Assignment #4 – CSC226

Instructor: Navid Mahdian

Flight Allocation Problem

Overview

An airline operates up to 26 different flight routes, labeled with capital letters `A` through `Z`. Each route represents a specific trip (e.g., `A` might be Vancouver → Calgary). On a given day, multiple flights for the same route may be requested. *Each individual flight requires exactly one pilot.*

The airline has exactly 10 pilots, labeled `0` through `9`, but not all pilots are qualified to fly every route. For each day, the airline publishes a flight schedule specifying:

- The number of flights requested for each route, and
- The pilots who are qualified for each route.

The task is to determine whether the requested flights can all be covered under these constraints:

- Each flight (for each request) must be assigned *exactly one pilot.*
- Each pilot can serve at most one flight per day.
- A pilot may only be assigned to a flight if that pilot is explicitly listed as qualified for its route.

If a valid assignment exists, your program must output one such assignment. If no valid assignment exists, it should output a failure indicator.

Input Format

The input contains multiple *days* of flight data. Each day's schedule ends with a blank line, and the entire input ends at the end-of-file marker.

Daily Flight Schedule Lines: Each line corresponds to requests for a single route and follows this structure:

1. One uppercase letter `(A--Z)`, the flight route.
2. One digit `(1--9)`, the number of flights requested that day for that route.
3. A space character.
4. One or more distinct digits `(0--9)`, indicating which pilots are qualified for this route.
5. A semicolon `;`.

For example:

```
A4 01234;
```

means:

- Route `A` must be flown 4 times today,
- Pilots `0, 1, 2, 3, 4` are qualified for route `A`.

After one or more such lines for a given day, a blank line follows. Then, if more days exist, the next day's schedule begins. The input terminates at end-of-file.

Output Format

For each day's schedule, output exactly one of the following:

- A 10-character string, where each position `(0--9)` corresponds to a pilot `(0--9)`. If a pilot is assigned to a route, place that route's letter in their position. If a pilot is unassigned, place an underscore _.

Or

- A single character `!` if no valid assignment can be made.

Sample Input

```
A4 01234;
Q1 5;
P4 56789;

A4 01234;
Q1 5;
P5 56789;
```

Sample Output

```
AAAA_QPPPP
!
```

Explanation of the Sample

Day 1

- Route `A` requires 4 flights, with pilots {0,1,2,3,4} qualified.
- Route `Q` requires 1 flight, only pilot {5} is qualified.
- Route `P` requires 4 flights, pilots {5,6,7,8,9} are qualified.

A valid assignment:

`AAAA_QPPPP`

where pilots 0,1,2,3 fly route `A`, pilot 5 flies route `Q`, pilots 6,7,8,9 fly route `P`, and pilot 4 is unassigned (_).

Day 2

- Route `A` requires 4 flights ({0,1,2,3,4}),
- Route `Q` requires 1 flight ({5}),
- Route `P` requires 5 flights ({5,6,7,8,9}).

Since `P` already needs 5 separate pilots, yet pilot 5 is also needed for `Q`, there is no way to satisfy all requests. Hence, the output is `!`.

Base Code Information

The provided base code is the same for both **C++ and Java**:

- The input is automatically parsed and stored in `flightPilotPairings`, which contains the parsed flight and pilot data.
- The function `solveAllocationProblem()` is provided as a placeholder where students must implement **Edmonds-Karp** for **maximum flow**.

Solution Approach: Edmonds-Karp Maximum Flow

You must model this as a bipartite matching problem using a flow network and then implement the **Edmonds-Karp** algorithm. The graph, and how you manage the flow, should be set up so that:

- Each flight can be matched to *exactly one* qualified pilot.
- Each pilot can be matched to *at most one* flight.

Assignment Instructions

- **Base code** in both **C++ and Java** is provided.
- You may write your solution in either **C++** or **Java**.
- Your code must compile; a program that does not compile will receive a score of 0.
- **No use of generative AI** is permitted. All code must be your own.
- You must ensure that your program **adheres to the given input and output format**.
- **Test your code** thoroughly before submitting; more comprehensive test data will be used to evaluate your solution.
- You may use the **algs4** library in your implementation. Its use is allowed but not required.
- You must submit a single **ZIP file named** `a4.zip` containing:
  - A **PDF file** named `a4.pdf` explaining your approach.
  - Your completed code file, named `Main.java` or `Main.cpp`.

Submission Structure

Your submission must follow this structure:

```
a4.zip
 |-- a4.pdf
 |-- Main.java   (if using Java)
 |-- Main.cpp    (if using C++)
```

Submissions that do not follow the specified naming convention will not be graded.