CS2001 – Data Structures Semester Project

Food Delivery Routing Optimization

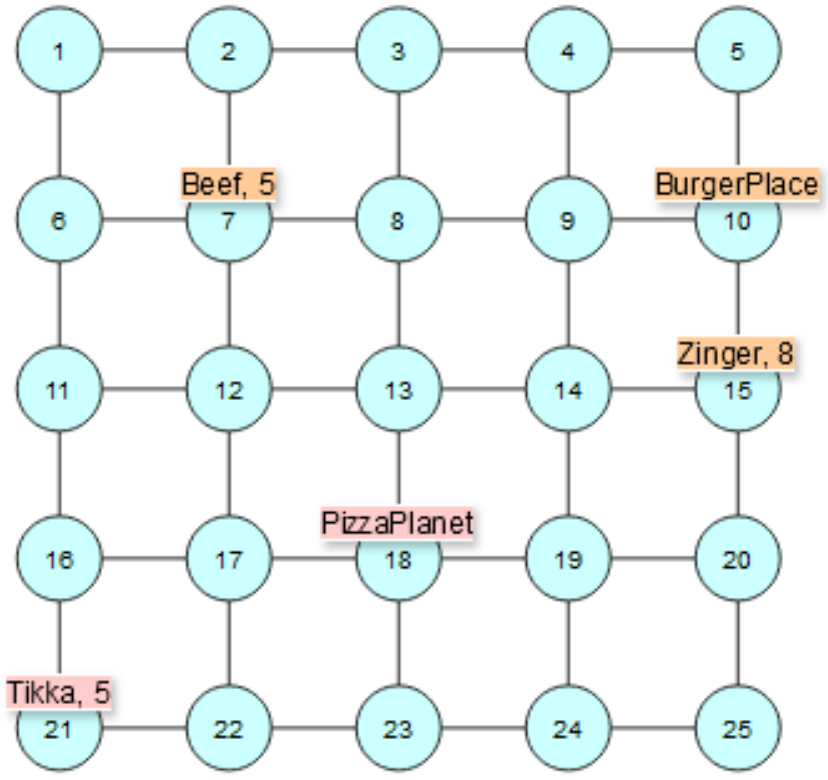
Due Date: Sunday, May 5th, 2024 11:59pm

Group Members: 3

Food Polar Bear, a delivery company operating in Grid City wants to optimize its route planning for delivering orders to customers efficiently. Grid City consists of multiple restaurants and customers scattered across a grid similar to Manhattan. The company needs to deliver orders from different restaurants to customers while considering various constraints, such as time windows for deliveries and minimizing travel time.

Suppose the company receives orders from two different restaurants: Burger Palace and Pizza Planet. Each restaurant has its own set of orders with specific delivery locations. The delivery personnel must plan their routes to cover the maximum number of deliveries in a day while adhering to the time constraints.

For instance, the delivery personnel start from Burger Palace and deliver orders to nearby customers. Then, they move to Pizza Planet and deliver orders from there. They optimize their routes to avoid backtracking and ensure timely deliveries. The total minimum time required to deliver all orders efficiently is calculated based on the route plan.



Instructions

- Represent the city map as an N x N grid where nodes should be numbered from 1 to N^2, going from left to right and top to bottom.
- Restaurants and customers are located at the nodes in a graph, and edges represent road connections.
- Each edge takes 1 unit time to travel.
- Consider time constraints for each delivery and plan routes accordingly to ensure timely deliveries.
- Optimize the routes to cover the maximum number of deliveries in a day while minimizing travel time.

Input

The input to the algorithm will be contained in a text file in the following order:

- The first line contains the number of test cases
- For each test case:
    - The first line contains grid size (N), number of riders (I), and number of restaurants (R)
    - For each restaurant:
        - The first line contains its name, its location, and the number of orders (O)
        - The next O lines contain the name, location, and delivery time limit of each order

Output

For each test case, the algorithm should output the total minimum time required to deliver all orders efficiently. For example, in the first test case shown below:

Rider 1: 10 (BurgerPalace) → 15 (Zinger) → 14 → 13 → 12 → 7 (Beef) = 5 time units
Rider 2: 18 (PizzaPlanet) → 17 → 16 → 21 (Tikka) = 3 time units
Total: 8 time units

Sample Input	**Sample Output**
2 5 2 2 BurgerPalace 10 2 Beef 7 5 (Order 1) Zinger 15 8 (Order 2) (Restaurant 1) PizzaPlanet 18 1 Tikka 21 5 (Restaurant 2) (Test Case 1) 5 2 1 CurryHouse 10 3 Chicken 2 7 ButterChicken 18 5 Biryani 15 2 (Test Case 2)	8 (Test Case 1) 8

Grading Rubric

Graph Creation (10 marks)

- The code correctly creates a graph representation of locations and their connections.
- The graph includes all locations and their corresponding edges based on the given input.
- The graph representation is appropriate for efficient traversal.

Input Handling (10 marks)

- The program correctly parses and handles input for the number of test cases, the number of restaurants, and orders.
- Input validation is implemented to handle potential errors or unexpected input formats.

Solution (30 marks)

- The students are able to explain and justify their proposed solution.
- The code correctly implements the algorithms and data structures required by the proposed solution.

Output Generation (10 marks)

- The code produces the correct output, displaying the minimum total time required to deliver all orders efficiently.
- The output format is clear and consistent across test cases.
- Output for both the provided example and additional test cases is accurate.

Code Organization and Readability (10 marks)

- The code is well-organized, with clear and meaningful variable names.
- Functions and logic are modular, promoting readability and maintainability.
- There are sufficient code comments explaining complex sections of the code.

Robustness and Error Handling (10 marks)

- The code handles edge cases and unexpected scenarios gracefully.
- There are appropriate error messages or feedback for users in case of invalid inputs or exceptional conditions.
- The program does not crash unexpectedly.

Efficiency (10 marks)

- The code is reasonably efficient in terms of time and space complexity.
- It avoids unnecessary computations or redundant data structures.
- The chosen algorithm scales well for larger inputs.

Work Division (10 marks)

- Every group member is able to justify their contribution in the project, particularly in code.