TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P1

PDF erstellt am: 19. März 2024

1 Vorbereitung

Um diese Programmieraufgabe erfolgreich durchführen zu können, müssen folgende Schritte umgesetzt werden:

1. Laden Sie das Framework `P1.zip` aus TUWEL herunter.
2. Entpacken Sie `P1.zip` und öffnen Sie das entstehende Verzeichnis als Projekt in IntelliJ (nicht importieren, sondern öffnen).
3. Öffnen Sie die nachfolgend angeführte Datei im Projekt in IntelliJ. In dieser Datei sind sämtliche Programmiertätigkeiten durchzuführen. Ändern Sie keine anderen Dateien im Framework und fügen Sie auch keine neuen hinzu.
   `src/main/java/exercise/StudentSolutionImplementation.java`
4. Füllen Sie Vorname, Nachname und Matrikelnummer in der Methode `StudentInformation provideStudentInformation()` aus.

2 Hinweise

Einige Hinweise, die Sie während der Umsetzung dieser Aufgabe beachten müssen:

- Lösen Sie die Aufgaben selbst und nutzen Sie keine Bibliotheken, die diese Aufgaben abnehmen.
- Sie dürfen beliebig viele Hilfsmethoden schreiben und benutzen. Beachten Sie aber, dass Sie nur die oben geöffnete Datei abgeben und diese Datei mit dem zur Verfügung gestellten Framework lauffähig sein muss.

3 Übersicht

In dieser Programmieraufgabe wird die Funktionsweise des bereitgestellten Frameworks sowie der allgemeine Ablauf von Programmieraufgaben in Algorithmen und Datenstrukturen nähergebracht.

Dazu werden mehrere Teilaufgaben zum Stable-Matching-Problem betrachtet, die jeweils unterschiedliche Laufzeiten aufweisen.

4 Theorie

Die notwendige Theorie kann in den Vorlesungsfolien „Einleitung - Das Stable-Matching-Problem“ sowie „Analyse von Algorithmen“ im Abschnitt „Laufzeiten einiger gebräuchlicher Funktionen“ gefunden werden.

5 Implementierung

5.1 Auffinden des Indizes

Implementieren Sie die Indexsuche in der Methode `int findIndex(int[] numbers, int element)`.

Mittels des Parameters `int[] numbers` wird ein Array mit Zahlen übergeben, in dem nach dem Parameter `int element` gesucht werden soll. Der Rückgabewert von `int findIndex(int[] numbers, int element)` soll jene Stelle (also jener Index) sein, an der `int element` erstmals in `int[] numbers` auftritt. Ist das gesuchte Element nicht enthalten, so geben Sie -1 zurück.

Beispiel: Wird im Array `[4, 1, 0, 2]` nach dem Element `0` gesucht, so soll 2 zurückgegeben werden, da 0 an Stelle 2 steht. Wird nach 6 gesucht, soll -1 zurückgegeben werden, da 6 nicht im Array enthalten ist.

5.2 Gale–Shapley-Algorithmus

Implementieren Sie den Gale-Shapley-Algorithmus in der Methode `void performGaleShapley(StableMatchingInstance instance, StableMatchingSolution solution)`.

`StableMatchingInstance instance` beinhaltet die Instanz des Stable-Matching-Problems, für das ein stabiles Matching bestimmt werden soll. Wie in der Vorlesung gehen wir auch hier vom anschaulichen Problemsetting aus, dass eine Zuordnung von Kindern zu Gastfamilien gesucht wird. Die Kinder und Familien sind jeweils von 0 bis $n-1$ durchnummeriert, wobei n die Anzahl der Kinder bzw. Familien ist. Folgende Methoden stehen bereit und dürfen von Ihnen verwendet werden:

- `int getN()`: Gibt die Größe der Instanz (= Anzahl der Kinder bzw. Familien) zurück.
- `int getFamilyOfChildAtRank(int child, int rank)`: Gibt für das Kind `int child` jene Familie zurück, die in seiner Präferenzliste an Stelle `int rank` steht. `int rank` geht dabei von 0 (höchste Präferenz) bis $n-1$ (niedrigste Präferenz).
- `int getChildOfFamilyAtRank(int family, int rank)`: Gibt für die Familie `int family` jenes Kind zurück, das in ihrer Präferenzliste an Stelle `int rank` steht. `int rank` geht dabei von 0 (höchste Präferenz) bis $n-1$ (niedrigste Präferenz).
- `int getRankOfFamilyForChild(int child, int family)`: Gibt die Präferenz des Kindes `int child` für Familie `int family` zurück. Der Rückgabewert geht dabei von 0 (höchste Präferenz) bis $n-1$ (niedrigste Präferenz).
- `int getRankOfChildForFamily(int family, int child)`: Gibt die Präferenz der Familie `int family` für Kind `int child` zurück. Der Rückgabewert geht dabei von 0 (höchste Präferenz) bis $n-1$ (niedrigste Präferenz).

Die letzten beiden genannten Methoden greifen auf Ihre Lösung von Abschnitt 5.1 zu.

Beispiel: Seien p_{child} bzw. p_{family} die Präferenzlisten aus der Sicht der Kinder bzw. Familien:

$$p_{\text{child}} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix} \quad p_{\text{family}} = \begin{bmatrix} 2 & 0 & 1 \\ 2 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

`getN()` gibt 3 zurück, da es 3 Kinder bzw. Familien gibt, die jeweils von 0 bis 2 nummeriert sind. `getFamilyOfChildAtRank(2, 1)` gibt 0 zurück, da für

Kind 2 die Familie 0 die Präferenz 1 hat. `getRankOfChildForFamily(1, 0)` gibt 2 zurück, da für Familie 1 das Kind 0 an Stelle 2 steht.

`StableMatchingSolution solution` verwaltet die Lösung für das Stable-Matching-Problem. Folgende Methoden stehen bereit und dürfen von Ihnen verwendet werden:

- `int getAssignedFamily(int child)`: Gibt jene Familie zurück, die dem Kind `int child` zugeordnet ist. Falls dem Kind `int child` keine Familie zugeordnet ist, wird -1 zurückgegeben.
- `int getAssignedChild(int family)`: Gibt jenes Kind zurück, das der Familie `int family` zugeordnet ist. Falls der Familie `int family` kein Kind zugeordnet ist, wird -1 zurückgegeben.
- `boolean isChildFree(int child)`: Gibt `true` zurück, falls das Kind `int child` keiner Familie zugeordnet ist und `false` andernfalls.
- `boolean isFamilyFree(int family)`: Gibt `true` zurück, falls die Familie `int family` keinem Kind zugeordnet ist und `false` andernfalls.
- `boolean assign(int child, int family)`: Ordnet das Kind `int child` der Familie `int family` zu. Diese Zuordnung erfolgt nur dann, wenn sowohl das Kind als auch die Familie frei sind. In diesem Fall wird `true` zurückgeben. Ist das Kind oder/und die Familie nicht frei, so erfolgt **keine** Zuordnung und es wird `false` zurückgegeben.
- `boolean setFree(int child)`: Löst die Zuordnung des Kindes `int child` zu seiner zugeordneten Familie auf. Wenn `true` zurückgegeben wird, war das Kind einer Familie zugeordnet. Wird `false` zurückgegeben, dann war das Kind keiner Familie zugeordnet, folglich gab es keine Zuordnung zum Auflösen.
- `boolean hasUnassignedChildren()`: Gibt `true` zurück, falls es noch nicht zugeordnete Kinder gibt und `false` andernfalls.
- `int getNextUnassignedChild()`: Gibt das nächste noch nicht zugeordnete Kind zurück. Dabei wird immer das freie Kind mit der niedrigsten Nummer gewählt. Gibt es kein freies Kind, so wird -1 zurückgegeben.

Das berechnete stabile Matching soll im zweiten Parameter `StableMatchingSolution solution` (ein vorinitialisiertes Objekt der Klasse StableMatchingSolution mit einer anfangs leeren Lösung) hinterlegt werden. Ein Rückgabewert wird nicht erwartet.

Hinweis: Verwenden Sie bei der Auswahl des nächsten Kindes immer `int getNextUnassignedChild()`. Es kann nämlich mehrere stabile Matchings geben und Reihenfolge der Auswahl des nächsten Kindes kann daher einen Einfluss auf das gefundene Matching haben.

Hinweis: Die Klasse `StableMatchingInstance` verfügt über die Methode `print()`, mit der Sie die Instanz auf die Console drucken können. Gleiches gilt für die Klasse `StableMatchingSolution`.

Hinweis: Wie können Sie sich für jedes Kind merken, welche Familien im Zuge des Gale-Shapley-Algorithmus noch nicht gewählt wurden?

5.3 Prüfung auf stabiles Matching

Implementieren Sie in der Methode
`boolean isStableMatching(StableMatchingInstance instance, StableMatchingSolution solution)` einen Algorithmus, der für die Instanz `StableMatchingInstance instance` bestimmt, ob das Matching in der Lösung `StableMatchingSolution solution` stabil ist.

`StableMatchingSolution solution` beinhaltet bereits ein perfektes Matching, das Sie nicht verändern sollen.

5.4 Brute-Force

Es ist auch ein Brute-Force-Ansatz implementiert, den Sie **nicht** mehr implementieren müssen. Der Brute-Force-Ansatz verwendet zur Bestimmung eines stabilen Matchings Ihre Methode aus Abschnitt 5.3.

6 Testen

Führen Sie zunächst die `main`-Methode in der Datei `src/main/java/framework/Exercise.java` aus.

Anschließend wird Ihnen in der Konsole eine Auswahl an Testinstanzen angeboten, darunter befindet sich zumindest `abgabe.csv`:

```
Select an instance set or exit:
[1] abgabe.csv
[0] Exit
```

Durch die Eingabe der entsprechenden Ziffer kann entweder eine Testinstanz ausgewählt werden oder das Programm (mittels der Eingabe von `0`) verlassen werden. Wird eine Testinstanz gewählt, dann wird der von Ihnen implementierte Programmcode ausgeführt. Kommt es dabei zu einem Fehler, wird ein Hinweis in der Konsole ausgegeben.

Relevant für die Abgabe ist das Ausführen der Testinstanz `abgabe.csv`.

Die weiteren Testinstanzen `indexsuche.csv`, `gale-shapley.csv`, `stablematching-positiv.csv`, `stablematching-negativ.csv` und `brute-force.csv` sind nur zum jeweiligen Testen der einzelnen Unteraufgaben gedacht. Beachten Sie, dass alle Stable-Matching-Probleme nur dann lauffähig sind, wenn die Indexsuche implementiert ist.

7 Evaluierung

Wenn der von Ihnen implementierte Programmcode mit der Testinstanz `abgabe.csv` ohne Fehler ausgeführt werden kann, dann wird nach dem Beenden des Programms im Ordner `results` eine Ergebnis-Datei mit dem Namen `solution-abgabe.csv` erzeugt.

Die Datei `solution-abgabe.csv` beinhaltet Zeitmessungen der Ausführung der Testinstanz `abgabe.csv`, welche in einem Web-Browser visualisiert werden können. (Auch Ergebnis-Dateien anderer Testinstanzen können zu Testzwecken visualisiert werden.) Öffnen Sie dazu die Datei `visualization.html` in Ihrem Web-Browser und klicken Sie rechts oben auf den Knopf *Ergebnis-Datei auswählen*, um `solution-abgabe.csv` auszuwählen.

Beantworten Sie basierend auf der Visualisierung die Fragestellungen aus dem folgenden Abschnitt.

8 Fragestellungen

Öffnen Sie `solution-abgabe.csv` und bearbeiten Sie folgende Aufgaben- und Fragestellungen:

1. Durch Klicken auf Gruppennamen in der Legende neben der Plots, lassen sich einzelne Gruppen aus- bzw. einblenden. Blenden Sie alles bis auf *Indexsuche* aus. Beschreiben Sie das Laufzeitverhalten anhand des Plots der *Indexsuche* und erklären Sie basierend auf Ihrer Implementierung, wieso dieses zustande kommt.

   Drücken Sie im Anschluss in der Menüleiste rechts über dem Plot auf den Fotoapparat, um den Plot als Bild zu speichern.

2. Blenden Sie nun alles bis auf *Gale Shapley* aus. Beschreiben Sie das Laufzeitverhalten anhand des Plots für *Gale Shapley* und erklären Sie basierend auf Ihrer Implementierung, wieso dieses zustande kommt.

   Erstellen Sie im Anschluss wieder mithilfe der Menüleiste ein Bild des Plots.

3. Blenden Sie nun alles bis auf *Stable Matching Positiv* aus. Beschreiben Sie das Laufzeitverhalten anhand des Plots für *Stable Matching Positiv* und erklären Sie basierend auf Ihrer Implementierung, wieso dieses zustande kommt.

   Blenden Sie nun auch *Stable Matching Negativ* ein und vergleichen Sie den Plot mit *Stable Matching Positiv*. Sind größere Unterschiede zu sehen? Wenn ja, wie erklären Sie sich diese?

   Erstellen Sie nun ein Bild der beiden Plots.

4. Blenden Sie nun alles bis auf *Brute Force* aus. Beschreiben Sie das Laufzeitverhalten anhand des Plots für *Brute Force*. Geben Sie darüber hinaus eine Schätzung für die Laufzeit bei einer Instanzgröße von $n = 11$ ab.

   Erstellen Sie wiederum ein Bild des Plots.

Falls sich im Zuge der Evaluierung die Darstellung der Plots auf ungewünschte Weise verändert (z.B. durch die Auswahl eines zu kleinen Ausschnitts), können Sie mittels Doppelklick auf den Plot oder Klick auf das Haus in der Menüleiste die Darstellung zurücksetzen.

Fügen Sie Ihre Antworten in einem Bericht gemeinsam mit allen erstellten Bildern der Visualisierungen der Testinstanz `abgabe.csv` zusammen.

9 Abgabe

Laden Sie die Datei `src/main/java/exercise/StudentSolutionImplementation.java` in der TUWEL-Aktivität *Hochladen Source-Code P1* hoch. Fassen Sie diesen Bericht mit den anderen für das zugehörige Abgabegespräch relevanten Berichten in einem PDF zusammen und geben Sie dieses in der TUWEL-Aktivität *Hochladen Bericht Abgabegespräch 1* ab.

10 Nachwort

Das Stable-Matching-Problem (auch als Stable-Marriage bekannt) hat viele Anwendungen, z.B. bei der Partner_innenwahl in Online-Dating-Apps, im Arbeitsmarkt (Industrie), oder bei der Schul- oder Studienplatzvergabe (Erziehung und Bildung). Hier haben sowohl Schüler_innen bzw. Bewerber_innen auf der einen Seite, als auch Schulen, Universitäten oder Firmen auf der anderen Seite, bestimmte Vorlieben dazu, wem sie gerne zugeordnet werden möchten. Stabilität ist besonders wünschenswert, weil sie den Teilnehmer_innen z.B. den Aufwand erspart, zu versuchen, von der ursprünglichen Zuordnung abzuweichen oder sich entgegen ihrer eigentlichen Vorlieben zu äußern, um eine vorteilhafte Zuordnung zu bekommen. In einigen Städten in den USA (z.B. New York City und Boston) und Ländern in Europa (z.B. Ungarn und Finnland) ist die Schulwahl so gestaltet, dass sie unter anderem die Stabilität berücksichtigt.

Das Konzept der Stabilität lässt sich verallgemeinern auf generelle (also nicht zwangsläufig bipartite) Graphen. Das darunterliegende Modell wird als *Stable-Roommates* bezeichnet, weil es z.B. Anwendung für Zuordnungen von Mitbewohner_innen in Zweierzimmer findet, wie z.B. im Studierendenwohnheim. Stable-Roommates kann auch verwendet werden, um Organtausch zwischen Patient_in-Spender_in-Paaren zu modellieren. Im Gegensatz zu Stable-Matchings existiert bei Stable-Roommates nicht immer eine stabile Zuweisung. Man kann aber in polynomieller Zeit feststellen, ob es eine solche gibt, und ggf. eine finden.

Der US Wissenschaftler Al Roth, der 2012 zusammen mit Lloyd Shapley auf Grund ihrer Beiträge zur Theorie der Matching-Märkte den Wirtschaftsnobelpreis erhalten hat, hat unter anderem die Märkte zwischen Ärzt_innen im Praktikum und Krankenhäusern, bei der Schulplatzvergabe

und beim Nierenaustauschprogramm in den USA durch Anwendungen von Stable Matchings oder Stable Roommates deutlich effizienter gemacht.

Das Thema Stable-Matching wird in der Master LVA "Algorithmic Social Choice" (192.118) ausführlich erläutert. Dort lernt man unter anderem, wie man feststellt, ob eine stabile Zuweisung in allgemeinen Graphen existiert, welche Optimalitätskriterien man anwenden kann, um zwei unterschiedliche stabile Matchings zu vergleichen, und wie man mit dem Fall umgeht, wenn die Präferenzen "Unentschieden" beinhalten.