Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Dr. Arne Schmidt
Ramin Kosfeld
Chek-Manh Loi

Sommersemester 2025

Klausur
Algorithmen und Datenstrukturen II
08.08.2025

Nachname: .....

Vorname: .....

Matr.-Nr.: .....

Studiengang: .....

☐ Bachelor ☐ Master ☐ Andere: .....

Hinweise:

- Bitte das Deckblatt in Druckschrift vollständig ausfüllen.
- Die Klausur besteht aus 14 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	10	14	13	10	15	15	13	10	100
Erreicht									

Aufgabe 1: Dynamic Programming - Subset Sum **(5+1+4 Punkte)**

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

Objekt i	1	2	3	4	5
Gewicht z_i	6	4	6	11	3

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																

b) Wie lässt sich in der Tabelle ablesen, ob die gegebene Instanz für SUBSET SUM für ein $Z \in \{0, \ldots, 15\}$ lösbar ist?

c) Betrachte das Problem INTEGER SUBSET SUM, bei dem beliebig viele Kopien aller Objekte zur Verfügung stehen. Wie lautet die Rekursionsgleichung, 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.

Aufgabe 2: Branch-And-Bound **(7+3+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	9	7	6
p_i	11	8	6

und $Z = 14$.

- 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 neue beste Lösung festzuhalten.
- Die einzelnen Schritte der Algorithmen, mit denen die Schranken bestimmt werden, brauchst Du *nicht* anzugeben.

Menge	Wert

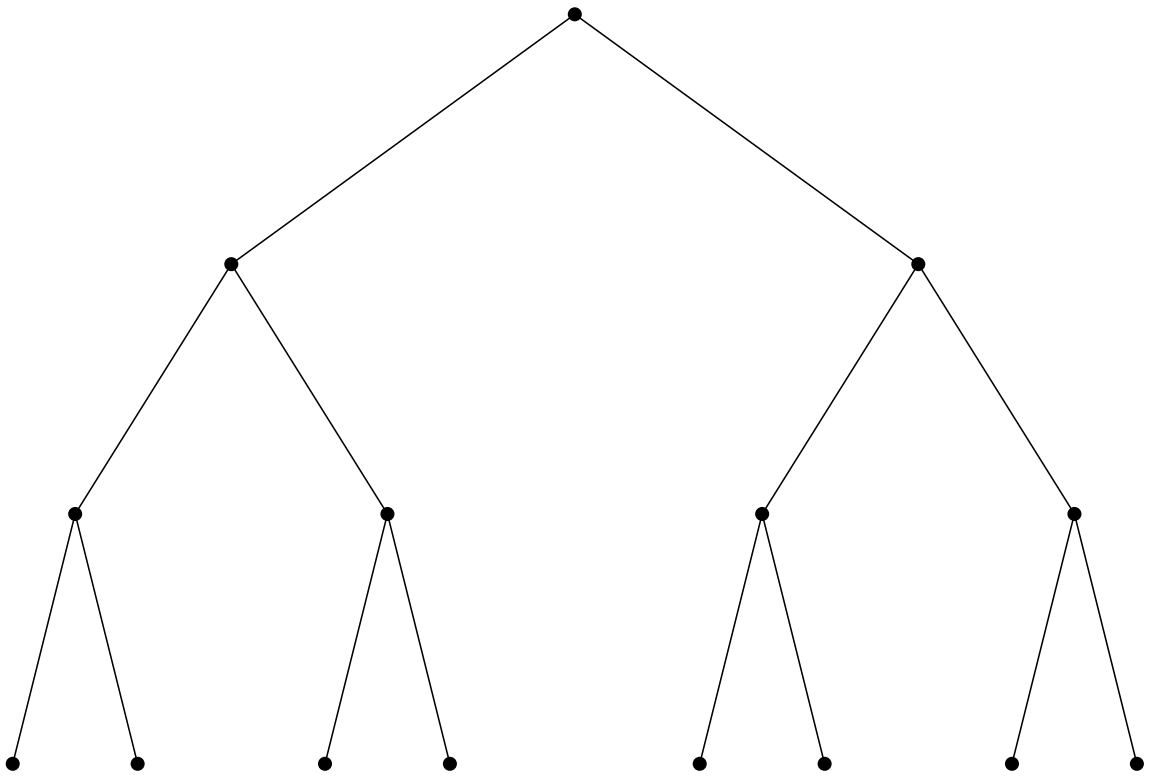

Abbildung 1: Ein Entscheidungsbaum.

b) Angenommen, wir wollen (aus welchen Gründen auch immer) beim Durchführen des Branch-and-Bound-Algorithmus für Maximum Knapsack auf die Berechnung einer unteren Schranke bei jedem rekursiven Aufruf verzichten. Erkläre, warum wir auch ohne einer solchen Berechnung trotzdem irgendwann untere Schranken finden und damit immer noch in der Lage sind, Teilbäume abzuschneiden.

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 Zahl der besuchten Knoten im Entscheidungsbaum beeinflussen kannst.

Aufgabe 3: Greedy-Algorithmen für Intervall-Cover (2+4+2+5 Punkte)
Wir betrachten das Problem IntervallCover, bei dem es darum geht, ein gegebenes Intervall I mit möglichst wenigen Intervallen aus der gegebenen Menge $C = \{[s_1, e_1), \ldots, [s_n, e_n)\}$ vollständig zu überdecken. Wir nehmen an, dass C so übergeben wird, dass es immer möglich ist, das Intervall I mit Intervallen aus C zu überdecken. Außerdem überlappt jedes Intervall in C an irgendeiner Stelle mit I.

a) Kreise in Abbildung 2 eine kleinstmögliche Zahl von Intervallen aus C ein, sodass diese Intervalle zusammen I vollständig überdecken.

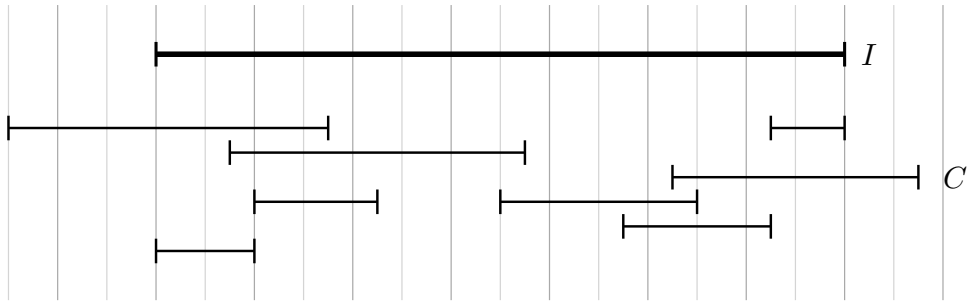

Abbildung 2: Eine Instanz von IntervallCover.

b) Führe folgenden Greedy-Algorithmus auf der Instanz in Abbildung 3 aus.

(i) Sortiere die Intervalle aus C nach deren Startpositionen von links nach rechts.

(ii) Für jedes Intervall i dieser Sortierung: Falls nach Entfernen von i das Intervall I immer noch überdeckt ist, entferne i.

(iii) Gib die übrig gebliebenen Intervalle zurück.

Nummeriere dazu sämtliche Intervalle entsprechend der Reihenfolge und markiere alle Intervalle mit einem Kreuz, die entfernt werden.

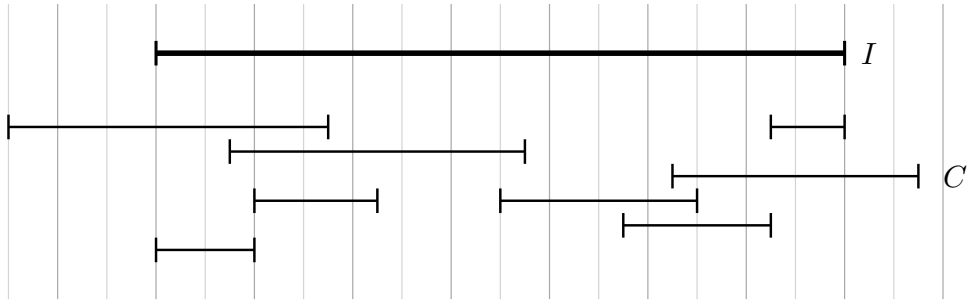

Abbildung 3: Eine Instanz von IntervallCover.

c) Zeige, dass Greedy für IntervallCover nicht optimal ist, wenn die Sortierung nach den Startpositionen der Intervalle von links nach rechts erfolgt.

d) Zeige, dass folgende Strategie IntervallCover optimal löst: Wir erweitern iterativ eine anfangs leere Lösungsmenge $M \subseteq C$, bis I vollständig von Intervallen aus M überdeckt ist. In jeder Iteration schreiben wir $R = [s, e)$ für das Intervall von I, was noch nicht von M überdeckt ist. Dann fügen wir wiederholt ein Intervall aus C zu M hinzu, was s beinhaltet und im resultierenden $R' = [s', e')$ das s' maximiert – also den bereits überdeckten Bereich.

Aufgabe 4: Algorithmenentwurf: Dynamic Programming (3+4+3 Punkte)
Gegeben sei ein Metallrohr H der Länge L und eine Preisliste $p_1, \ldots, p_n$, wobei p_i den Preis für ein Metallrohr der Länge i für $i = 1, \ldots, n$ angibt. Wir wollen H in kleinere Stücke teilen, um maximalen Profit zu bekommen, d.h. wir suchen Längen $\ell_1, \ldots, \ell_m$ mit $\sum_{j=1}^{m} \ell_j = L$ und $\sum_{j=1}^{m} p_{\ell_j}$ maximal.

a) Betrachte folgende Preisliste:

i	1	2	3	4	5
p_i	2	6	10	14	15

Was ist der maximale Profit für Rohre der Längen $0, \ldots, 8$? Fülle dazu die folgende Tabelle aus:

Länge	0	1	2	3	4	5	6	7	8
Profit									

b) Sei $\mathcal{P}(L)$ der maximale Profit, um ein Rohr der Länge L in Teilrohre der Längen $1, \ldots, n$ zu teilen. Stelle eine Rekursionsgleichung auf, um $\mathcal{P}(L)$ zu bestimmen.

c) Begründe kurz, dass deine Rekursionsgleichung aus b) korrekt ist.

Aufgabe 5: Approximation - Greedy$_k$ **(6+1+4+4 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	8	9	11	12
p_i	9	10	11	10

mit $Z = 20$.

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) Gib den Approximationsfaktor von $\textsc{Greedy}_k$ an (ohne Begründung).

c) Bei der Durchführung von $\textsc{Greedy}_k$ wird jeweils für die fixierten Elemente in $\overline{S}$ angenommen, dass sie auf jeden Fall gepackt werden. Wir betrachten jetzt eine modifizierte Variante von $\textsc{Greedy}_k$ mit dem Namen $\textsc{GreedyExclude}_k$, bei der die fixierten Elemente in $\overline{S}$ von einer möglichen Lösung *ausgeschlossen* werden. Auf den restlichen Elementen wird dann wie üblich $\textsc{Greedy}_0$ angewendet.

Führe $\textsc{GreedyExclude}_k$ für $k = 1$ auf der oben gegebenen Instanz aus:

- S_{Greedy} ist die Menge der von $\textsc{Greedy}_0$ gewählten Elemente
- G ist der Wert aller von $\textsc{Greedy}_0$ gewählten Elemente

$\overline{S}$	S_{Greedy}	G	G_k	S

d) Zeige, dass $\textsc{GreedyExclude}_k$ für $k = 5$ *keine* konstante Approximation für Maximum Knapsack ist. (Hinweis: Konstruiere als Gegenbeispiel eine Klasse von Instanzen, die ein Element $z_1 = p_1 = N$ enthält, wobei N beliebig groß gewählt werden kann.)

Aufgabe 6: Hashing **(5+1+3+2+4 Punkte)**

a) Betrachte ein anfangs leeres Array A der Größe 11 mit Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir Hashing mit offener Adressierung mit der folgenden Hashfunktion durch:

$$t(i, j) = j^2 - 2j + i \mod 11.$$

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:

$$4, \quad 6, \quad 7, \quad 3, \quad 2.$$

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
$A[j]$											

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

c) Kollisionen sind beim Hashing unvermeidbar. Welche zwei konzeptionell verschiedenen Ansätze gibt es, Kollisionen beim Hashing zu begegnen? Nenne jeweils einen Vorteil des jeweiligen Ansatzes.

d) Sei $h(x)$ eine Hashfunktion und $\text{Prob}(h(x) = j) = \frac{1}{m}$, wobei m die Größe der Hashtabelle bezeichnet. Wie lautet die Formel zur Bestimmung der Wahrscheinlichkeit, dass k Schlüssel kollisionsfrei in eine Hashtabelle eingefügt werden können?

e) Angenommen, du hast ein Array mit n Elementen gebeben, in dem positive ganze Zahlen enthalten sind. Deine Aufgabe ist, zu prüfen, ob es in diesem Array zwei aufeinanderfolgende Zahlen gibt (also z.B. 7 und 8, falls du beide irgendwo in dem Array findest). Beschreibe in Stichpunkten, wie du Hashing nutzen kannst, um dies in (erwarteter) *linearer* Laufzeit herauszufinden, also insbesondere ohne Sortieren. (Hinweis: Pseudocode ist nicht erforderlich.)

Aufgabe 7: Komplexität **(4+1+3+3+2 Punkte)**

a) Betrachte die folgende 3-SAT-Instanz S.

$$S(x_1, x_2, x_3, x_4) = C_1 \wedge C_2 \wedge C_3 =$$
$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

Wir wollen nun die Reduktion von 3-SAT auf VertexCover, wie in der Vorlesung gezeigt, am Beispiel von S durchführen. In Abbildung 4 sind bereits alle benötigten Knoten der aus S resultierenden VertexCover-Instanz eingezeichnet und einige Beschriftungen vorgenommen worden. Ergänze die korrekten Kanten in Abbildung 4.

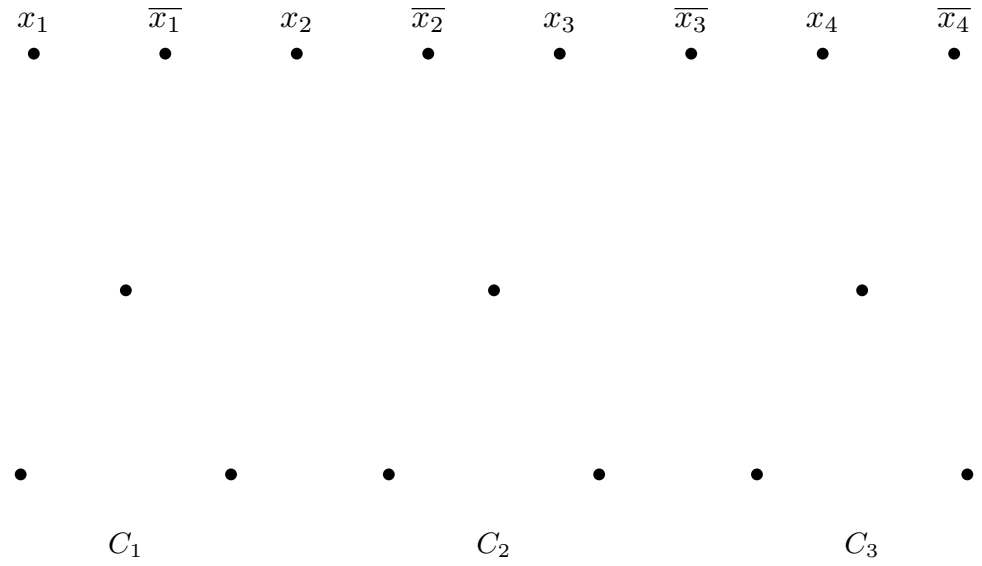

Abbildung 4: Eine unvollständige VertexCover-Instanz.

b) Markiere in Abbildung 4 ein minimales Vertex Cover.

c) Wie ist die Problemklasse P definiert? Gib außerdem ein Problem in P an und begründe die Zugehörigkeit.

d) Wie ist die Problemklasse NP definiert? Gib außerdem ein Problem in NP an und begründe die Zugehörigkeit.

e) Angenommen es gilt P $\neq$ NP. Gibt es Probleme in NP, die nicht NP-schwer sind? Begründe deine Antwort.

Aufgabe 8: Kurzfragen **(2+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 für Fractional Knapsack hat Laufzeit $\Theta(n^2)$. ☐
- Der Greedy-Algorithmus löst das Problem optimal. ☐
- Das Problem liegt in P. ☐

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

- ... Subset Sum hat Laufzeit $\Theta(Z \cdot 2^n)$. ☐
- ... Maximum Knapsack hat Laufzeit $\Theta(nZ)$. ☐
- ... Subset Sum kann genutzt werden, um Partition zu lösen. ☐

c) Betrachte den Branch-and-Bound-Algorithmus für Maximum Knapsack.

- Der Algorithmus löst Maximum Knapsack optimal. ☐
- Wir benutzen den Greedy-Algorithmus für Fractional Knapsack, um untere Schranken zu berechnen. ☐
- Der Wert P der unteren Schranke wird im Verlauf der Ausführung des Algorithmus nie kleiner. ☐

d) Der Algorithmus Greedy_k ...

- ... liefert stets eine Lösung, die mindestens so gut ist wie die von Greedy_0. ☐
- ... hat eine Laufzeit von $\Theta(k^n)$. ☐
- ... löst Maximum Knapsack optimal für $k := n$. ☐

e) Welche Aussagen sind wahr?

- Es existiert eine 2-Approximation für VertexCover. ☐
- Das Travelling Salesman Problem ist ein Maximierungsproblem. ☐
- Mithilfe eines Branch-and-Bound-Algorithmus kann man – zusammen mit einem Solver für lineare Programme (LPs) – Integer Programme (IPs) optimal lösen. ☐

Viel Erfolg ☺