↑ Für deinen Klausurcode ✂

Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

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

Sommersemester 2024

Klausur
Algorithmen und Datenstrukturen II
02.08.2024

Nachname: .....

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.
- 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	8	13	11	15	19	14	10	10	100
Erreicht									

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

a) Betrachte folgende Instanz für FRACTIONAL KNAPSACK:

i	1	2	3	4	5
z_i	6	3	9	10	5
p_i	10	1	12	8	10
RF					

mit $Z = 17$.

Wende den Greedy-Algorithmus für FRACTIONAL KNAPSACK auf diese Instanz an. Fülle dazu die Zeile 'Reihenfolge(RF)' in der obigen Tabelle mit der Reihenfolge, in der die Objekte betrachtet werden.

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				

b) Begründe kurz, ob die in a) erhaltene Lösung eine gültige Lösung für das MAXIMUM-KNAPSACK ist.

Aufgabe 2: Branch-And-Bound **(7+2+1+1+2 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	7	10
p_i	2	7	11

und $Z = 17$.

- 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

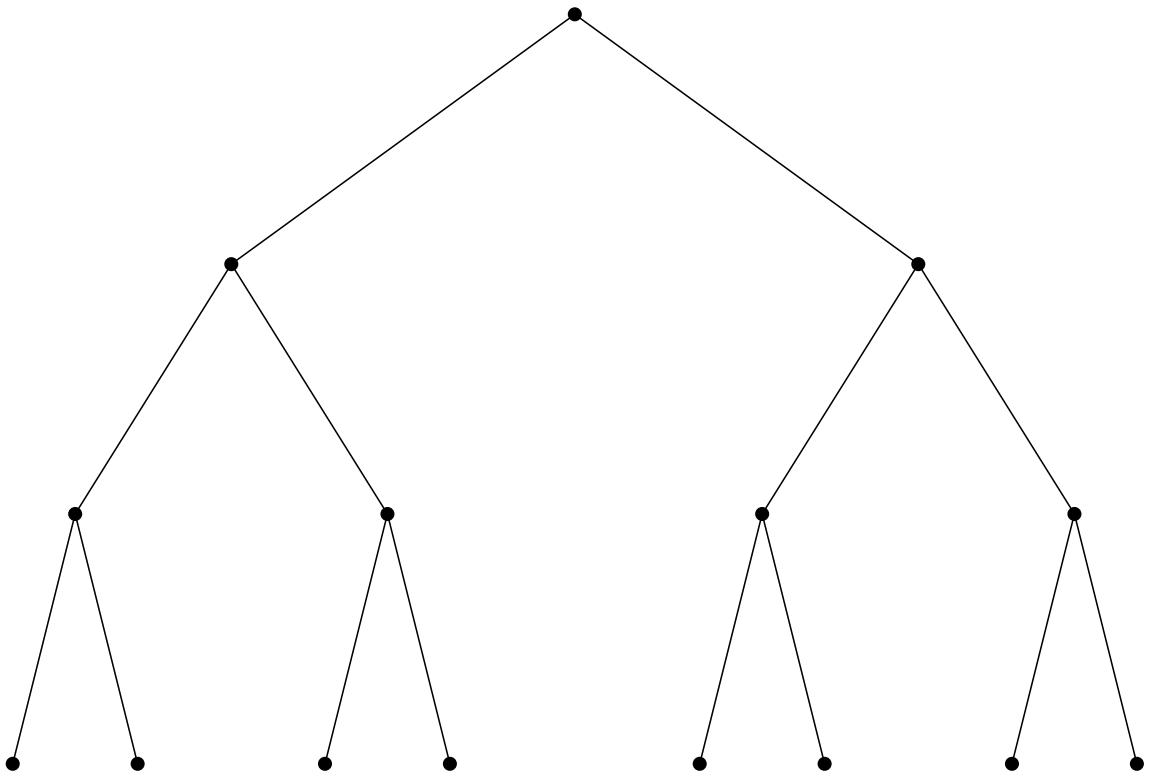

Abbildung 1: Ein Entscheidungsbaum.

b) Angenommen, bei der Durchführung des Branch-and-Bound-Algorithmus aus der Vorlesung wird über die letzte der n Variablen gebranched. In dem Fall sind bereits alle vorherigen Variablen fixiert. Können die beiden rekursiven Aufrufe an diesem Knoten (für $b_n = 0$ und $b_n = 1$) eine bessere untere Schranke identifizieren, als sie vor den beiden Aufrufen bekannt war? (Ohne Begründung)

c) Angenommen, der Branch-and-Bound-Algorithmus wählt bei jeder Ausführung erneut völlig zufällig aus, ob er an einem Knoten zunächst $x_i = 0$ oder $x_i = 1$ untersucht. Kann dann die Anzahl der rekursiven Aufrufe im Algorithmus bei mehrmaligen Durchführungen des Algorithmus variieren? (Ohne Begründung)

d) Kann man mit Branch-and-Bound-Algorithmen jedes NP-vollständige Probleme lösen? (Ohne Begründung)

e) Wie viele Blätter kann es in einem Enumerationsbaum über i Variablen maximal geben? (Ohne Begründung)

Aufgabe 3: Dynamic Programming - Subset Sum **(5+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	3	6	7	6	8

mit $Z = 14$

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
0															
1															
2															
3															
4															
5															

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

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 4: Modellierung: Knapsack nach Gewinn (3+4+3+5 Punkte)
Wir kennen bereits eine Rekursionsgleichung, um MAXIMUM KNAPSACK mithilfe von Dynamic Programming zu lösen. Anstatt in unserer Tabelle das Gewicht der Gegenstände aufsteigen zu lassen, wollen wir nun eine Rekursionsgleichung über den jeweiligen Gewinn formulieren. Dafür wollen wir eine Rekursionsgleichung $Q(y, i)$ finden, die das minimale Gewicht berechnet, was wir brauchen, um mit den ersten i Elementen einen Gewinn von y zu erreichen.

Für Minimierungsprobleme ist die Initialisierung der Tabelle besonders wichtig. Es gibt folgende Basisfälle:

$$Q(y, 0) = \begin{cases} 0 & , \text{ wenn } y = 0, \\ \infty & , \text{ sonst.} \end{cases}$$

a) Begründe kurz, warum wir bei diesem Problem die Initialisierung statt mit ∞ auch mit $Z + 1$ durchführen können. Z ist dabei das maximal erlaubte Gesamtgewicht des Rucksacks.

b) Gib eine Rekursionsgleichung für die verbleibenden Fälle an.

c) Mit der Rekursionsgleichung können wir nun alle Werte von $Q(y, i)$ berechnen. Allerdings wissen wir bisher nicht, wann wir aufhören können. Gib eine geeignete obere Schranke für y an.

d) Gegeben sind folgende Elemente.

	i	1	2	3	4
Gewicht	z_i	3	2	4	4
Nutzen	p_i	2	3	4	5

mit $Z = 8$

Bestimme für $Y = \{0, 1, \ldots, 9\}$ das minimale Gewicht $Q(Y, i)$, das benötigt wird, um mit den ersten i Elementen einen Gewinn von Y zu erreichen. (Hinweis: Anstatt ∞ in eine Zelle einzutragen, kannst du sie stattdessen einfach durchstreichen.)

Y \ i	0	1	2	3	4
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					

Aufgabe 5: Approximation - Greedy$_k$ **(6+2+2+3+2+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	2	3	8	13
p_i	5	6	9	13

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 die von Greedy$_k$ zurückgegebene Lösung und ihren Wert an.

c) Ist die von Greedy$_k$ erhaltene Lösung immer 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?

f) Aus der großen Übung kennen wir eine 2-Approximation zum Finden des minimalen Vertex Cover in einem Graphen G: Wir identifizieren zuerst ein inklusionsmaximales Matching M und wählen dann jeweils beide Knoten aller Kanten von M als Vertex Cover aus.

Zeige, dass diese Approximation scharf ist. Zeige dies, indem du eine Klasse von Instanzen angibst, bei denen die Zahl der Knoten beliebig groß gewählt werden kann, und wo der Approximationsfaktor jeweils 2 beträgt - egal, welches inklusionsmaximale Matching ausgewählt wird. Eine Begründung ist nicht notwendig.

Aufgabe 6: Hashing **(6+2+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 linearer Sondierung durch, und nutzen dafür die Hashfunktion

$$h(x) = x^2 + 4x \mod 11.$$

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

$$5, \quad 4, \quad 7, \quad 6, \quad 2.$$

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) 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 $0 \leq i < 11$ gilt: $\text{Prob}(h(x) = i) = \frac{1}{11}$. Mit welcher Wahrscheinlichkeit endet das Element in $A[3]$? Mit welcher Wahrscheinlichkeit endet das Element in $A[4]$?

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

d) Angenommen, du hast ein Array mit n Elementen gebeben, in dem positive und negative ganze Zahlen enthalten sind, aber keine 0. Deine Aufgabe ist, zu prüfen, ob es in diesem Array ein Paar von Zahlen gibt, die zusammengerechnet 0 ergeben (also z.B. 7 und -7, falls du beide irgendwo in dem Array findest). Beschreibe in Stichpunkten, wie du Hashing nutzen kannst, um dies in (erwarteter) *linearer* Laufzeit herauszufinden. (Hinweis: Pseudocode ist nicht erforderlich.)

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

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

b) Betrachte die folgende Instanz von VertexCover.

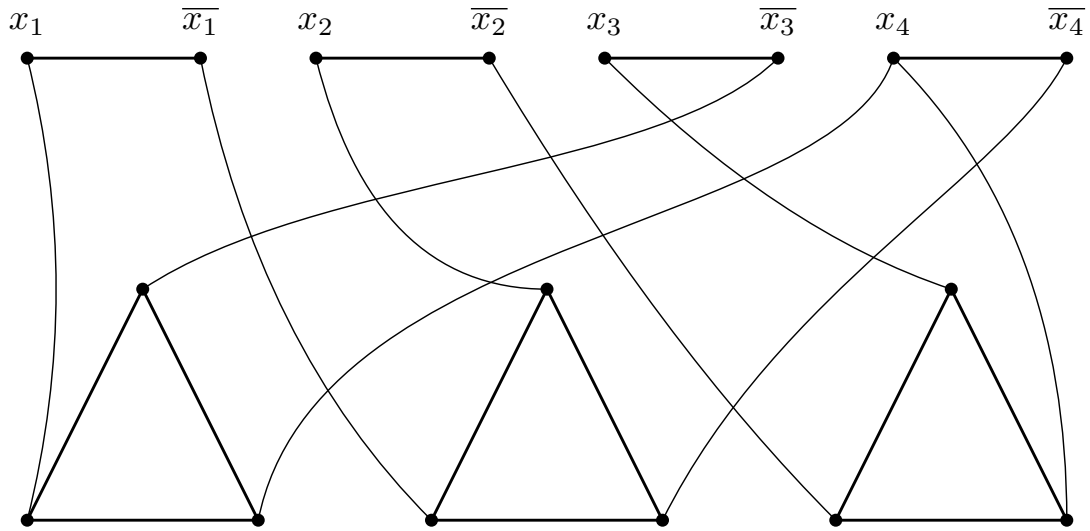

Abbildung 2: Eine VertexCover-Instanz.

Nimm an, dass die oben abgebildete VertexCover-Instanz aus einer 3-SAT-Instanz erzeugt wurde, wie in der Vorlesung gezeigt. Die Knoten, die den Variablenbelegungen in dieser Reduktion entsprechen, sind bereits entsprechend beschriftet. Gib die ursprüngliche 3-SAT-Formel an.

c) Angenommen 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 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 $\Theta(nZ)$. ☐
- ... MAXIMUM KNAPSACK löst das Problem in polynomieller Zeit. ☐

b) Welche Aussagen zu FRACTIONAL KNAPSACK sind korrekt?

- Der Greedy-Algorithmus für FRACTIONAL KNAPSACK hat Laufzeit $\Theta(n^2)$. ☐
- Das Problem liegt in P. ☐
- Falls alle Variablen in der Lösung ganzzahlig sind, ist das eine optimale Lösung für das MAXIMUM KNAPSACK. ☐

c) Sei A ein Problem in P. Welche Aussagen sind wahr?

- A lässt sich in $\mathcal{O}(n^2)$ lösen. ☐
- A lässt sich in polynomieller Zeit lösen. ☐
- Eine Lösung für A lässt sich in polynomieller Zeit verifizieren. ☐

d) Betrachte einen Branch-and-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. ☐

e) Der Algorithmus GREEDY_k

- ... liefert stets eine Lösung, die mindestens so gut ist wie die von GREEDY_0. ☐
- ... hat eine Laufzeit von $\Theta(k^n)$. ☐
- ... ist eine $(1 - \frac{1}{k+1})$-Approximation. ☐

Viel Erfolg ☺