2nd Lt David Crow

ENG/20M

Analysis of Algorithms Homework 1

**Chapter 1, Problem 4**

Gale and Shapley published their paper on the Stable Matching Problem in 1962; but a version of their algorithm had already been in use for ten years by the National Resident Matching Program, for the problem of assigning medical residents to hospitals.

Basically, the situation was the following. There were hospitals, each with a certain number of available positions for hiring residents. There were medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than there were slots available in the hospitals.

The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.)

We say that an assignment of students to hospitals is *stable* if neither of the following situations arises

* First type of instability: There are students and , and a hospital , so that
  + is assigned to , and
  + is assigned to no hospital, and
  + prefers to .
* Second type of instability: There are students and , and hospitals and , so that
  + is assigned to , and
  + is assigned to , and
  + prefers to , and
  + prefers to .

So we basically have the Stable Matching Problem, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students.

Show that there is always a stable assignment of students to hospitals­­ and give an algorithm to find one.

**Solution**

*Overall, this problem took me no more than 20 minutes. I’d rate it a 3 on the difficulty scale; repurposing the given algorithm was fairly straightforward.*

As our textbook says, “The Gale-Shapley algorithm is remarkably robust to variations on the Stable Matching Problem” (20). With that in mind, we can use a very similar algorithm.

I’ve taken the Gale-Shapley algorithm as shown in the text and modified it for this problem. Effectively, hospitals represent men and students represent women. Hospitals *propose* (i.e. send an offer) to students in order of preference. If a student is available, the student accepts the offer. If not, and if the student prefers the hospital to the one they’ve already accepted, they free the already-accepted hospital and accept the new offer; otherwise, the student rejects the offer. This process repeats until all hospitals fill all slots with students.

Here’s the algorithm:

1 Initially all and are free

2 While there is a hospital which still desires students and hasn’t

given an offer to every student

3 Choose such a hospital

4 Let be the highest-ranking student in ’s preference list that

has not yet given an offer to

5 If is available then

6 accepts ’s offer and has fewer slots available

7 Else has already accepted an offer from

8 If prefers to then

9 ’s slot remains open

10 Else prefers to

11 accepts the offer from

12 has one fewer slot available

13 has one more slot available

14 Endif

15 Endif

16 Endwhile

17 Return the set of hospital-student pairs

Because every hospital might give an offer to every student , this algorithm will terminate in no more than steps. In other words, it’s .

This algorithm will always generate a stable matching of hospitals and students. To prove this, let’s assume there exists some instability in , and we’ll then show that such an instability *can’t* exist. An instability exists if, for some matching and for some matching , both of the following are true:

* prefers to , and
* prefers to .

As the algorithm requires, sent its most recent offer to . This means one of three things: a) sent an offer to and was rejected, b) retracted its acceptance of the offer from , or c) did not send an offer to .

1. If rejected an offer from , we know that s had already accepted an offer from some other that prefers to . We also know that is the final hospital for , so we have two more cases.
   1. If , then prefers to .
   2. If , then prefers to , and thus prefers to .

In both cases, we see that prefers to . This contradicts our original assumption that prefers to .

1. Similarly, if retracted its acceptance of an offer from in lieu of another offer, we know that prefers that other offer from some to that of . This leads to the same two cases in (a), and thus must prefer to .
2. If sent an offer to before sending an offer to , we know that prefers . This contradicts our assumption that prefers .

Because our assumptions lead to a contradiction in every case, we know that our assumptions must be false and that such a matching cannot exist. Thus, our modified Gale-Shapley algorithm leads to a stable matching in every case.

**Chapter 1, Problem 6**

Peripatetic Shipping Lines, Inc., is a shipping company that owns *n* ships and provides service to ports. Each of its ships has a *schedule* that says, for each day of the month, which of the ports it’s currently visiting, or whether it’s out at sea. (You can assume the “month” here has days, for some .) Each ship visits each port for exactly one day during the month. For safety reasons, PSL Inc. has the following strict requirement:

(\*) No two ships can be in the same port on the same day.

The company wants to perform maintenance on all the ships this month, via the following scheme. They want to *truncate* each ship’s schedule: for each ship , there will be some day when it arrives in its scheduled port and simply remains there for the rest of the month (for maintenance). This means that will not visit the remaining ports on its schedule (if any) that month, but this is okay. So the *truncation* of ’s schedule will simply consist of its original schedule up to a certain specified day on which it is in a port ; the remainder of the truncated schedule simply has it remain in port .

Now the company’s question to you is the following: 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.

**Example.** Suppose we have two ships and two ports, and the “month” has four days. Suppose the first ship’s schedule is

*port ; at sea; port ; at sea*

and the second ship’s schedule is

*at sea; port ; at sea; port*

Then the (only) way to choose truncations would be to have the first ship remain in port starting on day 3 and have the second ship remain in port starting on day 2.

**Solution**

*Overall, this problem took me an hour or so. I’d rate it a 6 on the difficulty scale; it took me a while to determine the correct way to assign preferences to the ships and ports.*

Let’s again modify the given Gale-Shapley algorithm. For this algorithm, we’ll set a ship’s preferences according to how early said ship visits a port; that is,

* a ship prefers a port if it is scheduled to stop there before another port.

Ports prefer the opposite:

* a port wants the latest possible ship to remain at said port for the rest of the month.

With those preferences in mind, here’s the algorithm:

1 Initially all and are free

2 For each day of the month

3 While there is a ship which still needs a port and has a stop

at port scheduled for day

4 Choose such a ship

5 If already has a docked ship

6 must find a new port at which to dock

7 docks at

7 Else already has a docked ship

8 must find a new port at which to dock

9 docks at

10 Endif

11 Endfor

12 Endfor

13 Return the set of ship-port pairs

It’s possible that none of the ships find the correct port until the last day of the month, so our runtime of ships and days is . Phrased another way, for each day of the month, we could have ships stopping at ports; again, this gives .

This algorithm guarantees a successful set of truncations will always be found. Let’s prove this.

The problem statement tells us that there exist ships and ports, so we know it is *possible* to match every ship to one port. Additionally, we can assume that Peripatetic Shipping Lines, Inc. will not schedule two ships for the same port on the same day (or else they would violate their own rule).

If we assume (that is, the algorithm doesn’t generate a successful set of truncations), then there must exist a ship such that is unable to find a suitable port by the end of the month. This means that a) the last possible port for was already occupied by some other ship , or b) there were no remaining ports with which could dock.

According to the algorithm, a port will always prefer a later ship to an earlier one. Thus, (a) cannot hold because, once attempts to dock at an occupied port , will release its current ship and will then dock at .

Additionally, (b) cannot hold because we showed above that the number of ships is equal to the number of ports, and thus, for ships still searching, there are remaining ports. If any of the remaining ports are on the schedule for prior to port , then would have docked at before attempting to dock at . If the remaining ports are on the schedule after , then we have not reached the end of the month, and thus there are still ports for to check.

We have now shown that this algorithm will always generate a successful truncation.

**Chapter 2, Problem 3**

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

**Solution**

*Overall, this problem took me no more than 10 minutes. I’d rate it a 2 on the difficulty scale; determining which functions grew faster was very simple.*

1. Just by looking at the exponents, it’s obvious that , so .
2. Then, because the exponents are equal, we can simply compare bases to show that , so .
3. Additionally, it’s obvious that (because an exponent of is much worse than one of for all ), so .
4. We know that has to grow faster than (because of ) and slower than (because ). As shown in the below table (because I can’t tell by just looking at it), , so . That gives us our final answer of . In other words,

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |