Algorithms CS 310

Assignment 1- Gayle Shapley Algorithm

Hassaan Ahmad Waqar-22100137

So we basically have the Stable Matching Problem as presented in class, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical residents. There are two parts of this problem.

1. Write a short report that includes the following information:

(a) Show that there is always a stable assignment of residents to hospitals.

An instability is produced when either of the two conditions occur as stated in the question description: a) Hospital **h** does not propose to its preference **s'** but proposes to some other resident **s** below **s'** in the preference list.

b) When there is a possibility of a back-channel deal between hospitals and residents such that let suppose hospital **h** is assigned to resident **s** whereas its priority is **s'**. And **s'** is assigned to hospital **h'** even though s' prioritizes hospital **h**. This is an instability that must be avoided.

Claim: There is no unstable pair in Matching M*

Proof by Contradiction: At least one unstable pair exists.

Consider a pair **h-s** that does not exist in M*

Case 1	**Case 2**
h never proposed s h prefers resident s' to s s is down in the preference of h h–s is not unstable	h initially proposed to s s rejected h s prefers h' h-s in unstable

Only these two cases are possible for h-s not to be in Match M*

Since both these cases show that we end up with the h-s not being unstable, we have reached a negation of the contradiction we made. And hence, we show that GS produces a stable match.

---

b) Give an algorithm in pseudo code (either an outline or paragraph works) to find a stable assignment. Hint: it should be very similar to the Gale-Shapley Algorithm

Dividing this answer into two sections, algorithm and implementation.

Algorithm:	Implementation:
Since a hospital will be having multiple slots to fill, but a resident can be assigned to only one hospital **Store all data from file to appropriate structure** **While all slots of the hospitals have not filled** (• Hospital sends a proposal to a resident top on its priority (each hospital proposes to each resident once). • If the resident is unmatched, it makes a match (update relevant structures) • Else If the resident is already matched, it checks whether	**Note: This is tentative and can change when translating to code.** Structure details will be discussed here. **Admin Preference:** To store admin preference, we can make a dictionary where each key as a list. Key represents hospitals and list will have residents in order of priority **Resident Preference:** Same dictionary format with keys as residents and hospitals as values in a list. **Resident_Match:** Simple one dimensional array that stores matching for each resident. Index

the offer is from a better priority; if yes, create a new match and break the previous one (structures updated). • Else, reject offer.) Loops terminates when all hospitals have all slots filled.	refers to the resident and value refers to hospital. **Slots**: A queue that will have all hospital numbers corresponding to slots. E.g. if Hospital 0 has five slots, we will have five 0s in the queue. And so on. Loop will run over it.

(c) Give a proof of your algorithm's correctness. Remember that you must prove both that your algorithm terminates and gives a correct result.

There are two parts (third: stability already discussed in (a)) to prove correctness of the algorithm.

Proof of Termination

- Hospitals will propose residents in decreasing order of preference
- Once a resident gets matched, it will never get unmatched
- Each hospital sends a proposal to a resident only once.

This way, we can see all residents will get offered at least once, which implies that none will remain unmatched (else the algorithm has not terminated yet). Each time, a hospital sends a proposal to one resident. There will be a total of n x m proposals (worst case).

Proof of Correctness

Claim: GS Algorithm outputs a match

- Hospital will propose only if some of its slot is unfilled -> matches to <= x residents where x refers to number of slots of a particular hospital.

- Resident only keeps the best hospital it gets an offer from
  -> <=1 hospital. (< since #residents > #hospitals)

Claim: All Hospitals get matched

By Contradiction: There exists some hospital that did not get matched.

Since number of residents are > number of hospitals, let suppose there is a hospital that does not get matched.

Either it has not yet made a proposal. In this case, the algorithm has not yet terminated.
Other case is when it sent a proposal to every resident available but match broke with every resident. This is a contradiction since
a) once a resident is proposed, it can only trade up, and not be left matchless.

b) Number of residents are greater than number of slots in hospitals so some resident/s will remain unmatched when all slots are filled.

(d) Give the runtime complexity of your algorithm in Big O notation and explain why?

Algorithm will run in **O (n x m)** where n are the number of hospitals and m are the number of residents. Since slots of hospitals, let say x, are less than number of residents, x will remain smaller than m thus incorporated into n x m. Appropriate Structures will be initialized in Θ(n x m). Offers will be sent in O (n x m) and each offer take O (1). Maintaining set of unpaired hospitals in a queue will take O (1) for adding or removing.

e) Consider a Brute Force Implementation of the algorithm where you find all combinations of possible matching and verify if they are a stable marriage one by one. Give the runtime complexity of this brute force algorithm in Big O notation and explain why.

For a total of n hospital and m residents where m > n, and n hospitals have to choose between m residents, there will be a total of mCn combinations where each resident will be matched to each hospital. This information will need to be stored somewhere (in a list).

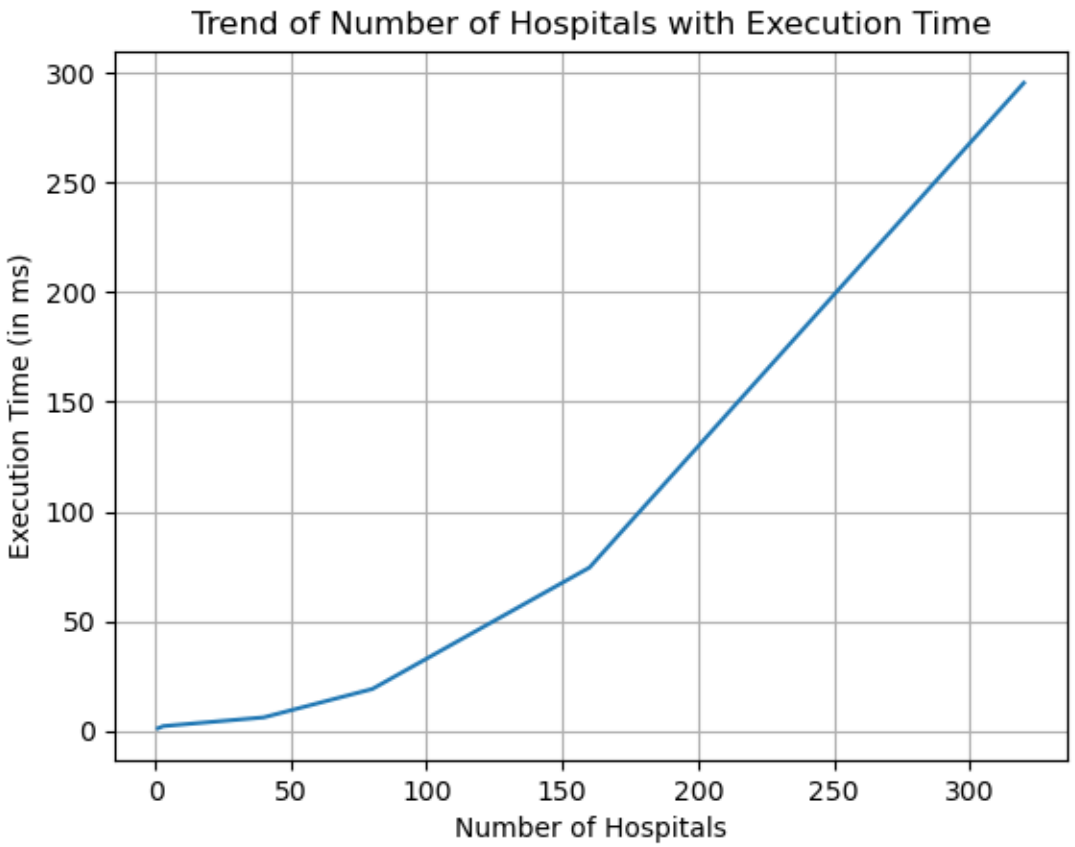
mCn means $\frac{m!}{n!(m-n)!}$ possibilities which will be in O (m!) since m! will incorporate n! terms too.

Therefore, Brute force will be in O (m!) where m is the number of residents since m > n.

f) (Graph on next page)

f) In the following two section you will implement a solution. In your report, use the 8 provided data files to plot the number of hospitals (x-axis) against the time in ms it takes for your code to run (y-axis).

Code was executed three times for each hospital and the average execution time was recorded.



Max time taken for 320 hospitals was around 295ms.