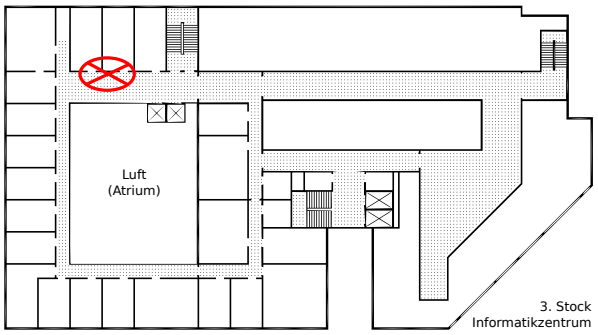
Abteilung Algorithmik **Sommer 2017**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Arne Schmidt

Algorithmen und Datenstrukturen II
Übung 2 vom 01. 05. 2017

Abgabe der Lösungen bis zum Montag, den 15. 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 45 Punkte, erreichbar (und abzugeben) sind aber lediglich Aufgaben im Gesamtwert von maximal 30 Punkten. Mehr wird nicht gewertet! Wähle zwei Aufgaben aus, die Du bearbeitest.

Aufgabe 1 (PARTITION - Dynamic Programming (DP)): In dieser Aufgabe wollen wir PARTITION mithilfe des DP-Algorithmus aus der Vorlesung (Algorithmus 1.11) lösen.

a) Für welches x und i kann man aus $\mathscr{S}(x,i)$ lesen, ob eine Lösung für PARTITION existiert?

b) Entscheide mit dem DP-Algorithmus, ob folgende PARTITION-Instanz lösbar ist:

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

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
0																			
1																			
2																			
3																			
4																			
5																			

c) Bisher entscheidet der DP-Algorithmus nur, ob eine Lösung existiert, aber nicht welche. Passe den Algorithmus an, sodass man eine Menge von Objekten erhält, die PARTITION bzw. SUBSET SUM löst!

(1+9+5 Punkte)

Aufgabe 2 (PARTITION - Greedy): Wir betrachten in dieser Aufgabe die Maximierungsvariante von PARTITION und einen möglichen Algorithmus, der dieses Problem lösen soll. In dieser Variante bekommen wir als Input eine Menge von Zahlen $z_1, \dots z_n$ und suchen eine Menge S, sodass $\sum_{i \in S} z_i \leq \sum_{i \notin S} z_i$ gilt und $\sum_{i \in S} z_i$ maximiert wird (vgl. Problem 1.7). Betrachte den unten angegebenen Algorithmus 1.

a) Wende Algorithmus 1 auf die folgende Instanz an:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
z_i	20	8	1	5	3	15	42	1	12	7	17	2	11	4

Gib dazu die Permutation $\pi(i)$ und nach jeder Iteration der **for**-Schleife das jeweils betrachtete Objekt, die Mengen A und B und den Wert der jeweiligen Mengen an. Welchen Wert besitzt die zurückgegebene Menge? Gibt es eine bessere Lösung?

b) Angenommen, alle z_i sind kleiner als $\frac{1}{2}\sum_{i=1}^{n} z_i$. Sei S die Menge, die von Algorithmus 1 zurückgegeben wird. Zeige: Es gibt eine Instanz, sodass $\frac{\sum_{i \in S} z_i}{\sum_{i \notin S} z_i} \leq \frac{1}{2}$, d.h. der Wert von S ist höchstens halb so groß wie der Wert der Elemente, die nicht in S liegen. Gibt es einen Algorithmus, der eine Lösung S mit $\frac{\sum_{i \in S} z_i}{\sum_{i \notin S} z_i} > \frac{1}{2}$ garantieren kann?

c) Betrachten wir einen Spezialfall, bei denen die Objekte $i < n$ die Gewichte $z_i = 2^{i-1}$ bekommen und das letzte, n-te Objekte das Gewicht $z_n = 1$. Zeige oder widerlege: Algorithmus 1 gibt für beliebiges n immer eine optimale Lösung zurück.

(*Hinweis:* $\sum_{i=0}^{n-1} 2^i = 2^n - 1$)

(9+4+2 Punkte)

```
1:  function MaxPartition(z_1, ..., z_n, Z)
2:      Sortiere z_i's absteigend, wodurch wir die Permutation π(i) erhalten.
3:      Erstelle Mengen A und B.
4:      for j = 1 to n do
5:          if Σ_{i∈A} z_i ≤ Σ_{i∈B} z_i then      ▷ Wert von A höchstens Wert von B?
6:              A ← A ∪ {π(j)}                      ▷ Ja: Füge π(j) zu A hinzu
7:          else
8:              B ← B ∪ {π(j)}                      ▷ Nein: Füge π(j) zu B hinzu
9:      if Σ_{i∈A} z_i ≤ Σ_{i∈B} z_i then
10:         return A
11:     else
12:         return B
```

Algorithmus 1: Ein Algorithmus zum Lösen von MAXIMUM PARTITION

Aufgabe 3 (Implementierung Knapsack): 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 package könnte sehr hilfreich sein! [1])

Nutze dazu die Javavorlage und die Testfälle, die auf der Vorlesungsseite[2] 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 asymptotische Laufzeit besitzt Dein Brute-Force-Algorithmus? Eine Angabe ohne Begründung ist ausreichend. (*Hinweise:* Wieviele Teilmengen gibt es von einer n-elementigen Menge? Wie lange benötigt man pro Teilmenge?)

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

Zur Abgabe: Ersetze 1234567 durch Deine Matrikelnummer und gib die (kommentierte) 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.

(10+3+2 Punkte)

[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/ss17/aud2/