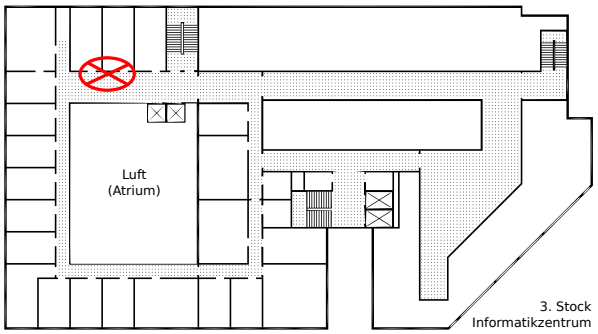
Abteilung Algorithmik **Sommer 2017**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Arne Schmidt

Algorithmen und Datenstrukturen II
Übung 3 vom 15. 05. 2017

Abgabe der Lösungen bis zum Montag, den 29. 05. 2017 um 13:15 im Hausaufgabenrückgabeschrank bei Raum IZ 337. Es werden nur mit einem dokumentechten Stift geschriebene Lösungen gewertet.
Bitte die Blätter zusammenheften und vorne deutlich mit eigenem Namen, Matrikel- und Gruppennummer, sowie Studiengang versehen!



Auf diesem Blatt gibt es 40 Punkte, erreichbar (und abzugeben) sind aber lediglich Aufgaben im Gesamtwert von maximal 30 Punkten. Mehr wird nicht gewertet! Wähle **drei** Aufgaben aus, die Du bearbeitest.

Aufgabe 1 (DP für Knapsack): Wir betrachten in dieser Aufgabe den Dynamic-Programming-Algorithmus für Knapsack aus der Vorlesung (Algorithmus 1.15).

a) Wende den Algorithmus auf folgende Instanz an:

i	1	2	3	4	5	6	7	8
z_i	5	6	8	5	6	1	3	4
p_i	1	4	4	2	3	1	1	4

und $Z = 18$.

Fülle dazu die folgende Tabelle aus (der Eintrag in der Zeile i und der Spalte x entspricht dem Wert $P(x, i)$):

$i \backslash x$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0																			
1																			
2																			
3																			
4																			
5																			
6																			
7																			
8																			

b) Ändere den Algorithmus so ab, damit er für Integer Knapsack funktioniert!

(8+2 Punkte)

Aufgabe 2 (DP Modellierung I - Ein Zwei-Spieler-Spiel): In dieser Aufgabe betrachten wir folgendes Zwei-Spieler-Spiel: Auf einem Tisch liegen n Münzen $1, \ldots, n$ mit jeweils Wert p_i in einer Reihe. Zwei Spieler, nennen wir sie Alice und Bob, ziehen nun abwechselnd eine Münze von einem der beiden Enden der Münzreihe. Sieger ist der Spieler, der am Ende einen größeren Gesamtmünzwert besitzt. Beispiel:

$$2, 5, 1, 10, 50, 2, 2, 10$$

Alice wählt zunächst die 10 und Bob danach eine zwei von rechts. Es bleibt über:

$$2, 5, 1, 10, 50, 2$$

Wählt Alice nun die rechte 2, kann Bob die Münze mit Wert 50 einstecken. Daher nimmt Alice die linke Münze. Da Bob die 50 nicht Alice überlassen möchste, wählt dieser die linke Münze. Es bleibt über:

$$1, 10, 50, 2$$

Die pfiffige Alice merkt, dass sie definitv die 50 einstecken wird, wenn sie die 1 für sich einnimmt. Bob muss dann die 10 oder 2 einstecken, wodurch die 50 frei wird. Das Spiel neigt sich nach weiteren Ziehungen zu Ende und die beiden Spieler können folgende Münzen gewählt haben:

Alice: {10, 2, 1, 50}; Bob: {2, 5, 10, 2}. Es steht 63 zu 19; ein klarer Sieg für Alice.

Da Alice immer gewinnen möchte, sucht sie nach einem Verfahren, ob und wie sie gewinnen kann. Konstruiere dazu einen Dynamic-Programming-Algorithmus, der für eine gegebene Reihe von Münzen entscheidet, ob Alice gewinnen kann. (*Hinweis:* Es darf angenommen werden, dass Alice anfängt zu ziehen.) **(10 Punkte)**

Aufgabe 3 (DP Modellierung II - Matrix Chain Multiplication): Gegeben sei eine Sequenz von Matrizen $\langle M_1, ..., M_n \rangle$. Für $i = 1, ..., n$ sei M_i eine $d_i \times d_{i+1}$-Matrix. Somit ist die Matrixmultiplikation $M_1 \cdot M_2 \cdot ... \cdot M_n$ wohldefiniert.

Anhand des folgendem Beispiels kann man erkennen, dass die Klammerung des Ausdrucks $M_1 \cdot M_2 \cdot ... \cdot M_n$ einen Einfluss auf die Anzahl der Multiplikationen von einzelnen Einträgen hat: $M_1 \in \mathbb{R}^5 \times \mathbb{R}^5$, $M_2 \in \mathbb{R}^5 \times \mathbb{R}^{100}$ und $M_3 \in \mathbb{R}^{100} \times \mathbb{R}^3$. Der Ausdruck $(M_1 \cdot M_2) \cdot M_3$ verursacht somit $5 \cdot 5 \cdot 100 + 5 \cdot 100 \cdot 3$ Multiplikationen, während $M_1 \cdot (M_2 \cdot M_3)$ nur $5 \cdot 100 \cdot 3 + 5 \cdot 5 \cdot 3$ Multiplikationen verursacht.

Entwirf einen Algorithmus, der mittels dynamischer Programmierung berechnet, wie viele Einzelmultiplikationen bei einer optimalen Klammerung nötig sind.

Verwende hierzu folgenden Ansatz: Für ein $k \in \{1, ..., n-1\}$ ergibt sich $(M_1 \cdot ... M_k) \cdot (M_{k+1} \cdot ... \cdot M_n)$. Die damit nötigen Multiplikationen sind die, die für die Berechnung von $M_1 \cdot ... \cdot M_k =: M$, $M_{k+1} \cdot ... \cdot M_n =: M'$ und $M \cdot M'$ nötig sind.

Hinweise zur Matrizenmultiplikation gibt es in der 3. großen Übung. **(10 Punkte)**

Aufgabe 4 (Implementierung Knapsack): Implementiere für KNAPSACK den sparsamen Dynamic-Programming-Algorithmus aus der Vorlesung (Algorithmus 1.17), sodass nicht nur der Wert einer Lösung, sondern auch eine Teilmenge der Objekte zurückgegeben wird, die diesen Wert realisiert.

Nutze dazu die Javavorlage und die Testfälle, die auf der Vorlesungsseite[1] zur Verfügung stehen.

a) Welche Instanzen kannst Du innerhalb von 10 Minuten lösen? *Hinweis:* Dein Algorithmus muss nicht alle Testinstanzen lösen, da diese teils sehr komplex sind.

b) Welche Instanzen sind wegen Speichermangels nicht lösbar (die Größe des *Java Heaps* soll nicht verändert werden)? Begründe Deine Antwort kurz!

Wir testen Deine Software mit
javac -cp *:. Knapsack1234567.java && java .cp *:. Knapsack1234567 < instance_xyz

Zur Abgabe: Ersetze 1234567 durch Deine Matrikelnummer und gib die Javadatei per Mail an Deinen entsprechenden Betreuer ab. Nenne in der Mail Name, Matrikel- und Gruppennummer. Es gilt dieselbe Frist wie für die anderen Aufgaben. **(8+1+1 Punkte)**

[1] `http://www.ibr.cs.tu-bs.de/courses/ss17/aud2/`