CMPT 435 - Fall 2022 - Dr. Labouseur

Final Project – Stable Marriage Problem

Genevieve Anderson
Genevieve.anderson1@marist.edu

December 14, 2022

1 Hospital Class

In my hospital class I create the framework for each hospital object. I begin by declaring the hospital id, each hospital's list of which residents they prefer, which residents are assigned to which hospital, and the capacity of the hospital [4-7]. Then, I initialize these variables in a constructor [9] and a parameterized constructor [16]. Lastly, I have a "toString()" function [23] which converts the id and array list into strings for printing purposes.

```
1  import java.util.*;
2
3  public class Hospital {
4      int id; // Hospital id
5      ArrayList<String> residentPrefList; // Hospital's list of which
        residents they prefer
6      ArrayList<Resident> residentAssigned; // Residents assigned to
       hospital
7      int cap; // Capacity of hospital
8
9      public Hospital() {
10         id = 0;
11         residentPrefList = new ArrayList<String>();
12         residentAssigned = new ArrayList<Resident>();
13         cap = 0;
14     }
15
16     public Hospital(int id, ArrayList<String> residentPrefList, int
        cap) {
17         this.id = id;
18         this.residentPrefList = residentPrefList;
19         residentAssigned = new ArrayList<Resident>();
```

```
20          this.cap = cap;
21      }
22
23      public String toString() {
24          String result = "id: " + id + "res pref list: " + Arrays.
        toString(residentPrefList.toArray());
25          return result;
26      }
27  }
```

2 Resident Class

In my resident class I create the framework for each resident object. I begin by declaring the resident id, each resident's list of which hospitals they prefer, and whether or not the resident is "free" [4-6]. Then, I initialize these variables in a constructor [8] and a parameterized constructor [14]. Lastly, I have a "toString()" function [20] which converts the id and array list into strings for printing purposes.

```
1  import java.util.*;
2
3  public class Resident {
4      int id; // Resident id
5      ArrayList<String> hospitalPrefList; // Resident's list of
       preferred hosptials
6      boolean free; // To determine whether they have been assignmd
       or not
7
8      public Resident() {
9          id = 0;
10         hospitalPrefList = new ArrayList<String>();
11         free = true;
12     }
13
14     public Resident(int id, ArrayList<String> hospitalPrefList) {
15         this.id = id;
16         this.hospitalPrefList = hospitalPrefList;
17         free = true;
18     }
19
20     public String toString() {
21         String result = "id: " + id + "hospital preferences list: "
        + Arrays.toString(hospitalPrefList.toArray());
22         return result;
23     }
24 }
```

3 Matching class

In my matching class I implement the stable marriage algorithm. First, I have my constructor [9] which takes in the list of residents and the list of hospitals. Then, I implement my "assign()" function [15] which is the stable marriage algorithm. I deal with one resident at a time. The first while loop [17] runs while the resident is free and has a list of preferred hospitals. Inside the loop, the resident proposes to its most preferred hospital [19]. Then, if the hospital is already fully subscribed [20], I find the least desired resident on the hospitals subscribed list [22]. I remove the "worst" resident from the hospital [23] and assign it to be free [24]. I then put the better resident in the old "worst" residents spot [27] and assign it to be un-free [28]. Now, moving on I see if the hospital has full capacity [29]. I find the new worst resident in the hospitals subscribed list [30]. Then, inside the for loop I go through for each successor of the "new worst" resident on the hospitals subscribed list and remove the new worst resident and the hospital from each others lists [33-37]. Lastly, I have a simple for loop which reassess which residents are free and which ones are not [40-45].

In the end of my matching class I have my "getWorst()" function [50]. Inside a nested for loop I go through the list from the back, since the back is where the worst resident would be located [52-60]. Through this I find the id of the worst resident and implement it in my stable marriage algorithm.

```
1  import java.util.*;
2
3  public class matching {
4
5      ArrayList<Resident> residents;
6      ArrayList<Hospital> hospitals;
7
8      // Constructor
9      public matching(ArrayList<Resident> residents, ArrayList<
       Hospital> hospitals) {
10         this.residents = residents;
11         this.hospitals = hospitals;
12     }
13
14     // Function for assigning
15     public void assign() {
16         Resident r = residents.get(0);
17         while (r.free == true && r.hospitalPrefList.size() > 0) {
18             int id = Integer.parseInt(r.hospitalPrefList.get(0).
       substring(1)); // Getting the id of the first hospital from the
        residents preferences
19             Hospital h = hospitals.get(id - 1); // Connecting the
       residents preference to the actual hospital in the array
20             if (h.residentAssigned.size() == h.cap) { // if h is
       fully subscribed
21                 // Based on h subscribed list, we are going to see
       whether or not h will be happy with the matches proposed by r
22                 Resident worstResident = getWorst(h); // Figuring
       out the least desired resident in the subscribed list
23                 h.residentAssigned.remove(worstResident); //
```

```
    Removing the resident from the hospitals subscribed list
24                  worstResident.free = true; // Assign resident to be
     free
25              }
26              System.out.println("Match found! Resident " + r.id + ",
     " + " Hospital " +  h.id);
27              h.residentAssigned.add(r); // Putting the better
    resident in the worst residents spot
28              r.free = false;
29              if (h.residentAssigned.size() == h.cap) {
30                  Resident newWorst = getWorst(h); // Worst resident
    assigned to hospital
31                  String f = "r" + newWorst.id;
32                  String s = "h" + h.id;
33                  for (int i = h.residentPrefList.indexOf(f) + 1; i <
     h.residentPrefList.size(); i ++) { // Loop for deleting all
    residents after the found worst resident
34                      String succesor = h.residentPrefList.get(i);
35                      h.residentPrefList.remove(i); // Remove from
    hospital
36                      residents.get(Integer.parseInt(succesor.
    substring(1)) - 1).hospitalPrefList.remove(s);
37                  }
38              }
39              // Reassess which residents are free
40              for (int i = 0; i< residents.size(); i++) {
41                  if (residents.get(i).free == true) {
42                      r = residents.get(i);
43                      i = residents.size(); // Stop looping
44                  }
45              }
46          }
47      }
48
49      // Function for figuring out hospitals least desired resident
    after the residents propose to the hospitals
50      public Resident getWorst(Hospital h) { // Taking in hospital
    because this is where the list of desired residents is
51          Resident worst = null;
52          for (int i = h.residentPrefList.size() - 1; i >= 0; i--) {
    // Starting from the back of the list to find the worst one
53              for (int j = 0; j < h.residentAssigned.size(); j++) {
54                  int id = Integer.parseInt(h.residentPrefList.get(i)
    .substring(1)); // Getting the number for the worst resident
55                  if (id == h.residentAssigned.get(j).id) { // Going
    through assigned residents
56                      worst = h.residentAssigned.get(j);
57                      j = h.residentAssigned.size();
58                      i = -1;
59                  }
60              }
61          }
62          return worst;
63      }
64  }
```

4 Unranked Class

In my unranked class I implement the second part of the assignment where residents rank their preferences, but the hospitals do not. I decided to place residents in their desired hospitals on a first come first serve basis.

I begin by declaring the array list of residents and hospitals [4, 5] and then initialize these variables in a constructor [8]. Then, I have my "unrankedAssign()" function [14]. Here, I go through the residents one by one inside a for loop [17]. Then, I get the id of the first hospital on the residents list [18,19]. Then, inside a while loop, I check that the hospital has not reached its capacity [20]. If the hospital is not full, then I place the resident inside.

"Stability" in this context does not have much meaning. I do not think that this algorithm would ever be beneficial in a real world application because it solely depends on the order in which the residents are in, that determines what hospital they will be in. Perhaps a better way of implementing this algorithm would have been to shuffle the residents prior so that there was an element of randomness in the order in which the residents are placed in hospitals. I created my own text file with a modified version of the data from the first part of the assignment.

```
1  import java.util.*;
2
3  public class Unranked {
4      ArrayList<Resident> residents;
5      ArrayList<Hospital> hospitals;
6
7      // Constructor
8      public Unranked(ArrayList<Resident> residents, ArrayList<
       Hospital> hospitals) {
9          this.residents = residents;
10         this.hospitals = hospitals;
11     }
12
13     // Function for assigning
14     public void unrankedAssign() {
15         int index = 0;
16         // I will be placing residents in their desired hosptials
       on a first come first serve basis
17         for (int i = 0; i < residents.size(); i++ ) { // going
       through the residents to put them in hospitals
18             String firstHospID = residents.get(i).hospitalPrefList.
       get(0); // getting the first hosptial from the residents list
19             Hospital hospital = hospitals.get(Integer.parseInt(
       firstHospID.substring(1)) - 1); // Getting the number of the
       hospital
20             while (hospital.cap == hospital.residentAssigned.size()
       ) { // while the hospital has not reached its capacity
21                 index++; // moving on to the next hosptial
22                 firstHospID = residents.get(i).hospitalPrefList.get
       (index);
23                 hospital = hospitals.get(Integer.parseInt(
       firstHospID.substring(1)) - 1); // Getting the number of the
```

```
    hospital
24              }
25              hospital.residentAssigned.add(residents.get(i));
26              System.out.println("Match found! Resident " + residents
    .get(i).id + ", Hospital " + hospital.id);
27          }
28      }
29  }
```

5 Main Program

In my main class I begin by creating an array list of residents and hospitals [9, 10]. For the first part of the assignment, I used a try and catch statement for uploading the file. Inside a while loop [17], I parse the file. If the file indicated a resident, I put each resident as their own object, and put their hospital preferences in a string array. Each instance of a resident was added to the residents array [32] which holds all residents. If the file indicated a hospital, I parsed the file to figure out the capacity and the id. I put the hospitals preferences for residents in a string array. Each hospital is its own object, and I added each instance of a hospital to the hospitals array which holds all hospitals [41]. Then, I creating a new instance of matching and run the stable marriage algorithm producing the expected results [49, 50]. I have pasted my results below! Note that the results first indicate resident 6 with hospital 4, but the algorithm corrects itself and further down places resident 6 with hospital 3.

Match found! Resident 1, Hospital 3
Match found! Resident 2, Hospital 1
Match found! Resident 3, Hospital 4
Match found! Resident 4, Hospital 3
Match found! Resident 5, Hospital 1
Match found! Resident 6, Hospital 4
Match found! Resident 7, Hospital 2
Match found! Resident 8, Hospital 1
Match found! Resident 9, Hospital 4
Match found! Resident 6, Hospital 3
Match found! Resident 10, Hospital 1
Match found! Resident 11, Hospital 5

For the second part of the assignment, I used a try and catch statement for parsing the file. I went through the same process as above if the text file indicated a resident. Then, if the text file indicated the hospital I did the same as above again, however, since the hospital does not have any preferences that code was removed. Then, I create a new unranked instance and run my unranked assigning algorithm which produces my expected results [90, 91].

Match found! Resident 1, Hospital 1
Match found! Resident 2, Hospital 1
Match found! Resident 3, Hospital 4
Match found! Resident 4, Hospital 3
Match found! Resident 5, Hospital 4
Match found! Resident 6, Hospital 1
Match found! Resident 7, Hospital 2
Match found! Resident 8, Hospital 3
Match found! Resident 9, Hospital 1
Match found! Resident 10, Hospital 3

Match found! Resident 11, Hospital 5

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.*;
import java.util.ArrayList;

public class Main {
    public static void main (String[] args) {
        ArrayList<Resident> residents = new ArrayList<Resident>();
        ArrayList<Hospital> hospitals = new ArrayList<Hospital>();

        System.out.println("------------------------------ Part 1: Ranked ------------------------------");

        try { //Trying to find the file
            File file = new File("final-project-data.txt");
            Scanner sc = new Scanner(file);
            while (sc.hasNextLine()) {
                String item = sc.nextLine();
                if (item.startsWith("r")) {
                    // index of will find where the colon is in each string - its in different spots each time because sometimes there are double digits
                    int id = Integer.parseInt(item.substring(1,item.indexOf(':'))); // putting r1,...,r11 each in their own resident object
                    String[] resHospPref = null;
                    if (id > 9) {
                        resHospPref = item.substring(5).split(" "); // putting the hospital preferences of the resident into a string array
                    }
                    else {
                        resHospPref = item.substring(4).split(" "); // putting the hospital preferences of the resident into a string array
                    }
                    ArrayList<String> arrayResHospPref= new ArrayList<String>();
                    Collections.addAll(arrayResHospPref, resHospPref); // putting as an array list
                    Resident res = new Resident(id, arrayResHospPref); // creating the instance of a resident
                    residents.add(res); // adding each instance of a resident to the array list which holds all residents
                }
                else if (item.startsWith("h")) {
                    int capacity = Integer.parseInt(item.substring(item.indexOf(':')+2, item.indexOf(':')+3)); // For figuring out what the capacity is for each hospital
                    int id = Integer.parseInt(item.substring(1,item.indexOf(':'))); // Figuring out hospital id
                    String[] hospResPref = item.substring(item.indexOf('-')+2).split(" "); // putting each hospital in its own object
```

```
38                  ArrayList<String> arrayHospResPref = new
    ArrayList<String>();
39                  Collections.addAll(arrayHospResPref,
    hospResPref);
40                  Hospital hosp = new Hospital(id,
    arrayHospResPref, capacity); // Adding each instance of a
    hospital to the array list which holds all hospitals
41                  hospitals.add(hosp);
42              }
43          }
44      }
45      catch (FileNotFoundException e) { //If we cant find the
    file
46          e.printStackTrace();
47      }
48
49      matching match = new matching(residents, hospitals);
50      match.assign();
51
52      System.out.println("------------------------------ Part 2:
    Unranked ------------------------------");
53
54      // Unranked
55      residents.clear();
56      hospitals.clear();
57
58      try { //Trying to find the file
59          File file = new File("Unranked.txt");
60          Scanner sc = new Scanner(file);
61          while (sc.hasNextLine()) {
62              String item = sc.nextLine();
63              if (item.startsWith("r")) {
64                  // index of will find where the colon is in
    each string - its in different spots each time because
    sometimes there are double digits
65                  int id = Integer.parseInt(item.substring(1,item
    .indexOf(':'))); // putting r1,...,r11 each in their own
    resident object
66                  String[] resHospPref = null;
67                  if (id > 9) {
68                      resHospPref = item.substring(5).split(" ");
     // putting the hospital preferences of the resident into a
    string array
69                  }
70                  else {
71                      resHospPref = item.substring(4).split(" ");
     // putting the hospital preferences of the resident into a
    string array
72                  }
73                  ArrayList<String> arrayResHospPref= new
    ArrayList<String>();
74                  Collections.addAll(arrayResHospPref,
    resHospPref); // putting as an array list
75                  Resident res = new Resident(id,
    arrayResHospPref); // creating the instance of a resident
76                  residents.add(res); // adding each instance of
    a resident to the array list which holds all residents
```

```
77                  }
78                  else if (item.startsWith("h")) {
79                      int capacity = Integer.parseInt(item.substring(
    item.indexOf(':')+2, item.indexOf(':')+3)); // For figuring out
     what the capacity is for each hospital
80                      int id = Integer.parseInt(item.substring(1,item
    .indexOf(':'))); // Figuring out hospital id
81                      Hospital hosp = new Hospital(id, null, capacity
    ); // Adding each instance of a hospital to the array list
    which holds all hospitals
82                      hospitals.add(hosp);
83                  }
84              }
85          }
86          catch (FileNotFoundException e) { // If we cant find the
    file
87              e.printStackTrace();
88          }
89
90          Unranked unranked = new Unranked(residents, hospitals);
91          unranked.unrankedAssign();
92      }
93  }
```