Lecture 2

Assignment 1 is out, due in February 15th

Will be going into more depth on the stable matching problem, and then on “math background”.

Example

Suppose we have students A, B, C and companies X, Y, Z, with their preferences as follows:

$A : X, Y, Z \quad B : X, Z, Y \quad C : Y, Z, X$

$X : B, A, C \quad Y : C, A, B \quad Z : A, B, C$

Is $(A, X), (B, Z), (C, Y)$ a stable matching?

No, B and X would rather defect to pair up with each other.

Gale-Shapley algorithm

```
Initially, all companies and students are unengaged.
While there is a company C that is unengaged and hasn't made an offer
to every student:
  Let S be the highest ranked student for C to whom C has not yet made
  an offer.
  If S is unengaged then
    (C, S) become engaged.
  Else
    Let C' be the company that S is engaged to.
    If S prefers C to C' then
      (C, S) become engaged
      C' becomes unengaged
    endif
  endif
endwhile
Output the set of engaged pairs
```

<u>Lemma 1</u>: Once a student receives an offer, she remains engaged until the end of the algorithm, and her job can only improve over time.

<u>*Proof*</u>: An unengaged student who receives an offer always becomes engaged.

An engaged student can only switch companies; they cannot become unengaged.

A student only switches to a company they prefer. ☐

<u>Lemma 2:</u> As the algorithm proceeds, the sequence of students engaged to a given company can only decrease in preference over time.

<u>Proof:</u> Companies make offers in decreasing preference order.

<u>Lemma 3:</u> If there are n companies, then the algorithm will terminate after at most n^2 iterations of the while loop.

<u>Proof:</u> Let $P(t)$ be the set of pairs (c, s) such that c has made an offer to s by the end of the tth iteration.

We have $|P(t+1)| = |P(t)| + 1$.

Since the sequence of values $|P(t)|$ is strictly increasing (by exactly 1 every time), and there are exactly n^2 pairs (c, s), the algorithm must terminate after at most n^2 steps. ☐

<u>Lemma 4:</u> The algorithm outputs a perfect matching. (every student has an offer, and every company is engaged)

<u>Proof:</u> New engagements are only created between parties who are not otherwise engaged (either because they were never previously engaged, or because they broke an engagement and took a new one), so the set of engagements is always a matching (I think this means you can’t have multiple people paired to a company and vice versa).

Suppose that at the end of the algorithm (which happens by Lemma 3), there is some company c and student s who are unengaged.

Then c must have made an offer to s. But once a student receives an offer, they remain engaged until the end of the algorithm (by Lemma 1).

This is a contradiction, so the matching must be perfect. ☐

<u>Theorem</u>: The matching output by the algorithm is stable.

<u>Proof:</u> Suppose (for a contradiction) that there is some pair (c, s) such that c prefers s to its assigned student s’ and s prefers c to their assigned company c’.

Then, c must have made an offer to s before s’ (since it makes offers in decreasing preference order). What went wrong?

Either:

- s had a job she preferred to c, so she rejected the offer, or
- s accepted c's offer, but later switched to a company that s preferred more.

Either way, s ends up with a job they prefers to c.

But s actually ends up with c', which is less preferred than c, which is a contradiction. $\square$

Therefore, the algorithm is correct.

We also showed that it runs in time $O(n^2)$, but to describe the input takes around $2n^2$ words (each company and student has n students/companies to list for their preferences), so in a way the running time increases linearly with input size, which is pretty good.

Math Background

Asymptotic notation

- We'll use big-O notation to express upper bounds on running times of algorithms.

Example: Running time $10n^2 + 2n + 5000 \in O(n^2)$

Definition: We say $f(n) \in O(g(n))$ if $\exists c > 0, n_0 \geq 0$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$

We usually consider algorithms to be efficient if they run in time $O(n^a)$ for some constant a (a polynomial time algorithm).