



Institut für Theoretische Informatik  
Peter Widmayer  
Markus Püschel  
Thomas Tschager  
Tobias Pröger

# Beispiellösung zur Prüfung Datenstrukturen und Algorithmen D-INFK

15. Dezember 2016

Name, Vorname: \_\_\_\_\_

Stud.-Nummer: \_\_\_\_\_

Ich bestätige mit meiner Unterschrift, dass ich diese Prüfung unter regulären Bedingungen ablegen konnte und dass ich die untenstehenden Hinweise gelesen und verstanden habe.

Unterschrift: \_\_\_\_\_

Hinweise:

- Ausser einem Wörterbuch dürfen Sie keine Hilfsmittel verwenden.
- Bitte schreiben Sie Ihre Studierenden-Nummer auf **jedes** Blatt.
- Melden Sie sich bitte **sofort**, wenn Sie sich während der Prüfung in irgendeiner Weise bei der Arbeit gestört fühlen.
- Bitte verwenden Sie für jede Aufgabe ein neues Blatt. Pro Aufgabe kann nur eine Lösung angegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen werden.
- Bitte schreiben Sie **lesbar** mit blauer oder schwarzer Tinte. Wir werden nur bewerten, was wir lesen können.
- Sie dürfen alle Algorithmen und Datenstrukturen aus der Vorlesung verwenden, ohne sie noch einmal zu beschreiben. Wenn Sie sie modifizieren, reicht es, die Modifikationen zu beschreiben.
- Die Prüfung dauert 120 Minuten.

**Viel Erfolg!**



Stud.-Nummer: \_\_\_\_\_

Aufgabe	1	2	3	$\Sigma$
Mögl. Punkte	14	6	13	33
$\Sigma$ Punkte				



**Aufgabe 1.**

/ 14 P

*Hinweise:*

- 1) In dieser Aufgabe sollen Sie **nur die Ergebnisse** angeben. Diese können Sie direkt bei den Aufgaben notieren.
- 2) Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung "Datenstrukturen & Algorithmen" verwenden, sind Erklärungen oder Begründungen nicht notwendig. Falls Sie jedoch andere Methoden benutzen, müssen Sie diese **kurz** soweit erklären, dass Ihre Ergebnisse verständlich und nachvollziehbar sind.
- 3) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

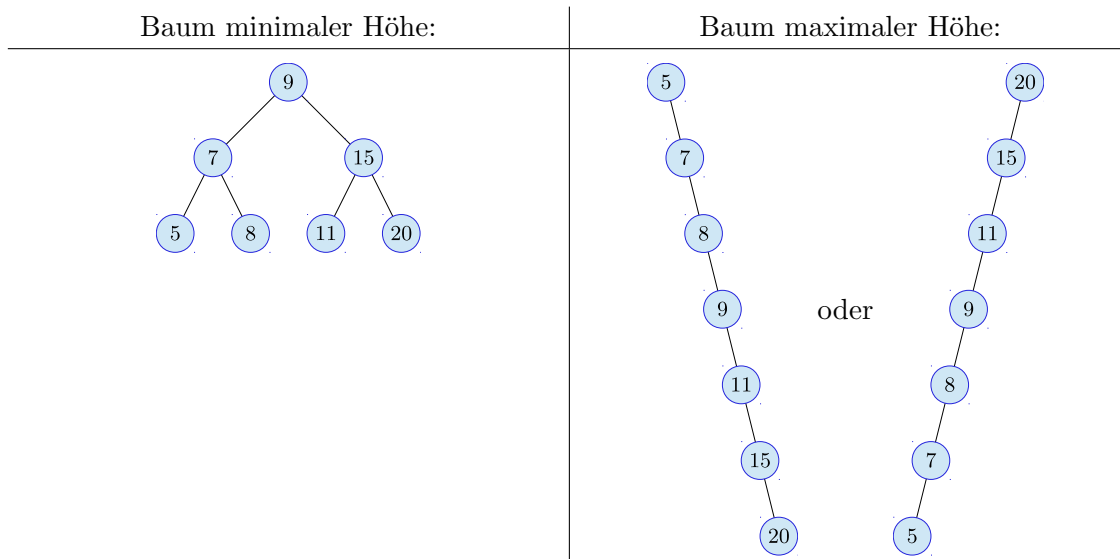
/ 1 P

- a) Fügen Sie die Schlüssel 4, 16, 20, 6, 12, 9, 5 in dieser Reihenfolge in die untenstehende Hash-tabelle ein. Benutzen Sie offenes Hashing mit der Hashfunktion  $h(k) = k \bmod 11$ . Lösen Sie Kollisionen mittels quadratischem Sondieren auf. Im Falle einer Kollision soll die Sondierung zunächst nach links und erst danach nach rechts erfolgen.

	12			4	16	6	5	9	20	
0	1	2	3	4	5	6	7	8	9	10

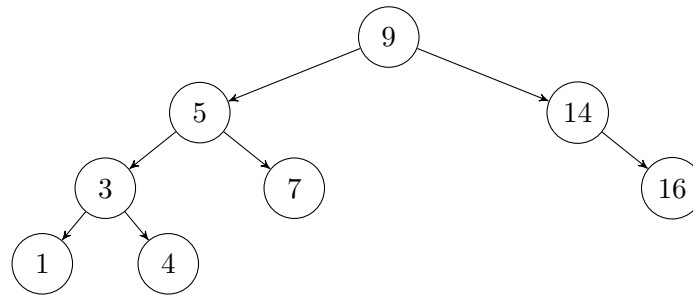
/ 1 P

- b) Gegeben sei die Schlüsselmenge  $\mathcal{K} = \{5, 9, 8, 11, 15, 7, 20\}$ . Zeichnen Sie die beiden binären Suchbäume, die genau die Schlüssel aus  $\mathcal{K}$  verwalten und die unter allen möglichen Suchbäumen minimale bzw. maximale Höhe haben.

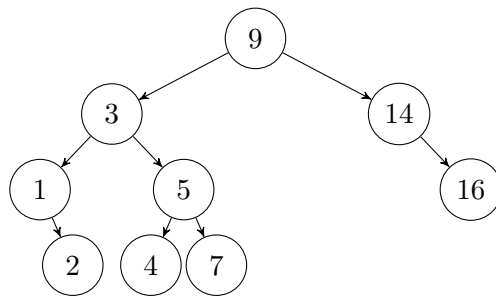


/ 1 P

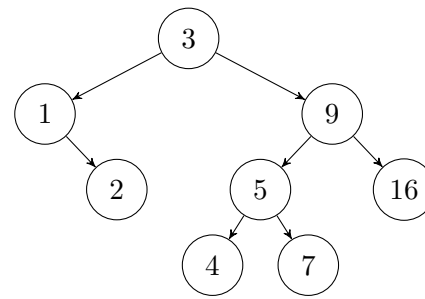
- c) Fügen Sie in den folgenden AVL-Baum *zuerst* den Schlüssel 2 ein und entfernen Sie *danach* den Schlüssel 14.



Nach Einfügen von 2:



Nach Löschen von 14:



/ 2 P

- d) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Jede korrekte Antwort gibt 0,5 Punkte, für jede falsche Antwort werden 0,5 Punkte abgezogen. Eine fehlende Antwort gibt 0 Punkte. Insgesamt gibt die Aufgabe mindestens 0 Punkte. Sie müssen Ihre Antworten nicht begründen.

*Eine Inorder-Traversierung eines binären Suchbaums erzeugt eine sortierte Liste der gespeicherten Schlüssel.*

WAHR     FALSCH

*Hat eine Folge von  $m$  Operationen im schlimmsten Fall Gesamtkosten  $\mathcal{O}(m)$ , dann hat jede einzelne dieser Operationen im schlimmsten Fall Kosten  $\mathcal{O}(1)$ .*

WAHR     FALSCH

*Sortieren durch Auswahl ist ein stabiles Sortierverfahren.*

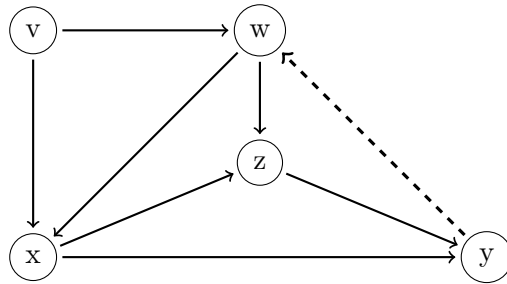
WAHR     FALSCH

*Auf einem komplett vorsortierten Array ist die Laufzeit von Sortieren durch Auswahl linear.*

WAHR     FALSCH

/ 1 P

e) Gegeben Sei der folgende Graph  $G = (V, E)$ . Geben Sie eine Menge  $E' \subset E$  kleinstmöglicher Kardinalität an, sodass  $G' = (V, E \setminus E')$  topologisch sortiert werden kann. Geben Sie ausserdem eine topologische Sortierung für  $G'$  an.



$E' = \underline{\hspace{2cm} (y, w) \hspace{2cm}}$

topologische Sortierung für  $G'$ :

           $v, w, x, z, y$           

/ 1 P

f) Führen Sie auf dem folgenden Array einen Aufteilungsschritt (in-situ, d.h. ohne Hilfsarray) des Sortieralgorithmus *Quicksort* durch. Benutzen Sie als Pivot das am rechten Ende stehende Element im Array.

11	5	9	6	1	8	3	4	2	12	7
1	2	3	4	5	6	7	8	9	10	11

2	5	4	6	1	3	7	9	11	12	8
1	2	3	4	5	6	7	8	9	10	11

/ 1 P

g) Das folgende Array enthält die Elemente eines in üblicher Form gespeicherten Min-Heaps. Entfernen Sie das minimale Element aus dem Heap, stellen Sie die Heapbedingung wiederher, und geben Sie das resultierende Array an.

2	8	10	9	12	15	11	13	14
1	2	3	4	5	6	7	8	9

8	9	10	13	12	15	11	14
1	2	3	4	5	6	7	8

/ 3 P

h) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} T(n/5) + 4n + 1 & n > 1 \\ 5 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (d.h. nicht-rekursive) und *möglichst einfache* Formel für  $T(n)$  an und beweisen Sie diese mit vollständiger Induktion.

*Hinweise:*

(1) Sie können annehmen, dass  $n$  eine Potenz von 5 ist.

(2) Für  $q \neq 1$  gilt:  $\sum_{i=0}^k q^i = \frac{q^{k+1}-1}{q-1}$ .

*Herleitung (falls benötigt):*

*Lösung:* Da wir annehmen dürfen, dass  $n$  eine Potenz von 5 ist, gilt  $n = 5^k$  für ein  $k \in \mathbb{N}$ . Wir teleskopieren, um auf eine Formel für  $T(n)$  zu kommen:

$$\begin{aligned} T(n) &= T(n/5) + 4n + 1 = (T(n/5^2) + 4n/5 + 1) + 4n + 1 \\ &= T(n/5^2) + 4n\left(\frac{1}{5^0} + \frac{1}{5^1}\right) + 2 = (T(n/5^3) + 4n/5^2 + 1) + 4n\left(\frac{1}{5^0} + \frac{1}{5^1}\right) + 2 \\ &= T(n/5^3) + 4n\left(\frac{1}{5^0} + \frac{1}{5^1} + \frac{1}{5^2}\right) + 3 = \dots \stackrel{!}{=} T(n/5^k) + 4n\left(\sum_{i=0}^{k-1} \frac{1}{5^i}\right) + k \\ &= T(5^k/5^k) + 4 \cdot 5^k \left(\sum_{i=0}^{k-1} \frac{1}{5^i}\right) + k = T(1) + 4 \cdot 5 \cdot \sum_{i=0}^{k-1} 5^i + k \\ &= 5 + 4 \cdot 5 \cdot \frac{5^k - 1}{5 - 1} + k = 5 + 5^{k+1} - 5 + k = 5^{k+1} + k = 5n + \log_5(n). \end{aligned}$$

*Geschlossene und vereinfachte Formel:*

$$T(n) = T(5^k) = 5^{k+1} + k$$



*Induktionsbeweis: Induktionshypothese* ( $k = 0$ ): Für  $k = 0$  ist  $T(n = 5^0) = T(1) = 5 = 5^1 + 0 = 5^{k+1} + k$ .

*Induktionsannahme:* Wir nehmen an, es gelte  $T(5^k) = 5^{k+1} + k$ .

*Induktionsschritt* ( $k \rightarrow k + 1$ ): Für  $k > 0$  gilt

$$\begin{aligned} T(n = 5^{k+1}) &= T(5^k) + 4 \cdot 5^{k+1} + 1 \stackrel{\text{Ind.-Ann.}}{=} 5 \cdot 5^k + k + 4 \cdot 5^{k+1} + 1 \\ &= 5^{k+1} + 4 \cdot 5^{k+1} + k + 1 = 5 \cdot 5^{k+1} + (k + 1) = 5n + \log_5(n). \end{aligned}$$

/ 1 P

- i) Geben Sie die asymptotische Laufzeit in Abhängigkeit von  $n \in \mathbb{N}$  für den folgenden Algorithmus (so knapp wie möglich) in  $\Theta$ -Notation an. Sie müssen Ihre Antwort nicht begründen.

---

```

1 for(int i = 1; i <= n/2; i += 2)
2     for(int j = n; j >= i; j -= 1)
3         for(int k = n; k > 2; k /= 2)
4             ;

```

---

*Lösung:* Die Schleife in Schritt 1 wird  $\Theta(n)$  Mal durchlaufen. Die Schleife in Schritt 2 wird  $\Theta(n - i)$  Mal durchlaufen, und da  $i \leq n/2$  für etwa  $n/4$  Werte von  $i$ , ist die Laufzeit von Schritt 2 in  $\Theta(n)$ . Die Schleife in Schritt 3 wird  $\Theta(\log n)$  Mal durchlaufen. Die Gesamtlaufzeit ist daher in  $\Theta(n^2 \log n)$ .

/ 1 P

- j) Geben Sie die asymptotische Laufzeit in Abhängigkeit von  $n \in \mathbb{N}$  für den folgenden Algorithmus (so knapp wie möglich) in  $\Theta$ -Notation an. Sie müssen Ihre Antwort nicht begründen.

---

```

1 for(int i = 1; i < n*n; i ++) {
2     for(int j = 1; j <= i; j *= 2)
3         ;
4     for(int k = 1; k*k <= n; k += 1)
5         ;
6 }

```

---

*Lösung:* Die äussere Schleife wird  $\Theta(n^2)$  Mal durchlaufen. Die erste innere Schleife wird  $\Theta(\log i) \subseteq \mathcal{O}(\log(n^2)) = \Theta(\log n)$  Mal durchlaufen, die zweite innere Schleife  $\Theta(\sqrt{n})$  Mal. Da beiden inneren Schleifen nacheinander ausgeführt werden, beträgt die Gesamtlaufzeit  $\Theta(n^2(\log n + \sqrt{n})) = \Theta(n^{5/2})$ .

/ 1 P

- k) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, so dass folgendes gilt: Wenn eine Funktion  $f$  links von einer Funktion  $g$  steht, dann gilt  $f \in \mathcal{O}(g)$ .

*Beispiel:* Die drei Funktionen  $n^3$ ,  $n^7$ ,  $n^9$  sind bereits in der entsprechenden Reihenfolge, da  $n^3 \in \mathcal{O}(n^7)$  und  $n^7 \in \mathcal{O}(n^9)$  gilt.

$$\sqrt{n^3}, \binom{n}{4}, n!, \frac{n^2}{\log n}, n \log n, 3^n, (\log n)^4$$

*Lösung:*  $(\log n)^4, n \log n, \sqrt{n^3}, \frac{n^2}{\log n}, \binom{n}{4}, 3^n, n!$

**Aufgabe 2.**

/ 6 P

Im Devisenhandel bezeichnet *Arbitrage* das Ausnutzen von Preisunterschieden, um durch mehrfaches Wechseln von Währungen einen Gewinn zu erzielen. Am 2. Juni 2009 zum Beispiel konnte 1 US-Dollar in 95.729 Yen gewechselt werden, 1 Yen in 0.00638 Britische Pfund und 1 Britisches Pfund in 1.65133 US-Dollar. Hätte ein Händler also 1 US-Dollar in Yen gewechselt, den erhaltenen Betrag in Britische Pfund und schliesslich diesen Betrag dann zurück in US-Dollar getauscht, dann hätte er  $95.729 \cdot 0.00638 \cdot 1.65133 \approx 1.0086$  US-Dollar erhalten, was einem Gewinn von 0.86% entspricht.

/ 3 P

- a) Gegeben seien  $n$  Währungen  $\{1, \dots, n\}$  und eine  $(n \times n)$ -Wechselkursmatrix  $R \in (\mathbb{Q}^+)^2$ . Für zwei Währungen  $i, j \in \{1, \dots, n\}$  kann eine Einheit der Währung  $i$  in  $R(i, j) > 0$  Einheiten der Währung  $j$  getauscht werden. Es soll entschieden werden, ob ein Arbitrage-Geschäft möglich ist, d.h., ob es eine Folge von  $k$  verschiedenen Währungen  $W_1, \dots, W_k \in \{1, \dots, n\}$  mit  $R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1) > 1$  gibt.

Modellieren Sie das Problem als Graphproblem. Konstruieren Sie aus der obigen Eingabe einen gerichteten, gewichteten Graphen  $G = (V, E, w)$ , der *genau dann* einen Kreis mit negativem Gewicht besitzt, wenn ein Arbitrage-Geschäft möglich ist. Begründen Sie, dass  $G$  genau dann einen Kreis negativer Länge enthält, wenn ein Arbitrage-Geschäft möglich ist.

*Lösung:* Wir erzeugen einen vollständigen Graphen mit der Knotenmenge  $V = \{1, \dots, n\}$ . Die Kante  $(i, j)$  bekommt das Gewicht  $w((i, j)) = -\ln(R(i, j))$ . Angenommen, ein Arbitrage-Geschäft mit den Währungen  $W_1, \dots, W_k \in \{1, \dots, n\}$  ist möglich. Dann gilt

$$\begin{aligned} R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1) &> 1 \\ \Leftrightarrow \ln(R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1)) &> \ln(1) = 0 \\ \Leftrightarrow \ln(R(W_1, W_2)) + \ln(R(W_2, W_3)) + \cdots + \ln(R(W_{k-1}, W_k)) + \ln(R(W_k, W_1)) &> 0 \\ \Leftrightarrow -\ln(R(W_1, W_2)) - \ln(R(W_2, W_3)) - \cdots - \ln(R(W_{k-1}, W_k)) - \ln(R(W_k, W_1)) &< 0 \\ \Leftrightarrow w((W_1, W_2)) + w((W_2, W_3)) + \cdots + w((W_{k-1}, W_k)) + w((W_k, W_1)) &< 0, \end{aligned}$$

folglich enthält  $G$  einen Zyklus mit negativem Gewicht. Da nur Äquivalenzumformungen durchgeführt wurden, gilt die Argumentation also in beide Richtungen.

/ 3 P

- b) Welcher Kürzeste-Wege-Algorithmus kann benutzt werden, um in einem Graphen Kreise negativer Länge zu erkennen? An welchem Knoten kann der Algorithmus beginnen? Welche Laufzeit (in Abhängigkeit von  $n$ ) erreicht der gewählte Algorithmus, wenn er auf den Graphen in a) angewendet wird?

*Lösung:* Der Algorithmus von Bellman und Ford ist eine sinnvolle Wahl, da er Zyklen mit negativem Gewicht erkennt. Da der Graph zusammenhängend ist und jeder Zyklus mit negativem Gewicht von jedem Knoten ausgehend erreicht werden kann, kann der Algorithmus bei jedem Knoten von  $G$  beginnen. Es gibt  $|V| = n$  Knoten und  $|E| = n(n-1) \in \Theta(n^2)$  viele Kanten. Der Algorithmus hat daher eine Laufzeit von  $\Theta(|V||E|) = \Theta(n^3)$ .



**Aufgabe 3.**

/ 13 P

Eine Firma hat den Auftrag bekommen, ein Kabel der Länge mindestens  $L$  zu produzieren. Da es im Lager viele bereits früher produzierte Kabelstücke gibt, soll das gewünschte Kabel aus diesen Teilstücken zusammengesetzt werden. Konkret gibt es im Lager  $n$  Kabelstücke. Kabelstück  $i$  hat Länge  $l_i$ . Werden zwei Kabelstücke mit Längen  $l_i$  und  $l_j$  zusammengesetzt, entsteht ein Kabel der Länge  $l_i + l_j$ . Um das entstehende Kabel nicht unnötig lang zu machen, soll es unter allen Möglichkeiten, ein Kabel der Länge  $\geq L$  herzustellen, minimale Länge haben. Sie dürfen annehmen, dass der Lagerbestand ausreichend gross ist, d.h., dass die Summe der Längen aller Kabelstücke mindestens  $L$  beträgt.

/ 9 P

- a) Geben Sie einen Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die minimale Länge eines Kabels berechnet, sodass dieses Kabel aus geeigneten Kabelstücken aus  $\{1, \dots, n\}$  zusammengesetzt werden kann und mindestens Länge  $L$  hat.

*Lösung:*

**Definition der DP-Tabelle:** Wir verwenden eine  $(n + 1) \times (1 + \sum_{i=1}^n l_i)$ -Tabelle  $T$  mit Einträgen aus  $\{\text{true}, \text{false}\}$ . Für  $i \in \{0, \dots, n\}$  und  $l \in \{0, \dots, \sum_{i=1}^n l_i\}$  habe  $T[i, l]$  genau dann den Wert  $\text{true}$ , wenn es eine Möglichkeit gibt, mit einer Auswahl geeigneter Kabelstücke aus  $\{1, \dots, i\}$  ein Kabel der Länge  $l$  zu erzeugen.

**Berechnung eines Eintrags:** Wir unterscheiden drei Fälle.

1. *Fall:*  $i = 0$ . Dies entspricht der Initialisierung, wenn keine Kabelstücke benutzt werden dürfen. Die einzig erreichbare Kabellänge ist 0, also setzen wir  $T[0, 0] \leftarrow \text{true}$  und  $T[0, l] \leftarrow \text{false}$  für jedes  $l \in \{1, \dots, \sum_{i=1}^n l_i\}$ .
2. *Fall:*  $i \in \{1, \dots, n\}$ ,  $l \in \{0, \dots, l_i - 1\}$ . Jedes zusammengesetzte Kabel mit Länge  $l < l_i$  benutzt das Kabelstück  $i$  nicht, weil dieses zu lang ist. Dann liegen alle benutzbaren Kabelstücke aus  $\{1, \dots, i\}$  in  $\{1, \dots, i - 1\}$ , und kann ein Kabel der Länge  $l$  aus geeigneten Kabelstücken aus  $\{1, \dots, i\}$  zusammengesetzt werden, dann kann dies auch mit geeigneten Kabelstücken aus  $\{1, \dots, i - 1\}$  geschehen. Deshalb setzen wir  $T[i, l] \leftarrow T[i - 1, l]$  für jedes  $l \in \{0, \dots, l_i - 1\}$ .
3. *Fall:*  $i \in \{1, \dots, n\}$ ,  $l \in \{l_i, \dots, \sum_{i=1}^n l_i\}$ . Mit den Kabelstücken  $\{1, \dots, i\}$  können wir genau dann ein Kabel mit Gesamtlänge  $l$  herstellen, wenn
  - das Kabelstück  $i$  benutzt wird und mit den Kabelstücken  $\{1, \dots, i - 1\}$  ein Kabel mit Gesamtlänge  $l - l_i$  hergestellt werden kann, oder wenn
  - das Kabelstück  $i$  nicht benutzt wird und mit den Kabelstücken  $\{1, \dots, i - 1\}$  ein Kabel mit Gesamtlänge  $l$  hergestellt werden kann.

Folglich wird  $T[i, l] \leftarrow T[i - 1, l] \vee T[i - 1, l - l_i]$  gesetzt.

**Berechnungsreihenfolge:** Wir berechnen die Einträge  $T[i, l]$  für aufsteigende  $i \in \{0, \dots, n\}$ . Für gleiche  $i$  berechnen wir die Einträge für aufsteigende  $l \in \{0, \dots, \sum_{i=1}^n l_i\}$ .

**Auslesen der Lösung:** Wir setzen  $l \leftarrow L$ . Falls  $T[n, l]$  nicht 'true' ist, inkrementieren wir  $l$  um 1, bis dies der Fall ist. Die gesuchte Länge ist  $l$ .

/ 2 P

- b) Beschreiben Sie detailliert, wie aus der DP-Tabelle abgelesen werden kann, welches Kabelstück in einer optimalen Lösung benutzt wird.

*Lösung:* Sei  $l$  die vom Algorithmus aus a) berechnete minimale Kabellänge. Wir setzen  $i \leftarrow n$  und wiederholen die folgenden Schritte, bis  $i = 0$  ist. Betrachte den Eintrag  $T[i, l]$ .

- Ist  $l \geq l_i$  und  $T[i, l] = T[i-1, l-l_i]$ , dann gibt es eine optimale Lösung, die das Kabelstück  $i$  benutzt, also geben wir  $i$  aus. Wir setzen  $l \leftarrow l - l_i$  und  $i \leftarrow i - 1$  und fahren analog fort.
- Ansonsten kann das Kabelstück  $i$  nicht verwendet werden, also wird nichts ausgegeben und  $l$  bleibt unverändert. Wir setzen aber  $i \leftarrow i - 1$  und fahren analog fort.

/ 2 P

- c) Geben Sie die Laufzeit des in a) und b) entwickelten Verfahrens an und begründen Sie Ihre Antwort. Ist die Laufzeit polynomiell?

*Lösung:* Die Zeit zur Berechnung eines Eintrags ist  $\Theta(1)$ . Da die Tabelle  $\Theta(n \sum_{i=1}^n l_i)$  Einträge besitzt, beträgt die Laufzeit des Algorithmus in a)  $\Theta(n \sum_{i=1}^n l_i)$ . Der Algorithmus in b) braucht zum Prüfen der zwei Fälle nur konstante Zeit, und da  $i$  von  $n$  bis 1 (bzw. 0) läuft, ist die Laufzeit  $\Theta(n)$ . Als Gesamtlaufzeit ergibt sich also  $\Theta((n \sum_{i=1}^n l_i) + n) = \Theta(n \sum_{i=1}^n l_i)$ . Diese ist pseudopolynomiell, weil  $\sum_{i=1}^n l_i$  Teil der Eingabe ist, aber durch nur  $\lceil \log(\sum_{i=1}^n l_i + 1) \rceil$  Bits dargestellt werden kann.