Übungsblatt 2

Abgabe: 14.05.2023

Auf diesem Übungsblatt macht für Aufgabe 1 die Person die Implementierung, die in einem Telefonbuch zuletzt gelistet würde. Die andere Person macht die Tests. Die restlichen Aufgaben werden gemeinsam bewertet.

Aufgabe 1 Ein Doodle-Post-Processor (70 %)

Aktuell findet die Tutoriumswahl in PI-2 ja nach dem Prinzip „Wer zuerst kommt, mahlt zuerst“ statt. Alternativ könnten wir auch ein Doodle mit den Terminen aufmachen, jede Studierende kreuzt an, zu welchen Terminen sie Zeit hat und wir verteilen auf Basis der Wahl die Studierenden auf die Tutorien. Die Methode, die diese Verteilung ermittelt, wollen wir auf diesem Übungsblatt implementieren.

Die Methode bekommt zwei Parameter:

1. Das Ergebnis der Wahl. Aus Datenschutzgründen werden hier Namen weggelassen, so dass es ein zweidimensionales Array ist, wo in jeder Zeile die ja/nein-Entscheidungen einer Studierenden für jedes Tutorium stehen.
2. Die Kapazitäten der Tutorien. Da es für den nachfolgenden Algorithmus keine Rolle spielt, wollen wir an dieser Stelle etwas flexibel sein und jedem Tutorium eine unterschiedliche Kapazität erlauben. Die Kapazitäten stehen in dem Array in derselben Reihenfolge wie die Termine in dem Doodle. Die zu implementierende Methode soll die Einträge in diesem Array anpassen, so dass sie widerspiegeln, welche Kapazität nach der Verteilung jeweils noch übrig ist.

Die zu implementierende Methode soll ein Array zurückliefern, in dem für jede Studierende die Nummer des Tutoriums steht, dem sie zugeordnet wurde. Tutoriumsnummern beginnen mit 0 und sind somit auch Indizes in die Tutoriumskapazitäten und die 2. Dimension des Wahlergebnisses.

Der Algorithmus der Verteilung besteht aus zwei ineinander geschachtelten Schleifen und funktioniert so:

1. Für alle Studierenden muss ihre aktuelle Zuordnung zu einem Tutorium gespeichert werden. Am Anfang sind sie keinem Tutorium zugeordnet.
2. Die äußere Schleife geht der Reihe nach alle Studierenden durch. Nach ihrem Ende muss die Zuordnung von Studierenden zu Tutorien zurückgegeben werden.
3. Die innere Schleife sucht für die aktuelle Studierende ein passendes Tutorium. Dazu wird ihre Tutoriumsnummer so lange erhöht, bis ein Tutorium gefunden wurde, das sie gewählt hat und in dem noch Platz ist. War sie bisher keinem Tutorium zugeordnet, beginnt die Suche bei Tutorium 0. Nach dem Ende der Schleife gibt es zwei Fälle: Entweder wurde ein passendes Tutorium gefunden, dann muss dessen verfügbare Kapazität angepasst werden, weil die Studierende nun in diesem Tutorium einen Platz einnimmt, und die äußere Schleife wird mit der *nächsten* Studierenden fortgesetzt. Oder es konnte kein passendes Tutorium gefunden werden, dann wird ihre Tutoriumswahl wieder auf „nicht zugeordnet“ gesetzt und die äußere Schleife wird bei der *vorherigen* Studierenden fortgesetzt (*Backtracking*). Beachtet, dass „Fortsetzen“ bedeutet, dass die vorherige Studierende aus ihrem aktuellen

Tutorium wieder ausgetragen werden muss, weil sie sich im nächsten Schleifendurchlauf ein neues suchen wird. Gibt es keine vorherige Studierende, weil die aktuelle bereits die erste ist, ist eine Verteilung nicht möglich und die Methode liefert *null* als Ergebnis zurück.

Beachtet, dass die innere Schleife immer hinter dem bisher für eine Studierende gewählten Tutorium weitermacht, d.h. sie beginnt nicht immer wieder von vorne. Das Backtracking bewirkt also, dass eine frühere Suche fortgesetzt wird. Hingegen beginnen alle Suchen für Studierende *nach* der aktuellen Studierenden (also mit größerem Index) immer von vorne, weil diese noch keinem Tutorium zugeordnet sind.

Tests. In dem bereitgestellten Projekt ist bereits eine Testklasse mit drei komplexen Tests enthalten. Allerdings stützen sich diese auf die bisher leere Methode *checkDistribution* ab. Vervollständigt die Methode durch geeignete Zusicherungen. Erweitert die Testklasse außerdem um einfachere Tests, die individuelle Eigenschaften eurer Implementierung überprüfen und nicht nur, ob das Ergebnis gültig ist.[1]

Aufgabe 2 Langer Atem (5 %)

In den Tests zu Aufgabe 1 ist ein Test mit *@Disabled* versehen und wird deshalb übersprungen. Entfernt das *@Disabled*. Was ist das Problem? Woran liegt das? Könnt ihr die Eingabedaten so ändern, dass der Test erfolgreich durchläuft, ohne die Wahl der Studierenden selbst oder die Größe der Tutorien zu ändern? Was könnte getan werden, um solche Situationen so weit wie möglich zu vermeiden?

Aufgabe 3 Wie langer Atem? (5 %)

Wie ist der *Worse-Case*-Aufwand des Algorithmus aus Aufgabe 1 für eine Anzahl an Studierenden s und Anzahl an Tutorien t?. Wie muss die Eingabe beschaffen sein, um diesen zu erreichen?

Aufgabe 4 Aufwändige Richtung (20 %)

Gegeben sind die folgenden zwei Methoden, die die Klasse *Array* aus Übungsblatt 1 benutzen:

```
1  void backward(final int n)
2  {
3    final Array a = new Array(0);
4    for (int i = n - 1; i >= 0; --i) {
5        a.set(i, i);
6    }
7  }
8
9  void forward(final int n)
10 {
11   final Array a = new Array(0);
12   for (int i = 0; i < n; ++i) {
13       a.set(i, i);
14   }
15 }
```

Wie ist der Aufwand von *backward* und *forward* im Sinne des O-Kalküls bezogen auf n?

[1] Das PITest-Plugin wird immer die Titelzeile der Klasse als ungetestet markieren, weil der Java-Compiler implizit einen Standardkonstruktor für die Klasse erzeugt, den ihr aber nicht testet, weil er ja auch gar nicht verwendet werden soll. Das ist in Ordnung.