CMPT 435 - Fall 2022 - Dr. Labouseur

Final Project

Shannon Brady

shannon.brady2@Marist.edu

December 8, 2022

1 Main

1. Line 24-30: Get the number of residents and hospitals in the file. Initialize arrays of Residents and Hospitals.
2. Line 32-48: Create all residents and place in residents array. Get each of the residents' preferences and place in a HashTable of resident preferences.
3. Line 52-77: Create all hospitals and place in hospitals array. Get each of the hospitals' preferences and place in a HashTable of hospital preferences.
4. Line 91-93: Create the stable matches and print the final pairings.
5. Line 100-103: Create a list of resident first choices.
6. Line 106-113: Calculate a sudo 'acceptance rate' for each hospital. This will be used to rank the hospitals later.
7. Line 115-117: Create a stable matches variation where hospitals do not rank residents. Print the final pairings.

```
1   import java.io.File;
2   import java.io.FileNotFoundException;
3   import java.util.ArrayList;
4   import java.util.Collections;
5   import java.util.Scanner;
6
7   public class Main {
8       public static void main(String args[]) {
9           Resident[] residents = null;
10          HashTable residentsPref = null;
11
12          Hospital[] hospitals = null;
13          HashTable hospitalsPref = null;
14
```

```java
        try {
            File file = new File("test.txt");
            Scanner myReader = new Scanner(file);
            int i = 0;
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();

                // Get the number of residents and hospitals
                // Use these values to intialize resident and hospital arrays, respectively
                if (data.contains("Config")) {
                    String[] configs = data.split(":")[1].trim().split(" ");
                    String numResidents = configs[0];
                    String numHospitals = configs[1];
                    residents = new Resident[Integer.parseInt(numResidents)];
                    hospitals = new Hospital[Integer.parseInt(numHospitals)];
                }

                if (data.startsWith("r")) {
                    residentsPref = new HashTable(residents.length+1);
                    while (data.startsWith("r")) {
                        String[] dataArr = data.split(": ");
                        String resKey = dataArr[0].replaceAll("[^0-9]", "");
                        String[] preferences = dataArr[1].split(" ");

                        // Place the new resident in the resident array
                        residents[i] = new Resident(dataArr[0], preferences[0]);

                        // Put resident preferences in a hashtable
                        for (int k=0; k<preferences.length; k++) {
                            residentsPref.put(Integer.parseInt(resKey), preferences[k]);
                        }
                        data = myReader.nextLine();
                        i++;
                    }

                }

                if (data.startsWith("h")) {
                    hospitalsPref = new HashTable(hospitals.length+1);
                    i = 0;
                    while (data.startsWith("h")) {
                        String[] dataArr = data.split(": ");
                        String hosKey = dataArr[0].replaceAll("[^0-9]", "");

                        String[] hosData = dataArr[1].split(" - ");
                        String capacity = hosData[0];
                        String[] preferences = hosData[1].split(" ");

                        // Place the new hospital in the hospital array
                        hospitals[i] = new Hospital(dataArr[0], Integer.parseInt(capacity));

                        // Put the hospital preferences in a hashtable
                        for (int k=0; k<preferences.length; k++) {
                            hospitalsPref.put(Integer.parseInt(hosKey), preferences[k]);
                    }

                    if (myReader.hasNextLine()) {
                        data = myReader.nextLine();
                    } else {
                        break;
                    }
                    i++;
                }

            }
        }
        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
        // -------------------------------------------------------------------------------

        StableMatching matchMaker = new StableMatching(residents, hospitals, residentsPref, hospitalsPref);

        // Pt. 1
        System.out.println("Stable Matching, Pt I");
        HashTable myStableMatches = matchMaker.doMatching();
        myStableMatches.printPairings();
        System.out.println();

        // -------------------------------------------------------------------------------

        // Pt.2
        // Create list of resident first choices
        ArrayList<String> firstChoices = new ArrayList<String>();
        for (int i=0; i<residents.length; i++) {
            firstChoices.add(residents[i].getFirstChoice());
        }
        // Set the 'acceptance rate' for each hospital
        // Multiply the number of occurrences by a constant factor to ensure rate < 1
        for (int i=0; i<firstChoices.size(); i++) {
            double occurrences = Collections.frequency(firstChoices, firstChoices.get(i));
            for (int j=0; j<hospitals.length; j++) {
                if (firstChoices.get(i).compareTo(hospitals[j].getName()) == 0) {
                    hospitals[j].setAcceptanceRate(hospitals[j].getCapacity() / (occurrences*10));
                }
            }
        }

        System.out.println("Stable Matching, Pt II");
        HashTable moreMatches = matchMaker.doMatchingVariation();
        moreMatches.printPairings();
    }
}
```

2 StableMatching

1. The StableMatching class has the following attributes:
   a) residents: an array of Residents
   b) hospitals: an array of Hospitals
   c) residentPref: a HashTable of resident preferences
   d) hospitalPref: a HashTable of hospital preferences

2.1 Stable Matching Pt. 1

1. Line 24-27: Create a stack of free residents. Push all Residents to the stack.
2. Line 30: Create an array of already assigned residents.
3. Line 33-37: While there are still free residents, pop the next resident and get their preferences.
4. Line 41-140: Loop through each of the resident's preferences
   a) Line 60-63: If the resident isn't already assigned, match the resident to the current hospital. Add the Resident to the assigned residents.
      i. Line 66-78: If the hospital is now full, remove the worst resident from the hospital preferences. Remove the hospital from the resident preferences.
   b) Line 85-11: Otherwise, check if there is an already matched resident we can switch with the current resident.
      i. Line 94-109: The hospital prefers the current resident over the one currently assigned. The removed resident becomes free again. Match the current resident with the current hospital.
   c) Line 114-135: Check if the hospital is now full. Same process as Line 66-78.

2.2 Stable Matching Pt. 2

1. Assumption: Hospitals with a lower 'acceptance rate' are inherently better, and therefore are more likely to be a resident's first choice.
2. Stability in this context: All unmatched residents have an equal chance at getting into the most selective hospital. More selective hospitals get to select their residents before less selective hospitals.
3. Residents that do not get their first choice are assigned the next available opening in their preferences list (if there is one). Given this, residents must choose there first choice on the basis of risk vs. outcome. Choosing a less selective hospital is safer based on the above assumption (less competition).
4. 'Better' hospitals will have the greatest ratio of residents that chose said hospital as their first choice to residents that had a different first choice. This means that the best hospitals have the greatest number of residents that actually want to be there. The contrary is true for the worst hospitals. This reinforces existing reputations.
5. Line 151: Shuffle the residents.
6. Line 154-157: Create an array of free residents
7. Line 160-161: Sort the hospitals from most selective to least selective

8. Line 163-181: Loop through all of the hospitals. If one of the free residents' first choice is that given hospital, match the resident and the hospital. Remove the resident from the free residents array. Break if any given hospital reaches capacity.
9. Line 184-206: For all residents that did not get their first choice, loop through each of their hospital preferences until an opening is found (if there is one). Match the resident with that given hospital.
10. Line 211-219: Method used to get the capacity of a Hospital based on a given hospital name.
11. Line 221-233: Method used to shuffle an array of Residents.

```java
1  import java.util.Random;
2  
3  public class StableMatching {
4  
5      private Resident[] residents = null;
6      private Hospital[] hospitals = null;
7      private HashTable residentsPref = null;
8      private HashTable hospitalsPref = null;
9  
10     public StableMatching(Resident _residents[],
11                           Hospital _hospitals[],
12                           HashTable _residentsPref,
13                           HashTable _hospitalsPref) {
14         this.residents = _residents;
15         this.hospitals = _hospitals;
16         this.residentsPref = _residentsPref;
17         this.hospitalsPref = _hospitalsPref;
18     }
19 
20     public HashTable doMatching() {
21         HashTable matches = new HashTable(hospitals.length+1);
22 
23         // All residents start out as free
24         Stack freeResidents = new Stack();
25         for (int i=residents.length-1; i>=0; i--) {
26             freeResidents.push(residents[i].getName());
27         }
28 
29         // Array of residents already assigned a hospital
30         String[] assignedResidents = new String[residents.length];
31 
32 
33         while (!freeResidents.isEmpty()) {
34             // Get the next resident and their hospital preferences
35             String currResident = freeResidents.pop().getName();
36             int resKey = Integer.parseInt(currResident.replaceAll("[^0-9]", ""));
37             LinkedList currResidentPref = residentsPref.get(resKey);
38 
39 
40             // Loop through the current resident's hospital preferences
41             Node hospital = currResidentPref.getHead();
42             while (hospital != null) {
43 
44                 // Get the hospital name, capacity, and key
```

```
                String hospitalName = hospital.getName();
                int hospitalCapacity = getCapactiy(hospital.getName());
                int hosKey = Integer.parseInt(hospitalName.replaceAll("[^0-9]", ""));

                // Check if the resident has already been assigned a hospital
                boolean alreadyAssigned = false;
                for (int i=0; i<assignedResidents.length; i++) {
                    if (assignedResidents[i] != null && currResident.compareTo(assignedResidents[i]) == 0) {
                        alreadyAssigned = true;
                    }
                }

                // Match all unassigned residents
                if (!alreadyAssigned) {
                    // If a hospital has room, provisionally assign the resident
                    if (matches.get(hosKey) == null ||
                            matches.get(hosKey).getSize() < hospitalCapacity) {
                        matches.put(hosKey, currResident);
                        assignedResidents[resKey-1] = currResident;

                        // Check if the hospital is now full
                        if (matches.get(hosKey).getSize() == hospitalCapacity) {
                            // Remove the worst candidate
                            LinkedList currHospitalPref = hospitalsPref.get(hosKey);

                            // Remove the resident from hospital preferences
                            int i = currHospitalPref.getSize()-1;
                            String removedRes = currHospitalPref.removeAt(i);

                            // Remove the hospital from resident preferences
                            int removeKey = Integer.parseInt(removedRes.replaceAll("[^0-9]", ""));
                            LinkedList removedPref = residentsPref.get(removeKey);
                            removedPref.removeNode(hospital.getName());
                        }

                        // Resident has been assigned to a hospital at this point, so we can go to the next resident
                        break;
                    } else {
                        // Resident is already assigned
                        // Check if we can switch the current resident with another already matched resident
                        LinkedList currHospitalPref = hospitalsPref.get(hosKey);
                        Node activeAssignment = null;

                        // Loop through current matches associated with current hospital, starting at the end of the list
                        int i = matches.get(hosKey).getSize()-1;
                        while (i >= 0) {
                            activeAssignment = matches.get(hosKey).getNode(i);

                            // The hospital prefers the current resident over the one currently assigned
                            if (currHospitalPref.getIndex(activeAssignment.getName()) > currHospitalPref.getIndex(currResident))
                                String removedResident = matches.get(hosKey).removeAt(i);

                                // We now have to reassign the removed resident
                                for (int j =0; j<assignedResidents.length; j++) {
                                    if (assignedResidents[j] != null && assignedResidents[j].compareTo(removedResident) == 0) {
                                        assignedResidents[j] = null;
                                        freeResidents.push(removedResident);
                                    }
                                }
                                // Create the new match with the current resident
                                matches.put(hosKey, currResident);

                                // We already found a matched resident that is a worse candidate than the current resident, so u
                                break;
                            }
                            i--;
                        }

                        // Check if the hospital is now full
                        if (matches.get(hosKey).getSize() == hospitalCapacity) {
                            currHospitalPref = hospitalsPref.get(hosKey);

                            // Remove the resident from hospital preferences
                            i = currHospitalPref.getSize()-1;
                            String removed = currHospitalPref.removeAt(i);

                            // Remove the hospital from the resident preferences
                            int removeKey = Integer.parseInt(removed.replaceAll("[^0-9]", ""));
                            LinkedList removedPref = residentsPref.get(removeKey);
                            removedPref.removeNode(hospital.getName());

                            // We have to reassign the removed resident now so...
                            for (int j=0; j<assignedResidents.length; j++) {
                                if (assignedResidents[j] != null && removed.compareTo(assignedResidents[j]) == 0) {
                                    assignedResidents[j] = null;

                                    // Remove the initial match associated with the removed resident and push to free residents
                                    freeResidents.push(removed);
                                }
                            }
                        }
                        // Resident has been assigned to a hospital at this point, so we can go to the next resident
                        break;
                    }
                }
                hospital = hospital.getNext();
            }
        }
        return matches;
    }

    // Matching variation where hospitals don't rank residents
    public HashTable doMatchingVariation() {
        HashTable matches = new HashTable(hospitals.length+1);

        // Place residents in random order
        shuffle(residents);

```

```java
        // All residents start out as free
        Resident[] freeResidents = new Resident[residents.length];
        for (int i=0; i<residents.length; i++) {
            freeResidents[i] = residents[i];
        }

        // Sort the hospitals from most selective to least selective
        InsertionSort insertionSortObj = new InsertionSort();
        insertionSortObj.sort(hospitals);

        for (int i=0; i<hospitals.length; i++) {
            int hosKey = Integer.parseInt(hospitals[i].getName().replaceAll("[^0-9]", ""));

            // Place each (randomly selected) resident in their first choice as long as capacity hasn't been reached
            for (int j=0; j<freeResidents.length; j++) {
                if (freeResidents[j] != null) {
                    Resident currResident = freeResidents[j];
                    if (currResident.getFirstChoice().compareTo(hospitals[i].getName()) == 0) {
                        freeResidents[j] = null;
                        matches.put(hosKey, currResident.getName());

                        // Check if hospital reached capacity
                        if (matches.get(hosKey).getSize() == hospitals[i].getCapacity()) {
                            break;
                        }
                    }
                }
            }
        }

        // Loop through all the residents that didn't get their first choice
        for (int i=0; i<freeResidents.length; i++) {
            if (freeResidents[i] != null) {
                int resKey = Integer.parseInt(freeResidents[i].getName().replaceAll("[^0-9]", ""));
                LinkedList currResidentPref = residentsPref.get(resKey);
                int j = 1;
                // Loop through each of the residents preferences until a spot is found
                while (freeResidents[i] != null) {
                    if (currResidentPref.getNode(j) == null) {
                        break;
                    } else {
                        String nextChoice = currResidentPref.getNode(j).getName();
                        int hosKey = Integer.parseInt(nextChoice.replaceAll("[^0-9]", ""));
                        int hospitalCapacity = getCapactiy(nextChoice);

                        if (matches.get(hosKey) == null || matches.get(hosKey).getSize() < hospitalCapacity) {
                            matches.put(hosKey, freeResidents[i].getName());
                            freeResidents[i] = null;
                        }
                        j++;
                    }
                }
            }
        }
        return matches;
    }

    // Returns the capcaity of a given hospital
    public int getCapacity(String hospitalName) {
        int capacity = 0;
        for (int i=0; i<hospitals.length; i++) {
            if (hospitals[i].getName().compareTo(hospitalName) == 0) {
                capacity = hospitals[i].getCapacity();
            }
        }
        return capacity;
    }

    public void shuffle(Resident[] array) {
        Random randomGen = new Random();
        int n = 0; // number of shuffled elements
        while (n < array.length-1) {
            n++;
            int randIndex = randomGen.nextInt(n); // select a random index value

            // swap the next array element with a random element
            Resident temp = array[n];
            array[n] = array[randIndex];
            array[randIndex] = temp;
        }
    }
}
```

3 Resident

1. The Resident class represents each resident in the Stable Matching scenario. Each Resident has the following attributes:
   a) name: resident name
   b) firstChoice: the resident's first hospital choice

```java
class Resident {

    private String name = null;
    private String firstChoice = null;

    public Resident(String _name, String _firstChoice) {
        this.name = _name;
        this.firstChoice = _firstChoice;
    }

    public String getName() {
        return this.name;
    }

    public String getFirstChoice() {
        return this.firstChoice;
    }
}
```

4

1. The Hospital class represents each hospital in the Stable Matching scenario. Each Hospital has the following attributes:
   a) name: hospital name
   b) capacity: hospital capacity
   c) acceptanceRate: a calculated sudo acceptance rate for each hospital

```
import java.text.DecimalFormat;

class Hospital {

    public static final DecimalFormat df = new DecimalFormat("0.00");

    private int capacity = 0;
    private String name = null;
    private double acceptanceRate = 0;

    public Hospital(String _name, int _capacity) {
        this.capacity = _capacity;
        this.name = _name;
        this.acceptanceRate = 0;
    }

    public void print() {
        System.out.println("Hospital: " + this.name +
                        "\nAcceptance Rate: " + df.format(this.acceptanceRate * 100));
    }

    public int getCapacity() {
        return this.capacity;
    }

    public String getName() {
        return this.name;
    }

    public double getAcceptanceRate() {
        return this.acceptanceRate;
    }

    public void setAcceptanceRate(double newRate) {
        this.acceptanceRate = newRate;
    }
}
```