Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Sommersemester 2022

Prof. Dr. Sándor Fekete
Matthias Konitzny

Klausur
Algorithmen und Datenstrukturen II
19.08.2022

Name: ....

Vorname: ....

Matr.-Nr.: ....

Studiengang: ....

Klausurraum: ....

Platznummer: ....

☐ 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 13 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	**13**	**12**	**15**	**18**	**14**	**11**	**9**	**8**	**100**
Erreicht									

Aufgabe 1: Greedy-Algorithmen - Maximum Knapsack (6+2+5 Punkte)

a) Betrachte folgende Instanz für Maximum Knapsack:

i	1	2	3	4	5
z_i	6	3	10	10	10
p_i	9	1	12	10	11
	0,67	3	0,83	1	0,91

mit $Z = 20$

13
+9

Greedy ∘

Wende den Greedy-Algorithmus für Maximum Knapsack auf diese Instanz an. Gib in jeder Iteration die folgenden Werte an: den aktuellen Gegenstand, ob er gepackt wird, das neue Gesamtgewicht und den neuen Gesamtwert. Nutze dafür die folgende Tabelle:

Iteration	Aktueller Gegenstand	Gepackt?	Gesamtgewicht	Gesamtwert
1	1	Ja 1	6	9
2	3	Ja 1	16	21
3	5	Nein 0	16	21
4	4	Nein 0	16	21
5	2	Ja 1	19	22

b) Ist die in a) erhaltene Lösung optimal? Begründe deine Antwort!

Nein, da es eine bessere Lösung existiert, nämlich die Items 3, 5 werden genommen, wobei Gesamtwert = 23 > 22

c) Zeige, dass der Greedy-Algorithmus für Maximum Knapsack keine Approximation liefert.

z.z. $\forall c \in (0;1]$ gibt es eine Instanz $0 < \frac{P_G}{P_{OPT}} < c$

i	1	2
z_i	1	B
p_i	1	$B-1$

$P_G = 1$
$P_{OPT} = B-1$

$\frac{P_G}{P_{OPT}} = \frac{1}{B-1} < c$
$\Leftrightarrow B-1 > \frac{1}{c}$
$\Rightarrow B > \frac{1}{c}+1$

Mit Archimedes
$\exists B \in \mathbb{N}: B > \frac{1}{c}+1$ für $\frac{1}{c}+1 \in \mathbb{R}_+$

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

a) Wende das dynamische Programm für Subset Sum auf folgende Instanz an.

Objekt i	1	2	3	4	5
Gewicht z_i	4	3	8	6	7

mit $Z = 16$

bei wiederholung: lag i-Objekte $S(x-z_i, i)$ 0
lag $(i-1)$-Objekte $S(x, i-1)$ 1 2

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.

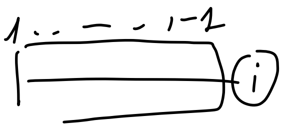
Capacity

i \ x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	0	0	0	0												
1	1																
2	1																
3	1																
4	1																
5	1																

b) Ist die Instanz von Subset Sum aus a) lösbar? Begründe deine Antwort!

Ja, da die letzte Spalte der Tabelle 1 enthält

nämlich die Lsg: Item 4, 5, 2



GPT

c) Wie lautet die Rekursionsgleichung, um Subset Sum zu lösen, wenn Objekte beliebig oft verwendet werden dürfen? Sei dazu $\mathcal{S}'(x,i) = 1$, wenn x mit den ersten i Objekten erzeugt werden kann und 0 andernfalls.

beliebig oft genommen

$$S(x,i) = \begin{cases} S(x, i-1) & x < z_i \text{, also } i \text{ wird sicher nicht verwendet; } S'(x,i) = 0 \\ \max(S(x, i-1), S(x-z_i, i)) & \text{sonst} \end{cases}$$

falls $S'(x,i) = 1$

falls $S'(x,i) = 0$

nicht mit ersten i Objekten erzeugt werden kann

Aufgabe 3: Branch-And-Bound **(10+5 Punkte)**

a) Wende den Branch-and-Bound-Algorithmus auf folgende, nach $\frac{z_i}{p_i}$ aufsteigend sortierte Instanz für Maximum Knapsack an.

i	1	2	3
z_i	5	8	11
p_i	9	9	11
			1

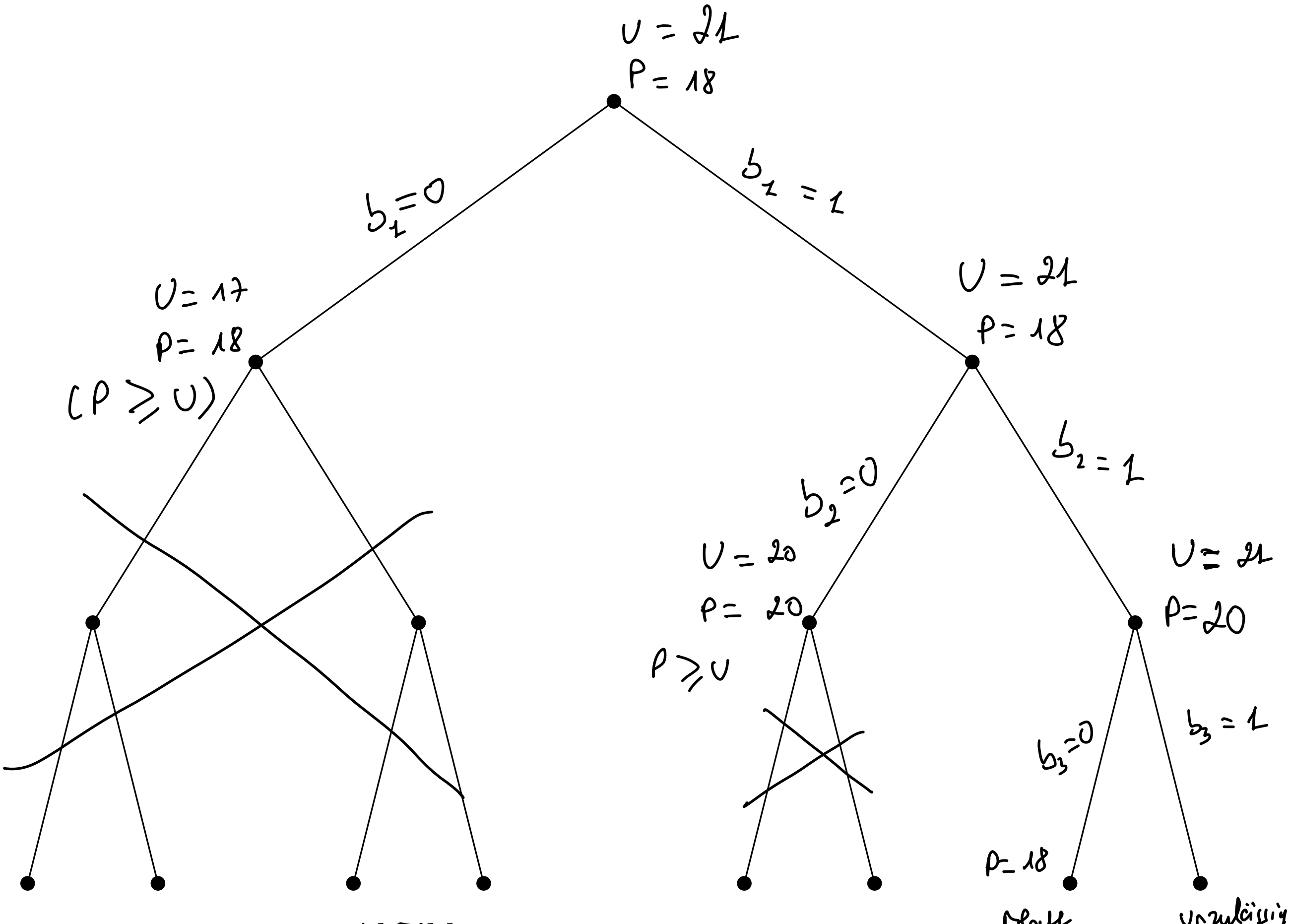
und $Z = 16$ $\qquad \frac{z_i}{p_i} \qquad \frac{16-5-8}{11}$

Beachte folgende Punkte:

- Benutze den Entscheidungsbaum aus Abbildung 1.
- 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.
- Halte in einer Tabelle fest, welche Objekte eine neue, beste Lösung liefern.

Menge	Wert
{1,2}	18
{1,3}	20

$\frac{16-8}{11}$



Term (Fun, V) : x+y
x+2

Abbildung 1: Ein Entscheidungsbaum.

b) Zeige: Für jedes $n \geq 1$ gibt es eine Instanz mit n Objekten, bei der der Branch-and-Bound-Algorithmus keine rekursiven Aufrufe startet.

Man kann immer so eine Instanz konstruieren, dass die ersten i Objekten ($i \leq n$) genau die Kapazität Z erreichen $\Rightarrow$ $P = U$. Nach dem Alg findet kein rekursive Aufrufe statt

Aufgabe 4: Approximation - Greedy$_k$ **(11+1+2+4 Punkte)**

In dieser Aufgabe betrachten wir den Algorithmus GREEDY$_k$ aus der Vorlesung. Wende GREEDY$_k$ auf die folgende Instanz an.

$n = 3$

i	1	2	3	4
z_i	2	4	14	15
p_i	7	9	11	7

mit $Z = 19$ und $k = 2$

a) 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 soll $\overline{S}$ lexikographisch sortiert sein: 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.)

$k = 2$

$\overline{S}$	$\sum_{i \in \overline{S}} z_i$	$Z - \sum_{i \in \overline{S}} z_i$	$G + \sum_{i \in \overline{S}} p_i$	G_k	S
$\emptyset$	0	19	16	16	{1,2}
{1}	2	17	16	16	{1,2}
{2}	4	15	16	16	{1,2}
{3}	14	5	18	18	{1,3}
{4}	15	4	14	18	{1,3}
{1,2}	6	13	16	18	{1,3}
{1,3}	16	3	18	18	{1,3}
{1,4}	17	2	14	18	{1,3}
{2,3}	18	1	20	20	{2,3}
{2,4}	19	0	16	20	{2,3}
{3,4}	29	–	–	20	{2,3}

b) Gib die von Greedy_k zurückgegebenen Lösungswerte an.

Menge: $\{2,3\}$

Wert: 20

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

GPT

Ja. Die Lösungen mit mehr als 3 Items kann nicht vorkommen, da die Summe der 3 kleinsten z_i größer als $Z = 19$ ist

↳ alle mögliche Lösungsmengen enthalten maximal 2 Items,

Mit Greedy_2 sind alle möglichen Lösungsmengen, die weniger als oder gleich 2 Items enthalten, untersucht.

↳ optimale Lösung kommt schon in Greedy_2 vor.

d) Kann k immer so gewählt werden, dass Greedy_k ein optimales Ergebnis zurückgibt? Begründe deine Antwort.

GPT

Ja.

Man zählt die Anzahl a an ersten Objekten mit kleinstem z_i und $\sum z_i > Z$

wähle $k = \max\{a-1, n\}$ n – Anzahl an Objekten

Es gilt, da die möglichen Lösungsmengen maximal $n-1$ Objekte enthalten und Greedy_{n-1} untersucht alle möglichen Lösungsmengen mit $(n-1)$ Objekten und weniger als $(n-1)$ Objekten

↳ die optimale Lösung gehört schon dazu

Aufgabe 5: Hashing **(7+1+1+3+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 der folgenden Hashfunktion durch:

$$t(x, i) = 5x^2 + 3 + i \cdot h(x) \mod 13 \quad \text{mit} \quad h(x) = (2x + 1 \mod 12) + 1$$

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

$$5, \quad 3, \quad 7, \quad 8 \quad 10$$

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
$A[j]$		7			~~8~~		10			3		5	

$h(5) = (2.5 + 1 \mod 12) + 1 = 12$

$t(5,0) = (5.25 + 3 + 0.h(5)) \mod 13 = \underline{11}$

$h(3) = (2.3 + 1 \mod 12) + 1 = 8$

$\hookrightarrow t(3,0) = (5.9 + 3 + 0.h(3)) \mod 13 = \underline{9}$

$h(7) = (2\ 7 + 1 \mod 12) + 1 = 4$

$\hookrightarrow t(7,0) = (5.49 + 3 + 0.h(7)) \mod 13 = 1$

$h(8) = (2.8 + 1 \mod 12) + 1 = 6$

$\hookrightarrow \underline{t(8,0)} = (5.64 + 3 + 0.h(8)) \mod 13 = 11$

$\hookrightarrow \underline{t(8,1)} = (5.64 + 3 + 1.h(8)) \mod 13 = 4$

$h(10) = (2.10 + 1 \mod 12) + 1 = 10$

$\hookrightarrow t(10,0) = (5.100 + 3 + 0.h(10)) \mod 13 = 9$

$\hookrightarrow t(10,1) = (5.100 + 3 + 1.10) \mod 13 = 6$

b) Wie ist der Belegungsfaktor β für eine Hashtabelle der Größe m mit n eingefügten Schlüsseln definiert?

$$\beta = \frac{n}{m}$$

c) Wie groß ist β in der Hashtabelle aus Aufgabenteil a) nach Einfügen der fünf Schlüssel?

$$\beta = \frac{5}{13}$$

d) Begründe kurz, warum beim Hashing mit offener Adressierung Schlüssel nicht einfach gelöscht werden dürfen. Beschreibe außerdem kurz, wie das Problem kurzfristig umgangen werden kann.

Beim löschen beendet sich der Search-vorgang früher als gewünscht

Die Plätze in der Hashtabelle werden als „schon besetzt", „noch nie besetzt", „wieder frei" markiert und die Suche wird nur beim „noch nie besetzt" abgebrochen

e) Beschreibe kurz, welches Problem auftritt, wenn die Lösung aus c) über einen längeren Zeitraum mit vielen Einfüge- und Löschoperationen verwendet wird.

Es gibt keine „noch nie besetzt" Felder

↳ Problem bei Suchenvorgang.

Aufgabe 6: Komplexität **(3+3+5 Punkte)**

a) Nenne ein Problem aus der Klasse NP und begründe kurz, warum es in NP liegt.

Subset Sum: NP, da man mit polynomieller Zeit eine Lösung verifizieren kann, ob sie gültig ist, indem man den Wert aller Objekte der Lösungen aufsummiert und checkt ob die Summe dem erwünschten Wert gleicht

Z

↓
O(n) Zeit

b) Beschreibe kurz, wie man NP-Schwere und NP-Zugehörigkeit zeigt. Nenne außerdem die Bezeichnung für ein Problem, welches beiden Klassen angehört.

+ NP-schwer: beweist es eine Polynomiell-Reduktion von einem bekannten NP-schwer Problem auf das betrachtete Problem existiert

+ NP: beweist dass jede Lösung in polynomieller Zeit verifiziert werden kann

+ NP-Vollständig

c) Betrachte die folgende Instanz von Maximum Knapsack:

x_1	$z_1 = p_1 =$	100110
$\overline{x_1}$	$z_2 = p_2 =$	100001
x_2	$z_3 = p_3 =$	10010
$\overline{x_2}$	$z_4 = p_4 =$	10101
x_3	$z_5 = p_5 =$	1000
$\overline{x_3}$	$z_6 = p_6 =$	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

und $Z = 111444$

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

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

Aufgabe 7: Dynamic Programming - Modellierung **(4+5 Punkte)**
Betrachte ein Gitter in dem bestimmte Zellen markiert sind. Das Gitter wird von einem $n \times n$ Array $M[1 \dots n, 1 \dots n]$ repräsentiert, in dem $M[i,j] = \textsc{True}$ eine markierte Zelle darstellt. Die Zelle $M[1,1]$ repräsentiert dabei die linke obere und $M[n,n]$ die untere rechte Ecke des Gitters.

Gesucht wird nun ein monotoner Pfad durch das Gitter, welcher links-oben startet und rechts-unten endet. Der Pfad kann immer nur nach rechts oder unten weiterführen und soll dabei möglichst viele der markierten Felder besuchen. Ein Beispiel ist in Abbildung 2 zu finden.

$P(10, j)$

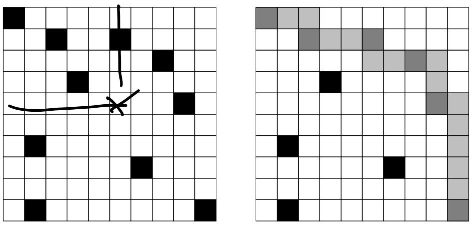


Abbildung 2: Beispielinstanz (links) mit monotonem Pfad welcher die meisten markierten Felder besucht (rechts).

a) Stelle eine Rekursionsgleichung $P(i,j)$ auf, welche für das Feld in der i-ten Zeile und j-ten Spalte die maximale Anzahl an markierten Feldern berechnet, die auf solch einem monotonen Pfad besucht werden können. Das Gitterfeld (i,j) ist dabei das letzte Feld auf dem Pfad.

Gesucht: wie viele markierte Felder maximal auf monotonem Pfad von $M[1,1]$ bis $M[i,j]$

2 Möglichkeiten:

a) die bessere Vorgänger liegt links neben uns

b) ______ liegt direkt über uns

$$P(i,j) = \begin{cases} M[i,j], & \text{wenn } i=j=1 \\ P(i,j-1) + M[i,j], & \text{sonst wenn } i=1 \\ P(i-1,j) + M[i,j], & \text{sonst } j=1 \\ \underline{\max}\{P(i-1,j), P(i,j-1)\} + \underbrace{M[i,j]}_{0 \text{ or } 1} & \text{sonst} \end{cases}$$

b) Entwirf einen Algorithmus in Pseudocode (maximal 15 Zeilen) welcher deine Rekursionsgleichung aus a) implementiert und das Problem löst.

Eingabe: $n \times n$ Array mit $M[i,j] \in \{0,1\}$ und $1 \leq i,j \leq n$
Ausgabe: Anzahl an bemerkten Feldern

```
Function P(i,j)
    if i=1 und j=1 then
        return M[1,1]
    if i=1 then
        return M[1,j] + P(1,j-1)
    if j=1 then
        return M[i,1] + P(i-1,1)
    return M[i,j] + max(P(i-1,j); P(i,j-1))
```

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?

- Es existiert ein Greedy-Algorithmus, der das Problem optimal löst. ☒
- Das Problem ~~liegt in NP~~. → P ☐
- Die optimale Lösung einer Instanz ist eine obere Schranke für die ganzzahlige Variante des Problems (MAXIMUM KNAPSACK). ☒

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

- ... im gesamten Enumerationsbaum. Subset sum ☐
- ... oberhalb von v. ☐
- ... unterhalb von v. 3-SAT ∈ P ☒

c) 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. Subset Sum ☐

d) Das dynamische Programm für MAXIMUM KNAPSACK ...

- ... hat eine Laufzeit von $O(nZ)$. ☒
- ... liefert immer eine optimale Lösung. ☒
- ... wird bei Branch&Bound für die untere Schranke verwendet. ☐

Viel Erfolg ☺