Stable Matching Report

Bergar Simonsen, Dan Vasile, Aleksandra Korach, Tase Comboeanu

September 2, 2015

Results

Our implementation produces the expected input on all input files. To verify that the algorithm runs correctly, we have implemented a small test tool which is described in the implementation section.

On input `sm-bbt-in.txt` we produce the following matches:

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

And on input `sm-friends-in.txt` we produce the following matches:

- Chandler – Monica
- Joey – Phoebe
- Ross – Rachel

Implementation details

Men and women are represented by integers, and are indexed from 0 to $n-1$. Their preferences are stored in two respective two-dimensional arrays of integers.

In order to ensure the constant running time of each iteration, which is required to make the whole algorithm run in time $O(n^2)$, we utilised a few additional structures described in Section 2.3 of Kleinberg and Tardos, *Algorithms Design*, Addison–Wesley 2005. We can find a free man who has not proposed to every woman in time $O(1)$, because we store them in an *ArrayList* from the Java Collections library.

Identification of the highest-ranked woman to whom a given man has not yet proposed also happens in time $O(1)$, thanks to an array of integers that for every man keeps track of the index of another entry on their preference list.

Moreover, we have another array of integers keeping track of the current partner of every woman. This allows us to check if a given woman is engaged and identify her current partner in time $O(1)$ as well.

Finally, to decide in time $O(1)$ whether a woman w prefers a man m over m', we keep an invert of womens' preference list (two-dimensional array), which is indexed by men and stores their ranks.

With these data structures our implementation runs in time $O(n^2)$ on inputs with n men and n women.

Test tool

In order to verify the correctness of the algorithm, we needed some way to test the results, i.e., compare our results to the expected result files handed out with the assignment. Since our own algorithm doesn't produce the output in the same order as the content of the expected output, a simple comparision of the two files (e.g., using diff) will not work. Therefore we have implemented our own test tool.

Our test tool is very simple. It reads two files, `file1` is the output from our own algorithm, and `file2` is the expected output. These files are then read line by line and added to a list `list1` and `list2` respectively. The content of these two files are then added to a HashSet (`resultSet`). Since a HashSet cannot contain duplicates, we can verify that the two files are identical if the following equation holds:

$$list1.length == list2.length == resultSet.length$$