24. OKTOBER 2013

DOKUMENTATION

ZU AUFGABENBLATT 02 AUS DER VORLESUNGSREIHE „ALGORITHMEN UND DATENSTRUKTUREN“

HAW HAMBURG

DOKUMENTATION

ZU AUFGABENBLATT 02 AUS DER VORLESUNGSREIHE „ALGORITHMEN UND DATENSTRUKTUREN“

AUFGABE 1

„Entwerfen Sie eine geeignet Schnittstelle. Verwenden Sie Fließkommazahlen für die Elemente.

Wenn Sie weitere Methoden für sinnvoll halten, so begründen Sie dies kurz.“

Das entsprechende Interface, das die geforderten Funktionen bereitstellt, heißt „QuadMatrix“. Im Anhang dieses Dokumentes ist die vollständige JavaDoc zu unserem entwickelten Projekt.

Es stellt außerdem folgende Funktionen bereit:

getSize() gibt die Länge einer Seite der Matrix zurück. Da es eine quadratische Matrix ist, lässt sich daraus auch wieder die maximale Anzahl der möglichen Werte berechnen lässt.

init() erstellt eine neue Instanz einer gegebenen Matrix-Implementation. Da es drei verschiedene Implementationen gibt, müssen wir den Implementationstyp der Ausgabe bei bspw. einer Addition auf den Implementationstyp der Eingabe anpassen. Dies wird durch **init()** ermöglicht.

equals() ermöglicht den Vergleich von Matrizen. Dies ist für JUnit-Tests erforderlich, und wird bei der Verwendung der Matrizen von Nutzen sein.

space() gibt den verwendeten Platz der einzelnen Instanz wieder. Diese Funktion wird zur Auswertung und Vergleich der einzelnen Implementationen genutzt.

time() gibt die bislang benötigte Zeit der einzelnen Instanz wieder. Diese Funktion wird zur Auswertung und Vergleich der einzelnen Implementationen genutzt.

timeReset() setzt den Zeitzähler einer Instanz auf 0 zurück. Wird eher zu Testzwecken nützlich sein.

setGen(int m, int n, double x) stellt eine Methode bereit, um Matrizen schnell zu befüllen. Diese Methode ist ausdrücklich nicht zum generellen Gebrauch zu nutzen. Sie setzt voraus, dass man keine vorhandenen Werte mit einer 0 überschreibt und wird genutzt, um später die nötigen Testwerte berechnen zu können.

AUFGABE 7

"Wieiviel Platz sparen wir mit den Listenimplementationen tatsächlich ein?"

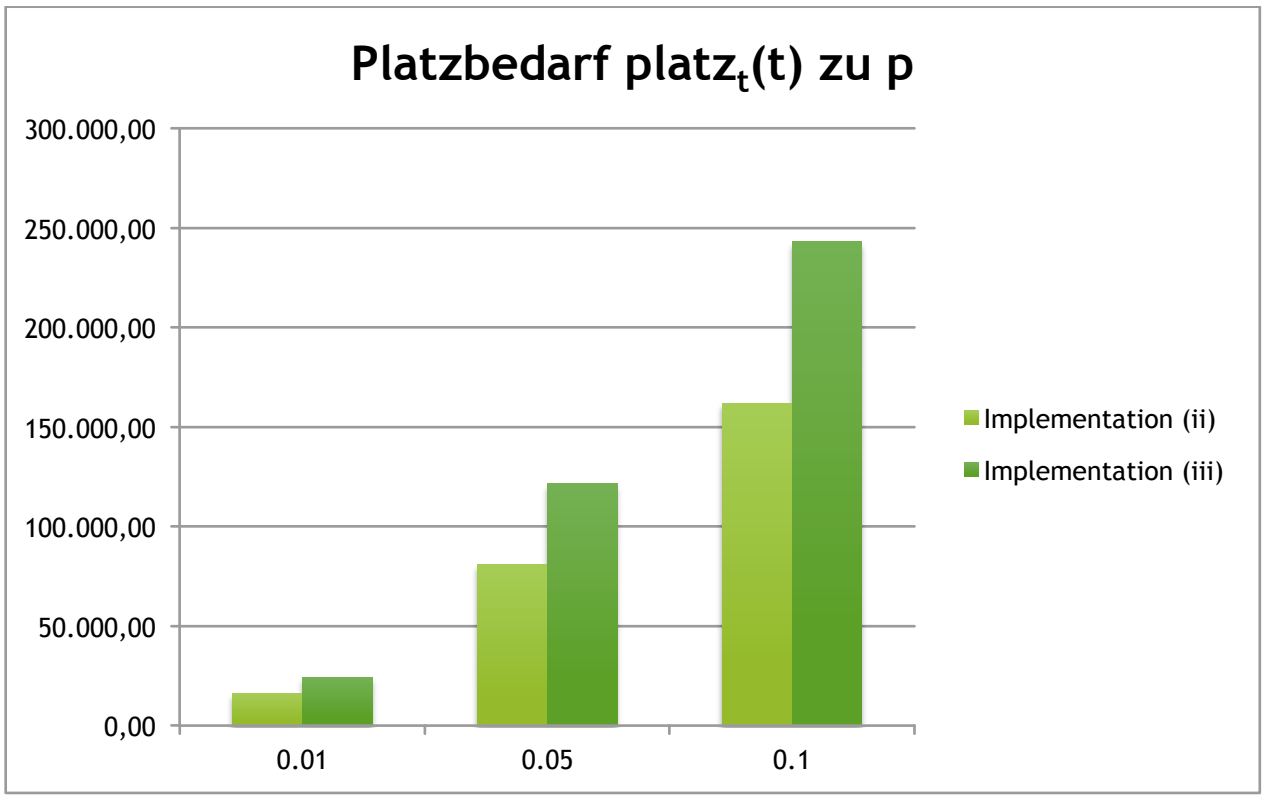
TEILAUFGABE B)

„Stellen Sie den Zusammenhang von Platzbedarf platz $_p(t)$ für diese Werte von p geeignet graphisch dar!"

Experiment: Erstellen Sie *t* Matrizen mit einer Größe von n x n (n sei die letzten 4 Stellen einer unserer Matrikelnummern). Es soll der durchschnittliche Platzverbrauch der jeweiligen Implementation berechnet werden. *t* soll dabei so dimensioniert werden, dass die ersten 5 Stellen der Werte „stabil" sind.

Erwartung: Der Platzverbrauch der Implementation (iii) sollte deutlich höher sein, da jedes Element in 3er Tupeln gespeichert wird = [Spalte,Zeile,Wert].

Ergebnis: Die Auswertung der Ergebnisse ergab folgendes Diagramm, das zeigt, dass die Implementation (iii) einen deutlich höheren Platzverbrauch aufweist. Das Ergebnis entspricht der Erwartung, dass der Platzverbrauch der Implementation (ii) zu 2/3 dem Platzverbrauch der Implementation (iii) entspricht.



TEILAUFGABE C)

„Bestimmen Sie für alle drei Implementationen zu den obigen Werten von p, welche Größe n sie im Mittel speichern können!“

QuadMatrixArrayImpl – Implementation (i) : space(p,n) = n^2 , da hier das Array in jedem Fall mit Werten instanziert wird.

QuadMatrixArrayListImpl – Implementation (ii) : space(p,n) = n*p*2 , da hier 2er Tupel (Spaltenindex und der zugehörige Wert) gespeichert werden, bei denen der Wert nicht 0 ist.

QuadMatrixListImpl – Implementation (iii) : space(p,n) = n*p*3, da hier 3er Tupel (Zeilenindex, Spaltenindex und der zugehörige Wert) gespeichert werden, bei denen der Wert nicht 0 ist.

TEILAUFGABE D)

„Bestimmen Sie aus Ihren Experimenten den Overhead der Listenimplementationen!“

Implementation (i) : Der Platz-Overhead beträgt genau n^2 , da das Array bereits mit 0’en vorbefüllt wird.

Implementation (ii) und (iii) : Der Platz-Overhead beträgt 0, da die Implementationen nicht mit Werten vorbefüllt werden. Sie sind von Anfang an leer.

AUFGABE 8

„Wieviel Zeit-Overhead haben die Listenimplementationen?
Sei n ein festes Argument. (Nehmen Sie die letzten 4 Ziffern Ihrer Matrikelnummer.)“

TEILAUFGABE B)

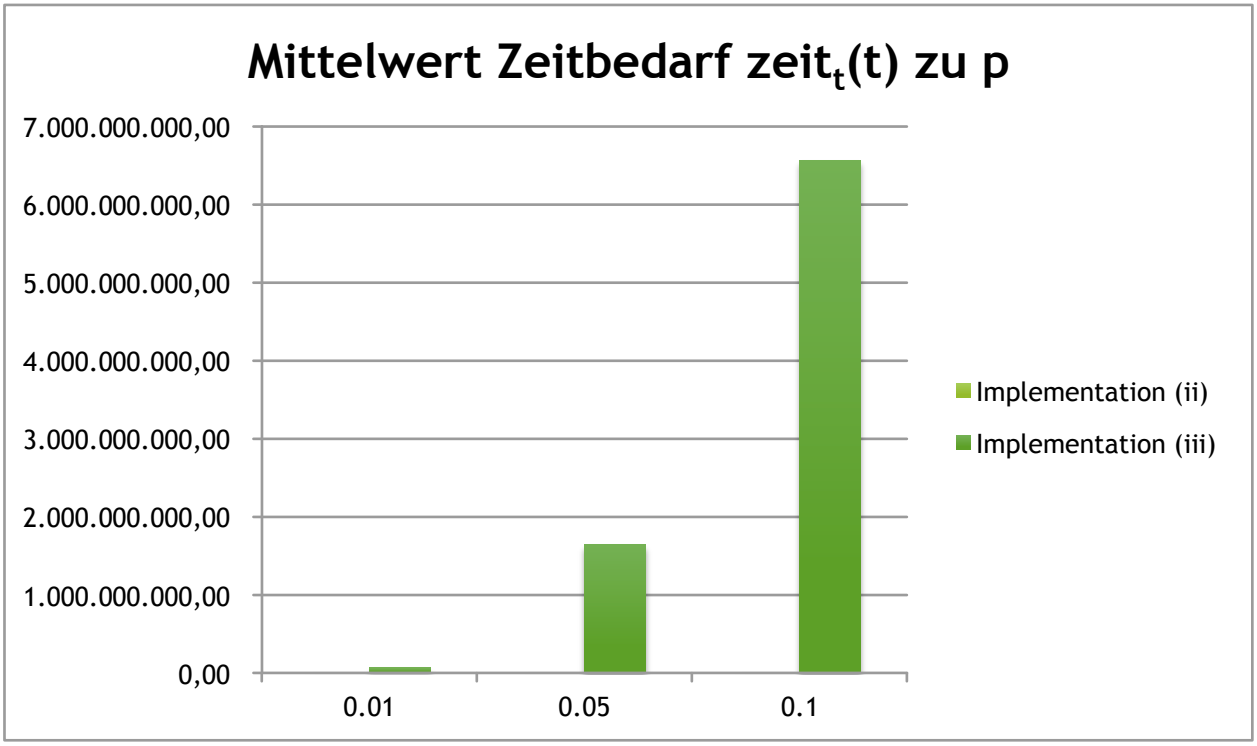
„Wiederholen Sie dieses Experiment t-mal (solange, bis wieder 5 Stellen stabil sind).
Geben Sie den Mittelwert und Streuung von $zeit_p(t)$ für diese Werte von p an und illustrieren Sie die Messreihe graphisch!“

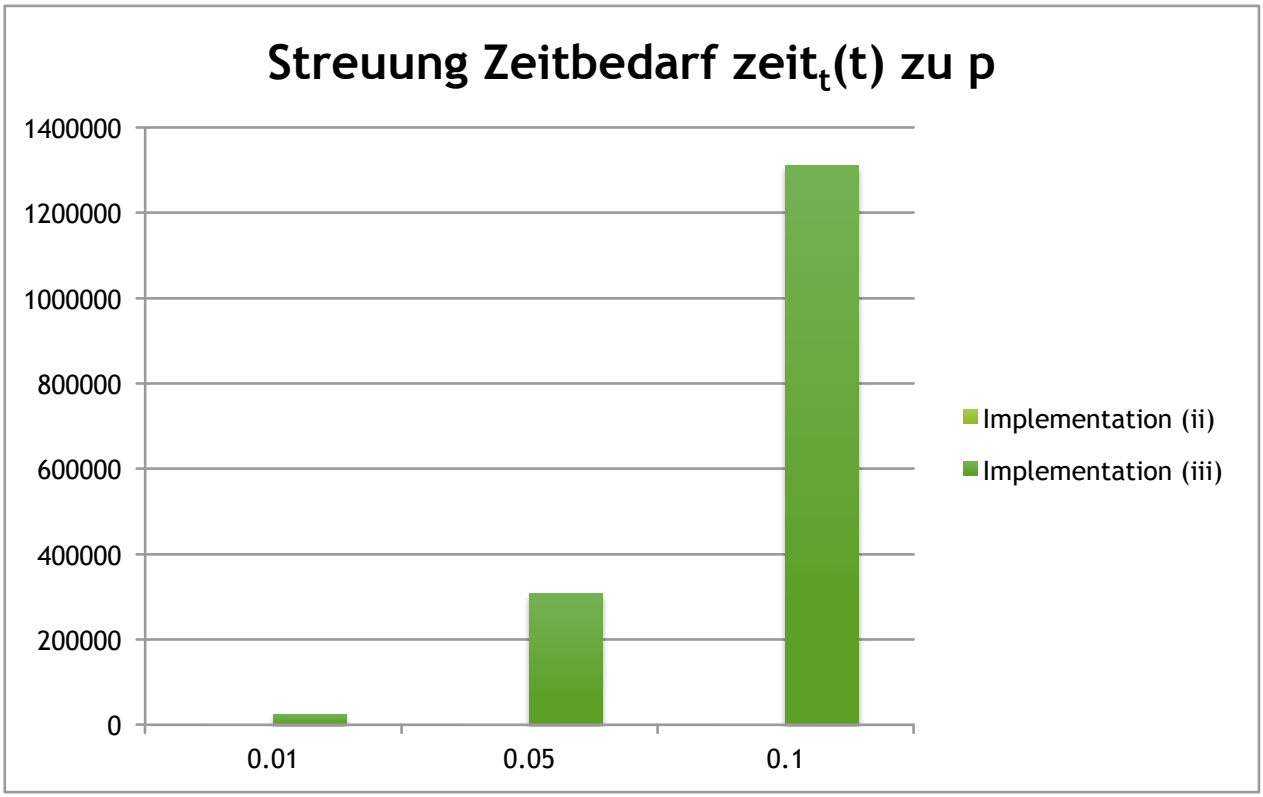
Experiment: Gleicher Experimentaufbau wie bei Aufgabe 7b, nur dass in diesem Experiment der Zeitverbrauch der einzelnen Implementationen gemessen wird.

Erwartung: Dieses Experiment sollte zeigen, dass die Implementation (i) im Verhältnis am schnellsten ist. Denn (ii) und (iii) müssen jedes Mal prüfen, ob der einzufügende Wert ein 0 ist und ob es an der einzufügenden Stelle schon etwas gespeichert ist. Dabei muss der Algorithmus die Stelle erst finden, und wenn sie vorhanden ist, dann löschen. Dies benötigt sehr viel Rechenzeit (je nachdem wie groß die Matrix letztendlich wird).

Ergebnis: Im Folgenden sind die Diagramme der Durchschnittsmesswerte und der Streuung der Messwerte zu sehen. Sie zeigen, dass auch hier die Implementationen (ii) und (iii) sehr langsam sind im Verhältnis zu (i).

Bei ersten Versuchen die Messreihe für (iii) zu erstellen, wurden die Berechnungszeiten derartig lang, dass in 9h nur für t=2000 ein Messwert erstellt werden konnte. Nach Erstellung und Nutzung der Methode **setGen** im Zufallsmatrixgenerator konnten die Messwerte nun in deutlich weniger Zeit berechnet werden.





TEILAUFGABE C)

„Bestimmen Sie aus Ihren Experimenten den Overhead der Listenimplementationen!“

Der Overhead der bei der Listimplementation entsteht liegt bei n^2 da im schlimmsten Fall jedes Element einmal anfassen muss.

Class AbstractQuadMatrix

```
java.lang.Object
    |
    +--A2.AbstractQuadMatrix
```

All Implemented Interfaces:

QuadMatrix

Direct Known Subclasses:

QuadMatrixArrayImpl, QuadMatrixArrayListImpl, QuadMatrixListImpl

---

< Constructors > < Methods >

---

public abstract class **AbstractQuadMatrix**
extends java.lang.Object
implements QuadMatrix

Constructors

AbstractQuadMatrix

```
public  AbstractQuadMatrix()
```

Methods

add

```
public QuadMatrix add(QuadMatrix mtx)
```

---

equals

```
public boolean equals(java.lang.Object mtx)
```

Overrides:

equals in class java.lang.Object

---

getSize

```
public int getSize()
```

---

mul

```
public QuadMatrix mul(QuadMatrix mtx)
```

---

mul

```
public QuadMatrix mul(double n)
```

---

pow

```
public QuadMatrix pow(int n)
```

---

time

```
public int time()
```

---

timeReset

```
public void timeReset()
```

Interface QuadMatrix

< Methods >

---

public interface **QuadMatrix**

Author:

Steffen Giersch, Birger Kamp, Maria Janna Martina Luedemann

Methods

add

```
public QuadMatrix add(QuadMatrix mtx)
```

Matrixaddition ueber zwei QuadMatrix Implementationen

Parameters:

mtx - Zu this zu addierende QuadMatrix

Returns:

Ergebnis der Matrixaddition (QuadMatrix)

---

equals

```
public boolean equals(java.lang.Object mtx)
```

Gleichheitsfunktion fuer QuadMatrix

Parameters:

mtx - Zu vergleichendes Objekt

Returns:

Sind die Matritzen gleich (true) oder nicht (false)

Overrides:

equals in class java.lang.Object

---

get

```
public double get(int m,
                  int n)
```

Gibt das Element am eingegebenen Index zurueck

Parameters:

m - Zeilenindex beginnend mit 1
n - Spaltenindex beginnend mit 1

Returns:

Element an diesem Index

---

getSize

`public int` **`getSize`**`()`

Gibt die Groesse (int) der Matrix zurueck

Returns:

Groesse der Matrix (int)

---

init

`public` `QuadMatrix` **`init`**`(int n)`

Gibt eine neue Instanz der QuadMatrix Implementation mit der Groesse n (int) zurueck

Parameters:

n - Groesse der zu erzeugenden Matrix

Returns:

Neue Instanz einer QuadMatrix Implementation

---

mul

`public` `QuadMatrix` **`mul`**`(``QuadMatrix` `mtx)`

Matrixmultiplikation in der Form this*mtx

Parameters:

mtx - Mit this zu multiplizierende Matrix

Returns:

Ergebnis der Matrixmultiplikation

---

mul

`public` `QuadMatrix` **`mul`**`(double n)`

Skalarmultiplikation fuer QuadMatrix

Parameters:

n - Skalar mit dem multipliziert werden soll

Returns:

Ergebnis der Skalarmultiplikation

---

pow

```
public QuadMatrix pow(int n)
```

Potenzierung von this zum Exponenten n (int)

Parameters:

n - Potenz (int)

Returns:

Ergebnis der Matrixmultiplikation

---

set

```
public void set(int m,
                int n,
                double x)
```

Setzt das Element am eingegebenen Index auf den Wert x (double)

Parameters:

m - Zeilenindex beginnend mit 1
n - Spaltenindex beginnend mit 1
x - Element an diesem Index

---

setGen

```
public void setGen(int m,
                   int n,
                   double x)
```

---

space

```
public int space()
```

Gibt den benoetigten Platz dieser Implementation wieder

Returns:

Benoetigter Platz (int)

---

time

`public int time()`

Gibt die benoetigte Anzahl der Dereferenzierungen vom letzten Reset bis jetzt wieder

Returns:

Anzahl der Dereferenzierungen

---

timeReset

`public void timeReset()`

Resetet die Anzahl der Dereferenzierungen

Class QuadMatrixArrayImpl

```
java.lang.Object
    |
    +--AbstractQuadMatrix
        |
        +--A2.QuadMatrixArrayImpl
```

All Implemented Interfaces:

QuadMatrix

< Constructors > < Methods >

---

public class **QuadMatrixArrayImpl**
extends AbstractQuadMatrix

Constructors

QuadMatrixArrayImpl

`public QuadMatrixArrayImpl(int n)`

Methods

get

```
public double get(int m,
                  int n)
```

---

init

```
public QuadMatrix init(int n)
```

---

set

```
public void set(int m,
                int n,
                double x)
```

---

setGen

```
public void setGen(int m,
                   int n,
                   double x)
```

---

space

```
public int space()
```

Class QuadMatrixArrayListImpl

```
java.lang.Object
    |
    +--AbstractQuadMatrix
        |
        +--A2.QuadMatrixArrayListImpl
```

All Implemented Interfaces:
QuadMatrix

< Constructors > < Methods >

---

public class **QuadMatrixArrayListImpl**

extends AbstractQuadMatrix

Constructors

QuadMatrixArrayListImpl

```
public  QuadMatrixArrayListImpl(int n)
```

Methods

get

```
public double get(int m,
                  int n)
```

---

init

```
public QuadMatrix init(int n)
```

---

set

```
public void set(int m,
                int n,
                double x)
```

---

setGen

```
public void setGen(int m,
                   int n,
                   double x)
```

---

space

```
public int space()
```

Class QuadMatrixGenerator

```
java.lang.Object
    |
    +--A2.QuadMatrixGenerator
```

< Constructors > < Methods >

---

public class **QuadMatrixGenerator**
extends java.lang.Object

Constructors

QuadMatrixGenerator

```
public  QuadMatrixGenerator()
```

Methods

random

```
public static QuadMatrix random(QuadMatrix mtx,
                                int x,
                                double n,
                                double m)
```

Gibt eine Zufallsgenerierte QuadMatrix der Implementation von mtx wieder

Parameters:

mtx - Typ der Matrix, der wiedergegeben werden soll
x - Groesse der Matrix
n - Wahrscheinlichkeit, dass ein Element nicht 0.0 ist
m - Reichweite von 0..m die ein Element haben kann

Returns:

Generierte Matrix

Class QuadMatrixListImpl

```
java.lang.Object
    |
    +--AbstractQuadMatrix
        |
        +--A2.QuadMatrixListImpl
```

All Implemented Interfaces:
QuadMatrix

< Constructors > < Methods >

---

public class **QuadMatrixListImpl**
extends AbstractQuadMatrix

Constructors

QuadMatrixListImpl

```
public  QuadMatrixListImpl(int n)
```

Methods

get

```
public double get(int m,
                  int n)
```

---

init

```
public QuadMatrix init(int n)
```

---

set

```
public void set(int m,
                int n,
                double x)
```

---

setGen

```
public void setGen(int m,
                   int n,
                   double x)
```

---

space

```
public int space()
```

Class main

```
java.lang.Object
    |
    +--A2.main
```

< Constructors > < Methods >

---

public class **main**
extends java.lang.Object

Constructors

main

```
public  main()
```

Methods

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args -

INDEX

A

add ... *1*
add ... *3*
AbstractQuadMatrix ... *1*
AbstractQuadMatrix ... *1*

E

equals ... *1*
equals ... *3*

G

get ... *3*
get ... *7*
get ... *8*
get ... *10*
getSize ... *2*
getSize ... *4*

I

init ... *4*
init ... *7*
init ... *8*
init ... *10*

M

main ... *11*
main ... *11*
main ... *11*
mul ... *2*
mul ... *2*
mul ... *4*
mul ... *4*

P

pow ... *2*
pow ... *5*

Q

QuadMatrix ... *2*
QuadMatrixArrayImpl ... *6*
QuadMatrixArrayImpl ... *6*
QuadMatrixArrayListImpl ... *7*
QuadMatrixArrayListImpl ... *8*
QuadMatrixGenerator ... *9*
QuadMatrixGenerator ... *9*
QuadMatrixListImpl ... *10*
QuadMatrixListImpl ... *10*

R

random ... *9*

S

set ... *5*
set ... *7*
set ... *8*
set ... *10*
setGen ... *5*
setGen ... *7*
setGen ... *8*
setGen ... *11*
space ... *5*
space ... *7*
space ... *8*
space ... *11*

T

time ... *2*
time ... *6*
timeReset ... *2*
timeReset ... *6*