CSE 551 - Foundations of Algorithms

Assignment 2

Submitted by: Sannidhya Pathania (spathan3)

Solution 1:

For time complexity in the big-O notation i.e. the west case scenario,

Let us assume that all m hospitals made proposals to all students and the last students accepted the proposal for the given hospital. This means each hospital made n offers, so the number of total offers made = nm.

So,

while hospital h_i has positions -------------------------------- n * m

  h_i offers to student s_j -------------------------------- 1

  //assuming that we are not using any data structure to check students' availability, it will take O(n) time to check the availability.

  if s_j is free then -------------------------------- 1

    s_j accepts the offer -------------------------------- 1

  else (s_j is already committed to a hospital h_k) -------------------------------- 1

    // If a student is committed to a hospital we have to traverse the list to get the preference. *n*

    if (s_j prefers h_k > h_i) then --------------------------------

      s_j remains committed to h_k --------------------------------

    else (s_j becomes committed to h_i) -------------------------------- n

      positions at h_k + 1 --------------------------------

      positions at h_j - 1 --------------------------------

So $T(m,n) = c_1(n*m) * n((c_2(1+1+1+1) + c_3(n)))$

$= c_1(n*m) * n(c_{23}(4 + n))$

$= c_1*c_{23}(n*m*n*(4 + n))$

$= c_{123} (n^3m + 4 n^2m)$

$=> T(m,n) = O(n^3m)$

So **$O(n^3m)$** should be the time complexity.

<u>Assumption 1:</u> The data structure is used to maintain the availability of students which reduces the time to check the availability from O(n) to O(1)

So $T(m,n) = c_1(n*m) * (c_2(1+1+1+1) + c_3(n))$

$= c_1(n*m) * (c_{23}(4 + n))$

$= c_1*c_{23}(n*m * (4 + n))$

$= c_{123} (n^2m + 4\,nm)$

$\Rightarrow T(m,n) = O(n^2m)$

So **$O(n^2m)$** should be the time complexity.

<u>Assumption 2:</u> If we assume that data structure was used to store the preferences of students in such a way that it takes constant time complexity to check the preference. For this, we will create inverses of hospital preference for students which will take O(m) time for each student and so O(nm) for all students.

So $T(m,n) = c_0(nm) + c_1(n*m) * (c_2(1+1+1+1) + c_3(1))$

$= c_0(nm) + c_1(n*m) (c_{23}(5))$

$= c_0(nm) + c_1*c_{23}(n*m*5)$

$= c_{023} (6\ nm)$

$\Rightarrow T(m,n) = O(nm)$

So **$O(nm)$** should be the time complexity.

<u>Solution 2:</u>

To prove the claim that the student-hospital match in the algorithm is stable, we need to prove that there are no unstable pairs i.e. we will prove by contradiction.

Before moving further we need to note these two observations:

1. Hospitals propose to Students in decreasing order of their preference.
2. Once a student is matched, he/she never becomes unmatched; only " trade-up".

Let S be the current Galey-Shapely matching. Consider a hospital h_i, and a student s_j, such that pair $(h_i – s_j)$ is an unstable pair in S.

Then, there exists another Galey-Shapely stable matching S*, where both h_i and s_j are matched to their more preferable stable partner.

- Case 1: h_i never proposed to s_j

  As we know according to observation 1, hospitals propose to students in decreasing order of their preference. So, If ($h_i - s_j$) is unstable pair then it means hospital h_i prefers its S* matching to student s_j and it shouldn't have proposed to student s_j. If this was the case then there was no reason for hospital h_i to leave its partner in S*.

  Therefore,

  - Hospital h_i prefers student s_j over its partner in S*.
  - ($h_i - s_j$) cannot be unstable pair.

- Case 2: h_i proposed to s_j

  As we know according to observation 2, students only "trade-up". So, this means hospital h_i proposed to student s_j, but was rejected either right away or later. Student s_j prefers its S* partner to h_i. But if this was the case, why will student s_j leave S* partner to pair up with hospital h_i (because the student only trades up).

  Therefore,

  - ($h_i - s_j$) cannot be unstable pair.

In either case, ($h_i - s_j$) is stable pair. Thus the student-hospital match in the algorithm is stable.

Solution 3:

Let the 6 students be s1, s2, s3, s4, s5, s6 and 2 hospitals be h1 and h2.

Given, h1 can take 2 students and h2 can take 3 students.

Let the preference list for hospitals and students as shown below:

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1

s4	h1	h2
s5	h2	h1
s6	h1	h2

Manually running the algorithm:

Hospital h1 will propose to student s3. s3 is free and will accept the offer.

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1
s4	h1	h2
s5	h2	h1
s6	h1	h2

Hospital h1 will propose to student s1. s1 is free and will accept the offer.

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1
s4	h1	h2
s5	h2	h1
s6	h1	h2

Now, the requirement for hospital h1 is completed and h2 will start proposing. h2 will propose to s5 and s5 being free will accept the offer.

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1
s4	h1	h2
s5	h2	h1
s6	h1	h2

Now, h2 will propose to s2 now, s2 being free will accept the offer.

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1
s4	h1	h2
s5	h2	h1
s6	h1	h2

Now, h2 will propose to s3 now, s3 isn't free but prefers h2 over h1 so will accept the offer from h2.

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1
s4	h1	h2
s5	h2	h1
s6	h1	h2

Now h1 is short by 1 so, it will propose to next preference, s2, and s2 is already committed but will accept the offer from h1 as it prefers h1 over h2.

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1
s4	h1	h2
s5	h2	h1
s6	h1	h2

Now h2 is short by 1 so, it will propose to next preference, s4, and s4 is free and will accept the offer from h2.

Hospital	0th	1st	2nd	3rd	4th	5th
h1	s3	s1	s2	s6	s4	s5
h2	s5	s2	s3	s4	s1	s6

Student	0th	1st
s1	h1	h2
s2	h1	h2
s3	h2	h1
s4	h1	h2
s5	h2	h1
s6	h1	h2

Output generated by the algorithm is

- Hospital h1 = (s1, s2)
- Hospital h2 = (s5, s3, s4)

Solution 4:

IDE used = Visual Studio Code

Java Code: Implementing the algorithm

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
import java.util.Set;

public class GS_StudentHospital {
   public static void main(String[] args){
       System.out.println("****************************************");
       System.out.println("Assignment 2: Q4");
       Scanner sc = new Scanner(System.in);
       System.out.println("Enter the number of students");
       int students = sc.nextInt();
```

```
        System.out.println("Enter the number of hospitals");
        int hospitals = sc.nextInt();
        int[][] stuPref = new int[students][hospitals];
        for(int i = 0; i<students; i++){
            System.out.println("Enter the preference of hospitals for student "+
(i+1));
            for(int j=0; j<hospitals; j++)
                stuPref[i][j] = sc.nextInt() - 1;
        }
        int[][] hosPref = new int[hospitals][students];
        for(int i = 0; i<hospitals; i++){
            System.out.println("Enter the preference of students for hospital "+
(i+1));
            for(int j=0; j<students; j++)
                hosPref[i][j] = sc.nextInt() - 1;
        }
        int[] req = new int[hospitals];
        for(int i=0; i<hospitals; i++){
            System.out.println("Enter the requirement for hospital "+ (i+1));
            req[i] = sc.nextInt();
        }
        Set<Integer>[] match = getMatchings(stuPref, hosPref, req);
        System.out.println("Stable matchings are as follows");
        for(int i=0; i<hospitals; i++){
            System.out.print("h" + (i+1)+": ");
            for(int mat : match[i])
                System.out.print("s" + (mat+1)+ " ");
            System.out.println();
        }
        System.out.println("****************************************");
    }

    public static Set<Integer>[] getMatchings(int[][] stuPref, int[][] hosPref, int[]
req){
        int students = stuPref.length;
        int hospitals = hosPref.length;
        //prefNumHospAt to keep track of proposals made by hospitals
        int[] prefNumHospAt = new int[hospitals];
        //studMatch to keep track of which student is committed to which hospital
        int[] studMatch = new int[students];
        //keeping track of unmatched hospitals and students
        Queue<Integer> freeHosp = new LinkedList<>();
        Set<Integer> freeStud = new HashSet<>();
        for(int i=0; i< hospitals; i++)
```

```
            freeHosp.offer(i);
        for(int i=0; i< students; i++)
            freeStud.add(i);

        Set<Integer>[] match = new HashSet[hospitals];
        inversePref(stuPref);

        while(!freeHosp.isEmpty()){
            int currentHosp = freeHosp.peek();
            int studNum = prefNumHospAt[currentHosp];
            int proposeStu = hosPref[currentHosp][studNum];
            if(studNum==0)
                match[currentHosp] = new HashSet<>();
            prefNumHospAt[currentHosp]++;

            if(freeStud.contains(proposeStu)){
                req[currentHosp]--;
                freeStud.remove(proposeStu);
                match[currentHosp].add(proposeStu);
                studMatch[proposeStu] = currentHosp;
            }
            else{
                int previousHosp = studMatch[proposeStu];
                if(stuPref[proposeStu][previousHosp] >
stuPref[proposeStu][currentHosp]){
                    req[previousHosp]++;
                    match[previousHosp].remove(proposeStu);
                    if(req[previousHosp]==1)
                        freeHosp.offer(previousHosp);
                    req[currentHosp]--;
                    match[currentHosp].add(proposeStu);
                    studMatch[proposeStu] = currentHosp;
                }
            }
            if(req[currentHosp]==0)
                freeHosp.poll();
        }
        return match;
    }
    public static void inversePref(int[][] stuPref){
        //inverse the student preferences for constant time access.
        int hosp = stuPref[0].length;
        int[] helper = new int[hosp];
        for(int i=0; i<stuPref.length; i++){
```

```
            for(int j = 0; j<hosp; j++)
                helper[j] = stuPref[i][j];
            for(int j = 0; j<hosp; j++)
                stuPref[i][helper[j]] = j;
        }
    }
}
```

Solution 5:

The output from manually running the algorithm matches with the output generated by the program.

Stable matchings are as follows

h1: s1 s2

h2: s3 s4 s5

```
spats@Sannidhyas-MacBook-Air GS_StudentHospital %  /usr/bin/env /L
sInExceptionMessages -cp /Users/spats/Library/Application\ Support
tHospital_9f52a38e/bin GS_StudentHospital
***************************************
Assignment 2: Q4
Enter the number of students
6
Enter the number of hospitals
2
Enter the preference of hospitals for student 1
1 2
Enter the preference of hospitals for student 2
1 2
Enter the preference of hospitals for student 3
2 1
Enter the preference of hospitals for student 4
1 2
Enter the preference of hospitals for student 5
2 1
Enter the preference of hospitals for student 6
1 2
Enter the preference of students for hospital 1
3 1 2 6 4 5
Enter the preference of students for hospital 2
5 2 3 4 1 6
Enter the requirement for hospital 1
2
Enter the requirement for hospital 2
3
Stable matchings are as follows
h1: s1 s2
h2: s3 s4 s5
***************************************
spats@Sannidhyas-MacBook-Air GS_StudentHospital %
```