Aufgabenblatt
Entwurf von Algorithmen

- Abgabetermin: **Samstag, 18.06.2016 23:55 Uhr**
- Zur Prüfungszulassung müssen in einem Aufgabenblatt mind. 25% der Punkte erreicht werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte bestanden werden.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Jede Hausaufgabe enthält eine optionale Zusatzaufgabe, die bei erfolgreicher Bearbeitung 1 Klausurpunkt zählt.
- Abgabe:
  - Die Abgabe erfolgt über CodeOcean[1]. CodeOcean ermöglicht die Bearbeitung und Ausführung der Quelldateien im Browser. Bei Bedarf können die Sources (Quelltexte) und Unit Tests von der Übungs-Webseite[2] heruntergeladen werden, um lokal zu entwickeln.
  - CodeOcean benutzt Java Version 8.
  - Alle Aufgaben sind über das CodeOcean einzureichen. CodeOcean ermöglicht das Anfügen von Kommentaren bei der Abgabe, darüber hinaus können auch Kommentare in den Source-Code geschrieben werden.
  - Geben Sie bei **jeder** Aufgabe beide Namen Ihrer Gruppe an.
  - Achten Sie darauf, dass Ihr Quelltext hinreichend kommentiert ist.

Aufgabe 1: Routenplanung

6 P

Prof. Naumann fährt mit seinem Auto von Newmark nach Reno. Sein Auto besitzt einen Tank, dessen maximales Füllvermögen für genau d Kilometer reicht. Die Gesamtstrecke der Reise z übersteigt die Reichweite eines Tanks ($d < z$), deswegen muss der Professor an einer oder mehreren der n Tankstellen auf dem Weg halten. Der Professor besitzt eine Karte, die die Entfernung zwischen zwei aufeinanderfolgenden Tankstellen angibt. Sei also t_i die Distanz zur von der $i-1$. zur i. Tankstelle ($0 \leq i < n$) und der Wert t_0 entspricht der Entfernung vom Startpunkt zur ersten Tankstelle. Zu Beginn der Reise ist sein Auto vollgetankt. Da Prof. Naumann keine Zeit an der Tankstelle zu verlieren beabsichtigt, möchte er so wenige Stopps wie möglich machen.

a) Implementieren Sie die Methode `getStops(int d, int z, int[] t)` der Klasse `assignment4.TripToReno`, die ihm eine minimale Liste der besuchenden Tankstellen effizient (greedy) zurückgibt. Nehmen Sie an, dass für alle Eingaben eine Liste von Tankstellen gefunden werden kann, deren Entfernung kleiner as d ist.

b) Diskutieren Sie die Zeitkomplexität ihrer Lösung im dafür vorgesehenen Quelltextkommentar.

c) Prof. Naumann fährt mit einer konstanten Geschwindigkeit `kmhProf` (z.B. 50km/h). Ein professioneller Fahrradfahrer fährt die gleiche Strecke mit einer Geschwindigkeit `kmhBike` (z.B. 30km/h) und nimmt mit ihm einen Kampf auf. Da Prof. Naumann jedoch sehr lange zum Tanken an den überfüllten Tankstellen braucht, (je Tankstelle 15 Minuten) möchte er wissen, ob das Fahrrad schneller sein wird. Implementieren Sie für ihn die Methode `timeCompare(int kmhProf, int kmhBike, int d, int z, int[] t)` der Klasse `assignment4.TripToReno`, die die Zeit in Minuten zurück gibt, die Prof. Naumann schneller ist - falls der Radfahrer schneller ist wird die Zeit negativ.

[1] https://open.hpi.de/courses/pt2-2016
[2] https://hpi.de/naumann/teaching/current-courses/ss-16/uebung-programmierechnik-ii.html

Aufgabe 2: Dateisortierung

6 P

a) Implementieren Sie das Sortierverfahren namens *Mergesort* zum Sortieren von Objekten der Klasse `ArrayList`[3] in der generischen Methode `sort(ArrayList<X> list)` der Klasse `assignment4.DivideAndConquer`.[4]

b) Implementieren Sie innerhalb der Klasse `assignment4.DivideAndConquer` die Methode `filesort(String inputFile, String outputFile, int maxLineCount)` zum Sortieren von Textdateien. Die Methode soll dabei die Eingabedatei zeilenweise lesen, die Zeilen lexikographisch sortieren und wieder (zeilenweise) in die Ausgabedatei schreiben.[5] Achten Sie bei Ihrer Implementierung darauf, dass zu jedem Zeitpunkt die maximale Anzahl von eingelesenen Zeilen im Speicher nur kleiner oder gleich `maxLineCount` sein darf. Nutzen Sie dazu temporäre Dateien, in denen Sie sortierte Zwischenergebnisse (Teillisten) speichern können und sortieren Sie diese Zwischenergebnisse in einer finalen `merge`-Phase in die Ausgabesdatei ein. Sie können beim `mergen` davon ausgehen, dass `maxLineCount` größer ist als die Anzahl der Teillisten. Achten Sie auch darauf, dass alle temporären Dateien nach dem Aufruf von `filesort` gelöscht werden und verwenden Sie als Encoding stehts `UTF-8`.

   Hinweis: Auf CodeOcean haben wir zum Testen die Dateien `fruits.txt` und `fruits.sorted.txt` bereitgestellt. In den Sources (auf der Lehrstuhlseite) gibt es noch die Dateien `numbers.txt` und `numbers.sorted.txt`, die wegen ihrer Größe nicht auf CodeOcean hochgeladen werden können.

[3] https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
[4] https://docs.oracle.com/javase/tutorial/extra/generics/index.html
[5] http://docs.oracle.com/javase/tutorial/essential/io/file.html#textfiles

Aufgabe 3: Rätsellösung

8 P

Sudoku ist ein Zahlenrätsel, dass auf einem 9×9 großen Spielfeld basiert. Das Spielfeld ist zusätzlich in 3×3 Blöcke aufgeteilt, die jeweils in 3×3 Felder unterteilt sind, insgesamt 81 Felder in 9 Zeilen und 9 Spalten. In einigen dieser Felder sind schon zu Beginn Ziffern zwischen 1 und 9 eingetragen ("Vorgaben"). Die Aufgabe besteht darin, die leeren Felder des Rätsels so zu füllen, dass in jeder der je neun Zeilen, Spalten und Blöcke jede Ziffer von 1 bis 9 genau einmal auftritt. Die drei Bereiche (Zeile, Spalte, Block) sind gleichrangige Einheiten oder Gruppen.

a) Implementieren Sie die Methode `solve(SudokuBoard sudoku)` der Klasse `assignment4.SudokuSolver`, die das gegebene Sudoku Spielfeld (das heißt ein Objekt der Klasse `SudokuBoard`) mittels Backtracking-Algorithmus löst und nur dann `true` zurückgibt, wenn eine korrekte Lösung gefunden wurde.

b) Implementieren Sie als nächstes die Methode `createSudoku(int allocations)` der Klasse `assignment4.SudokuSolver` zum Erstellen eines neuen (noch nicht gelösten) Sudokus. Das neue Sudoku soll dabei so viele Vorgaben besitzen, wie durch den Parameter spezifiziert ist (wenn also `allocations=3`, dann sollen 3 Felder belegt sein). Wichtig ist natürlich auch, dass das so erstellte Sudoku lösbar ist, wobei nicht verlangt ist, dass es eindeutig lösbar sein muss (es kann also auch mehrere richtige Lösungen für das Rätsel geben). Damit nicht immer die gleichen Rätsel erzeugt werden, müssen die Vorbelegungen (Positionen als auch Werte) weitestgehend zufällig gewählt werden. Verwenden Sie dazu den Pseudozufallsgenerator von Java (z.B. über die Methoden `java.util.Random#nextInt(int)` und `java.util.Collections#shuffle(List<?>)`).

Zusatzaufgabe: Kontakt zu Außerirdischen

Wir schreiben das Jahr 2056 und haben erstmals Kontakt zu Außerirdischen aufgenommen! In zwei unterschiedlichen Radaranlagen in Australien und Kalifornien wurden Nachrichten vom Exoplaneten Kepler-186f empfangen. Die Signale sind sehr schwach und von starken Störsignalen verzerrt. Nach zahlreichen schlaflosen Nächten konnte ein Forscherteam die Signale entschlüsseln und auf die Zeichen des Alphabets abbilden. Lediglich die Störsignale sind noch zu entfernen.
Glücklicherweise gibt es stets zwei Varianten jeglicher gesendeter Nachrichten – jene aus Kalifornien (Nachricht `a`) und die aus Australien (Nachricht `b`). Beide Nachrichten enthalten voneinander unabhängige Störsignale. Dabei ist anzunehmen, dass die tatsächliche Nachricht die längste Kette an Zeichen ist, welche in beiden Nachrichten in der gleichen Reihenfolge vorkommen. Diese Kette muss nicht zusammenhängend sein, sondern kann auch von Störsignalen unterbrochen sein.
Sie werden hiermit dazu beauftragt, ein Program zu schreiben, welches die tatsächliche Nachricht möglichst schnell dekodiert. Implementieren Sie dazu in der Klasse `assignment4.MessageReader` die Methode `String findMessage(String a, String b)`, welche die beiden Nachrichten `a` und `b` filtert und die tatsächliche Nachricht zurückgibt. Diese Dekodierungsleistung ist äußerst zeitkritisch, da noch unklar ist, ob uns die Außerirdischen friedlich gesonnen sind und die Börsenanleger schnell erfahren wollen, ob somit Investitionen in Blumen- oder Raketenaktien sinnvoller sind.

Gehen Sie bei der Lösung gemäß der dynamischen Programmierung vor. Ihr Algorithmus sollte eine in $\mathcal{O}(n^2)$ angesiedelte Zeitkomplexität haben. Beschreiben Sie wie üblich in Quelltextkommentaren Ihre Vorgehensweise und Annahmen.

Beispiele:
$a =$ `XHAYLLYIOWZIEZLT`; $b =$ `QQHALKKLOWJJELNT` $\rightarrow$ `HALLOWELT`
$a =$ `FFOOOOBBAARR`; $b =$ `FOOBAR` $\rightarrow$ `FOOBAR`
$a =$ `BANANENEIS`; $b =$ `ANANAS` $\rightarrow$ `ANANS`