**Technische Universität Braunschweig**
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Sommersemester 2018

Prof. Dr. Sándor P. Fekete
Arne Schmidt

# Klausur
# *Algorithmen und Datenstrukturen II*
# 03.08.2018

Name: ..................................

Vorname: ...............................

Matr.-Nr.: ..............................

Studiengang: ...........................

☐ Bachelor ☐ Master ☐ Andere

***Klausurcode:***
*Dieser wird benötigt, um das Ergebnis der Klausur abzurufen.*

**Hinweise:**

- Bitte das Deckblatt in Druckschrift vollständig ausfüllen.
- Die Klausur besteht aus 11 Blättern, bitte auf Vollständigkeit überprüfen.
- Erlaubte Hilfsmittel: Keine.
- Eigenes Papier ist nicht erlaubt.
- Die Rückseiten der Blätter dürfen beschrieben werden.
- Die Klausur ist mit 50 % der Punkte bestanden.
- Antworten die *nicht* gewertet werden sollen bitte deutlich durchstreichen. Kein Tippex verwenden.
- Mit *Bleistift* oder in *Rot* geschriebene Klausurteile können nicht gewertet werden.
- Werden mehrere Antworten gegeben, werten wir die mit der geringsten Punktzahl.
- Sämtliche Algorithmen, Datenstrukturen, Sätze und Begriffe beziehen sich, sofern nicht explizit anders angegeben, auf die in der Vorlesung vorgestellte Variante.
- Die Bearbeitungszeit für die Klausur beträgt 120 Minuten.

| **Aufgabe** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | $\Sigma$ |
|---|---|---|---|---|---|---|---|---|---|---|
| **Punkte** | **13** | **12** | **9** | **11** | **11** | **10** | **8** | **12** | **14** | **100** |
| **Erreicht** | | | | | | | | | | |
| **Note** | — | — | — | — | — | — | — | — | — | |

**Aufgabe 1: Greedy** **(4+9 Punkte)**

Betrachte folgende Instanz:

| Objekt | $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| Gewicht | $z_i$ | 6 | 14 | 5 | 7 | 2 | 7 | 4 | mit $Z = 23$ |
| Wert | $p_i$ | 5 | 14 | 3 | 4 | 1 | 8 | 3 | |

a) Betrachte die Instanz als eine Instanz von FRACTIONAL KNAPSACK und wende den fraktionalen Greedy-Algorithmus an. Gib in jeder Iteration den aktuellen Gegenstand, zu welchen Teilen er gepackt wird, sowie den Gesamtzustand (Was ist gepackt und wie hoch ist der aktuelle Gesamtwert und -gewicht?) an.

b) Wende $\textsc{Greedy}_0$ auf die Instanz an. Gib in jeder Iteration den aktuellen Gegenstand an, sowie die Menge der bereits gepackten Gegenstände mitsamt ihrem Gesamtgewicht und -wert. Ist die erhaltene Lösung optimal? Begründe Deine Antwort!

**Aufgabe 2: Dynamic Programming** **(6+4+2 Punkte)**

a) Betrachte folgende Instanz:

| Objekt | $i$ | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|
| Gewicht | $z_i$ | 6 | 3 | 4 | 8 | 3 | mit $Z = 14$ |
| Wert | $p_i$ | 5 | 4 | 3 | 9 | 4 | |

Wende den Dynamic-Programming-Algorithmus für Maximum Knapsack auf diese Instanz an, indem Du die folgende Tabelle ausfüllst und die Objekte in der Reihenfolge ihrer Indizes bearbeitest. (*Hinweis:* 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | |

b) Wie lautet die Rekursionsgleichung für Maximum Knapsack?

c) Wandle die Rekursionsgleichung für das Maximum Knapsack so ab, dass Objekte beliebig oft benutzt werden dürfen; es sollen also Instanzen von Integer Knapsack gelöst werden.

**Aufgabe 3: Branch and Bound** **(9 Punkte)**

Betrachte den Branch-and-Bound-Algorithmus für KNAPSACK aus der Vorlesung mit dem Berechnen von $\text{GREEDY}_0$ als untere Schranke und GREEDY für FRACTIONAL KNAPSACK als obere Schranke. Wende den Algorithmus auf folgende Instanz an. (Hinweis: Die Objekte sind für die Greedy-Algorithmen bereits vorsortiert.)

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $z_i = p_i$ | 10 | 5 | 4 | 8 |

und $Z = 18$

Beachte folgende Punkte:

- Benutze den Enumerationsbaum aus Abbildung 1.
- Beschrifte sowohl die Kanten mit der Auswahl, die getroffen wurde, als auch die Knoten mit den besten Schranken (obere und untere), die aktuell gelten.
- Sollte eine aktuelle Auswahl unzulässig sein, beschrifte den zugehörigen Knoten mit *unzulässig.*
- Sollten Kanten nicht benutzt werden, streiche sie durch.
- Halte in einer Tabelle fest, welche Belegung eine neue, aktuell beste Lösung liefert.

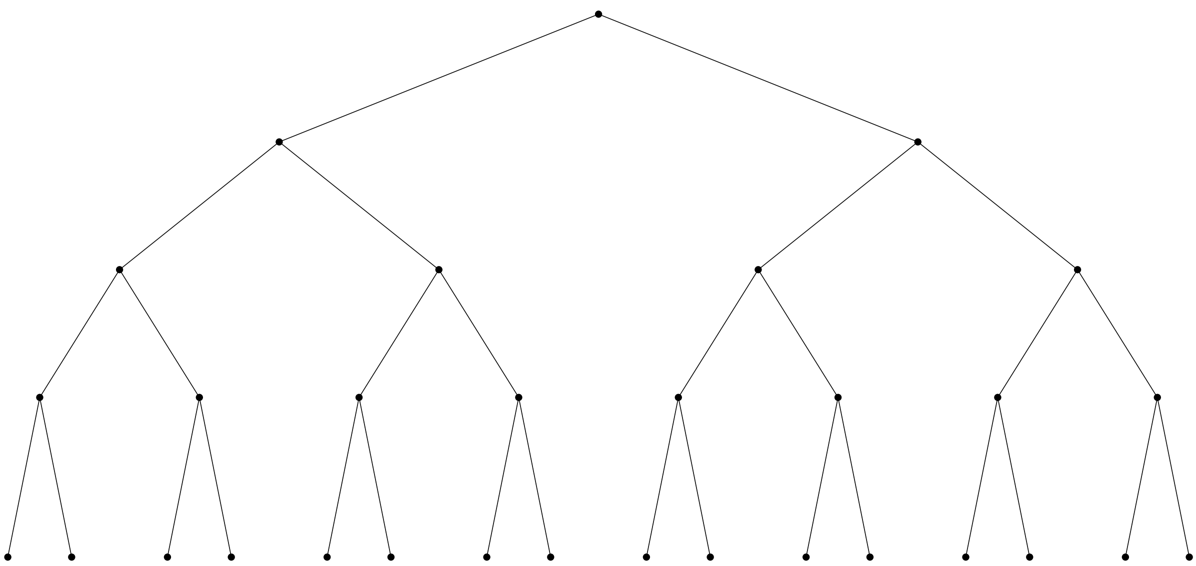

**Abbildung 1:** Der Enumerationsbaum

**Aufgabe 4: Modellierung** **(3+3+5 Punkte)**

Betrachte eine Menge $\mathcal{S}$ von $n$ Intervallen $I_1, \ldots, I_n$. Jedes Intervall $I_i$ besitzt einen Start $s_i$, ein Ende $e_i$ und einen Wert $w_i$. Es darf angenommen werden, dass die Intervalle bezüglich ihres Endes sortiert sind.

Wir suchen nun eine Menge $S \subseteq \mathcal{S}$ von Intervallen, die sich paarweise nicht überlappen und deren Gesamtwert maximal ist. Sei $\textsc{opt}(i)$ der größte Wert, den wir mit den ersten $i$ Intervallen erreichen können.

Sei pred$(i)$ die Funktion, die uns den Index des Intervalls zurückgibt, welches am spätesten endet aber noch vor Intervall $I_i$ liegt. Gibt es kein solches Intervall, so gibt pred$(i)$ den Wert 0 zurück. Formal: $\text{pred}(i) = \max(\{j \mid e_j \leq s_i\} \cup \{0\})$

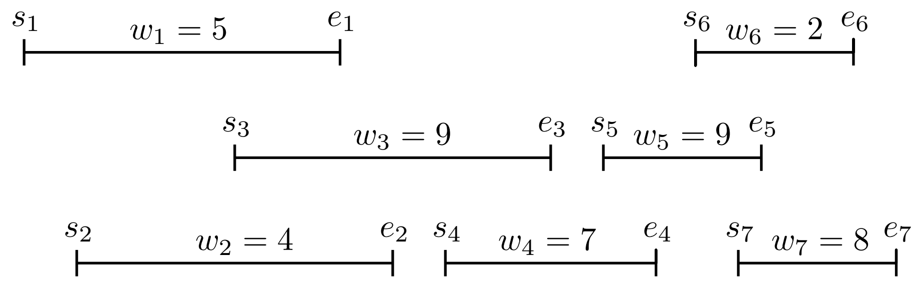


**Abbildung 2:** Eine Instanz mit sieben Intervallen $I_1, \ldots, I_7$

a) Gib für alle $1 \leq i \leq 7$ die Werte pred$(i)$ für die Intervalle aus Abbildung 2 an.

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| pred$(i)$ | | | | | | | |

b) Gib für alle $0 \leq i \leq 7$ die Werte $\textsc{opt}(i)$ für die Intervalle aus Abbildung 2 an.

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\textsc{opt}(i)$ | | | | | | | | |

c) Gib eine Rekursionsgleichung an, mit der man $\textsc{opt}(i)$ berechnen kann.

**Aufgabe 5: Approximation I** **(11 Punkte)**

Wende $\textsc{Greedy}_k$ auf die folgende Instanz an. (Hinweis: Die Objekte sind für $\textsc{Greedy}_0$ bereits vorsortiert.)

$$\begin{array}{r|cccc} i & 1 & 2 & 3 & 4 \\ \hline p_i = z_i & 6 & 7 & 2 & 3 \end{array} \text{ mit } Z = 10 \text{ und } k = 2.$$

Gib dazu die folgenden Mengen bzw. Werte tabellarisch an:

- $\overline{S}$
- $\sum\limits_{i \in \overline{S}} z_i$
- $Z - \sum\limits_{i \in \overline{S}} z_i$
- $A_{\overline{S}} := \textsc{Greedy}_0(\{z_i | i \notin \overline{S}\}, Z - \sum\limits_{i \in \overline{S}} z_i, \{p_i | i \notin \overline{S}\})$
- $\sum\limits_{i \in \overline{S}} p_i + A_{\overline{S}}$
- $G_k$
- $S$

Achte darauf, dass $\overline{S}$ mit der kleinsten Menge anfängt und mit der größten endet. Zusätzlich soll $\overline{S}$ lexikographisch sortiert sein, das heißt, für zwei gleichgroße Mengen $\overline{S}_1$ und $\overline{S}_2$ kommt $\overline{S}_1$ vor $\overline{S}_2$, falls das kleinste Element $x \in \overline{S}_1 \setminus \overline{S}_2$ kleiner ist als das kleinste Element $y \in \overline{S}_2 \setminus \overline{S}_1$.

**Aufgabe 6: Approximation II** **(6+4 Punkte)**

Betrachte den Algorithmus $\textsc{Greedy}_0$. Angenommen, jedes Objekt hat ein Gewicht von höchstens $\frac{Z}{\alpha}$ für ein festes $\alpha \in \{1, 2, 3, 4, \dots\}$. Wir nehmen außerdem an, dass $z_i = p_i$ für jedes Objekt $i$ gilt und dass die Summe aller $z_i$ die Kapazität $Z$ überschreitet.

a) Zeige: Nach Anwenden von $\textsc{Greedy}_0$ gilt $\sum_{i=0}^{n} x_i z_i \geq Z(1 - \frac{1}{\alpha})$.

b) Welche Approximationsgüte besitzt $\textsc{Greedy}_0$ mit diesen speziellen Objekten?

**Aufgabe 7: Komplexität** **(8 Punkte)**

Angenommen, wir haben einen Algorithmus $\mathcal{A}$, der Partition löst. Wie können wir nun $\mathcal{A}$ benutzen, um auch Subset Sum zu lösen? Begründe außerdem die Korrektheit deines Verfahrens! (Hinweis: Es darf angenommen werden, dass $Z \geq \frac{1}{2} \sum_{i=1}^{n} z_i$)

## Aufgabe 8: Hashing **(7+5 Punkte)**

a) Betrachte ein anfangs leeres Array $A$ der Größe 10, es gibt also die Speicherzellen $A[0], \ldots, A[9]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + 3ix \mod 10$$

Dabei ist $x$ ein einzusetzender Schlüssel und $i$ die Nummer des Versuches, $x$ in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in $A$ bekommt:

$$4, \quad 12, \quad 2, \quad 14$$

Dabei sollen die Schlüssel in der gegebenen Reihenfolge eingefügt werden und der Rechenweg soll klar erkennbar sein. Trage die Elemente in das Array in Tabelle 1 ein.

| **A[0]** | **A[1]** | **A[2]** | **A[3]** | **A[4]** | **A[5]** | **A[6]** | **A[7]** | **A[8]** | **A[9]** |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

**Tabelle 1:** Das Array $A$.

b) Gib eine Instanz mit höchstens 10 Schlüsseln an, sodass die in Teil a) gegebene Hashfunktion nicht jedem Schlüssel einen Platz in der Hashtabelle zuweisen kann. Zeige, dass deine Instanz diese Eigenschaft erfüllt.

**Aufgabe 9: Kurzfragen** **(2+2+2+2+6 Punkte)**

a) Was bedeutet es für ein Problem, wenn es zur Klasse $P$ gehört? Nenne ein Problem aus der Vorlesung, das in $P$ liegt!

b) Wie ist die Klasse $NP$ definiert? Begründe kurz, warum 0-1-Knapsack $\in NP$ gilt!

c) Wann ist ein Problem $NP$-schwer?

d) Wie zeigt man, dass ein Problem $NP$-vollständig ist?

e) Gib drei prinzipiell verschiedene Vorgehensweisen für ein NP-vollständiges Problem an, benenne jeweils die Vor- und Nachteile jeder Alternative und führe jeweils ein Beispiel für solch einen Algorithmus an. Fülle dazu die gegebene Tabelle 2 aus. (Hinweis: Es reicht, die Algorithmen zu benennen.)

| | | | |
|---|---|---|---|
| **Vorgehensweise** | | | |
| **Vorteil** | | | |
| **Nachteil** | | | |
| **Algorithmus** | | | |

**Tabelle 2:** Eine nicht vollständige Tabelle.

**Viel Erfolg** ☺