Übungsblatt 2

Abgabe bis: 2017-05-14, 23:59 Uhr MESZ

Auf diesem Übungsblatt wollen wir einen Teil eines eigenen Tutorieneinschreibetools entwickeln. Da wir in der Vorlesung das Konzept *Backtracking* als Entwurfsparadigma für Algorithmen kennengelernt haben, wollen wir das hier ausprobieren, um die Studierenden gemäß vorhandener Terminbewertungen so auf die Tutorien zu verteilen, dass niemand **völlig**[1] unzufrieden ist.

Unabhängig von diesem Übungsblatt wurde bereits eine Webapplikation entwickelt, die uns die Eingabedaten liefert. Studierende können in dieser Webapplikation die angebotenen Tutorien mit Sternen bewerten. Null Sterne bedeuten dabei die schlechteste, fünf Sterne die bestmögliche Bewertung. Studierende würden also keine Sterne vergeben für Tutorien, an denen sie keine Zeit (oder Lust) haben und fünf Sterne für ihre Wunschtermine. Die Ergebnisse dieser Bewertungen werden von unserem Tool dann entsprechend eingelesen.

Backtracking

Das zu verwendende Verfahren hängt von folgenden Konfigurationsgrößen ab:

- t: Anzahl Tutorien
- s: Anzahl Studierende
- st_{max} maximale Anzahl von Studierenden pro Tutorium
- r Mindestbewertung

Die Tutorien sind von 1 bis t nummeriert, die Studierenden analog von 1 bis s. Ferner muss gelten $s \leq st_{max} \cdot t$.

Eine Konfiguration K besteht dann aus den Tutorien mit den ihnen zugeordneten Studierenden. Eine solche Konfiguration ist genau dann *gültig*, wenn kein Tutorium überfüllt ist (d. h. die Anzahl der zugeordneten Studierenden ist nicht größer als st_{max}) und keine StudentIn einem Tutorium zugeordnet ist, für das sie eine geringere Bewertung als r abgegeben hat.

Die Anfangskonfiguration K_0 besteht aus t leeren Tutorien.

Folgekonfiguration Gegeben sei eine Konfiguration K_i. Dabei ist S_n ($n \in \{1, ..., s\}$) die StudentIn, die in dieser Konfiguration zuletzt zugeordnet wurde. Sie ist dem Tutorium T_j ($j \in \{1, ..., t\}$) zugeordnet. Im Fall der Anfangskonfiguration gibt es keine solche StudentIn S_n, n ist dann 0.

Zur Berechnung der Folgekonfiguration werden vier Fälle unterschieden:

Fall 1: K_i ist gültig und $n = s$

Die aktuelle Konfiguration ist eine gültige Lösung. Es gibt keine Folgekonfiguration.

[1] Das bedeutet hier, dass wir eine Mindestbewertung vorgeben und unser Tool dafür sorgt, dass niemand einem Tutorium zugeordnet wird, für dass er oder sie weniger Sterne als die Mindestbewertung vergeben hat.

Fall 2: K_i ist gültig und $n < s$
StudentIn S_{n+1} wird dem Tutorium 1 zugeordnet.

Fall 3: K_i ist nicht gültig und $j < t$
StudentIn S_n wird aus Tutorium T_j entfernt und Tutorium T_{j+1} zugeordnet.

Fall 4: K_i ist nicht gültig, $j = t$
Dann kann es ein größtes k' ($k' \in \mathbb{N}^+$) geben, so dass $S_{k'}$ einem Tutorium $T_z, z < t$ zugeordnet ist und alle S_k für $k \in \{k'+1, ..., n\}$ dem Tutorium T_t zugeordnet sind. Es werden alle Studierenden S_k für $k \in \{k'+1, ..., n\}$ aus dem Tutorium T_t entfernt. Die StudentIn $S_{k'}$ wird aus dem Tutorium T_z entfernt und dem Tutorium T_{z+1} zugeordnet.

Wenn es kein solches k' gibt, gibt es keine Folgekonfiguration.

Aufgabe 1 TheaPlusPlus [100 Punkte]

Ihr findet bei Stud.IP ein gepacktes Eclipse-Projekt *TheaPlusPlus*. In diesem Projekt befinden sich diverse Klassen, die einige Teile zur Lösung der folgenden Aufgabe bereits enthalten und auch einige Testklassen.

Aufgabe 1.1 TheaPP [90 Punkte]

Im Paket `pi.uebung02` gibt es bereits eine Klasse `TheaPP`. Ihr findet darin unvollständige Methoden, die ihr noch implementieren müsst. Haltet euch dabei an die *javadoc*-Kommentare, die ihr direkt bei den Methoden findet[2].

Ergänzt die JUnit-Tests in der Klasse `TheaPPTest` und testet damit.

Aufgabe 1.2 Main [10 Punkte]

Im Projektordner befinden sich auf oberster Ebene drei `csv`-Dateien. Sie sind für Tutoriumsgrößen von 10 Studierenden gedacht und aus ihren Namen könnt ihr entnehmen, wieviele Tutorien es jeweils geben soll.

Ergänzt die Klasse `TheaPP` um eine `main`-Methode. Für jede der drei `csv`-Dateien erzeugt ihr jeweils ein `TheaPPParser`-Objekt mit entsprechender Beschränkung und lest damit die Dateien ein.

Erzeugt dann `TheaPP`-Objekte mit passenden Beschränkungen und versucht dann Tutoriumszuordnungen mit verschiedenen Mindestbewertungen ausrechnen zu lassen. Die Ergebnisse einer Zuordnung lasst ihr dann auf der Konsole ausgeben, indem ihr die Methode *printDistribution()* aufruft.

Protokolliert die Ergebnisse dieser Aufrufe in eurer Abgabe, sofern ihr innerhalb einer angemessenen Zeit eine Antwort erhaltet.

[2] Hier gibt es Methoden, die den Zugriffsmodifikator package-private haben. Sie sind also nicht öffentlich, d. h. ihre Parameter kommen nicht von außen, sondern nur von Klassen innerhalb desselben Pakets. Pakete können vor der Auslieferung versiegelt werden, so dass niemand nachträglich Klassen in die Pakete „einschleusen“ kann. Insofern sind diese Methoden also von außen nicht verwendbar. Sie sind aber für sich automatisiert testbar, wenn die entsprechenden JUnit-Testklassen im selben Paket sind.

Aufgabe 2 BONUS: Was ist denn bloß los? [5 Punkte]

Was ist bei *TenTutorials.csv* und der Mindestbewertung 4 das Problem und wie wirkt es sich aus?

Könnt ihr die negativen Auswirkungen des Problems auf die Implementierung verringern oder beheben,

1. ohne am Quelltext Änderungen vorzunehmen?
2. indem ihr Änderungen am Quelltext vornehmt?