CSCE 586 HW 1

Marvin Newlin

11 October 2018

1. **Chapter 1 Problem 4**

Given *m* students and *n* hospitals with *m* > *n*, show that there is always a stable assignment of students to hospitals and give an algorithm to find one.

Solution:

As given in the problem statement, there are some assumptions that are made here. First, we have that *m* > *n* so there will be a surplus of students and not all students will be matched to hospitals. Additionally, 1 student is assigned to 1 hospital and no hospital is assigned more than 1 student.

Let *H* be the set of hospitals and *S* be the set of students. We have that |*H*| = *n* and |*S*| = *m*. Also, let *M* be the set of matches that are determined by our algorithm.

Description:

Note: This algorithm is exactly the Gale-Shapley algorithm defined on page 6 of the textbook but reworded with different notation.

Assume *h,h'* in *H* and *s* in S.

```
INIITIALIZE M to be an empty matching
WHILE (h is unmatched and hasn't proposed to all s)
    IF (s is free)
        s accepts h
        add h-s to M
    ELSE IF (s prefers h to current partner h')
        add h-s to M
        remove h'-s from M
    ELSE
        s rejects h
```

Proof of Correctness:

To prove this algorithm is correct, we need to show that the algorithm terminates, that it outputs a matching, and that that matching is stable. These are the same steps shown in the proof of the Gale Shapley algorithm in the textbook but modified to reflect the exact values of our problem.

Claim: The algorithm terminates in at most *m*n* iterations.

Proof: First we note that the hospitals propose in order of preference for students. Second, once a student is matched, they never become unmatched, they only upgrade.

Each hospital proposes to at most *m* students and since there are *n* hospitals, the algorithm will terminate after at most *m*n* iterations.

Claim: The algorithm outputs a matching, i.e. all hospitals are matched.

Proof: Suppose that our algorithm has output a matching and some hospital *h* in *H* has not been matched. Then, by our algorithm, ∃ *s* in S such that *s* was never proposed to. But we know that *h* proposed to every student since that is a condition of our while loop and *h* ended up unmatched. Thus, we have a contradiction. Therefore, all hospitals are matched.

Claim: The matching that our algorithm produces is a stable matching.

Proof: Suppose that the matching that the algorithm has output is not a stable matching. There are 2 conditions of instability. They are:

i) There exists *s,s'* in *S* and *h* in *H* such that
- *s* assigned to *h* and
- *s'* not assigned and
- *h* prefers *s'* to *s*

ii) There exists *s,s'* in *S* and *h, h'* in *H* such that
- *s* assigned to *h* and *s'* assigned to *h'* and
- *h* prefers *s'* and
- *s'* prefers *h*

This leaves us with 2 cases for our unstable matching.

Case 1: Our matching produced by the algorithm has produced the instability (i). This means that *s'* was higher on *h's* preference list than *s*. But since *h* ended up with *s* according to our algorithm, then *s* was higher on *h*'s preference list than *s'.* Thus, we have a contradiction and the case (i) instability cannot occur.

Case 2: Our matching produced a case (ii) instability. Assume, WLOG that the matching *h'-s'* was already in *M*. Since *h* prefers *s'*, *h* proposes to *s'* before *s* and since *s'* prefers *h*, then according to our algorithm *s'* should accept *h*. But since we ended up with *h-s* then this did not occur, so *h* must have preferred *s*, which is a contradiction.

Thus, since both cases of an unstable match cannot occur from our algorithm, then our algorithm has produced a stable match.

Runtime Complexity:

As shown in the proof of termination above, there are *n* hospitals and *m* students. Each hospital proposes to at most *m* students and since there are *n* hospitals the proposal process is repeated at most *n* times for all hospitals, so the complexity of the algorithm is O(*m***n*).

2. **Chapter 1 Problem 6**

Given the schedule for each ship, find a truncation of each so that condition (†) continues to hold: no two ships are ever in the same port on the same day. Show that such a set of truncations can always be found and give an algorithm to find them.

(†) No two ships can be in the same port on the same day

<u>Solution:</u>

This problem is an instance of the Stable Matching problem where we need to match *n* ships to *n* ports.

However, we can't simply match them, we need to figure out a way to match them given the constraints of the schedules of each ship and the condition from the problem statement, i.e. that no two ships can be in the same port on the same day.

Description:

What we have to do here is set up the preference lists so that we can use the Gale-Shapley stable matching algorithm to match the ships and the ports. Here's how we build the preference lists:

<u>Construction:</u>

Each ship ranks the ports it visits in chronological order based on its schedule.

Each port ranks the ships that visit it in reverse chronological order of each ship's visit.

Constructing the preference lists in this way, we can map the set of Ships to the set of men *M* and the set of ports can be matched to the set of women *W*. Note that we allow the ships to do the selecting. Then we perform the algorithm described on page 6 of the textbook to obtain the matching of ships to ports

Proof of Correctness:

We know that the G-S algorithm outputs a stable match. What we do not explicitly know is whether that stable match is a valid assignment of ships to ports.

Claim: A stable assignment of ships to ports does not violate condition (†).

Proof: Suppose that we have obtained a stable match from the G-S algorithm and we have ended up with an invalid assignment (i.e. it violates (†)). Then there is some ship s_i that passes through port p_k after another ship s_j has stopped there. With the preference relation we defined above this means that s_i prefers p_k to its actual port and p_k prefers s_i to s_j but then this is an unstable match, so we have a contradiction of our assumption that we had a stable match.

Runtime Complexity:

Since this is the G-S algorithm with 2 sets of size *n* we know that it runs in O(n^2) time. This is also shown as item 1.3 in chapter 1.

3. **Chapter 2 Problem 3**

Take the following list of functions and arrange them in ascending order of growth rate. That is, if function g(n) immediately follows function f(n)in your list, then it should be the case that f(n) is O(g(n)).

$f_1(n) = n^{2.5}$, $f_2(n) = \sqrt{2n}$, $f_3(n) = n + 10$, $f_4(n) = 10^n$, $f_5(n) = 100^n$, $f_6(n) = n^2 log(n)$

Solution:

We can easily order all but f_6 using the basic properties of exponentials & polynomials.

$f_2(n)$ is O($f_3(n)$) since $f_2(n) < f_3(n)$ for all n >= 1. $f_3(n)$ is clearly $O(f_1(n))$ and $f_1(n)$ is clearly O($f_4(n)$) since exponentials are bigger than polynomials. Then $f_4(n)$ is clearly O($f_5(n)$). So now we only have to place $f_6(n)$. We know that $f_6(n)$ is not O(n) since it contains an n^2. Thus, we guess it is probably between $f_3(n)$ and $f_1(n)$.

$$n^2 log(n) \leq c * n^{2.5} for c \in R$$

$$\rightarrow log(n) \leq c * n^{\frac{1}{2}}$$

$$\rightarrow log(n) \leq 1 * n^{\frac{1}{2}}$$

Thus, we have that $f_6(n)$ is O($f_1(n)$) so we can make our final ordering.

Order: f_2, f_3, f_6, f_1, f_4, f_5

Difficulty:

Time on problem 1 was about 20 minutes, difficulty 4

Time on problem 2 was about 45 minutes, difficulty 7

Time on problem 3 was about 10 minutes, difficulty 2