Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Datenstrukturen und Algorithmen (SS 2013)

Übungsblatt 5

Abgabe: Montag, **27.05.2013**, 14:00 Uhr

- Die Übungen sollen in Gruppen von zwei bis drei Personen bearbeitet werden.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auf die abgegebenen Lösungen.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auch in die Quellcode-Dateien.
- Geben Sie Ihre Lösungen am **Anfang** der Globalübung, montags, 14:00 Uhr, ab.
- Schicken Sie den jeweiligen Quellcode bitte per **E-Mail** direkt an Ihre/n Tutor/in.
- Geben Sie außerdem den ausgedruckten Quellcode zusammen mit den schriftlichen Lösungen ab.
- Zu spät abgegebene Lösungen werden nicht bewertet.
- Sofern nicht anders gefordert, müssen alle Lösungen und Zwischenschritte kommentiert werden.

Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Aufgabe 1 (Dynamische Programmierung [**10 Punkte**])

In rekursiven Programmen werden Werte oft unnötigerweise mehrfach berechnet, so z.B. bei der naiven Implementierung einer Funktion zur Berechnung der Fibonacci-Zahlen:

```
fibonacci(n)
    if (n < 2) return n;
    return fibonacci(n-1) + fibonacci(n-2)
```

Man mache sich das an einem Beispiel klar! Bei der dynamischen Programmierung geht man daher umgekehrt vor: Ausgehend vom Basisfall der Rekursion berechnet man weitere Werte und speichert diese für spätere Verwendung ab:

```
fib[0] = 0; // Basisfall der
fib[1] = 1; // Rekursion
for (i = 2; i <= n; ++i)
    fib[i] = fib[i-1] + fib[i-2]
// Ergebnis steht in fib[n]
```

Überzeugen sie sich, dass hier jede Fibonacci-Zahl nur einmal berechnet wird! In der folgenden Aufgabe soll diese Technik nochmals eingeübt werden.
Gegeben seien n verschiedene, positive Münzwerte $a_1, \ldots, a_n \in N^+$ und ein Betrag $m \in N_0$. Gesucht ist die Zahl der möglichen n-Tupel $(k_1, \ldots, k_n)$ mit

$$\sum_{l=1}^{n} k_l a_l = m$$

also die Zahl der Möglichkeiten, m durch die gegebenen Münzwerte darzustellen. Hierbei wollen wir optimistischerweise annehmen, dass von jeder Münzsorte beliebig viele zur Verfügung stehen. Außerdem vereinbaren wir folgenden Sonderfall: Der Betrag $m = 0$ lässt sich immer genau einmal durch 0 Münzen darstellen.
Wir verallgemeinern zunächst das Problem und fragen stattdessen nach der Zahl der Möglichkeiten einen Wert $0 \leq j \leq m$ durch nur $0 \leq i \leq n$ Münzwerte darzustellen. Diese Zahl nennen wir s_{ij}. Offensichtlich erhalten wir dann die Lösung zum ursprünglichen Problem in s_{nm}. Die Werte s_{ij} lassen sich bequem in einer Tabelle speichern deren Zeilen mit i und deren Spalten mit j nummeriert werden.

1. Welchen Wert hat s_{0j} für $0 \leq j \leq m$, d.h. welche Einträge stehen in der ersten Zeile der Tabelle? (Beachten Sie den Sonderfall für $j = 0$.) [1 Punkt]

2. Welchen Wert hat s_{i0} für $1 \leq i \leq n$, d.h. welche Einträge stehen in der ersten Spalte der Tabelle? [1 Punkt]

3. Es seien $n = 3, m = 10, a_1 = 2, a_2 = 5$ und $a_3 = 3$. Stellen Sie die zugehörige Tabelle auf und geben Sie die verschiedenen Möglichkeiten, den Betrag 10 darzustellen an. [2 Punkte]

4. Überlegen Sie sich eine Rekursions-Formel für den (i, j)-ten Eintrag der Tabelle. Was können Sie als Basis-Fall der Rekursion verwenden? [4 Punkte]

5. Skizzieren Sie einen Algorithmus der mittels der Rekursionsformel die Tabelle vollständig ausfüllt. Welche Zeit- und Platzkomplexität hat dieser Algorithmus in Abhängigkeit von n und m? [2 Punkte]

Aufgabe 2 (Sortieralgorithmen [**10 Punkte**])

(a) In der Vorlesung wurde das *Insertion-Sort*-Verfahren vorgestellt (Foliensatz 2.3.2, Folie 11 ff.). In dem zu dieser Übungsaufgabe bereitgestellten Quellcode finden Sie die Java-Klasse `InsertionSort.java`, die eine leere Methode `sort()` beinhaltet. Beim Instanziieren der Klasse wird das zu sortierende Array übergeben und in das Member-Array `int[] A` kopiert. Implementieren Sie die Methode `sort()` so, dass das Array `A` mit Hilfe des Insertion-Sort Algorithmus sortiert wird. Überprüfen Sie Ihre Ergebnisse durch Ausführen der von uns in der Klasse `MainClass` bereitgestellten `main`-Methode. [2 Punkte]

(b) Insertion-Sort hat eine *Worst-Case*- und *Average-Case*-Komplexität von $O(n^2)$, wobei n die Eingabegröße ist. Wie könnte man die Laufzeit des Algorithmus verbessern? Begründen Sie ihre Antwort und analysieren Sie die *Best*-, *Worst*- und *Average-Case*-Laufzeit. Was fällt Ihnen bezüglich der *Best-Case*-Laufzeit im Vergleich zur normalen Version des Algorithmus auf? [3 Punkte]

(c) Ein Sortieralgorithmus ist "stabil", wenn er die Reihenfolge zweier Elemente mit demselben Sortierschlüsselwert beim Sortieren erhält. Erklären Sie, warum Quick-Sort diese Eigenschaft nicht besitzt und erweitern Sie den Algorithmus so, dass er stabil ist. Sie brauchen hierzu nichts implementieren, sondern lediglich die notwendigen Erweiterungen in Worten erklären und ggf. in Form von Pseudocode notieren. [5 Punkte]