CSE551 Assignment 2

Jicheng Zhi

January 25, 2021

1. Calculate the time complexity in big-O notation of the algorithm. Show the steps of your work.

```
GS(hospitals, students)
 1  Q = Queue(hospitals)
 2  while Q is not empty
 3       h_i = Q.Poll()
 4       s = h_i.next-preference
 5       if s is free
 6            s.Accept(h_i)
 7       else
 8            h_j = s.current-offer
 9            if s prefers h_i to h_j
10                 s.Not-Accept(h_j)
11                 s.Accept(h_i)
12                 if h_j has positions
13                      Q.Offer(h_j)
14       if h_i has positions
15            Q.Offer(h_i)
```

Suppose we have N hospitals and M students. All hospitals have same preference $s_1, s_2, s_3, \ldots, s_{n-1}, s_n$, and all students also have same preference $h_n, h_{n-1}, \ldots, h_2, h_1$ (Note here we assume all hospitals has M positions, and all hospital has all students on preference list and vice versa).

In the worst case, if we add hospitals to the queue as sequence of $h_1, h_2, \ldots, h_{n-1}, h_n$. Then for each student s, the program will loop N times because the latter offer from h_i is always better than current one h_j, and h_j will be added to queue for it's next preference. So the program will loop at most $N \cdot M$ times, hence the time complexity is $O(MN)$.

2. Prove the claim that the medical student-hospital match generated in the algorithm is stable.

From the algorithm we can see:

(a) student always picks the best offer among all offers received

(b) hospital always gives offer to students it likes more

For a matching result (s, h), h is the best offer student s can choose among all the offers he/she received, but he/she may prefer hospital h', whose last choice is student s'. (s', h') means h' prefers s' to s, and h' has filled all positions before giving offer to s. (s, h') must be unstable. So both students and hospitals in result are stable.

3. Create an input set (test case) with 6 medical students and two hospitals, with h_1 can take 2 students and h_2 can take 3 students. Use the test case to manually test algorithm and give the output (match) generated by the algorithm.

   Hospitals:

   (a) Arkham, 2 positions, preference: Bob, Alice

   (b) Hilltop, 3 positions, preference: Carl, Alice, Bob

   Students:

   (a) Alice, preference: Arkham, Hilltop

   (b) Bob, preference: Hilltop, Arkham

   (c) Carl, preference: Hilltop Arkham

   (d) Danny, preference: Arkham Hilltop

   (e) Ellen, preference: Hilltop, Arkham

   (f) Frank, preference: Arkham Hilltop

   Procedures:

   (a) i. dequeue and get Arkham, try giving offer to Bob
   ii. Bob has no offer, so he/she accepts hospital Arkham
   iii. Arkham has 1 open positions, so enqueue Arkham

   (b) i. dequeue and get Hilltop, try giving offer to Carl
   ii. Carl has no offer, so he/she accepts hospital Hilltop
   iii. Hilltop has 2 open positions, so enqueue Hilltop

   (c) i. dequeue and get Arkham, try giving offer to Alice
   ii. Alice has no offer, so he/she accepts hospital Arkham

   (d) i. dequeue and get Hilltop, try giving offer to Alice
   ii. Alice has better offer from hospital Hilltop, he/she refuses to change
   iii. Hilltop has 2 open positions, so enqueue Hilltop

   (e) i. dequeue and get Hilltop, try giving offer to Bob
   ii. Bob reject current offer Arkham and accepts hospital Hilltop

4. Use a programming language (C++, Java, or Python) to implement the algorithm.

   Please check $Homework2.java$ which is in the same directory of this file. The code will also be posted at the end of this file.

   Note: all Map implementations are $LinkedHashMap$, so you don't have to worry about the iteration sequence!

5. Use the test case created in question 3 to test the program and submit the output generated by the program.

   $Arkham : [Bob, Alice]$
   $Hilltop : [Carl]$

Code in *Homework2.java*.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.Scanner;

public class Homework2 {

    static class Hospital {
        String name;

        // since we never look back, iterator is well enough to go
        Iterator<String> preferences;

        int positions;

        LinkedHashSet<String> offers;


        Hospital(String name, int positions, List<String> preferences) {
            this.name = name;
            this.preferences = preferences.iterator();
            this.positions = positions;
            this.offers = new LinkedHashSet<>();
        }

        public String nextPreference(){
            return this.preferences.next();
        }

        public boolean isOpen() {
            return positions > 0 && this.preferences.hasNext();
        }

        public void offerTo(String student) {
            positions--;
            offers.add(student);
        }

        public void cancelOffer(String student) {
            positions++;
            offers.remove(student);
        }
```

```
}

static class Student {

    String name;

    String offer;

    // hospital name => ranking
    Map<String, Integer> preference;

    Student(String name, List<String> preferences) {
        this.name = name;
        this.preference = new HashMap<>();

        for (int i = 0; i < preferences.size(); i++) {
            preference.put(preferences.get(i), i);
        }
    }

    void acceptOffer(String hospital) {
        this.offer = hospital;
    }

    // haven't receive any offer
    boolean isFree() {
        return offer == null;
    }

    // prefer [h] than current offer?
    boolean prefer(String h) {
        if (isFree()) {
            throw new IllegalStateException("student has no offer");
        }
        return preference.get(h) < preference.get(offer);
    }

    @Override
    public String toString() {
        return String.format("%s: %s", name, preference);
    }
}

public static void galeShapley(Map<String, Hospital> hospitals,
                               Map<String, Student> students) {

    Queue<Hospital> queue = new LinkedList<>(hospitals.values());

    while (!queue.isEmpty()) {
```

```
            Hospital h = queue.poll();
            String studentName = h.nextPreference();
            Student s = students.get(studentName);


            if (s == null) {
                throw new RuntimeException("no user find: " + studentName);
            }

            if(s.isFree() || s.prefer(h.name)){
                if (!s.isFree()) {
                    // return offer of current hospital
                    Hospital prev = hospitals.get(s.offer);
                    prev.cancelOffer(s.name);

                    if (prev.isOpen()) {
                        queue.add(prev);
                    }
                }

                s.acceptOffer(h.name);
                h.offerTo(s.name);
            }

            if (h.isOpen()) {
                queue.add(h);
            }
        }
    }

    /**
     * Input format:
     * First line contians 2 numbers: hospital count and student count
     * for next N lines, each line contains information of a hospital:
     * <hospital-name> <preference-count> <preference-1> <preference-2> ...
     * for next M lines, each line contains information of a student:
     * <student-name> <preference-count> <preference-1> <preference-2> ...
     * <p>
     * 2 6
     * Arkham 2 Bob Alice
     * Hilltop 3 Carl Alice Bob
     * Alice 2 Arkham Hilltop
     * Bob 2 Hilltop Arkham
     * Carl 2 Hilltop Arkham
     * Danny 2 Arkham Hilltop
     * Ellen 2 Hilltop Arkham
     * Frank 2 Arkham Hilltop
     * </p>
     *
     * Note: all {@code Map} implementations are {@code LinkedHashMap},
```

```
     * so you don't have to worry about the iteration sequence :)
     */
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int hospitalNum = scanner.nextInt();
        int studentNum = scanner.nextInt();

        Map<String, Hospital> hospitals = new LinkedHashMap<>();
        Map<String, Student> students = new LinkedHashMap<>();

        for (int i = 0; i < hospitalNum; i++) {
            String name = scanner.next();
            int position = scanner.nextInt();
            List<String> preference = new ArrayList<>();

            for (int j = 0; j < position; j++) {
                preference.add(scanner.next());
            }
            hospitals.put(name, new Hospital(name, position, preference));
        }


        for (int i = 0; i < studentNum; i++) {

            String name = scanner.next();
            int preferenceCnt = scanner.nextInt();
            List<String> preference = new ArrayList<>();

            for (int j = 0; j < preferenceCnt; j++) {
                preference.add(scanner.next());
            }

            students.put(name, new Student(name, preference));
        }

        galeShapley(hospitals, students);

        for (Hospital hospital : hospitals.values()) {
            System.out.printf("%s: %s\n", hospital.name, hospital.offers);
        }

        scanner.close();
    }

}
```