Aufgabenblatt 4
Entwurf von Algorithmen

- Abgabetermin: **Freitag, 13.06.14 23:59 Uhr**
- Zur Prüfungszulassung muss ein Aufgabenblatt mit mind. 25% der Punkte bewertet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Abgabe:
  - Geben Sie im Abgabesystem[1] eine ZIP-Datei ab, welche den Quelltext aller implementierten Klassen (Java 7), sowie entsprechende textuelle Lösungen enthält.
  - Achten Sie darauf, dass ihr ZIP-Archiv *keine* kompilierten Klassen (class-Dateien) enthält.
  - Textuelle Lösungen reichen Sie bitte in einer entsprechend benannten *PDF*-Datei (aufgabe_x.pdf) auf oberster Ebene im ZIP-Archiv ein.
  - Der Quelltext soll in einer Ordnerstruktur analog zu assignment3_sources.zip[2] vorliegen: Quellcode im `src` Ordner, etc.
  - Achten Sie darauf, dass Ihr Quelltext gut kommentiert ist.
  - Geben Sie bei der Einreichung Ihrer Lösung den Namen des anderen Gruppenmitglieds unter "Authors" an (pro Gruppe nur eine Abgabe).

Vorbereitung

Laden Sie sich das Archiv assignment4_sources.zip[2] herunter, es enthält das Gerüst (Klassen- sowie Methodendeklarationen und UnitTests), auf dem die folgenden Aufgaben aufbauen.

[1] `https://www.dcl.hpi.uni-potsdam.de/submit/`
[2] `http://hpi-web.de/fileadmin/hpi/FG_Naumann/lehre/SS2014/PTII/assignment4_sources.zip`

Aufgabe 1: Tankstellensuche

6 P

Professor 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. Prof. Naumann möchte so wenige Tankstopps 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.

   Beispiel: `d = 50`, `z = 100`, `int[] t = new int[] {34,37,19}`
   Lösung: `new int[] {0,1}`

b) Diskutieren Sie kurz die Komplexität (Laufzeit) ihrer Lösung und speichern Sie die Lösung als PDF-Datei (aufgabe_1.pdf) und fügen Sie die Datei dem ZIP-Archiv hinzu.

c) Professor 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 Professor Naumann jedoch sehr lange zum Tanken an den überfüllten Tankstellen braucht, (je Tankstelle 15 Minuten) möchte Professor Naumann wissen, ob er bei schneller sein kann als der Radfahrer. Implementieren Sie für ihn die Methode `timeCompare(int kmhProf, int kmhBike, int d, int z, int[] t)` der Klasse `assignment4.TripToReno`, die den Wert 1 zurückgibt, wenn Prof. Naumann schneller ist, -1 wenn der Fahrradfahrer schneller ist und 0, wenn beide gleich schnell sind.

Aufgabe 2: Teile und herrsche

6 P

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

b) Implementieren Sie die Methode `filesort(String inputFile, String outputFile, int maxLineCount)` der Klasse `DivideAndConquer` 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`.

[3] `http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html`
[4] `http://docs.oracle.com/javase/tutorial/extra/generics/methods.html`
[5] `http://docs.oracle.com/javase/tutorial/essential/io/file.html#textfiles`

Aufgabe 3: Sudoku

8 P

Sudoku ist ein Zahlenrätsel, dass auf einem 9×9 großen Spielfeld basiert. Das Spielfeld 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 nur einmal auftritt. Die drei Bereiche (Zeile, Spalte, Block) sind gleichrangige Einheiten oder Gruppen.[6]

a) Implementieren Sie die Methode `solve(SudokuBoard sudoku)` der Klasse `assignment4.SudokuSolver`, die das gegebene Sudoku Spielfeld (`SudokuBoard`-Objekt) 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<?>)`).

[6] `http://de.wikipedia.org/wiki/Sudoku`

Zusatzaufgabe: Das Netz der nächsten Generation

Ein großer Mobilfunkanbieter möchte in Deutschland ein Netz eigener Funkmasten aufstellen. Er hat dazu das Land in 656 kleine Gebiete unterteilt, die er bedienen möchte. Nun sucht er nach einer Möglichkeit, seine Masten optimal auf diese zu verteilen. In jedem Gebiet darf dabei höchstens ein Sendemast platziert werden. Da die Masten eine gewisse Reichweite haben, kann ein Gebiet aber auch von einem umliegenden anderen Gebiet aus mitversorgt werden; es muss also nicht in jedem Gebiet ein Mast stehen. Außerdem dürfen die Masten nur in Deutschland, nicht jedoch in angrenzenden Ländern aufgestellt werden. Es stehen Masten unterschiedlichen Typs zur Verfügung. Typ A ist relativ günstig und einfach, Typ B hat eine bessere Signalstärke und mittlere Reichweite, kostet aber auch mehr und Typ C ist der teuerste mit der höchsten Reichweite. Liegt ein Gebiet im Sendebereich mehrerer Masten, so addieren sich dort alle Signalstärken. Da der Anbieter seinen Kunden eine überall hohe Leistungsfähigkeit seines Services verspricht, soll z.B. in 90% der Gebieten eine Signalstärke von 3 oder besser herrschen. Der Ausbau darf das Budget (z.B. 100.000.000) nicht übersteigen. Der Projektverantwortliche möchte selbstverständlich die Kosten für dieses Vorhaben so gering wie möglich halten. Zusammenfassend sollte also möglichst wenig ausgegeben werden, um eine möglichst hohe Abdeckung mit ausreichend guter Signalstärke zu erreichen.

Das beauftragte Projektteam möchte zum Lösen dieser komplexen Aufgabe den Ansatz der Evolutionären Programmierung umsetzen. Dabei wird zunächst eine zufällige Generation von Problemlösungsversuchen (Individuen) generiert. Dann wird geprüft, ob eines der Individuen dieser Generation bereits die Kriterien erfüllt. Ist dies nicht der Fall, so wird jedem Individuum eine Fitness zugewiesen, die aussagt, zu welchem Grad es geeignet ist, um das Problem zu lösen. Fittere Individuen bekommen eine Chance zu überleben und es in die Nachfolgegeneration zu schaffen (Selektion). Individuen können außerdem zufälligen Änderungen unterliegen (Mutation) oder in ihren Merkmalen mit anderen gekreuzt werden (Crossover). Daraus ergibt sich die nächste Generation. Dieser Vorgang wird iterativ so lange wiederholt, bis schließlich eines der Individuen als Lösung für das gegebene Problem tauglich ist. Eine interessante Einführung in Genetische Algorithmen finden Sie im HPImgzn[7]. Für weitere Informationen zum Thema lesen Sie auch das zur Vorlesung empfohlene Buch "Algorithmen und Datenstrukturen"[8] oder den Wikipedia-Artikel[9].

Um das Programmgerüst zu starten, folgen Sie diesen Schritten:
`Run → Run Configurations... → Java Application → Right Click → New → Main Class: assignment4.evolution.EvolutionGUI → Run`

Sie sehen ein Fenster, in welchem bereits eine erste zufallsgenerierte Generation von Individuen angezeigt wird. Beim Klicken auf den Button "next generation" wird die Methode `evolve()` der Klasse `assignment4.evolution.Evolution` aufgerufen. Sie tut zunächst nichts, als die aktuelle Generation zu randomisieren. Die bisherige Implementierung enthält eine Klasse TransmitterMast, die für die einzelnen Sendemasten steht und die Klasse CountryMap, die Ihnen sagt, wo diese Masten zu platzieren sind. Ein Individuum, also ein Lösungsversuch, wird als MastDistribution umgesetzt. Die Menge Individuen wird in Generation implementiert und diese wird wiederum von der Hauptklasse Evolution verwendet. Ihre Aufgabe als Mitarbeiter des Projektteams ist es nun, sich mit der Idee der Evolutionären Programmierung sowie dem Quellcode vertraut zu machen und die Methode `evolve()` der Klasse `assignment4.evolution.Evolution` zu implementieren. Diese Methode soll die Überführung von einer Generation in die nachfolgende bewerkstelligen. Implementieren Sie auch etwaig benötigte Hilfsmethoden wie zum Beispiel `rateFitness(...)`, `mutate(...)`, `crossover(...)` und `isTerminated()`. Es ist Ihnen gestattet die Zahl der Individuen pro Generation sowie die Zahl der zu erzeugenden Generationen zu variieren (siehe `assignment4.evolution.Evolution#main(...)`). Denken Sie daran, den Code gut zu kommentieren, um eine spätere Nachvollziehbarkeit zu gewährleisten und fügen Sie den Code dem Abgabearchiv hinzu. Diskutieren Sie Ihre Strategien für Mutation, Rekombination, Fitness-Wert und Abbruchkriterium zusätzlich in einer PDF Datei (zusatz.pdf) und stellen Sie den Verlauf einer vollständigen Evolution an einem Beispiel grafisch dar, indem Sie die maximalen Fitnesswerte (y-Achse) für die unterschiedlichen Generationen (x-Achse) abbilden und fügen Sie dieses Diagramm ebenfalls dem PDF hinzu.

[7] Ausgabe 14, S. 50-53
[8] Kapitel I, S. 86-88
[9] `http://de.wikipedia.org/wiki/Genetische_Algorithmen`