Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Prof. Dr. Sándor Fekete
Dr. Phillip Keldenich

Sommersemester 2023

Klausur
Algorithmen und Datenstrukturen II
11.08.2023

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 14 Blättern, bitte auf Vollständigkeit überprüfen.
- Erlaubte Hilfsmittel: keine
- Eigenes Papier ist nicht erlaubt.
- Die Rückseiten der Blätter dürfen beschrieben werden.
- 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**	**Σ**
Punkte	**9**	**14**	**10**	**13**	**14**	**14**	**18**	**8**	**100**
Erreicht									

Aufgabe 1: Greedy-Algorithmen - Fractional Knapsack (5+1+1+1+1 Punkte)

a) Betrachte folgende Instanz für Fractional Knapsack:

i	1	2	3	4	5	6
z_i	6	3	9	10	10	5
p_i	9	1	12	7	8	10
z_i/p_i	0.66...	3	0.75	1.42...	1.25	0.5
RF						

mit $Z = 21$.

Wende den Greedy-Algorithmus für Fractional Knapsack auf diese Instanz an. Fülle dazu die Zeile 'RF' in der obigen Tabelle mit der Reihenfolge, in der die Objekte betrachtet werden. Trage also beim zuerst betrachteten Objekt eine 1 ein, beim zweiten eine 2, und so weiter.

Gib außerdem in jeder Iteration die folgenden Werte an: den aktuellen Gegenstand, zu welchem Anteil er gepackt wird, das neue Gesamtgewicht und den neuen Gesamtwert. Nutze dafür die folgende Tabelle. Falls Iterationen entfallen, streiche die entsprechende Zeile in der Tabelle.

Iteration	Aktueller Gegenstand	Gepackter Anteil	Gesamtgewicht	Gesamtwert
1				
2				
3				
4				
5				
6				

b) Ist die in a) erhaltene Lösung optimal für Fractional Knapsack (ohne Begründung)?

c) Angenommen, Du wolltest die Lösung von Aufgabenteil a) als obere Schranke für das ganzzahlige Maximum Knapsack nutzen. Wie gut kann die Lösung für Maximum Knapsack höchstens sein?

d) Ist Fractional Knapsack NP-schwer (ohne Begründung)?

e) Ist 0-1-Knapsack in NP (ohne Begründung)?

Aufgabe 2: Dynamic Programming - Maximum Knapsack (6+2+2+4 Punkte)

a) Wende das dynamische Programm für MAXIMUM KNAPSACK auf folgende Instanz an.

Objekt	i	1	2	3	4	5
Gewicht	z_i	2	1	6	4	5
Wert	p_i	5	4	3	9	4

mit $Z = 11$

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

i \ x	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2												
3												
4												
5												

b) Was ist eine optimale Lösungsmenge? Was ist der optimale Lösungswert?

c) Betrachte das Problem INTEGER KNAPSACK, bei dem Objekte nur ganzzahlig oft mitgenommen werden dürfen, wobei aber beliebig viele Kopien aller Objekte zur Verfügung stehen. Gib einen Ausdruck an, der beschreibt, wie viele Kopien eines Objekts mit Gewicht z_i bei einer Gewichtsschranke von Z höchstens mitgenommen werden dürfen.

d) Wie lautet die Rekursionsgleichung für INTEGER KNAPSACK? Sei dazu $P'(x, i)$ der beste Lösungswert, der mit den ersten i Objekten und einem Gesamtgewicht von höchstens x erreicht werden kann.

Aufgabe 3: Branch-And-Bound **(5+1+1+3 Punkte)**

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

i	1	2	3
z_i	1	10	6
p_i	2	11	6

und $Z = 16$.

- 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.
- Beschrifte die Kanten mit der Auswahl, die getroffen wurde.
- Beschrifte die Knoten mit den aktuell besten Schranken (obere und untere).
- Beschrifte einen Knoten mit *unzulässig*, falls die aktuelle Auswahl unzulässig ist.
- Sollten Kanten nicht benutzt werden, streiche sie durch.
- Nutze die Menge-Wert-Tabelle, um neue beste Lösungen festzuhalten.
- Die einzelnen Schritte der Algorithmen, mit denen die Schranken bestimmt werden, brauchst Du nicht anzugeben.

Menge	Wert

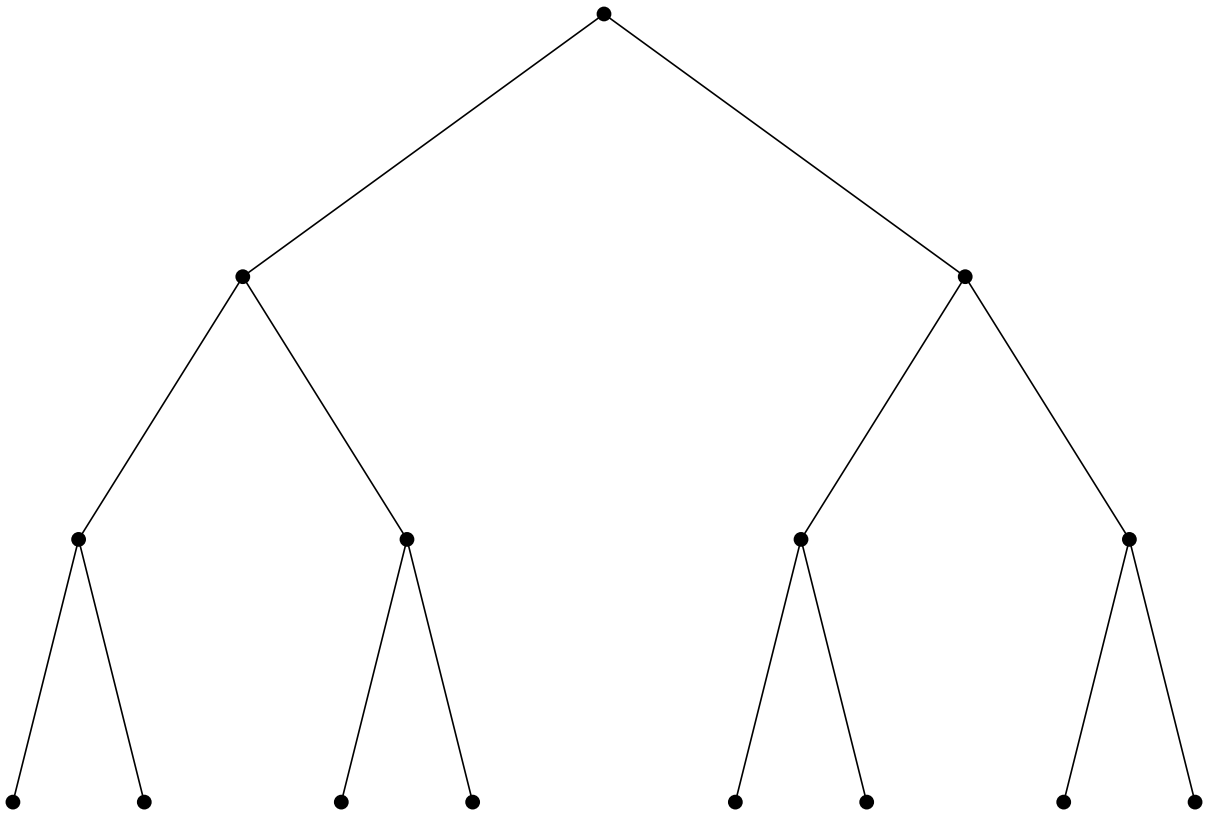

Abbildung 1: Ein Entscheidungsbaum.

b) Angenommen, Du dürftest die Reihenfolge der Objekte ändern. Könnte sich dadurch die Anzahl der rekursiven Aufrufe des Algorithmus ändern (ohne Begründung)?

c) Kann man mit Branch & Bound-Algorithmen NP-vollständige Probleme lösen?

d) Gib einen Ausdruck an, der die Anzahl der Blätter in einem vollständigen Enumerationsbaum für MAXIMUM KNAPSACK mit n Objekten beschreibt.

Aufgabe 4: Algorithmenentwurf: Dynamic Programming (3+5+5 Punkte)
Angenommen, wir werfen einen 6-seitigen Würfel n mal (oder n Würfel gleichzeitig) und zählen die Summe aller Augen zusammen. Dabei sind alle Zahlen zwischen n und $6n$ möglich. Einige Zahlen 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 Zahl 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) Gib eine Definition für $\varphi(1, x)$ an, die alle Werte von x abdeckt.

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

c) Beschreibe einen Dynamic Programming-Algorithmus, der $\varphi(n, x)$ berechnet. Nutze dabei Deine Formel aus b).

Aufgabe 5: Approximation - Greedy$_k$ **(6+2+2+2+2 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	1	2	5	14
p_i	5	6	7	14

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 $\overline{S}$ 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) Gib die von Greedy$_k$ zurückgegebene Lösung und ihren Wert an.

c) Ist die in a) erhaltene Lösung optimal? Begründe Deine Antwort.

d) Begründe, dass k immer so gewählt werden kann, dass GREEDY$_k$ eine optimale Lösung berechnet.

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

Aufgabe 6: Hashing **(7+3+2+2 Punkte)**

a) Betrachte ein anfangs leeres Array A der Größe 13 mit Speicherzellen $A[0], \ldots, A[12]$. In diesem Array führen wir Hashing mit offener Adressierung mit linearer Sondierung durch, und nutzen dafür die Hashfunktion

$$h(x) = 2x^2 + 3x \mod 13.$$

Dabei ist x ein Schlüssel. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

$$5, \quad 3, \quad 7, \quad 6, \quad 12.$$

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.

j	0	1	2	3	4	5	6	7	8	9	10	11	12
$A[j]$													

b) Angenommen, in die Tabelle aus a) soll noch ein weiteres Element einfügt werden, das nicht bereits in der Tabelle enthalten ist. Angenommen, das Element x wird zufällig so gewählt, dass für alle $0 \leq i < 13$ gilt: $\text{Prob}(h(x) = i) = \frac{1}{13}$. Mit welcher Wahrscheinlichkeit endet das Element in $A[4]$? Mit welcher Wahrscheinlichkeit endet das Element in $A[11]$?

c) Beschreibe kurz, welches Problem bei linearer Sondierung auftritt.

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

Aufgabe 7: Komplexität **(2+2+6+2+6 Punkte)**

a) Wie haben wir gezeigt, dass ein Problem NP-schwer ist?

b) Nimm an, dass P und NP verschieden sind. Nenne ein Beispiel für ein Problem, das in NP, aber nicht in P liegt.

c) In der Vorlesung haben wir eine Reduktion von 3-SAT auf Subset Sum gezeigt. Wende diese Reduktion auf die folgende Formel an.

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3).$$

d) Beim klassischen 3-SAT muss in jeder Klausel mindestens ein Literal erfüllt sein. Beim Problem 1-in-3-SAT muss in jeder Klausel *genau ein* Literal erfüllt sein. Eine Klausel wie $(x_1 \vee \overline{x_2} \vee x_3)$ wird also durch die Bedingung 'genau-eins-von$(x_1, \overline{x_2}, x_3)$' ersetzt, die wahr ist genau dann, wenn genau eines der Literale x_1, $\overline{x_2}$, x_3 wahr ist. Ist die Formel aus c) erfüllbar, wenn man sie als 1-in-3-SAT-Instanz betrachtet? Falls ja, gib eine erfüllende Belegung an!

e) Gib eine Reduktion von 1-in-3-SAT auf Subset Sum an.
Hinweis: Du kannst die Reduktion aus der Vorlesung anpassen; ein möglicher Lösungsweg braucht weniger Objekte als in der Vorlesung!

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

Kreuze an, welche Aussagen korrekt sind. Es gibt nur Punkte für vollständig korrekt angekreuzte Teilaufgaben. (Hinweis: In jeder Teilaufgabe ist immer mindestens eine Aussage korrekt.)

a) Welche Aussagen zu Fractional Knapsack sind korrekt?

- Der Greedy-Algorithmus hat Laufzeit $\Theta(n \log n)$. ☐
- Das Problem liegt in NP. ☐
- Falls man alle Objekte beliebig oft mitnehmen kann, gibt es einen Algorithmus, der die optimale Lösung in Zeit $\Theta(n)$ berechnet. ☐

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

- ... Subset Sum hat Laufzeit $\Theta(nZ)$. ☐
- ... Maximum Knapsack hat Laufzeit $\Theta(n^2 Z)$. ☐
- ... Maximum Knapsack löst das Problem in pseudopolynomieller Zeit. ☐

c) Falls es einen Algorithmus gibt, der Vertex Cover in polynomieller Zeit löst ...

- ... existiert für jedes NP-schwere Problem ein Algorithmus, der es in polynomieller Zeit löst. ☐
- ... gilt P = NP. ☐
- ... existiert ein Algorithmus der 3-SAT in polynomieller Zeit löst. ☐

d) Betrachte einen Branch & Bound-Algorithmus und seinen Suchbaum für ein Maximierungsproblem. Der Wert ...

- ... P der unteren Schranke wird im Verlauf des Algorithmus nie kleiner. ☐
- ... U der oberen Schranke wird auf jedem Pfad von der Wurzel zu einem Blatt nie größer. ☐
- ... U der oberen Schranke wird auf jedem Pfad von der Wurzel zu einem Blatt nie kleiner. ☐

Viel Erfolg ☺