technische universität dortmund

Fakultät für Informatik
Lehrstuhl 11

Erik Thordsen
Lena Krieg
Sebastian Konietzny

Sommersemester 2023
17. Mai

DAP2 Praktikum – Blatt 7

Abgabe: ab 23. Mai

Studienleistung

- Zum Bestehen des Praktikums muss jeder Teilnehmer die folgenden Leistungen erbringen:
  - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
  - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
- Im Krankheitsfall kann ein Testat bei Vorlage eines Attests in der folgenden Woche nachgeholt werden.
- Wenn ein Praktikumstermin auf einen Feiertag fällt, müssen Sie sich an einem beliebigen anderen Praktikumstermin in der gleichen Woche testieren lassen.
- Ansonsten kann die Testierung **nur in der zugeteilten Gruppe** garantiert werden.
- Bitte bereiten Sie den Tester sowie in den Aufgaben angegebene Beispieltests vor, bevor Sie sich testieren lassen!

Wichtige Information (im Moodle verfügbar)

- Beachten Sie die Erklärung des **Ablaufs (Blatt A)**.
- Beachten Sie die **Regeln und Hinweise (Blatt R)** in der aktuellsten Version!
- Beachten Sie die **Hilfestellungen (Blatt H)** in der aktuellsten Version!

Kurzaufgabe 7.1: Fibonacci (5 Punkte)

Die Fibonacci Zahlen sind, wie in der Vorlesung beschrieben, wie folgt definiert:

$$F_n = \begin{cases} n & \text{wenn } n \leq 1 \\ F_{n-1} + F_{n-2} & \text{sonst} \end{cases}$$

Eine naive Umsetzung der obigen Formel in einen rekursiven Algorithmus würde eine *exponentielle* Laufzeit haben. Hier soll die n-te Fibonacci Zahl mithilfe der *dynamischen Programmierung* effizient berechnet werden

Implementieren Sie die Klasse `AufgabeB7A1` mit den folgenden Methoden:

- `fibDyn(int n)` berechnet die n-te fibonaccizahl in $O(n)$ Schritten für eine nicht-negative Ganzzahl n.
- `main(string[] args)` verarbeitet wie üblich die Eingaben: Eine Zahl n wird eingegeben und die n-te fibonacci-Zahl berechnet und ausgegegen. Fehler und falsche Eingaben werden abgefangen.

Beispiele:

```
> java AufgabeB7A1 10
> 55
> java AufgabeB7A1 12
> 144
```

Geforderte Klassen und Methoden

```
public class AufgabeB5A1{
   public static void main(String[] args) {...}
   public static int fibDyn(int n) {...}
}
```

Kurzaufgabe 7.2: Rucksack (11 Punkte)

Das Rucksackproblem ist ein bekanntes kombinatorisches Optimierungsproblem. Es geht darum, aus einer gegebenen Menge von Gegenständen eine Teilmenge auszuwählen, die in einen Rucksack passt und dabei den größtmöglichen Gesamtwert hat.

Gegeben sei:

- n: Anzahl der Gegenstände
- C: Kapazität des Rucksacks
- w_i: Gewicht des Gegenstands i
- v_i: Wert des Gegenstands i

Gesucht ist:

- Eine Teilmenge $S \subseteq \{1, \ldots, n\}$ der Gegenstände, sodass die folgenden Bedingungen erfüllt sind:

$$\sum_{i \in S} w_i \leq C \quad \text{(Gewichtsbeschränkung)}$$
$$\text{maximiere} \sum_{i \in S} v_i \quad \text{(Wertmaximierung)}$$

Dabei kann das Problem unterteilt werden in Teilprobleme. $T[i][j]$ repräsentiert den maximalen Gesamtwert, der erreicht werden kann, wenn wir die ersten i Gegenstände betrachten und den Rucksack mit einer Kapazität von j füllen möchten. So kann T wie folgt berechnet werden:

$$T[i][j] = \begin{cases} 0 & \text{wenn } i = 0 \text{ oder } j = 0 \\ \max(\text{werte}[i-1] + T[i-1][j - \text{gewichte}[i-1]], T[i-1][j]) & \text{wenn gewichte}[i-1] \leq j \\ \text{tabelle}[i-1][j] & \text{sonst} \end{cases}$$

Nutzen Sie diese Definition von T um das Rucksackproblem mithilfe der *dynamischen Programmierung* zu lösen.

Implementieren Sie die Klasse `AufgabeB7A2` mit den folgenden Methoden:

- `void main(String[] args)`
  Bekommt einen Parameter welcher die Rucksackkapazität bestimmt. Außerdem werden die Gewichte sowie Gegenstandswerte als zwei Listen mit Leerzeile getrennt übergeben (s. Beispieldatei `B7A2Input1.txt`). Das Programm soll danach überprüfen, ob auch wirklich das Ende des Eingabestreams erreicht wurde und ansonsten eine entsprechende Meldung ausgeben und terminieren (s. Beispiele). Ausgegeben werden soll der Wert der optimalen Lösung des Rucksackproblems, also die *Summe der eingepackten Gegenstandwerte*.

- `int[] getInput(Scanner scanner) throws NumberFormatException`
  Liest aus dem Scanner die nächsten zusammenhängenden Zeilen aus Ganzzahlen ein. Sollte bis zu der nächsten Leerzeile oder bis zum Ende des Scanners eine Zeile aus einer Nicht-Ganzzahl bestehen, soll die `NumberFormatException` weitergereicht werden. Die gelesenen Ganzzahlen sollen als Array zurückgegeben werden.

- `int[][] knapsack(int[] values, int[] weights, int capacity)`
  Berechnet die optimale Lösung des Rucksackproblems und gibt eine Tabelle mit den Zwischenergebnissen zurück. Die Laufzeit soll in $O(\texttt{values.length} \cdot \texttt{capacity})$ sein.

Beispiele:

```
> cat B7A2Input1.txt | java AufgabeB7A2 10
> 14
> cat B7A2Input2.txt | java AufgabeB7A2 12
> Input did not end after second list.
> cat B7A2Input3.txt | java AufgabeB7A2 12
> The number of values does not match the number of weights.
```

Geforderte Klassen und Methoden

```
public class AufgabeB5A2{
   public static void main(String[] args) {...}
   public static int[] getInput(Scanner scanner) throws NumberFormatException
      {...}
   public static int[][] knapsack(int[] values, int[] weights, int capacity)
      {...}
}
```