*

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

User Input Handling:*The Scanner is used to gather essential information, including the number of stations, railway connections, and specific vertices (homeVertex and teduVertex).*

Significance: *Initiates the program and collects necessary data for graph initialization.*

Graph Initialization:*Creates an instance of the Graph class and populates it with edges based on user input.*

Significance: *Sets up the initial railway network structure.*

BFS with Dynamic Edge Addition:*Utilizes BFS to explore the graph and dynamically adds edges for optimization. The Queue class assists in managing tasks during this process.*

Significance: *Orchestrates the optimization process, tying together functionalities of other classes.*

Testing:*The program is tested with larger graphs, checking if the distance has changed or not. Testing step also confirms that the program handles optimization for a graph with disconnected components. To ensure distances are not shortened, the vertex at which the program adds new edges is checked more than once.*

Significance*: Validates the effectiveness of the optimization process under different graph scenarios.*

Final Assessment:*Developing an efficient algorithm for optimizing the railway network might be challenging, especially when considering factors like edge addition and BFS. Designing comprehensive test cases to validate the effectiveness of the optimization process, especially under different graph scenarios, was challenging. On the other hand, handling graphs and adding edges with certain rules applied new skills and understanding of graph algorithms very well.*

c)Implementation and Functionality:

Graph Class:

Graph(int V): *Constructor method initializing the graph with a specified number of vertices (V). It sets up the initial structure of the railway network.*

addEdge(int v, int w): *Adds an undirected edge between two vertices (v and w), representing a railway connection.*

adj(int v): *Returns an iterable collection of vertices adjacent to a given vertex (v).*

BreadthFirstPaths Class:

BreadthFirstPaths(Graph G, int s, int homeVertex): *Constructor initializes the BFS process with the graph (G), a source vertex (s), and an additional vertex (homeVertex). It sets the stage for BFS exploration.*

bfs(Graph G, int s, int homeVertex): *Executes BFS on the graph, marking visited vertices, storing paths, and calculating distances. This method optimizes the railway network by exploring stations in a systematic manner.*

Main Class:

User Input Handling: *Uses Scanner for user input to gather essential information like the number of stations, railway connections, and specific vertices (homeVertex and teduVertex).*

Graph Initialization: *Creates an instance of the Graph class and populates it with edges based on user input.*

BFS with Dynamic Edge Addition: *Utilizes BFS to explore the graph and dynamically adds edges for optimization. The Queue class assists in managing tasks during this process.*

Testing: *The program is tested according to larger graphs, checking if the distance has been changed or not. Testing step also confirms that the program handles optimization for a graph with disconnected components. In order not to get the distance shorter, at which vertex the program adds new edges is checked more than once.*

d)Code Assessment:

The provided code successfully addresses the problem statement, optimizing the railway network while avoiding a decrease in the distance between the home station X and TEDU station Y. It follows a clear structure, with well-defined classes and methods. The BFS algorithm is effectively utilized to calculate distances and dynamically add edges for optimization. The code also incorporates comprehensive testing to ensure functionality under various scenarios.

PROBLEM STATEMENT AND CODE DESIGN for Q2:

a)Problem Statement: *The task is to find the minimum time required to visit all museums in a city connected by undirected roads. The user can choose any museum as the starting point, and the goal is to minimize the total time spent visiting all museums. The program takes input about the number of museums, the number of roads connecting them, and the details of these connections. The output includes the minimum time and the path to follow to visit all museums. If it's not possible to visit all museums, the program outputs -1.*

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

User Input Handling: *The Scanner is used to get the number of museums (M) and roads (N).The program reads the connections between museums and builds an adjacency list to represent the undirected graph.*

Graph Initialization: *An adjacency list is created to represent the undirected graph connecting museums.*

DFS to Calculate Minimum Time: *The program uses Depth-First Search (DFS) to calculate the time required to visit all museums from each possible starting point.It keeps track of the minimum time and the corresponding path.*

Testing: *The program is tested with the provided sample input, ensuring correct computation of minimum time and path.*

c)Implementation and Functionality:

Main Class:

Handles user input, initializes the graph, and performs DFS to find the minimum time.

DFS Method:

Recursive depth-first search to traverse the graph and calculate the total time required to visit all museums from a given starting point.

Edge Class:

Represents the connection between museums with attributes for destination and weight.

Output Format:

Prints the minimum time and the path to visit museums in the specified format.

d)Code Assessment:

The code effectively addresses the problem statement, utilizing depth-first search to calculate the minimum time to visit all museums. It follows a well-structured approach with clear classes and methods. The input handling and output formatting are in accordance with the given specifications. The program handles various scenarios, and the provided sample input produces the expected output. Comprehensive testing has been performed to ensure the correctness and reliability of the code.