TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P1

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 einfache Algorithmen mit unterschiedlichen Laufzeiten zum Thema Sortieren herangezogen.

4 Theorie

Die notwendige Theorie kann in den Vorlesungsfolien „Analyse von Algorithmen“ gefunden werden.

5 Implementierung

5.1 Überprüfung auf Sortierung – `isSorted()`

Implementieren Sie die Methode `boolean isSorted(int[] numbers)`.

Diese Methode überprüft, ob das gegebene Array `int[] numbers` mit Integerzahlen aufsteigend sortiert ist und soll `true` zurückliefern, wenn `int[] numbers` aufsteigend sortiert ist und `false` andernfalls.

5.2 Insertion Sort – `insertionSort()`

Implementieren Sie die Methode `void insertionSort(int[] numbers)`.

Dem Parameter `int[] numbers` wird ein Array mit Integerzahlen überbeben. Ziel ist es, dieses Array mithilfe von **Insertion Sort** aufsteigend zu sortieren. Den Pseudocode finden Sie in den Vorlesungsfolien.

Die Funktion hat keine Rückgabe (`void`). Stattdessen sollen am Ende die Zahlen in sortierter Reihenfolge im übergebenen Array `int[] numbers` abgelegt sein.

5.3 Brute Force – `sortBruteForce()`

Implementieren Sie die Methode `void sortBruteForce(int[] numbers)`.

Mittels des Parameters `int[] numbers` wird ein Array mit Integerzahlen übergeben. Ziel ist es nun, mithilfe von **Brute Force** das Array zu sortieren, also der Reihe nach Permutationen des Arrays durchzugehen, bis eine sortierte Folge gefunden wird.

Um die Aufgabe zu erleichtern, wird Ihnen die Klasse `PermutationIterator` zur Verfügung gestellt, die Sie verwenden dürfen. Es handelt sich um eine Klasse, mit der Sie alle möglichen Permutationen der Zahlen aus `int[] numbers` erzeugen können.

`PermutationIterator iterator = new PermutationIterator(numbers)` erzeugt ein neues Objekt der Klasse `PermutationIterator` und weist es der Variablen `iterator` zu. Folgendermaßen können Sie dann Gebrauch vom Objekt `iterator` machen:

- `boolean hasNext()` gibt `true` zurück, falls es eine noch nicht betrachtete Permutation gibt und `false` andernfalls.
- `int[] next()` gibt die nächste Permutation zurück. Sind bereits alle Permutationen zuvor erzeugt worden, so wird `null` zurückgegeben.

Die Funktion hat keine Rückgabe (`void`). Stattdessen sollen am Ende die Zahlen in sortierter Reihenfolge im übergebenen Array `int[] numbers` abgelegt sein.

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 `isSorted.csv`, `insertionSort.csv` und `bruteForce.csv` sind nur zum jeweiligen Testen der einzelnen Unteraufgaben gedacht.

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 *isSorted true* aus. Beschreiben Sie das Laufzeitverhalten anhand des Plots für *isSorted true* und erklären Sie basierend auf Ihrer Implementierung, wie dieses Laufzeitverhalten zustande kommt.

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

Blenden Sie nun zusätzlich auch *isSorted false* ein und vergleichen Sie die Laufzeitkurven von *isSorted true* und *isSorted false*. Sind größere Unterschiede zu sehen? Wenn ja, wie erklären Sie sich diese? Erstellen Sie nun auch ein Bild dieses Plots.

2. Die drei Kategorien *Insertion Sort set1*, *Insertion Sort set2* und *Insertion Sort set3* beinhalten das Laufzeitverhalten für Insertion Sort, der auf drei unterschiedlichen Instanzsets ausgeführt wurde.

   Blenden Sie nun alles bis auf *Insertion Sort set1* aus. Beschreiben Sie das Laufzeitverhalten von *Insertion Sort set1* anhand des Plots und erstellen Sie mithilfe der Menüleiste ein Bild des Plots. Wiederholen Sie dies für *Insertion Sort set2* und *Insertion Sort set3*.

   Blenden Sie nun *Insertion Sort set1*, *Insertion Sort set2* und *Insertion Sort set3* gleichzeitig ein. Sind Unterschiede zwischen den drei Gruppen feststellbar? Wenn ja, wie erklären Sie sich diese basierend auf Ihrer Implementierung bzw. welche Rückschlüsse auf die drei Instanzgruppen können Sie ziehen?

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

3. Blenden Sie nun alles bis auf *Brute Force* aus. Beschreiben Sie das Laufzeitverhalten anhand des Plots für *Brute Force*. Geben Sie darüber hinaus eine plausible 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 den 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

Wenn wir die Effizienz eines Algorithmus experimentell bestimmen wollen, ist die Auswahl der Eingaben sehr wichtig. Zuerst sollten wir klären, ob wir (A) das worst-case Verhalten bestimmen wollen, oder (B) das Verhalten des Algorithmus an solchen Eingaben, die wir auch tatsächlich in der Praxis erwarten. Im Fall (A) sind manchmal zufällig generierte Eingaben sinnvoll, da diese keinerlei Struktur aufweisen. Wenn es sich z.B. um Graphen handelt, ist noch zu überlegen, welche Dichte wir erreichen wollen. Wir wählen eine reelle Zahl p zwischen 0 und 1, und fügen für jedes Knotenpaar u, v eine Kante zwischen u und v mit der Wahrscheinlichkeit p ein. Experimente können nun Zufallsgraphen mit verschiedenen Dichten umfassen. Im Fall (B) ist es am sinnvollsten, Daten aus realen Anwendungen zu verwenden. Hierfür gibt es oft auch öffentlich zugängliche Sammlungen von Testdaten (*benchmark repositories*).

Speziell bei schweren Problemen, die wir im Allgemeinen nicht in Polynomialzeit lösen können, unterscheiden sich experimentell beobachteten Laufzeiten zwischen zufälligen und realen Eingaben oft erheblich. Um dann eine sinnvolle Aussage über die empirische Effizienz eines Algorithmus treffen zu können, sollte das Experiment möglichst über eine breite Auswahl an Eingaben durchgeführt werden. Stellt sich heraus, dass bei mehreren zur Auswahl stehenden Algorithmen jeder auf anderen Eingaben am schnellsten ist, kann es günstig sein, alle diese Algorithmen in einem Portfolio zu vereinigen um mittels einer heuristischen Methode je nach Eingabe zu entscheiden, welcher Algorithmus des Portfolios für die jeweilige Eingabe verwendet werden soll.