Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Prof. Dr. Sándor P. Fekete
Dr. Christian Scheffer

Sommersemester 2016

Klausur
Algorithmen und Datenstrukturen II
01. Agust 2016

Name: .....

Vorname: .....

Matr.-Nr.: .....

Studiengang: .....

Bachelor ☐ Master ☐ Diplom ☐ Andere ☐

Mit der Veröffentlichung des Klausurergebnisses nur mit der Matrikelnummer über die Mailingliste und auf der Homepage bin ich einverstanden.

..... *Unterschrift*

Hinweise:

- Bitte das Deckblatt vollständig ausfüllen.
- Die Klausur besteht aus 12 Blättern, bitte auf Vollständigkeit überprüfen.
- Erlaubte Hilfsmittel: Keine.
- Eigenes Papier ist nicht erlaubt.
- Die Rückseiten der Blätter dürfen beschrieben werden.
- Die Heftung der Blätter darf nicht entfernt werden.
- Die Klausur ist mit 50% der Punkte bestanden.
- Antworten die *nicht* gewertet werden sollen bitte deutlich durchstreichen.
- Mit *Bleistift* oder in *Rot* geschriebene Klausurteile können nicht gewertet werden.
- Werden mehrere Antworten gegeben, werten wir die mit der geringsten Punktzahl.
- Die Bearbeitungszeit für die Klausur ist 120 Minuten.

Aufgabe	**1**	**2**	**3**	**4**	**5**	Σ
Punkte	**30**	**20**	**15**	**23**	**12**	**100**
Erzielte Punkte						

Aufgabe 1: Knapsack **(5+5+11+9 Punkte)**

Gegeben ist folgende BINARY-KNAPSACK-Instanz mit 6 Gegenständen:

i	1	2	3	4	5	6
Wert p_i	2	3	1	3	5	2
Gewicht z_i	5	4	6	10	2	3

Dabei darf der Rucksack mit einem Gesamtgewicht von maximal $Z = 20$ bepackt werden.

a) Ermittle mithilfe des Greedy-Algorithmus (GREEDY_0) aus der Vorlesung eine zulässige Lösung für die gegebene Instanz. Gib in jeder Iteration den aktuellen Gegenstand an, sowie die Menge der bereits gepackten Gegenstände mitsamt ihrem Gesamtgewicht und -wert.

b) Interpretiere die oben gegebene Instanz als Instanz des FRACTIONAL-KNAPSACK-Problems und berechne mithilfe des Greedy-Algorithmus aus der Vorlesung eine zulässige Lösung. Gib dabei 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. Ist diese Lösung optimal? Begründe deine Aussage.

c) Wende den Dynamic-Programming-Algorithmus aus der Vorlesung auf die gegebene Instanz an.

   Gib die dafür nötige Tabelle komplett mit allen Einträgen an und bearbeite die Gegenstände in der Reihenfolge ihrer Indizes.

d) Betrachte jetzt Branch-&-Bound.

(i) Welche untere Schranke lässt sich aus a) ableiten? Für welche Teilbäume gilt sie?

(ii) Welche obere Schranke lässt sich aus b) ableiten? Für welche Teilbäume gilt sie?

(iii) Was lässt sich aus der Kombination von (i) und (ii) folgern? Was bedeutet dies für den Algorithmus Branch-&-Bound, der diese beiden Schranken verwendet? Begründe Deine Antwort.

Aufgabe 2: Approximationsalgorithmen **(15+5 Punkte)**

a) Führe Algorithmus 1.26 ($\textsc{Greedy}_k$) mit $k = 2$ für die Eingabe $n = 5$, $z_1 = 1$, $z_2 = 1$, $z_3 = 1$, $z_4 = 1$, $z_5 = 2$, $p_1 = 5$, $p_2 = 1$, $p_3 = 3$, $p_4 = 2$, $p_5 = 3$ und $Z = 3$ aus. Gib für jede Iteration die folgenden Mengen bzw. Werte an:

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

b) Inwiefern ähnelt Algorithmus $\textsc{Greedy}_k$ für $k := n$ dem aus der Vorlesung bekannten Verfahren Branch-&-Bound? Inwiefern unterscheiden sich beide Verfahren? Begründe Deine Antwort.

Aufgabe 3: Hashing **(15 Punkte)**

Wir betrachten ein anfangs leeres Array A der Größe 7, es gibt also die Speicherzellen $A[0], A[1], \ldots, A[6]$. In diesem führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = (2 \cdot x \cdot i + x) \mod 7$$

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:

$$2, \quad 9, \quad 6, \quad 1, \quad 8, \quad 3, \quad 5$$

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

$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$

Abbildung 1: Die Hashtabelle.

Aufgabe 4: Dynamische Programmierung (3+3+3+3+3+3+5 Punkte)
Betrachte folgendes Problem: Gegeben ist eine Menge $\mathcal{M} := \{[b_i, e_i] | 1 \leq i \leq n\}$ von n Intervallen. Für $i \in \{1, ..., n\}$ sei $w_i > 0$ das Gewicht des Intervalles $[b_i, e_i]$. Gesucht ist eine Teilmenge $\mathcal{M}' \subseteq \mathcal{M}$, so dass die Intervalle aus $\mathcal{M}'$ sich nicht schneiden und so dass die Summe der Gewichte der Intervalle aus $\mathcal{M}'$ maximal ist.

Zur Vereinfachung nehmen wir an, dass die Intervalle aufsteigend bzgl. e_i nummeriert sind, d.h. es gilt $e_1 \leq e_2 \leq ... \leq e_n$. Weiterhin sei $pred(i)$ der Index des Intervalls, das am spätesten aber noch vor $[b_i, e_i]$ endet. Ein Beispiel ist in Abbildung 2 zu sehen.

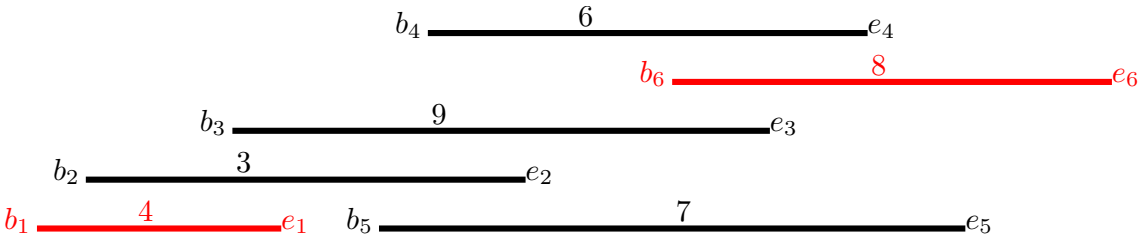

Abbildung 2: Beispielkonfiguration von 6 Intervallen $I_1 = [b_1 e_1], ... I_6 = [b_6, e_6]$ für Aufgabe 4. Die Zahlen über den Intervallen entsprechen deren Gewichten. Die Menge der rot markierten Intervalle bildet eine optimale Lösung mit dem Wert $4 + 8 = 12$. Die Intervalle 5 und 6 haben in der Summe ein höheres Gewicht, bilden aber keine korrekt Lösung, da die Intervalle sich überlappen (es gilt $b_6 < e_5$). In diesem Beispiel gilt $pred(1) = 0$, $pred(2) = 0$, $pred(3) = 0$, $pred(4) = 1$, $pred(5) = 1$ und $pred(6) = 2$.

Betrachte für das obige Problem das folgende dynamische Programm:

$$OPT(i) = \left\{ \begin{array}{cc} \max\{OPT(i-1), w_i + OPT(pred(i))\} & falls\ i > 0 \\ 0 & falls\ i = 0 \end{array} \right\}.$$

a) Gib die Werte $pred(1), ..., pred(6)$ für folgendes Beispiel an: $[b_1 = 1, 3 = e_1]$, $[b_2 = 2, 6 = e_2]$, $[b_3 = 3, 7 = e_3]$, $[b_4 = 5, 9 = e_4]$, $[b_5 = 8, 11 = e_5]$ und $[b_6 = 10, 12 = e_6]$.

b) Gib eine mathematische Formel zur Definition von $pred : \{1, ..., n\} \to \{0, ..., n-1\}$ an.

c) Gib einen Algorithmus an, der *pred* berechnet.

d) Gib die Werte $OPT(0), ..., \mathrm{OPT}(6)$ für das Beispiel aus Aufgabenteil a) mit den Gewichten $w_1 = 2$, $w_2 = 3$, $w_3 = 2$, $w_4 = 4$, $w_5 = 3$ und $w_6 = 3$ an.

e) Beschreibe in eigenen Worten, welche Bedeutung $OPT(i)$ für $i \in \{0, ..., n\}$ hat.

f) Gib einen Algorithmus an, der unter Verwendung der bereits berechneten Werte $pred[0], ..., pred[n]$ das obige dynamische Programm realisiert, also alle Werte $OPT(0), ..., OPT(n)$ berechnet.

g) Beweise mittels vollständiger Induktion über n, dass $OPT(n)$ der Wert einer optimalen Lösung ist.

Aufgabe 5: Kurzfragen **(2+2+2+2+2+2 Punkte)**

a) Lösungen für Fractional Knapsack sind obere Schranken für Integer Knapsack. ☐ wahr ☐ falsch

b) Der Dynamische-Programmierungs-Algorithmus für Knapsack verwendet Rekursion. ☐ wahr ☐ falsch

c) Die Laufzeit des Dynamischen-Programmierungs-Algorithmus für Knapsack hängt ausschlißlich von der Anzahl der zu betrachtenden Items ab. ☐ wahr ☐ falsch

d) Die Familie $\{\textsc{Greedy}_k \mid k \in \mathbb{N}\}$ ist ein PTAS für Knapsack. ☐ wahr ☐ falsch

e) Die Komplexitätsklasse P ist eine Menge von Algorithmen. ☐ wahr ☐ falsch

f) Wenn Knapsack polynomiell lösbar ist, dann ist auch 3SAT polynomiell lösbar. ☐ wahr ☐ falsch

Viel Erfolg ☺