

IFT2015-A25: HW2

October 5, 2025

General instructions

- **Due date: Monday, October 20th at 23:59.**
- This assignment is to be completed individually or in pairs.
- No plagiarism will be accepted.
- It is possible that clarifications or modifications to the instructions will be sent by email and posted on Studium.
- Each day late in submitting your work will incur a penalty of 5 points.
- All code must be written in Java.
- You must submit the required Java files (`Q1.java`, `Q2.java`).
- **If you are working as a team:** one person submits the required files and the **other** submits only a `.txt` file with their **partner's name**.
- Code evaluation may be automated; make sure you have **the same function signatures**.
- You may use `java.util.*` for both problems.
- You are allowed to create as many helper functions as you wish.
- **Don't forget to add comments to your implementations. Undocumented code will be penalized.**
- For any questions regarding HW2, please post them on the "TP2 Q&A" forum on Studium.
- Good luck!

Problem 1 (3pts)

You are the coordinator for a new book distribution event. At this event, each participant requests exactly one book. There are n participants, numbered from 1 to n . The book that participants i requests is numbered b_i (where $1 \leq b_i \leq n$). Multiple participants may request the same book.

The distributor will prepare exactly the right number of each book as requested, but the sequence in which the books are handed out is specified by another sequence s_1, s_2, \dots, s_n . Whenever a book is handed out, it should be given to the customer with the smallest number who is waiting for that book.

This distribution method can lead to disappointment. If a participant notices that someone who was originally behind them receives their book first, that participant's anger value increases by 1. To analyze improvements, please write a program in `Q1.java` that, **using the queue data structure**,

1. Prints the participant number who receives each book in the actual distribution sequence.
2. Prints the total anger value of all participants.

Example

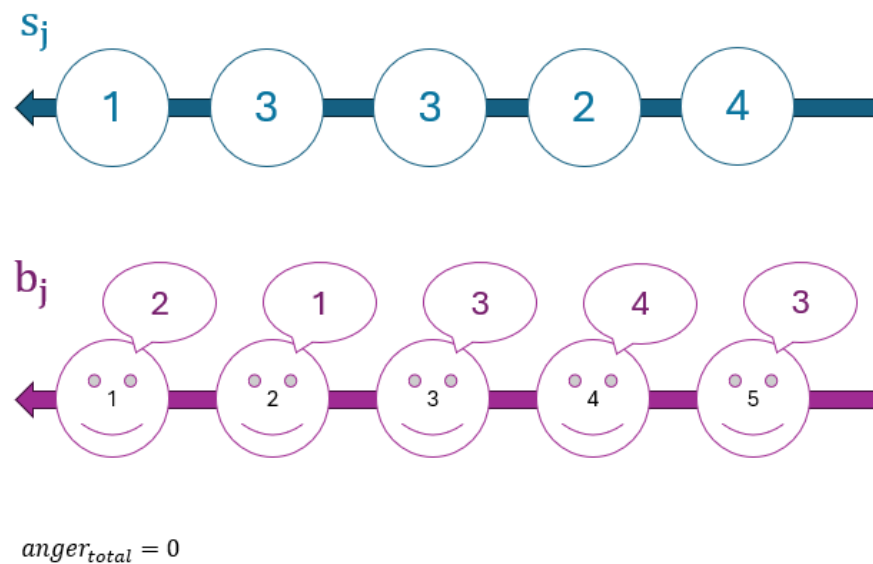


Figure 1: There are 5 participants, and they each requested books 2, 1, 3, 4, and 3, respectively. The books are handed out in the order 1, 3, 3, 2, 4.

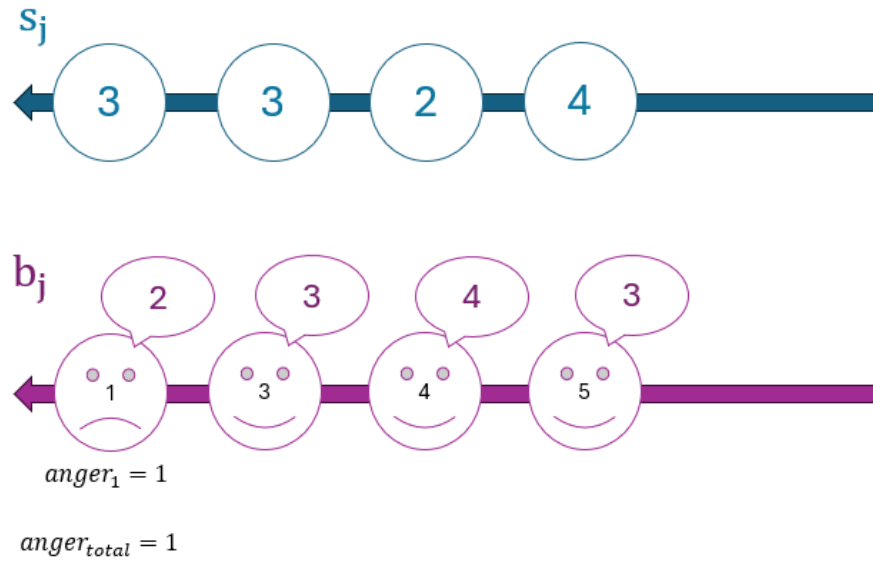


Figure 2: Since participant 2 requested book 1, they received their book before participant 1. The anger value of participant 1 increases by 1.

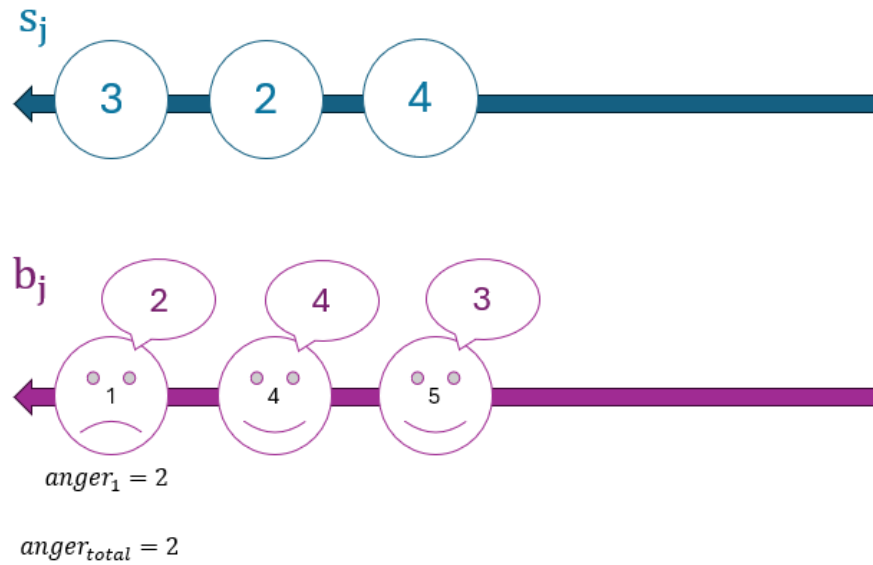


Figure 3: The next book handed out is book number 3. Participant 3 received their book. Since participant 1 hasn't received their requested book yet, their anger value increases by 1.

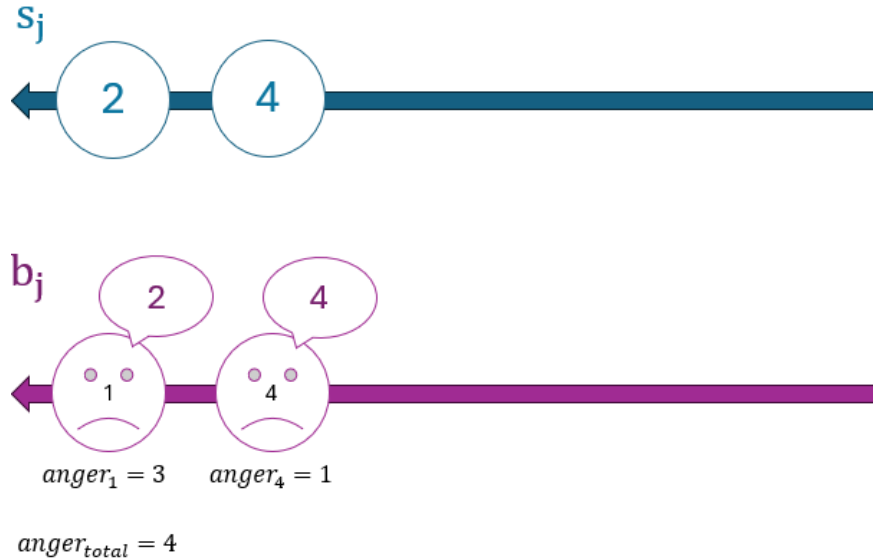


Figure 4: Participant 5 received their book 3. Since participants 1 and 4 haven't received their book, their anger value increases by 1.

The distribution continues until the queue is empty. The system prints the results:

```
participant served: 2 3 5 1 4
total anger: 4
```

Problem 2 (7pts)

Imagine you are designing an Employee Hierarchy Management System for a company. Each employee has a unique name and a level i that indicates their position within the organization's structure. The highest position is level 1, and only one employee in this rank. All other employees at level $i > 1$ work under an employee at level $i - 1$, who serves as their immediate boss. Your task is to manage this hierarchy and complete the methods in `Q2.java` to accomplish the following:

1. `void addEmployee(String name, int level, String bossName)`

- Adds a new employee to the company hierarchy. If the employee is at level 1, they are set as the root of the hierarchy. Otherwise, they report to an existing employee specified by their boss's name.
- Example: Suppose there are five employees:

```
addEmployee("Claude", 1, null); // Claude is the root boss at level 1.
addEmployee("Bob", 2, "Claude"); // Bob reports to Claude.
addEmployee("Elaine", 2, "Claude"); // Elaine also reports to Claude.
addEmployee("Alice", 3, "Bob"); // Alice reports to Bob.
```

```
addEmployee("David", 3, "Elaine"); // David reports to Elaine.
```

2. `void printChart(Node root)`

- Print the entire hierarchy in an indented format, **recursively**. Each direct report is prefixed by `|--` (one space) and indented further by `|` (three spaces) according to their depth in the tree.
- Example: Using the example in (1), `printChart(root)` prints

```
Claude
|-- Bob
|   |-- Alice
|-- Elaine
    |-- David
```

3. `void move(String employee, String boss)`

- Move an employee to report to a new boss.
- Example: `move("Alice", "Elaine")` results in Elaine becoming the immediate boss to Alice. `printChart(root)` prints

```
Claude
|-- Bob
|-- Elaine
    |-- David
    |-- Alice
```

4. `void remove(String employee, String boss)`

- Remove an employee from the hierarchy, reassigning their direct reports to a new boss.
- Example: Using the example in (1), `remove("Bob", "Elaine")` removes Bob, and Alice now reports to Elaine. `printChart(root)` prints

```
Claude
|-- Elaine
    |-- David
    |-- Alice
```

5. `void printAllBoss(String employee)`

- Print the chain of supervisors from the given employee up to the root boss.

- Example: Using the example in (1), `printAllBoss("Alice")` prints Alice -> Bob -> Claude.

6. `void printLevels(Node root)`

- Print all employees at each level of hierarchy using **queue**. Specify the level.
- Example: Using the example in (1), `printLevels(root)` prints

1: [Claude]
2: [Bob, Elaine]
3: [Alice, David]

7. `String lowestCommonBoss(String e1, String e2)`

- Return the name of the lowest common boss between two employees.
- Example: Using the example in (1), `lowestCommonBoss("Alice", "David")` returns Claude.