Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Prof. Dr. Sándor Fekete
Ramin Kosfeld
Chek-Manh Loi

Sommersemester 2026

Klausur
Algorithmen und Datenstrukturen II
07.08.2026

Nachname: .....

Vorname: .....

Matr.-Nr.: .....

Studiengang: .....

☐ Bachelor ☐ Master ☐ Andere: .....

Hinweise:

- Bitte das Deckblatt in Druckschrift vollständig ausfüllen.
- Die Klausur besteht aus 13 Blättern, bitte auf Vollständigkeit überprüfen.
- Die Bearbeitungszeit für die Klausur beträgt 120 Minuten.

- 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.
- Sofern nicht anders angegeben, sind alle Graphen als einfache Graphen zu verstehen.

Aufgabe	1	2	3	4	5	6	7	8	Σ
Max	13	16	10	10	14	17	10	10	100
Erreicht									

Aufgabe 1: Dynamic Programming - Subset Sum **(7+2+4 Punkte)**

a) Wende das dynamische Programm für SUBSET SUM auf folgende Instanz an.

Objekt i	1	2	3	4	5	6
Gewicht z_i	4	7	6	9	3	2

mit $Z = 15$

Fülle hierzu die folgende Tabelle aus, wobei der Eintrag in Zeile i und Spalte x dem Wert $\mathcal{S}(x, i)$ entspricht. Nullen müssen nicht eingetragen werden.

i \ x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
4																
5																
6																

b) Ist die SUBSET SUM-Instanz aus a) lösbar? Begründe deine Antwort!

c) Betrachte das Problem PLUS ONE SUBSET SUM, bei dem für jedes Objekt entweder das Gewicht des Objekts selber genutzt werden kann, oder das Gewicht plus 1. Wie lautet die Rekursionsgleichung für dieses Problem? Sei dazu $\mathcal{S}'(x, i) = 1$, wenn x mit den ersten i Objekten erzeugt werden kann und 0 andernfalls.

Aufgabe 2: Branch-And-Bound **(8+4+4 Punkte)**

a) Wende den Branch-and-Bound-Algorithmus für MAXIMUM KNAPSACK aus der Vorlesung auf folgende sortierte Instanz an.

i	1	2	3
z_i	1	2	8
p_i	2	3	9

und $Z = 9$.

- Benutze den Entscheidungsbaum aus Abbildung 1.
- Der Branch-and-Bound-Algorithmus aus der Vorlesung trifft Entscheidungen auf den Variablen $b_1, b_2, \ldots, b_k$ in genau dieser Reihenfolge. Für jedes b_i wird zuerst $b_i = 0$ im linken Teilbaum betrachtet, und anschließend $b_i = 1$ im rechten.
- Beschrifte die Kanten mit der Auswahl, die getroffen wurde.
- Beschrifte die Knoten mit der aktuell besten lokalen oberen Schranke U und der global bis hierhin besten unteren Schranke P.
- Beschrifte einen Knoten mit *unzulässig*, falls die aktuelle Auswahl unzulässig ist.
- Sollten Kanten im Baum nicht benutzt werden, streiche sie durch.
- Nutze die Menge-Wert-Tabelle, um eine neue beste Lösung festzuhalten.
- Die einzelnen Schritte der Algorithmen, mit denen die Schranken bestimmt werden, brauchst du *nicht* anzugeben.

Menge	Wert

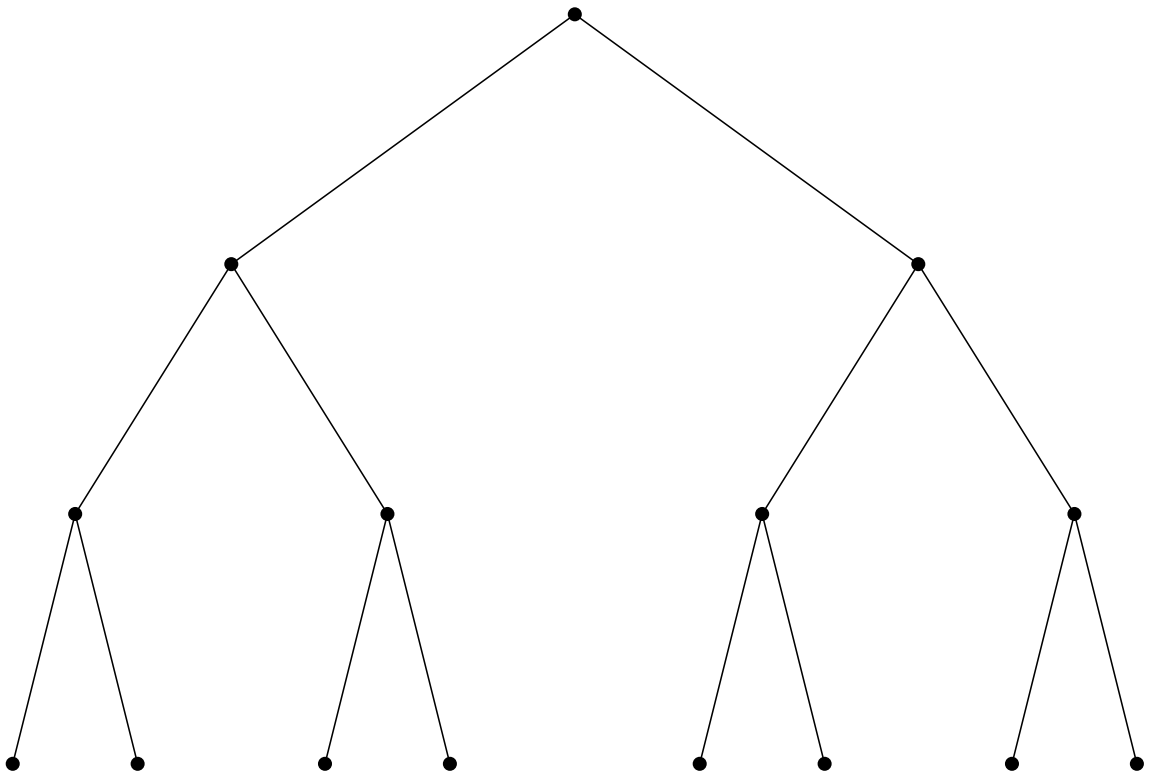

Abbildung 1: Ein Entscheidungsbaum.

b) Gib eine Knapsack-Instanz mit vier Elementen und $Z = 10$ an, für die unser Branch-and-Bound-Algorithmus keinen einzigen rekursiven Aufruf startet, also im Wurzelknoten terminiert. Begründe kurz, warum sich deine Instanz so verhält.

c) Angenommen, du implementierst einen Branch-and-Bound-Algorithmus für ein eigenes Problem und bist mit der Performance des Algorithmus unzufrieden. Nenne vier Stellen im Algorithmus, an denen du die Anzahl der besuchten Knoten im Entscheidungsbaum beeinflussen kannst.

Aufgabe 3: Gewichtete Hörsaal-Belegung **(3+3+4 Punkte)**

Wir betrachten die gewichtete Variante des Problems HÖRSAAL-BELEGUNG. Hier ist eine Menge von n Intervallen $C = \{[s_1, e_1), \ldots, [s_n, e_n)\}$ gegeben, denen jeweils ein Gewicht $w_1, \ldots, w_n \in \mathbb{R}^+$ zugeordnet ist. Das Ziel ist es, eine Teilmenge von Intervallen aus C so auszuwählen, dass alle Intervalle disjunkt sind, und das Gesamtgewicht maximiert ist.

a) Markiere in der abgebildeten Instanz eine optimale Lösung, indem du alle Intervalle der Lösung einkreist. Welches Gewicht hat deine Lösung?

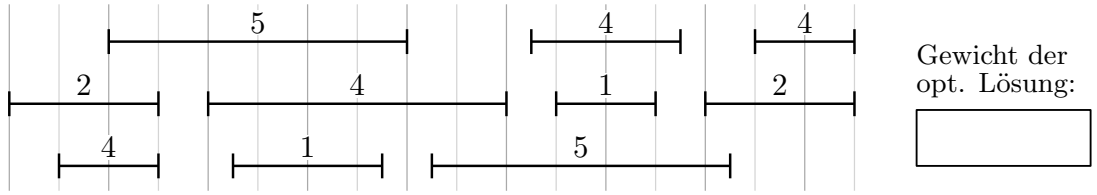

Abbildung 2: Eine Instanz des gewichteten Hörsaal-Belegungsproblems. Die Intervalle sind mit ihrem jeweiligen Gewicht beschriftet.

b) Eine Möglichkeit, das Problem zu lösen, ist ein Greedy-Algorithmus, der immer das Intervall mit dem höchsten Wert wählt. Gib eine Instanz bestehend aus vier Intervallen an, die zeigt, dass solch ein Greedy-Algorithmus nicht optimal ist.

c) Sei $\mathcal{A}$ ein Algorithmus, der die ungewichtete Variante von HÖRSAAL-BELEGUNG optimal löst. Sei $\mathcal{I}$ die Menge aller Instanzen des gewichteten Hörsaal-Belegungsproblems, bei denen alle Gewichte im Intervall $[1, 2]$ liegen. Zeige: $\mathcal{A}$ ist ein $1/2$-Approximationsalgorithmus für alle Instanzen in $\mathcal{I}$.

Aufgabe 4: Algorithmenentwurf: Dynamic Programming (3+3+4 Punkte)

Angenommen, wir werfen einen 6-seitigen Würfel n mal (oder n Würfel gleichzeitig) und addieren die gewürfelten Zahlen. Dann ist die resultierende Summe ein Wert zwischen n und $6n$. Manche Summen sind allerdings wesentlich wahrscheinlicher als andere. Das Ergebnis n erhalten wir z.B. nur, wenn jeder Wurf eine Eins ergibt; dagegen gibt es für $n > 1$ viele Möglichkeiten, das Ergebnis $\lfloor 3.5n \rfloor$ zu erwürfeln.

In dieser Aufgabe wollen wir zählen, wie viele Möglichkeiten es für n Würfel gibt, eine bestimmte Summe x zu erwürfeln. Wir wollen also einen Algorithmus entwickeln, der die Funktion $\varphi(n, x)$ berechnet, welche zu einer beliebigen natürlichen Zahl x angibt, wie viele Möglichkeiten es gibt, diese mit n Würfelwürfen zu erreichen. Es soll $\varphi(n, x) = 0$ sein, falls es nicht möglich ist, x mit n Würfen zu erreichen (weil x zu klein oder zu groß ist). Wir setzen daher $\varphi(0, 0) = 1$ und $\varphi(0, x) = 0$ für $x > 0$.

Für unseren Algorithmus wollen wir dynamische Programmierung verwenden.

a) Fülle die Tabelle mit allen Möglichkeiten aus, mit drei Würfeln die Summe 5 zu erreichen.

Würfel 1	Würfel 2	Würfel 3

b) Gib eine Definition für $\varphi(1, x)$ an, die alle Werte von x abdeckt.

c) Gib eine Formel an, mit der sich $\varphi(n + 1, x)$ berechnen lässt, wenn man alle Werte von $\varphi(n, x)$ kennt.

Aufgabe 5: Approximation - Greedy$_k$ **(8+3+3 Punkte)**

In dieser Aufgabe betrachten wir den Algorithmus GREEDY$_k$ mit $k = 2$ aus der Vorlesung auf der folgenden Instanz.

i	1	2	3	4
z_i	9	6	7	8
p_i	13	8	8	9

mit $Z = 21$.

a) Wende den Algorithmus GREEDY$_k$ mit $k = 2$ auf die Instanz an. Gib die folgenden Mengen bzw. Werte in jeder Iteration von GREEDY$_k$ tabellarisch an:

- $\overline{S}$: Menge fixierter Objekte
- $\sum_{i \in \overline{S}} z_i$: Gewicht der fixierten Objekte
- $Z - \sum_{i \in \overline{S}} z_i$: Restkapazität
- $G + \sum_{i \in \overline{S}} p_i$: Wert der fixierten Objekte plus Greedy auf nicht fixierten Objekten
- G_k: Wert der bisher besten gefundenen Lösung
- S: Lösungsmenge der bisher besten Lösung

Achte darauf, dass $\overline{S}$ mit einer kleinsten Menge anfängt und mit einer größten Menge endet. Zusätzlich sollen die Mengen $\overline{S}$ wie in den Hausaufgaben lexikographisch sortiert betrachtet werden: 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 $y \in \overline{S}_2 \setminus \overline{S}_1$. (Hinweis: Die Menge $X \setminus Y$ enthält Elemente aus X, die nicht in Y vorkommen.)

$\overline{S}$	$\sum_{i \in \overline{S}} z_i$	$Z - \sum_{i \in \overline{S}} z_i$	$G + \sum_{i \in \overline{S}} p_i$	G_k	S

b) Begründe, dass k immer so gewählt werden kann, das $\textsc{Greedy}_k$ eine optimale Lösung berechnet.

c) Wir haben in der Vorlesung gelernt, dass $\textsc{Greedy}_k$ eine Familie von Algorithmen mit polynomieller Laufzeit ist. Warum ergibt das mit der Aussage aus b) keinen Beweis für $P = NP$?

Aufgabe 6: Hashing **(6+1+3+3+4 Punkte)**

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

$$h(j, i) = j^2 + j + i \mod 10.$$

Dabei ist j der einzufügende Schlüssel und i der Versuchszähler, der anfangs immer 0 ist. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

$$4, \quad 6, \quad 7, \quad 3, \quad 1.$$

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

k	0	1	2	3	4	5	6	7	8	9
$A[k]$										

b) Wie heißt die Sondierung, die in a) genutzt wird?

c) Angenommen, in der Tabelle aus a) soll noch ein weiteres Element eingefügt werden, das nicht bereits in der Tabelle enthalten ist. Angenommen, das Element x wird zufällig so gewählt, dass für alle $0 \leq k < 10$ gilt: $\text{Prob}(h(x) = k) = \frac{1}{10}$. Mit welcher Wahrscheinlichkeit landet das Element in $A[5]$? Mit welcher Wahrscheinlichkeit landet es in $A[8]$?

d) Begründe kurz, warum beim Hashing mit offener Adressierung Schlüssel nicht einfach gelöscht werden dürfen.

e) Angenommen, du hast ein Array mit n Elementen gegeben, in dem ganze Zahlen enthalten sind. Deine Aufgabe ist, zu prüfen, ob es in diesem Array zwei Zahlen gibt, die in Summe 42 ergeben. Beschreibe in Stichpunkten, wie du Hashing nutzen kannst, um dies in (erwarteter) *linearer* Laufzeit herauszufinden.

Aufgabe 7: Komplexität **(3+3+4 Punkte)**

a) Betrachte die folgende Instanz von 0-1-KNAPSACK:

$$\begin{array}{rr} z_1 = p_1 = & 100010 \\ z_2 = p_2 = & 100101 \\ z_3 = p_3 = & 10101 \\ z_4 = p_4 = & 10010 \\ z_5 = p_5 = & 1000 \\ z_6 = p_6 = & 1111 \\ z_7 = p_7 = & 200 \\ z_8 = p_8 = & 100 \\ z_9 = p_9 = & 20 \\ z_{10} = p_{10} = & 10 \\ z_{11} = p_{11} = & 2 \\ z_{12} = p_{12} = & 1 \end{array} \quad \text{und } Z = 111444$$

Nimm an, dass die oben abgebildete 0-1-KNAPSACK-Instanz aus einer 3-SAT-Instanz mit Hilfe der in der Vorlesung vorgestellten Reduktion erzeugt wurde. Gib die ursprüngliche 3-SAT-Formel an.

b) Nenne ein Problem, das *NP*-vollständig ist. Beweise, dass es in *NP* liegt.

c) In der Vorlesung wurde die Reduktion von 3-SAT auf 0-1 KNAPSACK mit dem Zielwert

$$Z = \sum_{i=0}^{n-1} 10^{m+i} + \sum_{i=0}^{m-1} x \cdot 10^i,$$

für $x = 4$ vorgestellt, wobei n die Anzahl der Variablen und m die Anzahl der Klauseln in der 3-SAT-Formel ist.

Wie kann man die Reduktion anpassen, sodass sie auch für $x = 3$ korrekt funktioniert?

Aufgabe 8: Kurzfragen **(2+2+2+2+2 Punkte)**

Kreuze an, welche Aussagen korrekt sind. (Hinweis: In jeder Teilaufgabe ist immer mindestens eine Aussage korrekt.)

a) Welche Aussagen zu Fractional Knapsack sind korrekt?

Der Greedy-Algorithmus für Fractional Knapsack hat Laufzeit $\Theta(n^2)$. ☐

Es existiert ein Greedy-Algorithmus, der das Problem optimal löst. ☐

Die optimale Lösung hat immer einen echt größeren Wert als die Lösung derselben Instanz als ganzzahlige Variante (Maximum Knapsack). ☐

b) Welche Aussagen zu den in der Vorlesung vorgestellten dynamischen Programmen sind korrekt? Das dynamische Programm für ...

... Maximum Knapsack löst das Problem in polynomieller Zeit. ☐

... Subset Sum hat Laufzeit $\Theta(nZ)$. ☐

... Subset Sum kann genutzt werden, um Partition zu lösen. ☐

c) Sei v ein Knoten im Enumerationsbaum für ein Maximierungsproblem und UB eine dazu passende obere Schranke. UB gilt in jedem Fall ...

... im gesamten Enumerationsbaum. ☐

... oberhalb von v. ☐

... unterhalb von v. ☐

d) Es gibt eine ...

... $1/2$-Approximation für Maximum Knapsack. ☐

... 1-Approximation für Fractional Knapsack. ☐

... 2-Approximation für Minimum Vertex Cover. ☐

e) Falls ein Algorithmus existiert, welcher 3-SAT in polynomieller Zeit löst ...

... existiert auch ein Algorithmus, der jedes *NP*-schwere Problem in polynomieller Zeit löst. ☐

... gilt $P = NP$. ☐

... existiert ein Algorithmus, der Maximum Knapsack in polynomieller Zeit löst. ☐

Viel Erfolg ☺