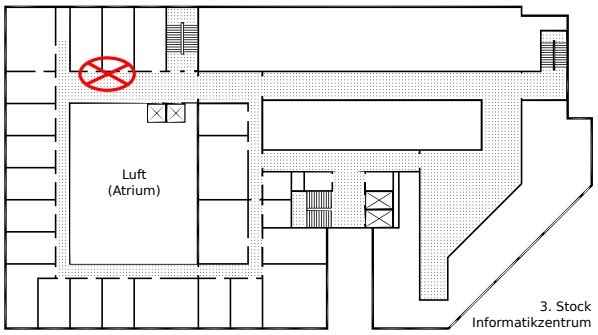
Abteilung Algorithmik **Sommer 2018**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Arne Schmidt

Algorithmen und Datenstrukturen II
Übung 3 vom 17. 05. 2018

Abgabe der Lösungen bis zum Donnerstag, den 07. 06. 2018 um 13:15 im Hausaufgabenschrank bei Raum IZ 337. Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.
Bitte die Blätter zusammenheften und vorne deutlich mit eigenem Namen, Matrikel- und Gruppennummer, sowie Studiengang versehen!



Dieses Blatt ist in drei Bereiche unterteilt: Allgemeiner Teil (A), theoretischer Teil (T) und praktischer Teil (P). Abgegeben muss der A-Teil und entweder T oder P. Mögliche Kombinationen sind also A und T bzw. A und P mit einem Gesamtwert von jeweils 30 Punkten. Die Auswahl gilt nur für dieses Blatt, d.h. auf dem nächsten Blatt kann sich wieder umentschieden werden.

A — Allgemeiner Teil

Aufgabe 1 (DP Modellierung - Matrix Chain Multiplication): (10 Punkte)
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.

Sei $\text{OPT}(i, j)$ die kleinste Anzahl an Multiplikationen, die für die Multiplikation der Matrizen M_i bis M_j nötig sind. Entwirf eine Rekursionsgleichung, die $\text{OPT}(i, j)$ berechnet.

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.

(Hinweis: Eine Multiplikation von einer $d_a \times d_b$-Matrix mit einer $d_b \times d_c$-Matrix benötigt insgesamt $d_a \cdot d_b \cdot d_c$ Operationen. Das Ergebnis der Multiplikation liefert eine $d_a \times d_c$-Matrix.)

Aufgabe 2 (DP für KNAPSACK): **(10 Punkte)**
Wir betrachten in dieser Aufgabe den Dynamic-Programming-Algorithmus für KNAPSACK aus der Vorlesung (Algorithmus 1.15).

Wende den Algorithmus auf folgende Instanz an:

i	1	2	3	4	5	6	7	8	9
z_i	5	6	8	5	6	1	3	4	4
p_i	1	4	4	2	3	1	1	4	5

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																			
9																			

T — Theoretischer Teil

Aufgabe 3 (KNAPSACK und der größte gemeinsame Teiler): **(7 Punkte)**
Aus der Vorlesung ist bekannt, dass das Dynamische Programm für KNAPSACK eine Laufzeit von $O(nZ)$ und einen Speicherbedarf von $O(Z)$ in der sparsamen Variante besitzt.

Wie kann der größte gemeinsame Teiler genutzt werden, um sowohl Laufzeit als auch Speicherplatz zu reduzieren? Beweise außerdem die Korrektheit deiner Methode! (Hinweis: Der größte gemeinsame Teiler von Zahlen $a_1, \ldots, a_n$ ist die größte Zahl, die $a_1, \ldots, a_n$ ganzzahlig teilt.)

Aufgabe 4 (3-PARTITION): **(3 Punkte)**
Betrachte folgendes Problem:

Gegeben: $n := 3k$ ganze Zahlen $x_1, \ldots, x_n$ mit $k \in \mathbb{N}$ und eine weitere Zahl $K \in \mathbb{N}$.

Frage: Gibt es eine Aufteilung der n Zahlen in k Dreiergruppen, sodass die Summe in jeder Dreiergruppe den Wert K besitzt?

Eine Idee diese Frage zu beantworten ist die folgende (eine formale Beschreibung ist in Algorithmus 1 zu sehen):
Suche drei Zahlen, die sich zu K aufsummieren. Falls es diese drei nicht gibt, gib *false* zurück. Andernfalls lösche diese drei Zahlen. Falls nun keine Zahlen übrig sind, gib *true* zurück. Wiederhole andernfalls die Prozedur.

Beantwortet dieser Algorithmus die Frage immer korrekt? Begründe deine Antwort!

```
function ThreePartition(x_1, ..., x_n)
    X := {x_1, ..., x_n}
    while X ≠ ∅ do
        if Existieren x_h, x_i, x_j mit x_h + x_i + x_j = K then
            X := X \ {x_h, x_i, x_j}          ▷ Entferne x_h, x_i und x_j aus X
        else
            return false
    return true
```

Algorithmus 1: Ein möglicher Algorithmus zum Lösen von 3-Partition.

P — Praktischer Teil

Aufgabe 5 (Implementierung Knapsack): (10 Punkte)
Implementiere für Knapsack den Dynamic-Programming-Algorithmus aus der Vorlesung (Algorithmus 1.15 bzw. 1.17), sodass nicht nur der Wert einer Lösung, sondern auch eine Teilmenge der Objekte zurückgegeben wird, die diesen Wert realisiert. (Hinweis: Es bietet sich an, in einem zweiten Array zu speichern, welches Objekt zuletzt hinzugefügt wurde. Das Speichern von *KnapsackSolution*-Instanzen für jedes $P(x, i)$ kann zu Speicherproblemen führen.)

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

a) Löse **alle zufällig generierten** Instanzen, d.h. diejenigen, die sich im Ordner *random* befinden (die Lösungen werden in csv-Dateien gespeichert). (Hinweis: Gegebenenfalls muss die Java-Heap Größe erhöht werden. Mit dem Parameter -Xmx4g können bis zu 4GB bereitgestellt werden; mehr sollte nicht nötig sein[2].)

b) Wie schnell wird die Lösung berechnet? Erstelle dazu folgenden *Scatter-Plot*:

- Die x-Achse entspricht der Anzahl der Objekte
- Die y-Achse entspricht dem Quotienten aus der Laufzeit für das Lösen und dem Produkt aus Objektanzahl und Kapazität der Instanz (formal: sei t die Zeit, die zum Lösen gebraucht wurde, n die Anzahl der Objekte, Z die Kapazität. Dann entspricht die y-Achse dem Wert $\frac{t}{nZ}$).

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

Zur Abgabe: Ersetze 1234567 durch Deine Matrikelnummer und gib sowohl die Javadatei, die vom Programm generierten csv-Dateien als auch den zu konstruierenden Scatter-Plot (z.B. als pdf-Datei) per Mail an Deinen entsprechenden Betreuer ab. Nenne in der Mail Name, Matrikel- und Gruppennummer. Es gilt dieselbe Frist wie für die anderen Aufgaben.

[1] `http://www.ibr.cs.tu-bs.de/courses/ss18/aud2/`

[2] Solltest du nicht genügend Speicher zur Verfügung haben, löse so viele Instanzen wie möglich. Informiere deinen Betreuer bei der Abgabe, dass aus Speicherproblemen nicht alle Instanzen gelöst werden konnten und nenne wie viel Speicherplatz du zur Verfügung hattest.