Notes 9-11

1 Runtime of the Gale-Shapley algorithm

Previously, to prove that the algorithm terminates, we showed that the while loop of the algorithm executes at most n^2 times. If each execution of the while loop takes c time for some constant number c, then we can say the worst-case runtime is cn^2. However, if each execution of the while loop takes n time, then the worst-case runtime is $n \cdot n^2 = n^3$. We will give implementation details which we can argue only give the runtime of n^3:

- list of free men in queue/linked list
- engagements stored in two arrays: wife[m] and husband[w] (e.g., if m is engaged to w, then wife[m] $= w$ and husband[w] $= m$)
- Men proposing:
  - count[m] : count of # of proposals made by m
- Women rejecting/accepting : pref[w] : array of all men in order of preference for w.

Due to the implementation in the last bullet, when a woman must decide if she prefers her current engagement over a new proposal, she must search through her preference list which takes n time in the worst-case. This occurs once in each step of the while loop, resulting in a worst-case runtime of n^3.

To alleviate this, we can do some pre-processing on each pref[w] array. Define mvpref in the following way: if m is in array index i of pref[w], then let mvpref[w][m] $:= i$. [1] Intuitively, mvpref[w][m] returns m's rank in w's preference list. Then, we can quickly check if w prefers her current match or not: w prefers m over m' if mvpref[w][m] $<$ mvpref[w][m']. This take constant time, since array lookup is constant time. Creating mvpref from pref takes about n time for each woman for a total of n^2 time; however, we must only create the array once at the beginning of the algorithm, not in each execution of the while loop. Now the while loop executes in c time for some constant c, and we do roughly n^2 steps of preprocessing, so the total worst-case runtime is $cn^2 = \mathcal{O}(n^2)$ (see later section for definition of $\mathcal{O}$).

[1] This technique is sometimes called making an "inverse array", because the data becomes the new index and the index becomes the new data.

2 Men-optimality vs. women-optimality

Consider the two example stable matchings shown in class (figure to be added soon). Note that the Gale-Shapley algorithm gives the blue matching (try running it with pencil and paper to test your understanding of the algorithm).

We call this matching *man-optimal*, since men get matched with their "best possible" partner. Women would prefer the matching in red. Although Z would prefer to be matched with A or B, there is no stable matching in which Z is matched with A or B. So we define *man-optimal* in the following way: women w is a *valid partner* for man m if there exists a stable matching where m is paired with w. Thus, A is not a valid partner for Z, since there is no stable matching in which they are paired. A *man-optimal* stable matching : every man is matched to his best (most preferred) valid partner. The man-optimal stable matching is unique!

Claim 1. *The Gale-Shapley algorithm returns the man-optimal stable matching.*

Find the proof in the textbook.

3 Algorithm analysis: runtime

We are generally consider with *worst-case* analysis[2]. The intuition is that algorithms work fast on "easy" inputs. What we are concerned with is how long the algorithm will take on "hard" inputs. E.g., the example stable matching problems we looked at are easy to solve even by hand; however, consider the National Resident Matching Program. In that case, the stable matching inputs are very large size, and thus an efficient algorithm is needed.

To clarify what we mean by large input size, there is typically an obvious parameter by which to scale the problem size. In the Gale-Shapley algorithm, when $\#men = \#women = n$, we call n the size of the problem. It is important to be clear what the complexity parameter is when giving a runtime analysis.

3.1 Asymptotic notation

In the Gale-Shapley algorithm, we said the algorithm takes cn^2 for some constant c (c does not scale with the input size). This constant c is affected by implementation details, and things like whether you are using assembly language or Java-level instructions, and other things of this nature. With asymptotic notation, we want (1) to ignore this constant c and (2) ignore what happens for small n.

Let $T(n)$ be the worst-case running time of an algorithm. We say $T(n) = \mathcal{O}(g(n))$ means $\exists c > 0, n_0 > 0$ such that $\forall n \geq n_0, T(n) \leq c \cdot g(n)$. The existence of some $c > 0$ addresses point (1) from the previous paragraph. The existence of some n_0 for which the inequality holds for $n \geq n_0$ addresses point (2) from

[2]Average-case analysis is common, but typically a more advanced algorithm analysis technique.

the previous paragraph. $\mathcal{O}$ is a sort of "asymptotically less than or equal to". In other words, $T(n) = \mathcal{O}(n)$ mean $T(n)$ is less than or equal to $g(n)$, ignoring some constant factors and small values of n.

3.2 Example 1:

Let $T(n) = pn^2 + qn + r$ where $p, q, r \geq 0$.

Claim 2. $T(n) = \mathcal{O}(n^2)$

Proof. Suffices to find $c, n_0 > 0$ such that $\forall n \geq n_0$, $pn^2 + qn + r \leq c \cdot n^2$. Let $n_0 = 1$, $c = p + q + r$. Then $pn^2 + qn + r \leq pn^2 + qn^2 + rn^2 = cn^2$. □