CS340 - Stable Matching Write-up Example

Kyu Chang and Kewei Qu
Collaborators: Sorelle Friedler

September 5, 2016

1 Problem A

1.1 Description

Initially, all applicants and employers are unpaired. An employer e, who is not paired with an applicant makes an offer to applicant a, where a is the highest-ranked applicant in e's preference list who e has not made an offer to yet. a can be either unpaired or a can be already paired with another employer e'. If a is unpaired then pair e and a. If a is already paired with e' then check if a prefers e to e'. If a prefers e to e' then unpair e' and a and pair e and a. Otherwise, e remains unpaired. Continue this process until every employer is paired or every employer who is not paired makes an offer to every applicant.

1.2 Pseudocode

```
Function findStableMatching(E, A)
    All e ∈ E and a ∈ A are unpaired
    while there is an e unpaired that hasn't made an offer to every a do
        choose such an e
        let a be the highest-ranked applicant in e's preference list who e has not
        made an offer to yet
        if a is unpaired then
            pair a and e
        end
        else
            a is currently paired with 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
```

1.3 Time Analysis

Each iteration consists of some employer making an offer to an applicant the employer has never made an offer to before. Let n be $|E|$ and $|A|$. Since each employer can make at most n offers and there are n employers. There can be at most n^2 iterations of the $While$ loop.

In order to run the algorithm in $O(n^2)$, the following operations need to be done in $O(1)$.

1. Checking/choosing if there is an e unpaired that hasn't made an offer to every a
2. Choosing a who is highest-ranked applicant in e's preference list who e has not made an offer to yet
3. Checking if a is paired/unpaired
4. Checking if a prefers e' to e

1.3.1 Data Structure

Each employer's ranked list of applicants can be stored in a linked list, where the head represents the highest-rank applicant that the employer has not made an offer yet. When an employer makes an offer then we can replace the head of list with head.next. These ranked lists is stored in dictionary where the key is e and the value is the linked list. This allows (2) to be done in $O(1)$, and it takes $O(n^2)$ to build a dictionary of linked lists.

Employers who are unpaired that haven't made an offer to every applicant can be stored in a queue. When the algorithm unpairs an employer e we push e to the queue. We pop any e that has made an offer to all applicants from the queue by checking the linked list described above. The queue combined with the dictionary allows (1) to be done in $O(1)$. Building this queue takes $O(n)$.

A dictionary with key a and value e or $None$ can be used to represent the pair (a, e). This allows (3) to be done in $O(1)$. Building such dictionary takes $O(n)$.

Applicants' preference of employers is stored as a dictionary of dictionaries. The key of the outer dictionary is a, the key of the inner dictionary is e, and the value of the inner dictionary is an integer used to represent the numerical preference. This data structure allows (4) to be done in $O(1)$ and it takes takes $O(n^2)$ to build it.

With the above data structures, which take $O(n^2)$ to build, the algorithm takes $O(n^2)$ since there can be at most n^2 iterations of the $While$ loop and all operations inside the loop can be achieved in $O(1)$.

Thus, the time complexity of the algorithm is $O(n^2)$.

1.4 Proof of Correctness

Lemma. *An applicant remains paired from the moment at which they receive the first offer and the sequence of employers to which this applicant is paired only gets better.*

Proof. From the algorithm execution we see that if the applicant is not paired at the moment when an employer makes an offer, the applicant pairs up with the employer. The applicant only switches employers when this applicant prefers the current employer making the offer to the one they are already paired up with, and never loses employment once paired. □

Proof of Correctness. To prove that this algorithm works, three parts need to be shown.

1. proof of termination
2. proof of perfect matching (every employer is matched with an applicant)

3. proof of stability, i.e. there does not exist a situation where an employer e is paired with an applicant a, but prefer applicant a' and applicant a' is paired with employer e' but prefer employer e.

Proof of termination:
Pseudocode and time analysis sections show that at most n^2 employer-applicant pairings are made. Since only new pairings are ever offered, because the employer only goes down their list and never re-makes offers, the algorithm terminates in at most $O(n^2)$ iterations of the while-loop.

Proof of perfect matching (by contradiction):
Suppose there is some employer e who didn't hire (wasn't matched). Then there is some applicant a who didn't get a job, since there are n employees and n applicants. Since the algorithm never allows an applicant to go without a job once a first offer is made (see the above Lemma), this means that a was never offered a job. Since a was never offered a job by any employer, that means that e never offered a a job. But the algorithm requires every employer to offer every applicant a job before giving up on matching that employer. This is a contradiction, so all employers and applicants are matched by the algorithm.

Proof of stability (by contradiction):
Suppose towards contradiction that employer e is paired with applicant a' but prefers applicant a, and applicant a is paired with employer e' but prefers employer e (i.e., there exists an unstable pair). Either 1) e never offered a a job or 2) e did offer a a job.

In case 1, e never offered a a job. Since e is paired with a' and e makes employment offers in preference order, e must prefer a' to a (i.e., e never got to a when going down their list). But this contradicts the assumption that e prefers a to a'.

In case 2, since e is not paired with a, a must have rejected e in favor of some other employer. Since a's options only get better (by the Lemma), a must prefer e' to e. But this contradicts the assumption that a prefers e to e'.

Since both cases end in contradiction, the matching is stable. □