DT8014 Algorithms Exercise (2014)

Week 1 – Stable Matching

Roland Philippsen

November 2, 2014

Imagine that there are N students and N projects proposed by local companies. Each student must do one project, and each project must be done by one student. Such a one-to-one assignment is called a **matching**. In order to be as fair as possible in assigning students to projects, it has been decided to proceed as follows.

1. Each company talks with every student.
2. Each student lists all projects by preference, from their most to least favorite.
3. The companies do the same, listing the students by decreasing preference.
4. A professor receives all lists and figures out a so-called **stable matching**. This means that no student and company would rather be paired with each other, rather than remain with the partners they have been assigned to.

The concept of stable matching is rather subtle, so make sure you understand it. For example, assume you were paired with your second choice. Your favorite company was thus paired with someone else. If that other student comes before you in the company's preference list, then the pairing is still stable, because your favorite company has no reason to pair with you instead. If, however, your favorite company would *also* prefer you over their currently assigned student, then the overall match would be unstable.

In other words, if there exist even a single case where two non-paired parties would *both* rather be with each other than with their current partners, we have an unstable matching.

1. If you are given the preference lists and a suggested matching, how would you determine whether it is stable?
2. How does your stability-checking algorithm scale with N?
3. Given the following preference lists, figure out a stable matching. The letters A—D denote students, and the numbers 1—4 are the companies.

student preference				
A:	1	3	2	4
B:	3	4	1	2
C:	4	2	3	1
D:	3	2	1	4

company preference				
1:	B	A	C	D
2:	D	A	B	C
3:	A	C	B	D
4:	B	C	A	D

4. Formulate your match-finding algorithm as pseudo code. How does it scale with N?

   Notice that you do not need to find an algorithm that is as fast as possible. Just write your suggestion down formally, and assess its complexity.