Design and Analysis of Algorithms

Ekesh Kumar[*]

February 5, 2020

These are my course notes for CMSC 451: Design and Analysis of Algorithms, taught by Professor Clyde Kruskal. Please e-mail corrections to `ekumar1@terpmail.umd.edu`

Contents

[*]Email: `ekumar1@terpmail.umd.edu`

§1 Tuesday, January 28, 2020

§1.1 Introduction

This is CMSC 451: Design and Analysis of Algorithms. We will cover graphs, greedy algorithms, divide and conquer algorithms, dynamic programming, network flows, NP-completeness, and approximation algorithms.

- Homeworks are due every other Friday or so; NP-homeworks are typically due every other Wednesday.
- There is a 25% penalty on late homeworks, and there's one get-out-of-jail free card for each type of homework.

§1.2 Stable Marriage Problem

As an introduction to this course, we'll discuss the **stable marriage problem**, which is stated as follows:

> Given a set of n men and n women, match each man with a woman in such a way that the matching is *stable*.

What do we mean when we call a matching is "stable"? We call a matching *unstable* if there exists some man M who prefers a woman W over the woman he is married to, and W also prefers M over the man she is currently married to.

In order to better understand the problem, let's look at the $n = 2$ case. Call the two men M_1 and M_2, and call the two women W_1 and W_2.

- First suppose M_1 prefers W_1 over W_2 and W_1 prefers M_1 over M_2. Also, suppose that M_2 prefers W_2 over W_1 and W_2 prefers M_2, then
- If both W_1 and W_2 prefer M_1 over M_2, and both M_1 and M_2 prefer W_1 over W_2, then it's still easy to see what will happen: M_i will always match with W_i.
- Now let's say M_1 prefers W_1 to W_2, M_2 prefers W_2 to W_1, W_1 prefers M_2 to M_1, and W_2 prefers M_1 to M_2. In this case, the two men rank different women first, and the two women rank different men first. However, the men's preferences "clash" with the women's preferences. One solution to this problem is to match M_1 with W_1 and M_2 with W_2. This is stable since both men get their top preference even though the two women are unhappy.

The solution to the problem starts to get a lot more complicated when the people's preferences do not exhibit any pattern. So how do we solve this problem in the general case? We can use the **Gale-Shapley algorithm**. Before discussing this algorithm, however, we can make the following observations about this problem:

- Each of the n men and M woman are initially unmarried. If an unmarried man M chooses the woman W who is ranked highest on their list, then we cannot immediately conclude whether we can match M and w in our final matching.This is clearly the case since if we later find out about some other man M_2 who prefers W over any other woman, W may choose M_2 if she likes him more than M. However, we cannot immediately rule out M being matched to W either since a man like M_2 may not ever come.

- Just because everyone isn't happy doesn't mean a matching isn't stable. Some people might be unhappy, but there might not be anything they can do about it (if nobody wants to switch).

Moreover, we introduce the notion of a man *proposing* to a woman, which a woman can either accept or reject. If she is already engaged and accepts a proposal, then her existing engagement breaks off (the previous man becomes unengaged).

Now that we've introduced these basic ideas, we can now present the algorithm:

```
# Input:  A list of n men and n women to be matched.

# Output:  A valid stable matching.

stable_matching {
   set each man and each woman to "free"
   while there exists a man m who still has a woman w to propose to {
     let w be the highest ranked woman m hasn't proposed to.

     if w is free {
        (m, w) become engaged
     } else {
        let m' be the man w is currently engaged to.
        if w prefers m' to m {
           (m', w) remain engaged.
        } else {
           (m, w) become engaged and m' loses his partner.
        }
     }
   }
}
```

Proposition 1.1
The Gale-Shapley algorithm terminates in $\mathcal{O}(n^2)$ time.

Proof. In the worst case, n men end up proposing to n women. The act of proposing to another person is a constant-time operation. Thus, the $\mathcal{O}(n^2)$ runtime is clear. □

§2 Thursday, January 30, 2020

§2.1 Optimality and Correctness of Gale-Shapley

Last time, we introduced the Gale-Shapley algorithm to find a stable matching. Today, we'll prove that the algorithm is correct (i.e. it never produces an unstable matching), and it is optimal for men (i.e. the men always end up for their preferred choice).

First, we'll show that the algorithm is correct:

Proposition 2.1

The matching generated by the Gale-Shapley algorithm is never an unstable matching.

Proof. Suppose, for the sake of contradiction, that m and w prefer each other over their current partner in the matching generated by the Gale-Shapley algorithm. This can happen either if m never proposed to w, or if m proposed to w and w rejected m. In the former case, m must prefer his partner to w, which implies that m and w do not form an unstable pair. In the latter case, w prefers her partner to m, which also implies m and w don't form an unstable pair. Thus, we arrive at a contradiction. □

Next, we'll prove that the algorithm is optimal for men. However, before presenting the proof, observe that it is not too hard to see intuitively that the algorithm "favors" the men. Since the men are doing all of the proposing and the women can only do the deciding, it turns out that the men always ends up with their most preferred choice (as long as the matching remains stable).

Proposition 2.2

The matching generated by the Gale-Shapley algorithm gives men their most preferred woman possible without contradicting stability.

Proof. To see why this is true, let A be the matching generated by the men-proposing algorithm, and suppose there exists some other matching B that is better for at least one man, say m_0. If m_0 is matched in B to w_1 which he prefers to his match in A, then in A, m_0 must have proposed to w_1 and w_1 must have rejected him. This can only happen if w_1 rejected him in favor of some other man — call him m_2. This means that in B, w_1 is matched to m_0 but she prefers m_2 to m_0. Since B is stable, m_2 must be matched to some woman that he prefers to w_1; say w_3. This means that in A, m_2 proposed to w_3 before proposing to w_1, and this means that w_3 rejected him. Since we can perform similar considerations, we end up tracing a "cycle of rejections" due to the finiteness of the sets A and B. □

§3 Tuesday, February 4, 2020

Today, we'll recap graph terminology and elementary graph algorithms.

§3.1 Graph Terminology

Definition 3.1. A **graph** $G = (V, E)$ is defined by a set of vertices V and a set of edges E.

The number of vertices in the graph, $|V|$, is the **order** of the graph, and the number of edges in the graph, $|E|$, is the **size** of the graph. Typically, we reserve the letter n for the order of a graph, and we reserve m for the size of a graph.

Definition 3.2. We say a graph is **directed** if its edges can only be traversed in one direction. Otherwise, we say the graph is **undirected**.

Definition 3.3. A graph is called **simple** if it's an undirected graph without any loops (edges that start and end at the same vertex).

Definition 3.4. A graph is **connected** if for every pair of vertices u, v, there exists a path between u and v.

§3.2 Graph Representations

There are two primary ways in which we can represent graphs: **adjacency matrices** and **adjacency lists**.

An adjacency matrix is an $n \times n$ matrix `A` in which `A[u][v]` is equal to 1 if the edge (u, v) exists in the graph; otherwise, `A[u][v]` is equal to 0. Note that the adjacency matrix is symmetric if and only if the graph is undirected.

On the other hand, an adjacency list is list of lists in which each list describes the set of neighbors of a single vertex in the graph.

Each graph representation has its advantages and disadvantages.

	Adjacency Matrix	Adjacency List
Storage	$\mathcal{O}(n+m)$	$\mathcal{O}(n^2)$
Add vertex	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
Add edge	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Remove vertex	$\mathcal{O}(n+m)$	$\mathcal{O}(n^2)$
Remove edge	$\mathcal{O}(m)$	$\mathcal{O}(1)$