CS340 Analysis of Algorithms

Handout:	**0**	**Professor:**	**Dianna Xu**
Title:	**Stable Matching**	**E-mail:**	dxu@brynmawr.edu
Date:		**URL:**	**http://cs.brynmawr.edu/cs340**

Sample description, pseudo code, time analysis and proof of correctness for Gale-Shapley. Refer to lecture notes and the text book for definitions of matching, perfect matching and stable matching.

Recall that given a set of employers E, a set of applicants A, $|E|=|A|=n$, and $2n$ preference lists, each of size n, we ask for a set of stable matchings M, which is a set of n pairs (e,a), where $e \in E$ and $a \in A$.

1 Description

Intially all employers and applicants are unpaired. We start with any unpaired employer e. e will make an offer to the highest ranked applicant in its preference list, to whom e has not made an offer to before. When receiving an offer, if an applicant a is unpaired, it becomes paired with e. Otherwise a is already paired with some e'. If a prefers e' to e, e remains unpaired. Otherwise, a becomes paired with e and e' becomes unpaired. Offering continues as long as there is an unpaired e that hasn't made an offer to every a.

2 Pseudocode

```
Function Gale-Shapley($E$,$A$)
    all $e$ in $E$ and $a$ in $A$ are unpaired
    while there is an $e$ unmatched that hasn't made an offer to every $a$ do
        choose such an $e$
        let $a$ be the highest-ranked applicant in the preference list of $e$, to whom $e$
        has not made an offer
        if $a$ is unpaired then
            pair $a$ and $e$
        end
        else
            $a$ is currently paired with some $e'$
            if $a$ prefers $e'$ to $e$ then
                $e$ remains unpaired
            end
            else
                $a$ is paired with $e$ and $e'$ becomes unpaired
            end
        end
    end
    return the set of pairs
```

3 Time Analysis

An employer never makes an offer to the same applicant again, therefore there are at most n^2 such pairings, and `while` loop runs $O(n^2)$ times. In order for the entire loop to run in $O(n^2)$, the following operations in the loop body must be performed in $O(1)$:

(1) Checking/choosing if there is an e unpaired that hasn't made an offer to every a, line 3

(2) Choosing a who is the highest-ranked applicant in e's preference list to whom e has not made an offer to yet, line 5

(3) Checking if a is paired/unpaired, line 6

(4) Checking if a prefers e' to e, line 11

All other lines in the loop body are clearly $O(1)$ operations, but lines 3, 5, 6 and 11 require proper data structure organization to achieve $O(1)$. The justifications below show why a search is not needed to perform any of the operations, which would explode time to $O(n)$.

3.1 Data Structures

Employers and applicants are assigned unique integer IDs $\{e_1 = 0, ..., e_n = n-1\}$, $\{a_1 = 0, ..., a_n = n-1\}$.

Unpaired employers who have not made an offer to every applicant are kept in a queue. Algorithm puts all employers in this queue intially and the employer at the head of the queue starts making offers. Employer preference lists are stored as sorted arrays of n applicants, in order of preference. When making offers, an employer will simply move down the sorted preference array and make an offer to the next applicant. When a matching is made, the employer is dequeued and the index of the next preferred applicant is saved. If an employer becomes unmatched (through switching), that employer is added back to the end of the queue. Thus (1) and (2) can both be done in $O(1)$ and the `while` loop terminates when the queue is empty.

An integer array `M` of size n, indexed by the applicant IDs and storing matched employer IDs is used to keep track of the matchings. Intially, all entries are -1 which indicates that all applicants are unmatched. Whenever an (e_i, a_j) pairing is matched, we set `M[`a_j`]` to e_i. Therefore, "if a is paired" can be decided by "`if (M[`a_j`] != -1)`" and is $O(1)$. This answers (3).

Applicant preference lists are stored in one $n \times n$ `int` array that is indexed by the IDs, storing ranks as integers (where 1 is the hightest ranked and n is the lowest). For example, applicant a_k with a preference list $\{e_3, e_{n-1}, e_n, ..., e_2, e_1\}$, will store the following values in row a_k in the applicant preference array `AP`:

$$\begin{aligned}
\texttt{AP}[a_k][e_1] &= \texttt{n} \\
\texttt{AP}[a_k][e_2] &= \texttt{n-1} \\
\texttt{AP}[a_k][e_3] &= \texttt{1} \\
&\ldots \\
\texttt{AP}[a_k][e_{n-1}] &= \texttt{2} \\
\texttt{AP}[a_k][e_n] &= \texttt{3}
\end{aligned}$$

Therefore, "if a_k prefers e_i to e_j" can be decided by "`if (AP[`a_k`][`e_i`] < AP[`a_k`][`e_j`])`", which is clearly $O(1)$. This answers (4).

An important note: although these $O(1)$ operations can also be achieved by using dictionaries/hashmaps, they are less efficient than directly indexing into arrays, especially in terms of space. Dictionaries/hashmaps require load factors to be below 70%-80% in order to resolve collisions, which means 20%-30% wasted space. If the size of the data is known, arrays should be the choice for $O(1)$ operations. Arrays have no collisions and lookup in arrays is $\theta(1)$ instead of $O(1)$.

Overall, the data structures size is $O(n^2)$, with the following components:

1. a queue of size n
2. employer preference lists: n sorted arrays, each of size n, for a total of n^2
3. applicant preference lists: a $n \times n$ array, for a total of n^2
4. an array of matchings, of size n

Because the input data format is unspecified, we assume that it will come in organized in the data structures as we prefer. Therefore we will skip time analysis for the construction/initialization of the preference lists. The queue intitally contains all employers (in no particular order) and this takes n. The initialization of the the array of matchings takes place in line 2 and takes n. Therefore the entire algorithm runs in $n+n+O(n^2)=O(n^2)$. We have shown that `Gale-Shapley` runs in $O(n^2)$, however, the n here is the number of employers or applicants, NOT the total input size. What is our input size? The largest component in the input is the preference lists, which as shown in the space analysis above, is $O(n^2)$. If we assign $N=n^2$, we can easily see that `Gale-Shapley` has input size $O(N)$ and runs in $O(N)$ as well. Therefore `Gale-Shapley` is linear.

4 Correctness Proof

Termination is easily argued because the sets E and A are finite and each employer only makes an offer to each applicant once. As discussed in the time analysis, there are at most n^2 such pairings and the `while` loop terminates after $O(n^2)$ iterations.

We will now prove that the matching M is perfect. In other words, all employers and applicants are matched at the end of the algorithm.

Proof: by contradiction. Suppose there is some employer e who didn't hire. Because we have n employers and n applicants, there must be some applicant a who didn't get a job. Note that by construction of the algorithm, an applicant will never become unmatched once an initial job offer was made (switching unmatches employers, but not applicants). Therefore e didn't offer a a job. But e was required to offer all applicants a job. Contradiction. ■

Now we show that all matchings are stable.

Proof: by contradiction. Suppose that there is a pair $(e,a) \in M$ that is unstable. This means that there exists an a' that e prefers more than a. In addition, a' must also prefer e more than its current matching e'. In other words, (e,a) being unstable indicates that there is another unstable pair (e',a'), because e and a' want to switch. How did e and a' end up not paired? There are two cases for us to consider.

1. e never offered a' a job. This is not possible because e prefers a' more than a and would have offered to a' before offering to a.
2. e did offer a' a job, but a' either didn't take it or switched later in favor of its current match. This is also impossible because a' is currently paired with e', which is less preferred than e.

Both cases are impossible. Contradiction. ■