Aufgabenblatt 3
Eigenschaften von Algorithmen

- Abgabetermin: **Freitag, 23.05.14 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 assignment3_sources.zip[2] herunter, es enthält das Gerüst (Klassen- sowie Methodendeklarationen und UnitTests), auf dem die folgenden Aufgaben aufbauen.

Aufgabe 1: Korrektheit

5 P

Gegeben sei die Datenstruktur `assignment3.SquareMatrix`, die eine quadratische Matrix repräsentiert.

a) In die Methode `multiply(double alpha)` der Klasse `assignment3.SquareMatrix`, die eine Skalarmultiplikation auf einem Matrix Objekt ausführen soll, hat sich ein kleiner Fehler eingeschlichen. Finden und korrigieren Sie diesen Fehler.

b) Die formale Verifikation von Algorithmen ist sehr aufwändig und kann nicht für alle Methoden in einem Softwaresystem durchgeführt werden. In der Praxis die empirische Methode des Testens durchgesetzt. Für Java stehen Entwicklern zum Überprüfen der Korrektheit eines Algorithmus in Form von Tests Frameworks wie JUnit zur Verfügung.

   Informieren Sie sich über unterschiedliche Metriken zum Messen der Testabdeckung[3][4]. Implementieren Sie im Unittest `assignment3.SquareMatrixTest` (im `test` Ordner) Testfälle, sodass für die Methoden `multiply(double alpha)`, `multiply(Matrix b)`, `min()` und `max()` der Klasse `assignment3.SquareMatrix` eine vollständige C_1-Testabdeckung (Zweigüberdeckung/Branch coverage) erreicht wird. **Tipp:** Code coverage Werkzeuge, wie z.B. das Eclipse-plugin EclEmma[5], können bei der Ermittlung der Testabdeckung helfen.

[1] `https://www.dcl.hpi.uni-potsdam.de/submit/`
[2] `http://hpi-web.de/fileadmin/hpi/FG_Naumann/lehre/SS2014/PTII/assignment3_sources.zip`
[3] `http://de.wikipedia.org/wiki/Dynamisches_Software-Testverfahren`
[4] `http://www.oio.de/testcoverage-clover.htm`
[5] `http://www.eclemma.org/`

Aufgabe 2: Zeitkomplexität

9 P

Im Folgenden soll die Laufzeitkomplexität der Multiplikationsoperationen der Klasse `assignment3.SquareMatrix` (`multiply(double alpha)` sowie `multiply(SquareMatrix b)`) analysiert werden.

a) Diskutieren Sie die Laufzeitkomplexität der beiden Multiplikationsoperationen theoretisch. Speichern Sie die Lösung als Textdatei (aufgabe_2.pdf) und fügen Sie die Datei dem ZIP-Archiv hinzu.

b) Bestimmen Sie die Laufzeitkomplexität der beiden Multiplikationsoperationen empirisch. Erstellen sie dazu eine der Klasse `RuntimeTestExec` mit einer `main` Methode. In dieser `main` Methode sollen die Operationen für zufällig belegte Matrizen verschiedener Größen (z.B. $1, 10, 100, 1000$) ausgeführt werden. Messen Sie die Laufzeit der Operationen mittels java.lang.System.nanoTime(). Wiederholen Sie ihre Messungen und protokollieren Sie Ihre Experimente; geben Sie im Protokoll alle für die Messung relevanten Einflussparameter an. Stellen Sie die Gesamtergebnisse tabellarisch oder graphisch dar und diskutieren Sie die Beobachtungen in der PDF-Datei (aufgabe_2.pdf) und fügen Sie die Datei dem ZIP-Archiv hinzu.

c) Bestimmen Sie, ob die folgenden Aussagen wahr sind und argumentieren sie warum. Fügen sie Ihre Diskussion dem ZIP-Archiv in einer PDF-Datei (aufgabe_2.pdf) hinzu:

1) $O(100 * N^2) \subseteq O(N^3)$
2) $10 * N + 20 \in O(N)$
3) $10 * N^{\frac{3}{2}} \in O(N \log N)$

Aufgabe 3: Platzkomplexität

6 P

Im Folgenden soll die Speicherplatzbedarf der Implementierung der Multiplikationsoperationen aus `assignment3.SquareMatrix` (`multiply(double alpha)` sowie `multiply(SquareMatrix b)`) analysiert werden.

a) Diskutieren Sie den Speicherplatzbedarf der beiden Multiplikationsoperationen theoretisch. Speichern Sie die Lösung als Textdatei (aufgabe_3.pdf) und fügen Sie die Datei dem ZIP-Archiv hinzu.

b) Bestimmen Sie dden Speicherplatzbedarf der beiden Multiplikationsoperationen empirisch. Erstellen Sie dazu eine der Klasse `MemoryTestExec` mit einer `main` Methode. In dieser `main` Methode sollen die Operationen für Matrizen verschiedener Größen (z.B. $1, 10, 100, 1000$) ausgeführt werden. Messen Sie den Speicherverbrauch in der Laufzeitumgebung mit Hilfe der Methoden `gc()`, `totalMemory()` und `freeMemory()` des `Runtime.getRuntime()`-Objektes[6]. **Tipp**: Beachten sie, dass die JVM in der das Programm läuft mit konkreten Heapspace-Parametern `-Xms1G -Xmx1G` gestartet wird. Wiederholen Sie ihre Messungen und protokollieren Sie Ihre Experimente; geben Sie im Protokoll alle für die Messung relevanten Einflussparameter an. Stellen Sie die Gesamtergebnisse tabellarisch oder graphisch dar und diskutieren Sie die Beobachtungen kurz in einer PDF-Datei (aufgabe_3.pdf) und fügen Sie die Datei dem ZIP-Archiv hinzu.

[6] `http://www.vogella.com/tutorials/JavaPerformance/article.html#runtimeinfo_memory`

Zusatzaufgabe: Kursevaluation

Zur Kursevaluation an einer Universität wird die Motivation und Zufriedenheit der Studenten ermittelt. Dazu wird von jedem Studenten eines Kurses sowohl der Motivations- als auch der Zufriedenheitsgrad als Ganzzahl gespeichert. Der Professor möchte nun wissen, was die q-t kleinste Summe aus Motivations- und Zufriedenheitsgrades ist. Dabei können Motivations- und Zufriedenheitswerte von dem gleichen oder unterschiedlichen Studenten stammen.

Gegeben seien zwei Arrays `long[] m` und `long[] z` der Länge K, die die Motivations- bzw. Zufriedenheitswerte der K Studenten beinhalten. Wir können die Summen also ermitteln, indem wir zwei Element `m[i]` und `z[j]` addieren. Für jede Anfrage q_x soll nun also der q_x. kleinste Wert ermittelt werden (q_i entspricht also auch einer Ganzzahl).

Implementieren Sie zum effizienten Lösen dieser Aufgabe die Methode `minSum(int[] q, long[] m, long[] z)` der Klasse `assignment3.Evaluation`. Die Funktion soll die q_x. kleinste Summe (jedes Element in `int[] q` entspricht einem q_x-Wert) aus `long[] m` und `long[] z` ermitteln und zurückgeben.

Zum Beispiel ergibt sich für die die Motivationswerte $m = [1, 2, 3]$, die Zufriedenheitsgrade $z = [4, 5, 6]$ und die drei Anfragen $q_0 = 4$, $q_1 = 9$, $q_2 = 1$ ($q = [4, 9, 1]$), der Aufruf `minSum(new int[] {4, 9, 1}, new long[] {1, 2, 3}, new long[] {4, 5, 6})`. Das erwartete Ergebnis lautet: $[7, 9, 5]$, denn die 4. kleinste Summe ($q_0 = 4$) lautet 7, die 9. kleinste Summe ($q_1 = 9$) 9 und die kleinste Summe ($q_2 = 1$) 5. Dies ergibt sich aus den folgenden 9 Summen:

- $1 + 4 = 5$
- $2 + 4 = 6$
- $1 + 5 = 6$
- $1 + 6 = 7$
- $2 + 5 = 7$
- $3 + 4 = 7$
- $2 + 6 = 8$
- $3 + 5 = 8$
- $3 + 6 = 9$

Wertebereiche

- $1 \leq K \leq 20000$
- $1 \leq Q \leq 500$
- $\forall i < Q : 1 \leq q_i \leq 10000$
- $\forall i < K : 1 \leq m_i \leq 10^{18}$
- $\forall i < K : 1 \leq z_i \leq 10^{18}$