Project Assignment (16%)

Submission deadline: Thursday, **May 02, 2024 (23:59)**

Important Notes for Submission:

1- When submitting your work, you must use Blackboard. **No other means of submission, such as email, are accepted.**

2- Assignments must be completed **in pairs, with two students forming a team**. You are not allowed to discuss solutions, collaborate on writing solutions, or copy and paste solutions that are not your own. It is your responsibility to ensure that your solution is original and not retrieved by others.

3- If plagiarism is detected, you will receive a grade of zero (0) for the entire assessment.

4- Submit your work according to the instructions provided below prior to the deadline.

5- Along with the Word submission file, you must submit the Java source files for the project. Put both the Word file and the Java source files in a folder named ***Project_QUID1_QUID2***. Then, compress this folder and submit it.

6- In the Word document, include some screenshots of your program's input/output.

7- If you have any questions, please contact the Teaching Assistant.

Programming Project Assignment: Elevator Optimization

Objective: Design and implement an elevator optimization program in Java using two different algorithm design techniques: Dynamic Programming, and Greedy Algorithms. The goal is to minimize the total number of floors people have to walk either up or down, while respecting the constraint of making at most k stops on any given run.

Problem Description: Assume that you work in a very tall building with a slow elevator. The elevator is frustratingly interrupted when people press buttons for many neighboring floors during a trip. Encountering interruptions during an upward journey becomes especially frustrating when individuals press buttons for several neighboring floors, like 13, 14, and 15. The ascent is repeatedly interrupted three times, once at each of these floors. A more considerate approach would be for those passengers to collectively agree to press only the button for floor 14. Subsequently, individuals on floors 13 and 15 could opt to use the stairs for a single floor, taking advantage of the opportunity to walk. We presume that the cost of ascending a flight of stairs is equivalent to descending – considering the health benefits of exercise. To resolve ties between solutions with equal costs, management suggests prioritizing elevator stops at the lowest floor possible, as this consumes less electricity. Importantly, the elevator is not obligated to stop precisely at the floors specified by riders; for instance, if riders indicate floors 27 and 29, the elevator can choose to stop at floor 28 instead.

Your task is to optimize the elevator's stops to minimize the total walking distance for the riders.

Specifications:

1. Riders enter their intended destinations at the beginning of the trip.
2. The elevator decides which floors to stop at along the way, limiting to at most k stops on any given run.
3. The goal is to minimize the total number of floors people have to walk either up or down.
4. Break ties among equal-cost solutions by giving preference to stopping the elevator at the lowest floor possible.
5. The elevator is smart enough to know how many people want to get off at each floor.
6. The elevator does not necessarily have to stop at one of the floors the riders specified; it can choose a nearby floor instead.

Implementation Guidelines:

1. **Dynamic Programming:**
    - Formulate the elevator optimization problem with overlapping subproblems and optimal substructure.
    - Design a dynamic programming algorithm to efficiently solve the problem and minimize walking distance.

2. **Greedy Algorithm:**

    - Devise a greedy algorithm that makes locally optimal choices at each step to minimize the total walking distance.

    - Consider the tie-breaking rule and how it influences the elevator's behavior.

Deliverables:

1. Two Java programs implementing the elevator optimization algorithm for each of the specified techniques.

2. A report documenting the design choices, algorithms' complexity analysis, and evaluation of the implemented solutions.

3. Conduct comparative study using different test cases and demonstrate the program's functionality, including examples that showcase the tie-breaking rule and optimization results.

Additional Notes:

- Use appropriate data structures and object-oriented principles in your implementation.

- Clearly, comment your code for better understanding.

- Provide insights into the time and space complexity of each algorithm.

Submission: Submit your Java programs, report, and any additional documentation as a compressed archive (e.g., ZIP) via Blackboard.

Due Date: 02 May 2024, Thursday, at 23:59

Note: This is a team work and it's essential that each team consists of exactly two students – neither more nor fewer.