Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Tobias Pröger

Beispiellösung zur Prüfung
Datenstrukturen und Algorithmen
D-INFK

25. Januar 2014

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 Fremdsprachenwörterbuch 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 180 Minuten.

Viel Erfolg!

Stud.-Nummer: _____

Aufgabe	1	2	3	4	5	**Σ**
Mögl. Punkte	9	8	8	12	13	50
$\sum$ Punkte						

Aufgabe 1.

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) Im untenstehenden Array sind die Elemente eines Max-Heaps in der üblichen Form gespeichert. Wie sieht das Array aus, nachdem das Maximum entfernt wurde und die Heap-Bedingung wieder hergestellt wurde?

27	17	20	15	7	9	13	8	2	5	3	1	6
1	2	3	4	5	6	7	8	9	10	11	12	13

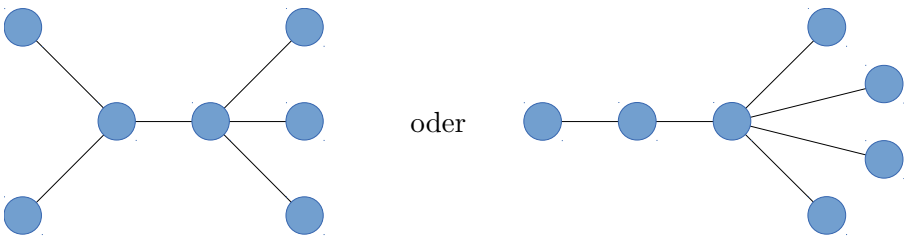
20	17	13	15	7	9	6	8	2	5	3	1

1 P (b) Gegeben sei ein zusammenhängender, ungerichteter Graph $G = (V, E)$ mit $n = |V|$ Knoten und $m = |E|$ Kanten. Wie viele Knoten und Kanten besitzt ein Spannbaum von G?

Knoten: n Kanten: $n - 1$

1 P (c) Zeichnen Sie einen zusammenhängenden Graphen mit 7 Knoten, der ein maximales Matching der Grösse 2 besitzt.

Hier sind verschiedene Lösungen möglich, zum Beispiel



3 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 Postorder-Traversierung eines binären Suchbaums erzeugt eine absteigend sortierte Liste der gespeicherten Schlüssel.	☐ Wahr	☒ Falsch
Es gibt einen AVL-Baum, bei dem mehr als die Hälfte seiner inneren Knoten nicht perfekt balanciert sind (d.h., einen Balancierungsfaktor ungleich 0 haben).	☒ Wahr	☐ Falsch
Zu jedem AVL-Baum gibt es eine Einfügereihenfolge, die zu genau diesem Baum führt, ohne dass Rotationen stattfinden.	☒ Wahr	☐ Falsch
Das Einfügen eines neuen Schlüssels in einen AVL-Baum führt selbst im schlimmsten Fall zu nur einer (einfachen oder Doppel-)Rotation.	☒ Wahr	☐ Falsch
Das Einfügen eines neuen Elements in einen Fibonacci-Heap erfordert im schlimmsten Fall nur konstante Zeit.	☒ Wahr	☐ Falsch
Sortieren durch Einfügen kann als stabiles Sortierverfahren implementiert werden.	☒ Wahr	☐ Falsch

1 P (e) Gegeben sei ein Segmentbaum zur Verwaltung ganzzahliger Intervalle mit Intervallgrenzen aus $\{1, \ldots, n+1\}$ für eine Zweierpotenz $n = 2^k$, $k \in \mathbb{N}$. Geben Sie die Anzahl Knoten an, die eine Aufspiessanfrage für einen Punkt $i \in \{1, \ldots, n+1\}$ maximal besucht.

Maximale Anzahl: $2k + 1$

1 P (f) Fügen Sie die Schlüssel 12, 19, 6, 15, 13, 2, 28 in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie Double Hashing mit der Hashfunktion $h(k) = k \bmod 11$, und benutzen Sie $h'(k) = 1 + (k \bmod 9)$ zur Sondierung.

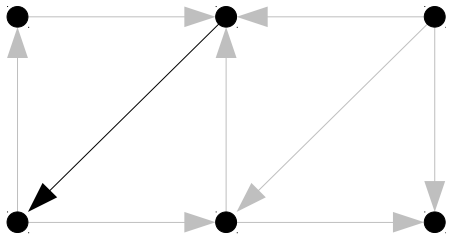
Bei Sondierung nach links $(h(k) - jh'(k) \bmod 11)$:

0	1	2	3	4	5	6	7	8	9	10
28	12	13		15		6		19		2

Bei Sondierung nach rechts $(h(k) + jh'(k) \bmod 11)$:

0	1	2	3	4	5	6	7	8	9	10
	12	13		15	2	6		19		28

1 P (g) Markieren Sie in untenstehendem Graphen $G = (V, E)$ eine *kleinstmögliche* Menge S an Kanten, sodass der Graph $G' = (V, E \backslash S)$ topologisch sortiert werden kann.



Aufgabe 2.

1 P (a) 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.

- $\frac{3^n}{n^3}$
- 10^{10}
- $\log(n^n)$
- $n!$
- $n^3 + n$
- $\sqrt{3^n}$

Lösung: Es gilt $\log(n^n) = n \log n$. Wegen $\lim_{n\to\infty} \frac{\sqrt{3^n}}{3^n/n^3} = \lim_{n\to\infty} \frac{n^3(3^n)^{1/2}}{3^n} = \lim_{n\to\infty} \frac{n^3}{3^{n/2}} = 0$ ist $\sqrt{3^n} \in o(3^n/n^3)$. Nach der Stirling-Formel ist $n! \geq \left(\frac{n}{e}\right)^n \geq 3^n$ für $n \geq 9$, also ist $\frac{3^n}{n^3} \in \mathcal{O}(n!)$. Die einzige gültige Reihenfolge ist daher

$$10^{10}, \log(n^n), n^3 + n, \sqrt{3^n}, \frac{3^n}{n^3}, n!$$

4 P (b) Gegeben ist die folgende Rekursionsgleichung:

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

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

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

$$\begin{aligned} T(n) &= 5T(n/5) + n + 4 = 5(5T(n/5^2) + n/5 + 4) + n + 4 \\ &= 5^2T(n/5^2) + 2n + 4(5^0 + 5^1) = 5^2(5T(n/5^3) + n/5^2 + 4) + 2n + 4(5^0 + 5^1) \\ &= 5^3T(n/5^3) + 3n + 4(5^0 + 5^1 + 5^2) = \ldots = 5^kT(n/5^k) + kn + 4\sum_{i=0}^{k-1} 5^i \\ &= 5^{\log_5(n)}T(n/5^{\log_5 n}) + n\log_5 n + 4\sum_{i=0}^{\log_5 n - 1} 5^i = n + n\log_5 n + n - 1 = n\log_5 n + 2n - 1. \end{aligned}$$

Wir beweisen nun unsere Annahme durch vollständige Induktion über k.

Induktionsverankerung ($k = 0$): Es gilt $T(5^0) = T(1) = 1 = 1\log_5 + 2 \cdot 1 - 1$.

Induktionsannahme: Für ein $k \in \mathbb{N}_0$ sei $T(5^k) = 5^k k + 2 \cdot 5^k - 1$.

Induktionsschritt ($k \to k + 1$):

$$\begin{aligned} T(5^{k+1}) &= 5T(5^k) + 5^{k+1} + 4 \overset{\text{Ind.-Ann.}}{=} 5(5^k k + 2 \cdot 5^k - 1) + 5^{k+1} + 4 \\ &= 5^{k+1}k + 5^{k+1} + 2 \cdot 5^{k+1} - 5 + 4 = 5^{k+1}(k+1) + 2 \cdot 5^{k+1} - 1 = n\log_5 n + 2n - 1. \end{aligned}$$

1 P (c) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für das folgende Programmstück (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = 1; i <= n; i += 10) {
2         for(int j = 1; j <= n/2; j += 4)
3                 ;
4 }
```

Lösung: Die innere Schleife wird höchstens $\lceil n/2 \rceil$ Mal durchlaufen, die äussere höchstens n Mal. Damit ist die Gesamtlaufzeit höchstens $\mathcal{O}(n^2)$. Andererseits wird die innere Schleife mindestens $\lfloor n/8 \rfloor$ Mal durchlaufen, die äussere mindestens $\lfloor n/10 \rfloor$ Mal. Damit ist die Laufzeit durch $\Omega(n^2)$ nach unten beschränkt, und die Gesamtlaufzeit beträgt $\Theta(n^2)$.

1 P (d) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für das folgende Programmstück (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

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

Lösung: Die Schleife in den Zeilen 2–3 wird genau $\lceil \sqrt{n} \rceil$ Mal durchlaufen, die Schleife in den Zeilen 4–5 genau $\lfloor \log_2 n \rfloor$ Mal. Wegen $\lfloor \log_2 n \rfloor \in o(\sqrt{n})$ ist die Laufzeit der Schritte 2–5 asymptotisch $\Theta(\sqrt{n})$. Da die äussere Schleife genau n Mal durchlaufen wird, beträgt die Gesamtlaufzeit $\Theta(n\sqrt{n})$.

1 P (e) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für das folgende Programmstück (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen, und dürfen weiterhin der Einfachheit halber annehmen, dass n eine Potenz von 2 ist.

```
1 int f(int n) {
2         if(n <= 1) { return 1; }
3         else { return f(n/2)+f(n/2); }
4 }
```

Lösung: Ohne die rekursiven Aufrufe in Schritt 3 ist die Laufzeit konstant, d.h. durch eine Konstante C nach oben beschränkt. Beziehen wir die rekursiven Aufrufe ein, dann beträgt die Laufzeit maximal

$$T(n) = C + 2T(n/2), T(1) = C.$$

Durch Teleskopieren erhalten wir $T(n) = C(2n - 1) \in \Theta(n)$. Die Rechnung und der Beweis verlaufen analog zu b).

Aufgabe 3. In dieser Aufgabe geht es um die Berechnung des Umrisses von Skylines. Die Skyline einer Stadt (der Umriss) ergibt sich wie der Schatten einer Menge rechteckiger Hochhäuser, die allesamt auf dem Boden aufsitzen und als orthogonale Recktecke wie im Bild links gegeben sind. Der Umriss ist das orthogonale Polygon, das die Vereinigung aller solchen gegebenen Rechtecke beschreibt, wie im Bild rechts zu sehen. Er kann durch die Menge seiner Eckpunkte $p_1, \ldots, p_k$ beschrieben werden.

7 P a) Entwerfen Sie einen möglichst effizienten Scanline-Algorithmus, der als Eingabe eine Menge von n orthogonalen Rechtecken erhält, und die Eckpunkte $p_1, \ldots, p_k$ des Umrisses berechnet.

Lösung:

Haltepunkte: Wir benutzen eine vertikale Scanline, die von links nach rechts läuft. Haltepunkte sind die x-Koordinaten der Rechtecke.

Scanline-Datenstruktur: Als Datenstruktur benutzen wir einen AVL-Baum. Dieser speichert die Höhen aller Rechtecke, deren Schnitt mit der Scanline nicht leer ist. Zusätzlich benutzen wir eine lineare Liste L zur Speicherung der Punkte des Umrisses.

Aktualisierung: Wir unterscheiden zwei Fälle.

1. Fall: Ein neues Rechteck startet. Wir fügen die Höhe dieses Rechtecks in die Scanline-Datenstruktur ein. Ist sie grösser als die bisherige maximale Höhe, dann fügen wir zwei Punkte zum Umriss hinzu. Sie haben die x-Koordinate der Scanline und die alte bzw. die neue maximale Höhe als y-Koordinate.

Links: Ein neues Rechteck startet, aber die maximale Höhe bleibt unverändert; folglich muss nichts ausgegeben werden. Rechts: Ein neues Rechteck startet, und die maximale Höhe ändert sich; folglich müssen q_1 und q_2 der Liste L hinzugefügt werden.

2. Fall: Ein Rechteck endet. Wir gehen analog zum vorigen Fall vor und entfernen das Rechteck aus der Scanline-Datenstruktur. Wird dadurch die maximale Höhe verringert, dann fügen wir die entsprechenden Punkte zu L hinzu.

Auslesen der Lösung: Die Menge der Punkte des Umrisses ist in der Liste L gespeichert.

1 P b) Geben Sie die Laufzeit Ihres in a) entwickelten Algorithmus an und begründen Sie Ihre Antwort.

Lösung: Das Sortieren der Haltepunkte kann in Zeit $\mathcal{O}(n \log n)$ durchgeführt werden. An jedem der $\Theta(n)$ Haltepunkte muss entweder eine Zahl in die Scanline-Datenstruktur eingefügt oder aus der Datenstruktur entfernt werden. Bei Verwendung eines AVL-Baums kann dies in Zeit

$\mathcal{O}(\log n)$ realisiert werden. Das Einfügen in die Liste L der Punkte des Umrisses kann in Zeit $\mathcal{O}(1)$ realisiert werden, wenn an den Listenanfang eingefügt wird (oder ein Zeiger auf das Listenende verwaltet wird). Damit fällt an jedem Haltepunkt nur Zeit $\mathcal{O}(\log n)$ an, und die Gesamtlaufzeit beträgt $\mathcal{O}(n \log n)$.

Aufgabe 4. Ein Versandhandel bietet die Artikeltypen $\{1, \ldots, n\}$ an. Wir bestellen von jedem Artikeltyp einige Exemplare, und beauftragen für den Transport $m \leq n$ Kuriere. Allerdings transportiert nicht jeder Kurier jeden Artikeltypen (z.B. kann ein Fahrradkurier keinen Fernseher ausliefern). Für jeden Kurier $j \in \{1, \ldots, m\}$ sei $T(j) \subseteq \{1, \ldots, n\}$ die Menge der Artikeltypen, die er transportieren kann. Wir möchten entscheiden, ob die Artikel(exemplare) so auf die Kuriere aufgeteilt werden können, dass alle Artikel befördert werden und jeder Kurier nur einmal fahren muss.

4 P a) Angenommen, von jedem Artikeltyp $i \in \{1, \ldots, n\}$ wird genau ein Exemplar bestellt, und jeder Kurier $j \in \{1, \ldots, m\}$ soll pro Fahrt nur ein einziges Artikelexemplar transportieren. Modellieren Sie das o.g. Problem als Flussproblem. Beschreiben Sie dazu die Konstruktion eines geeigneten Netzwerks $G = (V, E, c)$ mit der Knotenmenge V sowie der Kantenmenge E, und welche Kapazitäten den Kanten zugewiesen werden. Wie kann aus dem Wert eines maximalen Flusses abgelesen werden, ob eine geeignete Verteilung existiert oder nicht?

Lösung: Dies entspricht dem Matching-Problem. Für jeden Artikeltyp $i \in \{1, \ldots, n\}$ erzeugen wir einen Knoten A_i, für jeden Kurier $j \in \{1, \ldots, m\}$ erzeugen wir einen Knoten K_j. Zusätzlich werden eine Quelle s und eine Senke t erzeugt. Für jeden Artikeltyp $i \in \{1, \ldots, n\}$ fügen wir die Kante (s, A_i) hinzu, und für jeden Kurier $j \in \{1, \ldots, m\}$ die Kante (K_j, t). Zusätzlich wird für jedes $j \in \{1, \ldots, m\}$ und jedes $i \in T(j)$ eine Kante (A_i, K_j) eingefügt. Allen Kanten wird die Kapazität 1 zugewiesen. Wir erzeugen also ein Netzwerk $N = (V, E, c)$ mit

$$\begin{aligned} V &:= \{A_i \mid i \in \{1, \ldots, n\}\} \cup \{K_j \mid j \in \{1, \ldots, m\}\} \cup \{s\} \cup \{t\}, \\ E &:= \{(s, A_i) \mid i \in \{1, \ldots, n\}\} \cup \{(A_i, K_j) \mid j \in \{1, \ldots, m\}, i \in T(j)\} \cup \\ &\qquad \{(K_j, t) \mid j \in \{1, \ldots, m\}\}, \\ c(e) &:= 1 \text{ für alle } e \in E. \end{aligned}$$

Eine Aufteilung der Artikelexemplare auf die Kuriere, sodass jeder Kurier nur einmal fahren muss, existiert genau dann, wenn ein maximaler Fluss mit Wert n existiert.

3 P b) Nun wird von jedem Artikeltyp $i \in \{1, \ldots, n\}$ nicht nur ein, sondern $f(i) \in \mathbb{N}_0$ Exemplare bestellt, und jeder Kurier $j \in \{1, \ldots, m\}$ kann pro Fahrt nicht nur einen, sondern bis zu $k(j)$ Artikelexemplare insgesamt befördern. Beschreiben Sie, wie Sie die Lösung aus a) modifizieren können, um dieses allgemeinere Problem zu lösen. Wie gross muss nun der Wert eines maximalen Flusses sein, damit jeder Kurier nur einmal fahren muss?

Lösung: Für allgemeine $f(i)$ und $k(j)$ kann die Konstruktion aus a) weiterhin verwendet werden, nur die Kapazitäten müssen angepasst werden. Wir setzen

$$\begin{aligned} c((s, A_i)) &:= f(i) \text{ für alle } i \in \{1, \ldots, n\} \\ c((A_i, K_j)) &:= f(i) \text{ für alle } j \in \{1, \ldots, m\}, i \in T(j) \\ c((K_j, t)) &:= k(j) \text{ für alle } j \in \{1, \ldots, m\} \end{aligned}$$

Eine geeignete Verteilung existiert genau dann, wenn im obigen modifizierten Netzwerk ein maximaler Fluss mit Wert $\sum_{i=1}^{n} f(i)$ existiert (diese Summe entspricht genau der Anzahl der zu transportierenden Artikelexemplare).

2 P c) Nennen Sie einen Algorithmus, der das Problem aus b) möglichst effizient löst, und geben Sie die Laufzeit im schlimmsten Fall in Abhängigkeit von der Anzahl der Artikeltypen n und der Anzahl der Kuriere m an. Begründen Sie Ihre Antwort.

Lösung: Das Netzwerk besitzt $2 + n + m \in \Theta(n)$ viele Knoten. Da $|T(j)| \leq n$ ist, ist die Anzahl der Kanten durch $n + m + nm \in \Theta(nm)$ nach oben beschränkt. Der beste (in der Vorlesung vorgestellte) Algorithmus für diese Eingabe ist der Algorithmus von Dinic. Er löst das Problem in Zeit $\mathcal{O}(|V|^2|E|) = \mathcal{O}(n^2(mn)) = \mathcal{O}(mn^3)$.

3 P d) Angenommen, das Flussproblem aus b) wurde bereits gelöst, d.h. zu einem maximalen Fluss ϕ kennen Sie den Fluss ϕ_e auf jeder Kante e, und angenommen, eine geeignete Verteilung wie oben beschrieben existiert tatsächlich. Beschreiben Sie detailliert einen Algorithmus, der aus den ϕ_e eine solche Verteilung berechnet. Konkret soll eine Menge M berechnet werden; diese enthält ein Tripel (j, i, k) genau dann, wenn der Kurier j genau k Exemplare vom Artikel i transportiert. Für das obige Beispiel ist $M = \{(1, 3, 4), (2, 1, 2), (2, 3, 1), (3, 2, 2)\}$. Welche Laufzeit hat Ihr Verfahren, wenn auf jedes ϕ_e in Zeit $\Theta(1)$ zugegriffen werden kann?

Lösung: Wir iterieren über alle Kanten e des Netzwerks, die von einem Artikel A_i zu einem Kurier K_j verlaufen. Ist der Fluss auf e grösser als 0 (d.h., $\phi_e > 0$), dann geben wir das Tripel (j, i, ϕ_e) aus. Die Laufzeit ist linear in der Anzahl der Kanten, also durch $\mathcal{O}(mn)$ nach oben beschränkt.

Aufgabe 5. In der Schweiz existieren die folgenden Münzwerte: 5 Rappen, 10 Rappen, 20 Rappen, 50 Rappen, 1 Franken, 2 Franken, 5 Franken. Damit lassen sich alle Geldbeträge darstellen, deren Wert in Rappen durch 5 teilbar ist. In dieser Aufgabe befassen wir uns mit der Anzahl verschiedener Möglichkeiten, einen gegebenen Betrag durch irgendeine Menge von Münzen zu erreichen. Beispielsweise kann der Betrag von 20 Rappen auf vier verschiedene Arten erreicht werden:

1) $5 + 5 + 5 + 5$ Rappen,
2) $5 + 5 + 10$ Rappen,
3) $10 + 10$ Rappen,
4) 20 Rappen.

Beachten Sie, dass es dabei auf die Reihenfolge der Münzen nicht ankommt; $5 + 5 + 10$ Rappen ist also dieselbe Menge von Münzen wie $5 + 10 + 5$ Rappen. Für diese Aufgabe gehen wir von der allgemeinen Annahme aus, dass n Münzen mit den Werten $M_1, \ldots, M_n \in \mathbb{N}$ (in der gleichen Einheit, z.B. Rappen) gegeben sind. O.B.d.A. seien diese aufsteigend geordnet und paarweise verschieden, d.h. $M_1 < M_2 < \cdots < M_n$. Für den Schweizer Franken z.B. hätten wir die Münzwerte $M_1 = 5$, $M_2 = 10$, $M_3 = 20$, $M_4 = 50$, $M_5 = 100$, $M_6 = 200$, $M_7 = 500$ (jeweils in Rappen ausgedrückt).

8 P a) Gegeben sei ein ganzzahliger Betrag $B \in \mathbb{N}$ in der gleichen Einheit wie die Münzen $M_1, \ldots, M_n$. Geben Sie einen möglichst effizienten Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die Anzahl der Möglichkeiten, den Betrag B mit den Münzen $M_1, \ldots, M_n$ darzustellen, berechnet.

Lösung:

Definition der DP-Tabelle: Wir verwenden eine $(n+1) \times (B+1)$-Tabelle T. Für $i \in \{0, \ldots, n\}$ und $b \in \{0, \ldots, B\}$ sei $T[i, b]$ die Anzahl der Möglichkeiten, den Betrag b mit den Münzen $1, \ldots, i$ darzustellen. Dabei muss nicht jede der Münzen $1, \ldots, i$ benutzt werden; zur Darstellung dürfen lediglich keine anderen als die Münzen $1, \ldots, i$ benutzt werden.

Berechnung eines Eintrags: Wir unterscheiden drei Fälle.

1. Fall: $b = 0$. Für jedes $i \in \{0, \ldots, n\}$ gibt es genau eine Möglichkeit, den Betrag 0 mit den Münzen $1, \ldots, i$ darzustellen (nämlich die, keine Münze zu verwenden). Wir setzen also $T[i, 0] = 1$ für alle $i \in \{0, \ldots, n\}$.

2. Fall: $i = 0$. Unter Verwendung von keiner Münze kann kein Betrag $b > 0$ dargestellt werden. Wir setzen also $T[0, b] = 0$ für alle $b \in \{1, \ldots, B\}$.

3. Fall: $i \in \{1, \ldots, n\}$, $b \in \{1, \ldots, B\}$. Die Anzahl der Möglichkeiten, den Betrag b mit den Münzen $1, \ldots, i$ darzustellen, ist die Summe aus

- der Anzahl der Möglichkeiten, den Betrag b mit den Münzen $1, \ldots, i-1$ darzustellen (in diesem Fall wird keine Münze mit Wert M_i benutzt), und
- der Anzahl der Möglichkeiten, den Betrag $b - M_i$ mit den Münzen $1, \ldots, i$ darzustellen (in diesem Fall wird die Münze M_i mindestens einmal benutzt, darf aber zur Darstellung des Betrags $b - M_i$ erneut benutzt werden).

Folglich wird $T[i, b] = T[i-1, b] + T[i, b - M_i]$ gesetzt.

Berechnungsreihenfolge: Wir berechnen die Einträge $T[i, b]$ für aufsteigende $i \in \{0, \ldots, n\}$. Für gleiche i berechnen wir die Einträge für aufsteigendes $b \in \{0, \ldots, B\}$.

Auslesen der Lösung: Die gesuchte Anzahl findet sich im Eintrag $T[n, B]$.

2 P b) Geben Sie die Laufzeit der Lösung aus Aufgabenteil a) an und begründen Sie Ihre Antwort. Ist die Laufzeit polynomiell?

Lösung: Die Zeit zur Berechnung eines Eintrags ist $\Theta(1)$. Da die Tabelle $\Theta(nB)$ Einträge besitzt, beträgt die Gesamtlaufzeit $\Theta(nB)$. Sie ist pseudopolynomiell, weil B Teil der Eingabe ist, aber durch nur $\lceil \log(B + 1) \rceil$ Bits dargestellt werden kann.

3 P c) Beschreiben Sie detailliert, wie durch Rückverfolgung in der Lösungstabelle gemäss b) *eine* mögliche Darstellung des Betrags B gefunden werden kann, sofern überhaupt eine existiert. Geben Sie auch die Laufzeit dieses Algorithmus an.

Lösung: Wir setzen $i = n$ und $b = B$ und starten die Rückverfolgung im Eintrag $T[n, B]$. Hat dieser den Wert 0, dann ist der Betrag B nicht mit den Münzen $1, \ldots, n$ darstellbar. Ansonsten setzen wir die Rückverfolgung entweder bei $T[i - 1, b]$ oder bei $T[i, b - M_i]$ fort, je nachdem, welcher dieser Einträge ungleich 0 ist (sind beide ungleich 0, dann kann die Rückverfolgung bei einem beliebigen der beiden Einträge fortgesetzt werden). Wird die Rückverfolgung bei $T[i, b - M_i]$ fortgesetzt, dann wird zusätzlich die Zahl i ausgegeben. Auf diese Weise wird die Rückverfolgung fortgesetzt, bis ein Eintrag $T[i, b]$ mit $b = 0$ erreicht wird. Da in jedem Schritt entweder i oder b um mindestens 1 verringert werden, beträgt die Laufzeit $\Theta(n + B)$.