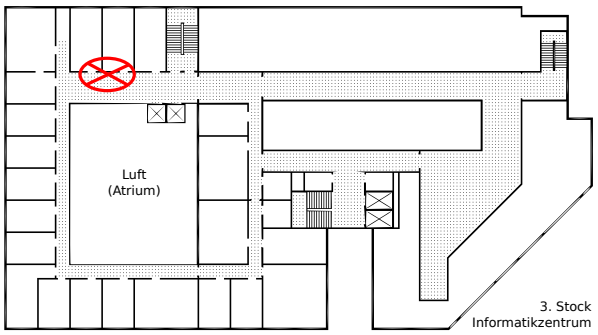
Abteilung Algorithmik **Sommer 2018**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Arne Schmidt

Algorithmen und Datenstrukturen II
Übung 2 vom 03. 05. 2018

Abgabe der Lösungen bis zum Donnerstag, den 17. 05. 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 (Subset Sum - Dynamic Programming (DP)): **(10 Punkte)**
Wende den DP-Algorithmus für Subset Sum auf folgende Instanz an:

Objekt	i	1	2	3	4	5
Gewicht	z_i	6	4	3	5	3

und $Z = 19$

Fülle hierzu folgende Tabelle aus (der Eintrag in der Zeile i und der Spalte x entspricht dem Wert $\mathscr{S}(x, i)$):

$i \backslash x$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0																				
1																				
2																				
3																				
4																				
5																				

Aufgabe 2 (DP Modellierung I - Ein Zwei-Spieler-Spiel): (5+2+3 Punkte)
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 möchte, wählt dieser die linke Münze. Es bleibt über:

$$1, 10, 50, 2$$

Die pfiffige Alice merkt, dass sie definitiv die 50 einstecken wird, wenn sie die 1 für sich beansprucht. 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, welches entscheidet, ob und wie sie gewinnen kann.

a) Sei $\textsc{opt}(i, j)$ der optimale Wert, den Alice erreichen kann, wenn die Münzen auf Positionen i bis j benutzt werden. Konstruiere eine Rekursionsgleichung, die $\textsc{opt}(i, j)$ berechnet. (Hinweis: Betrachte zusätzlich zu $\textsc{opt}(i, j)$ den Münzwert $C_{i,j}$ der Münzen von i bis j. Von diesem Wert wird Bob so wenig wie möglich überlassen!)

b) Angenommen, Alice beginnt mit dem Ziehen. Welchen Wert muss $\textsc{opt}(1, n)$ überschreiten, damit Alice weiß, dass sie gewinnt? Was passiert, wenn Bob mit dem Ziehen beginnt? Welche Bedingung muss nun gelten, damit Alice weiß, dass sie gewinnt?

c) Betrachte alle möglichen $\textsc{opt}(i, j)$ mit $1 \leq i \leq j \leq n$. Bestimme wie viele verschiedene $\textsc{opt}(i, j)$ existieren!

T — Theoretischer Teil

Aufgabe 3 (Subset Sum - Mehr Dynamic Programming): (1+5+4 Punkte)
Betrachten wir noch einmal das Problem Subset Sum und das Dynamische Programm dazu.

a) Wie kann man mit Hilfe des Dynamischen Programms für Subset Sum erkennen, ob es eine Lösung für Partition gibt?

b) Wandle das Dynamische Programm (Algorithmus 1.11) so ab, dass in jedem $\mathscr{S}(x, i)$ die Menge ausgelesen werden kann, die Subset Sum mit den ersten i Objekten und Zielwert x erfüllt. (Hinweis: Eine nicht-existente Menge beschreiben wir mit nil, leere Mengen mit $\emptyset$ oder {}.)

c) Wandle das Dynamische Programm (Algorithmus 1.11) so ab, dass Objekte beliebig oft genutzt werden dürfen.

P — Praktischer Teil

Aufgabe 4 (Implementierung Knapsack): **(10 Punkte)**
Implementiere für KNAPSACK einen Brute-Force-Algorithmus, d.h. einen Algorithmus, der jede Belegung der Objekte in dem Rucksack testet. (Hinweis: Die Funktion *Combinations* aus dem commons-math Paket[1] könnte sehr hilfreich sein! Der Algorithmus sollte nicht mehr als 30 Zeilen Code besitzen.)

Nutze dazu die Javavorlage und die Testfälle, die auf der Vorlesungsseite[2] zur Verfügung stehen.

a) Starte dein Programm mit allen Instanzen mit bis zu 30 Objekten.
(Hinweis: Diese Instanzen sollten nur einige wenige Minuten dauern!)

b) Der Brute-Force-Algorithmus hat eine Mindestlaufzeit von $\Omega(2^n)$, da alle Möglichkeiten durchprobiert werden. Um welchen Faktor steigt die Anzahl der Möglichkeiten, wenn man 10 Objekte dazu nimmt (unabhängig davon wie viele wir vorher hatten)?

c) Wie lange wird das Lösen einer Instanz mit 40 oder 50 Objekten etwa dauern? Nutze zur Beantwortung dieser Frage die Ergebnisse aus a) und b).

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 als auch die vom Programm generierten csv-Dateien per Mail an Deinen entsprechenden Betreuer ab. Nenne in der Mail Name, Matrikel- und Gruppennummer. Es gilt dieselbe Frist wie für die anderen Aufgaben. Aufgabenteile b) und c) sollen wie Teil A schriftlich eingereicht werden.

[1] `http://commons.apache.org/proper/commons-math/javadocs/api-3.4/org/apache/commons/math3/util/Combinations.html`

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