Stable Matching

Thore Husfeldt and Riko Jacob

August 22, 2015

Description

Implement Gale and Shapley's deferred acceptance algorithm for stable matching and run it on several test inputs.

Files

The files for this exercise are on LearnIT.

Input

A typical input file is shown below. The first zero or more lines start with "#" and are ignored. Then comes a line of the form "$n = int$" (no whitespace around the equality sign) defining n. The following $2n$ lines describe $m_1, w_1, \ldots, m_n, w_n$, in that order. Every line starts with an identifying integer id, starting with 1, followed by a single space, followed by a nonempty string name of non-whitespace characters (such as letters, digits, punctuation). Odd-numbered lines are men, even-numbered lines are women. Then follows a blank line. Finally, there is a line for every id (in order) that describes a preference list. The line starts with id, followed by a colon and a single space. Then follows a permutation of the ids of the opposite gender, separated by a single space.

```
# Stable matching instance based on CBS's show "The Big Bang Theory"
#
n=4
1 Sheldon
2 Amy
3 Rajesh
4 Penny
5 Howard
6 Bernadette
7 Leonard
8 Emily

1: 2 4 6 8
2: 1 7 5 3
3: 6 4 2 8
4: 7 3 1 5
5: 6 4 8 2
```

```
6: 5 3 7 1
7: 4 8 6 2
8: 7 5 1 3
```

Output

The expected output contains one line for every matching. Every line contains the name of the man, followed by space, two hyphens, space, and the name of the matched woman. Example:

```
Sheldon -- Amy
Rajesh -- Emily
Howard -- Bernadette
Leonard -- Penny
```

Requirements

Language Java.

Infrastructure Your program has to take its input either from standard input (STDIN) or from a specified file. It has to write to standard output (STDOUT). Assuming all the files are in the right place, we must be able to run the program from the command line with at least one of the following two commands:

```
% java GS < sm-kt-p-4.in
% java GS sm-kt-p-4.in
```

(Your're welcome to write your code in Eclipse or whatever rocks your boat. But the above has to work.)

Correctness The course data file directory (linked from the course home page) contains a number of pairs of files called "sm-*.in" and "sm-*.out" that contain matching input and expected output. Your solution must be consistent with these files.

Solution quality

Minimal, acceptable solution Implement the Gale–Shapley algorithm so that it works correctly. Especially, your program shall give the correct answer on all test cases, except for the two big files. Write code that is as short and clear and crisp as possible. This solution will earn you full credit for this lab.

Better solution Make your algorithm run in time $O(n^2)$. Especially, that means that you must be able to handle a woman's comparison of

two male ranks in constant time. Section [KT, 2.3] explains how to do this; the modification to your code should be minimal – it didn't cost me a single line of extra code.

Questions

What about coding paradigms? I don't much care. You can follow the book and make an array-based solution, where people are basically integers that are used as indices into various arrays. You can also try for an object-oriented approach where people are objects, and the data structures use Java's collections package. I've done both.

Style? Your code should be as neat as you can make it. Don't declare variables or methods that you never use. What about documentation? I don't much care. I'd much rather have your code be as short and sweet as possible, instead of full of comments. Especially, don't waste time explaining the algorithm. My largest solution doesn't use much more than 100 lines of code, and neither should yours.

Deliverables

1. The source code for your implementation
2. A report in PDF. Use the report skeleton in the `doc` directory.

Tips

- Algorithmically, this is supposed to be an easy exercise. The algorithm is simple and spelled out in detail on p. 6 in the book, and many implementation issues are addressed in section 2.3. However, what is probably going to take you a lot of time is reading the input, especially if you haven't done that kind of thing before. I suggest the following: First, hard-code the input, in a convenient data structure, while you get the algorithm to work. This will allow you to change your mind about how the data structures should look. Only when everything seems to look OK, read (and parse!) your input from standard input.
- Learn `java.util.Scanner`. You want something like:
  `Scanner SC = new Scanner(System.in)`
- When debugging, don't throw away your test cases! Instead, make additional input–output file pairs. Write another program that runs your Gale–Shapley implementation on all test files (both the ones I have made and those you may have produced during debugging) and checks them against the corresponding output

files. This requires nontrivial file operations (opening matching filenames, reading directories, comparing files), and Java may not be the best language to do that in. (You could learn Perl or shell scripting.)

- What annoyed me most about input are the indices. In my array-based solution, after changing my minds a million times, I settled for indexing the men $0, \ldots, n-1$ and the women $0, \ldots, n-1$. This made the parsing stage slightly more complicated, because the input file uses a different convention, but cleaned up the rest of my code a lot.

FILE	n	Description
sm-friends	3	Based on the popular US TV show Friends
sm-illiad	62	Based on Homer's Iliad
sm-kt-p-*	2	Examples in [KT] on page 4 and 5
sm-random-*	5, 50, 500	Random instances of various sizes
sm-worst-*	5, 50, 500	Worst case instances of various sizes