# Übungsblatt 12

## 2 Aufgaben, 20 Punkte

Objektorientierte Modellierung und Programmierung (inf031) Sommersemester 2020
Carl von Ossietzky Universität Oldenburg, Fakultät II, Department für Informatik

Dr. C. Schönberg

---

**Ausgabe:** 2020-07-07 14:00 **Abgabe:** 2020-07-14 12:00

**Aufgabe 1:** *Fibonacci-Zahlen* **(1 + 3 + 2 + 3 + 1 + 1 Punkte)**

Erinnern Sie sich an die Fibonacci-Zahlen:

$$f(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 1, & \text{falls } n = 1 \\ f(n-1) + f(n-2), & \text{sonst} \end{cases}$$

Gegeben sei folgende Basis-Klasse zur Berechnung:

```
1 public abstract class Fibonacci {
2
3   public abstract long calculate(int n);
4
5 }
```

**a)** Schreiben Sie eine Klasse
**`public class FibonacciRecursive extends Fibonacci`**,
welche die `calculate`-Methode *rekursiv* implementiert.

**b)** Schreiben Sie eine Klasse
**`public class FibonacciParallel extends Fibonacci`**,
welche die `calculate`-Methode *rekursiv mit mehreren Threads* implementiert. Jeder rekursive Aufruf soll in einem eigenen Thread erfolgen.

**c)** Schreiben Sie eine Klasse
**`public class FibonacciDynamic extends Fibonacci`**,
welche die `calculate`-Methode mittels *Dynamischer Programmierung mit Memoisation* implementiert. Verwenden Sie als Speicher eine Datenstruktur, die sich bei Bedarf leicht erweitern lässt. So können Sie die Datenstruktur über mehrere Aufrufe der Methode hinweg bestehen lassen, so dass spätere Aufrufe von früheren Berechnungen profitieren können, und nicht nur die Berechnung des aktuellen Aufrufs beschleunigt wird.

**d)** Schreiben Sie eine Klasse
**`public class FibonacciDynamicParallel extends Fibonacci`** **oder**
**`public class FibonacciDynamicParallel extends FibonacciDynamic`**,
welche die `calculate`-Methode mittels *Dynamischer Programmierung mit Memoisation* **und dabei** *mit mehreren Threads* implementiert.

**e)** Schreiben Sie eine JUnit-Testklasse
`class FibonacciTest`,
welche alle vier Implementierungen nacheinander für die Zahlen

$$3, 5, 8, 12, 9, 18, 15, 10, 7, 11, 20$$

testet. Dazu sollen jeweils die vier Ergebnisse auf Gleichheit geprüft werden, unter der Annahme, dass eins davon wohl korrekt sein wird oder zumindest nicht alle vier den gleichen Fehler enthalten.

**f)** Erweitern Sie die JUnit-Testklasse so, dass sie auch die durchschnittliche Zeit für die Berechnung durch die einzelnen Verfahren misst. Ein naïver Beobachter könnte vermuten, dass die Verfahren der Reihe nach immer effizienter die Ergebnisse berechnen, also dass gilt:
timeRecursive $<$ timeParallel $<$ timeDynamic $<$ timeDynamicParallel.
Vergleichen Sie die gemessenen Ergebnisse mit dieser Vermutung und erklären Sie eventuelle Abweichungen.

**Aufgabe 2:** *TSP* **(8 + 1 Punkte)**

Gegeben sei ein Flugnetz (London, Paris, Frankfurt, Bremen, Stockholm, Wien) in Form der folgenden Adjazenzmatrix (redundante Zellen leergelassen):

| | Bremen | Frankfurt | London | Paris | Stock-holm | Wien |
|---|---|---|---|---|---|---|
| Bremen | - | 340 | 629 | 652 | 899 | 760 |
| Frankfurt | | - | 630 | 469 | 1199 | 603 |
| London | | | - | 341 | 1430 | 1233 |
| Paris | | | | - | 1544 | 1035 |
| Stockholm | | | | | - | 1244 |
| Wien | | | | | | - |

Für eine *Tabusuche* sei folgende Konfiguration gegeben:

- Startlösung $x_{start} = (BFLPSWB)$
- Gütefunktion $q(x) =$ Länge der Rundreise, sodass $q(x_{start}) = 4859\ km$
- Nachbarschaftsmenge $N(x) =$ Permutationen von $x$ durch Austauschen von zwei disjunkten Kanten (also beispielsweise Ersetzen von BF und SW durch BS und FW – wenn Sie sich den Graphen aufzeichnen, werden Sie schnell erkennen, dass es beim Entfernen von zwei disjunkten Kanten immer nur genau eine Möglichkeit gibt, wieder einen Zyklus herzustellen)
- Auswahl der Nachbarlösung $x'$: Wahl des Nachbarn, der die größte Verbesserung der Gütefunktion mit sich bringt
- Wartezeit $w = 3$

Angenommen, diese Tabusuche sei bereits seit einigen Durchlaufen aktiv und sei nun in folgendem fiktiven Zustand:

- Aktuelle Lösung $x = (BSFPLWB)$
- Beste Lösung $x_{best} = x_{start} = (BFLPSWB)$
- Tabuliste = $\{(BSFWPLB, 2), (BSLPFWB, 1), (BSFPLWB, 3)\}$

**a)** Führen Sie **einen Durchlauf** des Algorithmus der Tabusuche ausgehend von diesem Zustand aus durch. Geben Sie anschließend die Nachbarschaftsmenge $N(x)$, die ausgewählte Lösung $x'$, die beste Lösung $x_{best}$ sowie die Tabuliste an.

**b)** Welche Lösung $x_{best} \in N(x)$ würde bei der lokalen Suche ausgewählt werden?