
Lösungshinweise

zu den Übungsaufgaben

Wir stellen hier Lösungshinweise zu ausgesuchten Übungsaufgaben des Buches *Grundlegende Algorithmen* (ISBN 978-3-528-13140-1) zusammen. Dabei werden hier **keine** vollständigen Lösungen angegeben, sondern lediglich Hinweise zur Lösung oder explizite Lösungen zum Vergleich (je nachdem, was für die einzelne Aufgabe sinnvoller ist).

Im Folgenden bedeutet —, dass (noch) kein Lösungshinweis zur Verfügung steht.

Lösungshinweise zu Kapitel 1

Lösungshinweis 1.1 Beweis durch vollständige Induktion.

Lösungshinweis 1.2 Für a) wähle Algorithmus A, für b) wähle Algorithmus B.

Lösungshinweis 1.3 Einige Beispiele:

- Eingabegröße für T_1 bei 1M: etwa 5,184,000;
- Eingabegröße für T_2 bei 1d: etwa 103,708;
- Eingabegröße für T_3 bei 1h: etwa 1,897;
- Eingabegröße für T_4 bei 1m: etwa 271;
- Eingabegröße für T_5 bei 1s: etwa 15.

Lösungshinweis 1.4 Zu berechnen ist $T_i(10n)/T_i(n)$. Damit gilt für T_1 : Faktor 10; T_2 : Faktor fast 10; T_3 : Faktor etwa 3.3 (Wurzel 10); T_4 : Faktor etwa 2.15 (dritte Wurzel aus 10) T_5 : wird um etwa 2.1 additiv größer

Lösungshinweis 1.5 —

Lösungshinweis 1.6 —

Lösungshinweis 1.7 Die RAM-Programme der Aufgaben 1.5 mit 1.9 sind im Prinzip leicht, aber technisch aufwendig. Für 1.7 geben wir exemplarisch ein RAM-Programm an (es gibt auch andere).

Wir lösen die Endrekursion des iterierten Quadrierens auf. Da wir in der RAM die Bits nur in der Reihenfolge vom niederwertigsten (LSB) zum höchstwertigsten (MSB) erhalten, für das iterierte Quadrieren die Reihenfolgen aber vom MSB zum LSB benötigen, werden die Bits in den Registern 4 bis m zwischengespeichert und im Register 4 wird im Wesentlichen die Nummer des höchsten verwendeten Registers gespeichert.

```
01: LOAD  #4
02: STORE 3
03: ODD  1          // momentanes LSB ermitteln
04: STORE @3        // und ablegen
05: LOAD  3          // c(3) erhöhen
06: ADD  1
07: STORE 3
08: SHIFT 1          // x halbieren
09: STORE 1
10: IF =0 GOTO 12    // Ist c(1)==0, dann haben wir alle Bits (x==c(1))
11: GOTO  3
12: LOAD  3
13: SUB   #4
14: IF =0 GOTO 26    // Ist c(3)=4, dann ist der Bitstring abgearbeitet
15: ADD   #3
16: STORE 3          // letztendlich c(3) dekrementiert
17: LOAD  1          // Eine Bitstelle, also Ergebnis verdoppeln
18: ADD  1          // (beim ersten Durchlauf geschieht hier nichts
19: STORE 1          // 0+0=0)
20: ODD  @3          // War das Bit gleich 1
21: IF =0 GOTO 14    // Wenn nein, dann nächstes Bit
22: LOAD  1          // Wenn ja, dann noch c(2)==y drauf addieren
23: ADD  2
24: STORE 1
25: GOTO 12
26: END
```

Lösungshinweis 1.8 —

Lösungshinweis 1.9 —

Lösungshinweis 1.10 a) n Stellen.

b) Es werden genau dann die i -letzten Stellen inspiziert, wenn die Binärzahl mit einer 1 gefolgt von $i - 1$ Nullen endet. Damit ist die Wahrscheinlichkeit, dass genau i Stellen inspiziert werden gerade 2^{-i} . Summation von $i = 1$ bis $i = n$ über

$i \cdot 2^{-i}$ liefert

$$\sum_{i=1}^n \frac{i}{2^i} = 2 - 1/2^n.$$

Lösungshinweis 1.11 Betrachte $(a \cdot 2^{2k+1} + c) \cdot (b \cdot 2^{2k+1} + d)$. Die ersten $2k$ Bits liefern $a \cdot b$, die letzten $2k$ Bits $c \cdot d$.

Lösungshinweis 1.12 Es gilt:

- $f_8 = o(f_3)$,
- $f_3 = o(f_4)$,
- $f_4 = o(f_1)$,
- $f_1 = o(f_9)$,
- $f_9 = o(f_{10})$,
- $f_{10} = o(f_2)$,
- $f_2 = o(f_6)$ und
- $f_6 = o(f_5)$.

Weiter ist $f_7 = o(f_6)$. f_7 ist mit den anderen Funktionen (außer f_5 und f_6) unvergleichbar.

Lösungshinweis 1.13 Beweis mit vollständiger Induktion. Alternativ betrachte man die mit $(x-1)$ multiplizierte Gleichung.

Lösungshinweis 1.14 Ja, es gilt.

Lösungshinweis 1.15 Die Implikation von links nach rechts gilt, die umgekehrte Implikation nicht.

Lösungshinweis 1.16 Gilt nicht.

Lösungshinweis 1.17 Nachrechnen und mehrfache Anwendung von l'Hospital.

Lösungshinweis 1.18 $f_1 = \Theta(\log(n))$, $f_2 = \Theta(1)$, $f_3 = \Theta(n \log(n))$, $f_4 = \Theta(n)$, $f_5 = \Theta(\log^3(n))$.

Lösungshinweis 1.19 $f(n) = 1 + (-1)^n$ und $g(n) = 1 - (-1)^n$.

Lösungshinweis 1.20 Das c verändert sich in jedem Induktionsschritt und ist somit nicht konstant.

Lösungshinweise zu Kapitel 2

Lösungshinweis 2.1 Nach jedem Vergleich wissen wir von der größeren Zahl, dass sie nicht das Minimum sein kann. Werden weniger als $n - 1$ Vergleiche ausgeführt, so gibt es mindestens zwei Zahlen, die noch für das Minimum in Frage kommen. Wir können beide Werte im Nachhinein geeignet erniedrigen, ohne dass die vorherigen Vergleiche dadurch falsch werden. Je nachdem, wie wir das tun, ist entweder die eine oder die andere das Minimum. Ein beliebiger Algorithmus mit maximal $n - 2$ Vergleichen kann also nicht feststellen, welche der beiden nun wirklich das Minimum ist.

Lösungshinweis 2.2 Erweitere den Schlüssel k des i -ten Datensatzes zu (k, i) und verwende eine lexikographische Ordnung, die auf der ersten Komponente die ursprüngliche Ordnung verwendet.

Lösungshinweis 2.3 Die obere Schranke ist klar, da es maximal $n(n + 1)/2$ Verschiebungen geben kann.

Bei n Elementen ist die Wahrscheinlichkeit, dass das letzte Element Rang k hat gerade $1/n$. Somit ist die erwartete Anzahl an Verschiebungen gerade die Summe von $i = 0$ bis $i = n$ über i/n , was $\sum_{i=0}^n i/n = (n + 1)/2$ ergibt.

Per Induktion lässt sich zeigen, dass dann die erwartete Anzahl von Verschiebungen durch $(n - 1)^2/4$ nach unten beschränkt ist.

Lösungshinweis 2.4 Die Aussage gilt.

Lösungshinweis 2.5 Man kann zeigen: In der i -ten Phase wird das i -te größte Element an die richtige Position gebracht.

Lösungshinweis 2.6 Man mische jeweils zwei der k Folgen zu einer neuen Folge zusammen. Man erhält dann $k/2$ Folgen der Länge $2n/k$. Dies führt man fort, bis nur noch eine Folge übrig ist. In der i -ten Phase werden also je zwei Folgen der Länge $2^{i-1}n/k$ zu einer Folge der Länge $2^i n/k$ zusammengemischt. Dabei gibt es je Phase $k/2^i$ Mischvorgänge. Insgesamt benötigt jede Phase maximal n Vergleiche und da es maximal $\log(k)$ Phasen gibt, maximal $n \log(k)$ Vergleiche:

$$T(n) < k \cdot T(n/k) + n \log(k).$$

Per Induktion kann man zeigen, dass $T(n) = O(n \log(n))$ gilt.

Lösungshinweis 2.7 Mit vollständiger Induktion und Fallunterscheidung, ob n gerade oder ungerade ist. Für den Fall n ungerade ist der Beweis technisch etwas aufwendiger.

Lösungshinweis 2.8 $T(n) = n - 1$.

Lösungshinweis 2.9 Man muss nur zeigen, dass innerhalb eines Levels die Knoten aufsteigend nummeriert werden und dass der linkeste Knoten eines Levels eine um eins größere Nummer als der rechteste Knoten des vorherigen Levels erhält. Dies lässt sich mit Abzählen leicht nachweisen.

Lösungshinweis 2.10 Laufzeit $\Theta(n \log(n))$.

Lösungshinweis 2.11 Für $k = 2$ ist der beschriebene Algorithmus vom Prinzip her derselbe wie Heapsort. Die Details der Implementierung weichen jedoch etwas voneinander ab.

Zu Beginn werden in den Gruppen die Minima bestimmt. Dies benötigt $O(n)$ Vergleiche. Anschließend werden in m Gruppen von jeweils maximal k Elementen die Maxima bestimmt. Da dies etwa n Mal wiederholt wird, ergibt sich eine Laufzeit von $O(nmk) = O(kn \log_k(n))$.

Dies geht etwas effizienter für große k , wenn man nicht nur in jeder Gruppe das Minimum bestimmt, sondern die Elemente als sortiertes Feld verwaltet. Wenn man die Gruppen als Baum interpretiert, hat der Baum insgesamt $O(n/k)$ Knoten (d.h. Gruppen). Das Sortieren aller Gruppen benötigt daher insgesamt

$$O(n/k \cdot k \log(k)) = O(n \log(k)).$$

Das Einfügen eines neuen Elements mittels binärer Suche in eine Gruppe kann in Zeit $O(\log(k))$ geschehen. Somit benötigt die zweite Phase maximal Zeit

$$O(nm \log(k)) = O(n \log_k(n) \log(k)) = O(n \log(n)).$$

Lösungshinweis 2.12 Ersetze in den beiden **do-while**-Schleifen

$$(\text{array}[i] < \text{array}[\text{pivot}]) \quad \text{bzw.} \quad (\text{array}[j] > \text{array}[\text{pivot}])$$

durch

$$(\text{array}[i] \leq \text{array}[\text{pivot}]) \quad \text{bzw.} \quad (\text{array}[j] \geq \text{array}[\text{pivot}]).$$

Lösungshinweis 2.13 Betrachte H_n als Ober- bzw. Untersumme des Integrals über dx/x mit geeigneten Grenzen.

Lösungshinweis 2.14 Man rechnet leicht nach, dass die Höhe des entstehenden Rekursionsbaumes durch $\log(n)/\log(1/(1-\varepsilon)) = O(\log(n))$ beschränkt ist (wobei ε konstant ist). Auf jedem Level benötigen alle Partitionierungen der einzelnen Teilintervalle (die ja eine Partition von $[1 : n]$ sein müssen) maximal Zeit $O(n)$. Somit ist der Zeitbedarf insgesamt $O(n \log(n))$.

Lösungshinweis 2.15 Quicksort: (3,3,1,1,2); wobei das letzte Element als Pivot gewählt wird.

Heapsort: (1,2,2,2); Beachte, dass das Feld bereits die Heap-Bedingung erfüllt.

Lösungshinweis 2.16 Zeige mittels vollständiger Induktion, dass

$$V_{wc}(n) \geq 1/2(n-1)n.$$

Dabei zeigst sich, wie die worst-case Eingabe aussieht.

Lösungshinweis 2.17 Nachrechnen.

Lösungshinweis 2.18 Verfahre analog zu Mergesort und teile die Folge in eine linke und rechte Folge. Bestimme rekursiv die optimalen Teilsequenzen. Die optimale Teilsequenz ergibt sich aus den beiden optimalen Lösungen der Rekursion sowie einer optimalen Folge die über den Schnittpunkt hinweggeht. Letztere kann in linearer Zeit gefunden werden.

Lösungshinweis 2.19 Fleißaufgabe. Mittels linearer Suche ist der resultierende Entscheidungsbaum in der Abbildung auf der nächsten Seite angegeben.

Lösungshinweis 2.20 Fleißaufgabe.

Lösungshinweis 2.21 Verwende einen Entscheidungsbaum: Da es

$$\frac{(n+m)!}{n! \cdot m!}$$

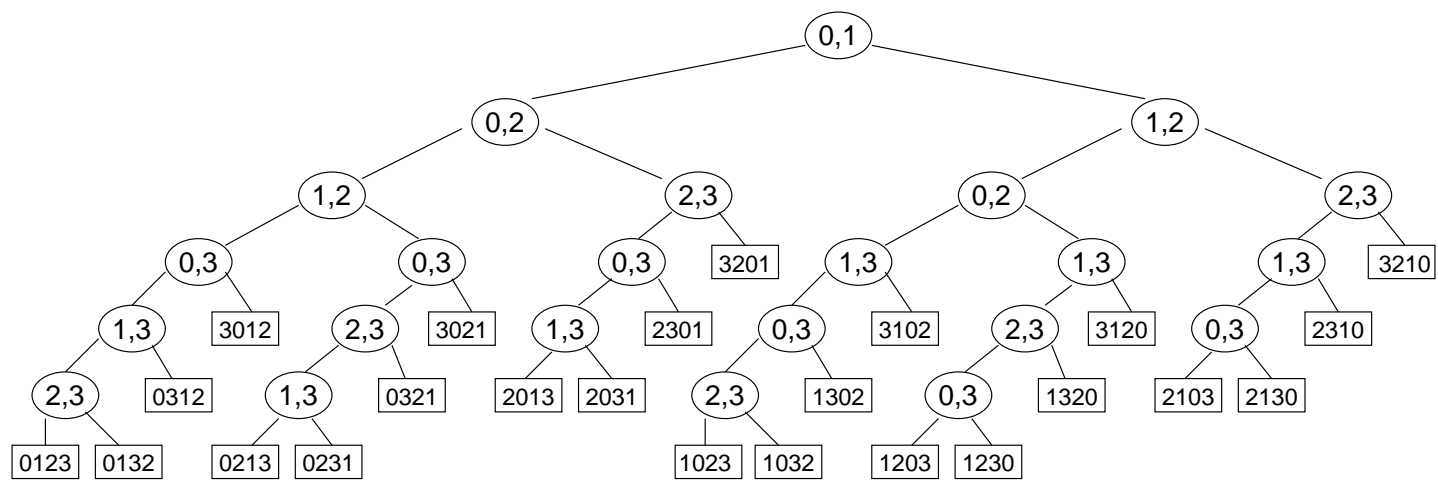
verschiedene Lösungen gibt, ist eine untere Schranke $\log((n+m)!/(n!m!))$.

Lösungshinweise zu Kapitel 3

Lösungshinweis 3.1 Beweis durch Induktion. Dabei darf man die $n-1$ Vergleiche für das Partitionieren nicht vergessen (man könnte das Partitionieren bei der Medianbestimmung miterledigen, dies wurde hier aber nicht vorausgesetzt, und den Term $O(n)$ mit 0 ansetzen).

Lösungshinweis 3.2 Es sind sogar nur 6 Vergleiche ausreihend und notwendig.

- Bilde zunächst zwei Paare von Elementen und vergleiche diese jeweils miteinander (1. und 2. Vergleich).
- Vergleiche die beiden größeren miteinander (3. Vergleich). Das Maximum muss nun Rang 1 oder zwei besitzen und kommt als Median nicht in Frage.
- Nun haben wir ein Paar von Elementen, die bereits miteinander verglichen wurden und zwei Elemente, die jeweils nicht mit den restlichen drei Elementen verglichen wurden. Vergleiche die beiden einzelnen Elemente (4. Vergleich), so dass wir wieder zwei Paare von Elementen erhalten, die miteinander verglichen wurden.



- Vergleiche nun wieder die beiden Maxima der Paare (5. Vergleich). Dieses muss nun den Rang 1 oder zwei besitzen, d.h. wir kennen jetzt die beiden größten Elemente.
- Im letzten und 6. Vergleich vergleichen wir jetzt unter den drei verbliebenen Elementen, die beiden Elemente, die noch keinen Vergleich gegen diese Elemente verloren haben. Das größere davon besitzt den Rang 3 und ist der gesuchte Median.

Die allgemeine untere Schranke aus diesem Kapitel liefert die hier gesuchte untere Schranke.

Lösungshinweis 3.3 Nachrechnen.

Lösungshinweis 3.4 Für $k = 3$ zeigt sich, dass die lineare Laufzeit nicht zu halten ist.

Für ungerade $k \geq 5$ ist die Analyse nicht einfach, da es aufwendig ist, die minimale Anzahl von Vergleichen zur Medianbestimmung zu ermitteln. Eine ausführliche Analyse (aufwendig!) zeigt, dass die Werte zuerst absteigen und dann wieder zunehmen, wobei dies nur für sehr große n gilt.

Lösungshinweis 3.5 Man vergleiche die Mediane der beiden Listen. Wenn beide gleich sind, hat man den Median der Gesamtmenge gefunden. Andernfalls kann man $(m + n)/2$ Elemente als Kandidaten für den Median der Gesamtmenge ausschließen und man wendet das Verfahren rekursiv auf die verbleibenden Liste der Länge $m/2$ bzw. $n/2$ an.

Lösungshinweis 3.6 a) Bestimme zuerst mit Hilfe des Schemas eines fast vollständigen binären Baumes das Maximum ($n - 1$ Vergleiche). Bestimme dann das Maximum aller Verlierer gegen das eigentliche Maximum ($\log(n) - 1$ Vergleiche).
b) —

Lösungshinweis 3.7 Wir diskutieren nur den Fall, dass n gerade ist. Führe zuerst $n/2$ Vergleiche von disjunkten Paaren aus ($n/2$ Vergleiche). Die jeweils größeren bzw. kleineren Elemente sammle in der Menge G bzw. L . Bestimme in G das Maximum und in L das Minimum ($2(n/2 - 1) = n - 2$ Vergleiche).

Wir diskutieren nur den Fall, dass n gerade ist. Wir führen folgende Bezeichnungen für Elemente ein

- \circ bedeutet, das Element war noch an keinem Vergleich beteiligt;
- $+$ bedeutet, das Element hat alle bisherigen Vergleiche gewonnen;
- $-$ bedeutet, das Element hat alle bisherigen Vergleiche verloren;

- $*$ bedeutet, das Element hat bereits Vergleiche sowohl gewonnen als auch verloren.

Elemente $*$ brauchen wir nicht mehr zu berücksichtigen, sie können weder das Maximum noch das Minimum sein. Demnach machen nur noch folgende Vergleiche einen Sinn (beachte die Symmetrie):

- \circ/\circ geht über in $+/-$;
- $\circ/+$ geht schlimmstenfalls in $-/+$ über;
- $\circ/-$ geht schlimmstenfalls in $+/-$ über;
- $+/+$ geht in $+/*$ über;
- $+/-$ geht schlimmstenfalls in $+/-$ über (Vergleich somit sinnlos!);
- $-/-$ geht in $-/*$ über;

Somit ist der Vergleich \circ/\circ der fruchtbarste, er liefert zwei Einschränkungen. Alle anderen (außer $+/-$) liefern im schlimmsten Fall nur eine weitere Einschränkung. Man überlegt sich leicht, dass mindestens $2n - 2$ Einschränkungen benötigt werden, bevor ein beliebiger Algorithmus sowohl das Maximum als auch das Minimum verkünden kann.

Somit kann ein cleverer Algorithmus mit $n/2$ Vergleichen vom Typ \circ/\circ insgesamt maximal n Einschränkungen holen; dann gibt es nur noch Elemente vom Typ $+$ oder $-$. Die benötigten restlichen $n - 2$ Einschränkungen müssen dann mit $n - 2$ Vergleichen vom Typ $+/+$ bzw. $-/-$ gemacht werden. Daher sind mindestens $3n/2 - 2$ Vergleich nötig.

Lösungshinweis 3.8 Konstruiere nebenher einen Graphen, der für jedes Ergebnis $x \leq y$ die Kante (x, y) aufnimmt. Zum Schluss entferne alle ausgehenden Kanten des Medians m (d.h. $(m, *)$). Man kann zeigen, dass die Zusammenhangskomponente mit m genau die k Elemente umfasst, die größer gleich m sind.

Lösungshinweis 3.9 Bestimme den Median der Menge und partitioniere die Menge in zwei Mengen, eine mit Elementen kleiner gleich und eine mit Elementen größer als der Median. Suche rekursiv in der richtigen Menge weiter.

Lösungshinweise zu Kapitel 4

Lösungshinweis 4.1 Nein, da nicht surjektiv.

Lösungshinweis 4.2 Es muss $\text{ggT}(\ell, n) = 1$ gelten, da sonst die verallgemeinerte Hashfunktion nicht surjektiv ist. Primäre und sekundäre Häufung bleiben bestehen.

Lösungshinweis 4.3 —

Lösungshinweis 4.4 Ja, durch sukzessives Einfügen der Elemente in den Suchbaum und anschließendes Traversieren des Suchbaumes in Preorder. Der Zeitbedarf ist dann $O(n \log(n))$.

Lösungshinweis 4.5 Durch Induktion und Ausnutzen der Rekursion

$$|T(h)| \geq 1 + |T(h-1)| + |T(h-2)|.$$

Lösungshinweis 4.6 Die minimale Anzahl für gerades h ist $2^{h/2+2} - 3$, für ungerades h ist $3 \cdot 2^{(h+1)/2} - 3$. Die maximale Anzahl ist $2^{h+1} - 1$.

Lösungshinweis 4.7 —

Lösungshinweis 4.8 Für einen Hinweis lese man das Kapitel nochmal.

Lösungshinweis 4.9 Einsetzen und Nachrechnen.

Lösungshinweis 4.10 Maximale Anzahl: $(b^{h+1} - 1)/(b - 1)$.

Minimale Anzahl: $1 + (a^h - 1)/(a - 1)$ (die Wurzel darf ein Kind haben!).

Lösungshinweis 4.11 —

Lösungshinweis 4.12 Fleißaufgabe.

Lösungshinweise zu Kapitel 5

Lösungshinweis 5.1 Starte an einem beliebigen Knoten und laufe entgegen der Kantenrichtung (dies ist nach Voraussetzung eindeutig). Entweder trifft man auf einen Knoten mit Eingangsgrad 0 oder man findet einen Zyklus (da der Graph endlich ist).

Lösungshinweis 5.2 \Rightarrow : Angenommen, $|E| \neq |V| - 1$. Betrachte unter allen solchen Graphen einen mit kleinster Knotenanzahl. Dann enthält G keinen Knoten mit Grad 1, denn das Entfernen des Knotens samt seiner inzidenten Kante würde zu einem kleineren Graphen mit denselben Eigenschaften führen.

Ist $|E| < |V| - 1$, dann kann G nicht zusammenhängend sein. Sei also $|E| \geq |V|$. Nach Annahme hat jeder Knoten mindestens Grad 2. Starte an einem beliebigen Knoten und durchlaufe den Graphen. Sobald man einen Knoten zum zweiten Mal besucht, hat man einen Kreis gefunden (dies muss geschehen, da der Graph endlich und zusammenhängend ist sowie jeder Knoten mindestens Grad 2 hat). Dies ist ein Widerspruch zur Azyklizität von G .

\Leftarrow : Angenommen G sei azyklisch. Wähle unter allen solchen Graphen einen mit minimaler Knotenanzahl. Dann enthält G keinen Knoten mit Grad 1, denn das Entfernen des Knotens samt seiner inzidenten Kante würde zu einem kleineren Graphen mit denselben Eigenschaften führen. Da die Summe der Knotengrade gleich der doppelten Kantenanzahl ist und jeder Knoten Grad mindestens 2 hat, gilt $|E| \geq |V| > |V| - 1$, was ein Widerspruch zur Voraussetzung ist.

Lösungshinweis 5.3 Induktion über die Anzahl der Blätter.

Lösungshinweis 5.4 Unterteile die $n \times n$ -Matrix in Teilmatrizen der Größe $k \times k$ mit $k = \sqrt{\log(n)}$. Es gibt nur $2^{k^2} = O(n)$ verschiedene $(k \times k)$ -Matrizen mit Einträgen aus $\{0, 1\}$. Diese können vorab in Platz $O(n \log(n))$ gespeichert werden. Jeder echte Block der Adjazenzmatrix wird dann durch einen Verweis auf diese vorberechneten Blöcke repräsentiert und benötigt daher Platz

$$n/k \cdot n/k = O(n^2 / \log(n)).$$

Lösungshinweis 5.5 Es müssen nur die im Text genannten Charakterisierungen der entsprechenden Kanten berücksichtigt werden.

Lösungshinweis 5.6 Eine Vorwärtskante würde man in einem ungerichteten Graphen zuerst andersherum als Rückwärtskante identifizieren.

Eine Querkante würde man in einem ungerichteten Graphen zuerst andersherum als Baumkante identifizieren.

Lösungshinweis 5.7 In gerichteten Graphen entstehen Baumkanten, Querkanten und Rückwärtskanten.

Bei ungerichteten Graphen entstehen Baumkanten und Querkanten.

Lösungshinweis 5.8 \Rightarrow : In einen Eulerkreis wird jeder Knoten genau so oft verlassen, wie er besucht wurde. Somit muss der Knotengrad gerade sein.

\Leftarrow : Starte an einem beliebigen Knoten v und durchlaufe den Graphen, solange es geht. Man kann zeigen, dass man wieder an v endet, da man jeden Knoten den man besucht wieder verlassen kann (da der Knotengrad gerade ist) außer bei v . Hat man einen Eulerkreis konstruiert, so ist man fertig. Sonst gibt es noch unbesuchte Kanten. Entferne den bisher gefundenen Kreis. Im neuen Graph hat jeder Knoten weiterhin geraden Grad. Starte an einem beliebigen Knoten und finde einen neuen Kreis. Zum Schluss muss aus der Menge der gefundenen Kreise ein neuer, einziger Kreis konstruiert werden (dies ist möglich, da G nach Voraussetzung zusammenhängend ist).

Lösungshinweis 5.9 Die Wurzel des Baumes ist der erste Knoten in der Preorder. Wir finden jetzt die Teillisten für die Teilbäume, die an den Kindern der

Wurzel gewurzelt sind. Der zweite Knoten der Preorder-Liste ist die Wurzel des ersten Teilbaumes. In der Postorder-Liste gehören alle Knoten vor diesem Knoten zum ersten Teilbaum. Der auf diese Menge von Knoten in der Preorder-Liste folgende Knoten ist die Wurzel des zweiten Teilbaumes usw. Die Teilbäume selbst werden dann aus den Bruchstücken der Preorder- und Postorder-Liste rekursiv konstruiert.

Lösungshinweis 5.10 Verwende eine rekursive Tiefensuche; Sobald eine Rückkante gefunden wird, ist der Graph kein DAG.

Die topologische Sortierung erhält man, indem man einen globalen Zähler mit $n = |V|$ initialisiert und nach dem Abschluss der rekursiven Tiefensuche an einem Knoten dort den Wert n vergibt und anschließend n um 1 dekrementiert.

Lösungshinweis 5.11 Traversiere den Graphen und füge für jede Kante (u, v) die Kante (v, u) hinzu.

Lösungshinweis 5.12 Sei $T(i, j, k)$ die Anzahl Operationen, die benötigt werden, um $D_{i,j}^k$ zu berechnen. Dabei sei $T(i, j, 0) = 1$. Dann gilt $T(i, j, k) = 3^k$ (Beweis mittel Induktion). Also ist die Gesamtlaufzeit $\Theta(n^2 3^n)$.

Lösungshinweis 5.13 $G = (\{a, b, c\}, \{(a, b, 2), (a, c, 3), (c, b, -2)\})$.

Lösungshinweis 5.14 Sobald ein Wert $D_{i,i}^k$ negativ wird, gibt es einen Kreis negativer Länge.

Lösungshinweis 5.15 Beweis mit Induktion.

Lösungshinweis 5.16 Wir nehmen hier schlingenfreie Graphen an, d.h. es gilt $(v, v) \notin E$. Weiter gilt $E^- = E - \text{sgn}^*(E^*)$, wobei sgn^* alle positiven Elemente auf 1 setzt und die Elemente der Hauptdiagonale auf 0 setzt.

Lösungshinweis 5.17 Analog wie bei normalen Heaps.

Lösungshinweis 5.18 Schlüssel und Name eines Elementes werden im Folgenden als identisch betrachtet:

- `insert(0); insert(n); insert(n - 1); delete_min();`
- `insert(0); insert (n + 1); insert(n - 2); delete_min();`
- `decrease_key(n + 1, 0); insert(n + 1); insert (n - 3); delete_min();`
- ...
- `decrease_key(n + 1, 0); insert(n + 1); insert(n - i); delete_min();`

- ...
- `decrease_key(n+1,0); insert(n+1); insert(1); delete_min();`
- `decrease_key(n+1,0); delete_min();`

Zum Schluss hat man einen Baum der aus einer linearen Liste mit n Elementen besteht, wobei $4n - 2$ Operationen benötigt werden.

Lösungshinweis 5.19 `delete(k)={decrease_key(k,min-1); delete_min();}`

Lösungshinweis 5.20 Betrachte einen minimalen Spannbaum T . Enthält T die Kante e nicht, so ist nichts zu zeigen. Sei also e eine Kante von T , die die beiden Teilbäume T_1 und T_2 verbindet. Verfolge nun den Pfad, der durch den restlichen Kreis induziert wird, beginnend vom Endknoten von e , der in T_1 liegt. Dieser Pfad enthält eine Kante e' , die von T_1 zu T_2 geht. Da e' ebenfalls eine Kante des Kreises mit maximaler Kante e ist, ist das Gewicht von e' höchstens so groß wie von e und somit ist T_1 und T_2 verbunden mit e' ebenfalls ein minimaler Spannbaum.

Lösungshinweis 5.21 In den U.S.A. ist er optimal, in Utopia nicht (betrachte den Wechsel von beispielsweise 30 Einheiten). Wie wäre es bei den Nennwerten $\{1, 5, 25, 100\}$?

Lösungshinweis 5.22 Angenommen, ein minimaler Spannbaum enthält e . Nach dem Entfernen von e aus dem Spannbaum bleiben zwei Bäume übrig. Beim Durchlaufen der restlichen Kanten des gegebenen Kreises trifft man auf mindestens eine Kante e' , die zwischen den beiden Bäumen verläuft. Nach Voraussetzung muss das Gewicht von e' gleich dem von e sein.

Lösungshinweis 5.23 Ein Binomial-Baum k -ter Ordnung ist rekursiv wie folgt definiert. Ein Binomial-Baum 0-ter Ordnung besteht aus nur einem Knoten. Ein Binomial-Baum k -ter Ordnung entsteht aus zwei Binomial-Bäumen $(k-1)$ -ter Ordnung, indem man die Wurzel des einen Baumes zum Kind des anderen Baumes macht.

Ein Binomial-Baum k -ter Ordnung besitzt offensichtlich 2^k Knoten und hat eine Tiefe von k . Man kann weiter zeigen, dass man sich einen Binomial-Baum k -ter Ordnung auch als eine Wurzel vorstellen kann, an dem für jedes $i \in [0 : k-1]$ ein Binomial-Baum i -ter Ordnung als Kind angehängt wird.

Sei im Folgenden $k := \log(n/2)$. Man erzeugt zunächst mit Union-Operationen einen Binomial-Baum k -ter Ordnung (mit $n/2$ Knoten). Hängt man diesen Binomial-Baum (mittels einer Union-Operation) an einen anderen Knoten als Kind an und führt ein Find (mit Pfadkompression) auf das tiefste Blatt in diesem Baum durch, so kann man zeigen, dass im Wesentlichen wieder ein Binomial-Baum

gleicher Ordnung entsteht (an deren Wurzel nur ein weiterer Knoten hängt). Diese Find-Operation verursacht Kosten in Höhe von $\Omega(\log(n))$. Da nun noch $n/2 - 1$ Knoten übrig sind, kann man dies noch $\Omega(n/2)$ mal wiederholen und erhält einen Gesamtaufwand von $\Omega(n \log(n))$.

Lösungshinweise zu Kapitel 6

Lösungshinweis 6.1 Initialisiere eine neue Variable `found=0`; und ersetze im Programm `return i`; durch `found++`; `break`; sowie `return -1`; durch `return found`;

Lösungshinweis 6.2 `border[] = (0, 0, 1, 0, 1, 0, 1, 2, 3, 4)`;

Lösungshinweis 6.3 `S[] = (12, 12, 12, 12, 12, 12, 12, 7, 12, 12, 3, 1)`

Lösungshinweis 6.4 Der Einfachheit halber nehmen wir an, dass n gerade und m ungerade ist: Wähle $t = (ab)^{n/2}$; $s = a(ab)^{(m-1)/2}$.

Lösungshinweis 6.5 Der Suffix-Trie besitzt $\Theta(n^2)$ Knoten.

Lösungshinweis 6.6 Fleißaufgabe.

Lösungshinweis 6.7 Konstruiere den Suffix-Baum für die konkatenierte Zeichenreihe und entferne Teilbäume, die Dollarzeichen enthalten. Alternativ konstruiere einen Suffix-Baum für t_1 . Konstruiere dann den Suffix-Baum für t_i innerhalb des bereits konstruierten Suffix-Baumes.

Lösungshinweis 6.8 Konstruiere zuerst einen verallgemeinerten Suffix-Tree T für $x, y \in \Sigma^*$. Markiere die Blätter mit x oder y , je nach dem, ob der zugehörige Suffix aus x oder y stammt. Der tiefste Knoten (gemäß der String-Tiefe) in T , der in seinem Teilbaum sowohl ein mit x als auch mit y markiertes Blatt besitzt, korrespondiert zum längsten gemeinsamen Teilwort.

Lösungshinweis 6.9 Suche nach v in ww .

Lösungshinweis 6.10 Suche nach dem zweiten Vorkommen von v in vv .

Lösungshinweis 6.11 —

Lösungshinweis 6.12 —

Lösungshinweise zu Kapitel 7

Lösungshinweis 7.1 Seien e und e' zwei neutrale Elemente eines Monoids, dann gilt: $e = ee' = e'$.

Sei a ein Gruppenelement und b sowie c zwei Inverse zu a , d.h. es gilt $ab = e$ sowie $ac = e$ (wobei e das eindeutige neutrale Element ist).

Wir zeigen zunächst, dass $ba = e$ ist (das Monoid ist nicht notwendig abelsch!). Sei dazu d das Inverse zu ba , d.h. $bad = e$. Dann gilt:

$$ba = bae = ba(bad) = b(ab)ad = bead = bad = e.$$

Betrachte jetzt:

$$b = be = b(ac) = (ba)c = ec = c.$$

Lösungshinweis 7.2 Beweis durch Induktion.

Lösungshinweis 7.3 Schreibe die Elemente als Potenzen des erzeugenden Elementes und verwende die vorhergehende Aufgabe.

Lösungshinweis 7.4 \Rightarrow : Wenn $(M', *)$ eine Untergruppe ist, dann ist M' nach Definition auch abgeschlossen.

\Leftarrow : Sei $a \in M'$ und sei $M = \{b_1, \dots, b_k\}$. Betrachte $a * b_1, \dots, a * b_k$. Da dies auch Elemente von M sind und M eine Gruppe ist, müssen alle Elemente paarweise verschieden sein. Es gibt also ein $j \in [1 : k]$ mit $a * b_j = a$, d.h. $b_j = 1$ und somit ein neutrales Element in M' . Weiterhin muss es ein $i \in [1 : k]$ mit $a * b_i = b_j = 1$. Also existiert zu a auch ein Inverses in M' . Da a ein beliebiges Element aus M' war, muss es zu jedem Element aus M' ein Inverses geben.

Lösungshinweis 7.5 Sei z das Inverse zu $0 * a$. Betrachte:

$$0 = 0 * a + z = (0 + 0) * a + z = (0 * a + 0 * a) + z = 0 * a + (0 * a + z) = 0 * a + 0 = 0 * a.$$

Lösungshinweis 7.6 Die Halbringeigenschaften nachrechnen.

Lösungshinweis 7.7 Sei $a \neq 0$ und $b \neq 0$, dann existieren die Inversen $ac = 1$ und $bd = 1$. Dann folgt aus $ab = 0$, dass $abdc = 0dc$. Mit Aufgabe 7.5 folgt dann: $1 = 0$.

Lösungshinweis 7.8 —

Lösungshinweis 7.9 $\{e\}$ und G .

Lösungshinweis 7.10 —

Lösungshinweis 7.11 Verwende den Satz aus Aufgabe 7.4.

Lösungshinweis 7.12 \Rightarrow : Wenn $p \in \mathcal{P}$ eine Primzahl ist, dann ist $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ sowie $\varphi(p) = p - 1$. Somit folgt die Behauptung aus einem Satz von Euler.

\Leftarrow : Sei d ein Teiler von p , d.h. $da = p$ für ein geeignet gewähltes a . Nach der rechten Seite gilt $d^{p-1} = 1 \bmod p$, d.h. $d^{p-1} - 1 = qp$ für ein $q \in \mathbb{N}$. Somit gilt unter Berücksichtigung von $p \geq 2$: $d(d^{p-2} - qa) = 1$. Diese Gleichung über \mathbb{N} kann dann und nur dann gelten, wenn $d = 1$.

Aus der Kontraposition der Richtung von links nach rechts kann so erst einmal kein sinnvoller randomisierter Primzahltest abgeleitet werden, da $\mathbb{Z}_n \setminus \{0\}$ im Allgemeinen keine multiplikative Gruppe ist und über die Anzahl der Zeugen nichts ausgesagt werden kann.

Lösungshinweis 7.13 Beweis durch Induktion.

Lösungshinweis 7.14 —

Lösungshinweis 7.15 Fleißaufgabe.

Die Rekursionsgleichung lautet: $B(n) = \sum_{i=1}^{n-1} B(i) \cdot B(n-i)$. Das Ergebnis lautet: $\frac{1}{n} \cdot \binom{2n-2}{n-1}$.

Alternativmöglichkeit: Stelle eine bijektive Beziehung zwischen geordneten erweiterten binären Bäumen und Klammerausdrücken her.

Lösungshinweis 7.16 Es gilt offensichtlich:

$$\begin{aligned} A(n, m) &= 1 + A(n-1, m) + A(n-1, m-1), \\ A(n, n) &= 0, \\ A(n, 0) &= 0. \end{aligned}$$

Wir setzen jetzt $B(n, m) := 1 + A(n, m)$. Dann gilt

$$B(n, m) = B(n-1, m) + B(n-1, m-1) \quad \text{sowie} \quad B(n, n) = B(n, 0) = 1.$$

Also ist $B(n, m) = \binom{n}{m}$. Damit kennen wir auch A .

Lösungshinweis 7.17 —

Lösungshinweise zu Kapitel 8

Lösungshinweis 8.1 \Rightarrow : Definiere zuerst $\psi(x) := \min \{n \in \mathbb{N} : \varphi(n) = x\}$. Zeige jetzt noch, dass ψ injektiv ist, wenn φ surjektiv ist.

\Leftarrow : Definiere $\varphi(n) := \psi^{-1}(n)$, wenn $\psi^{-1}(n)$ existiert und $\varphi(n) := m$ für ein festes $m \in M$ sonst. Zeige jetzt noch, dass φ surjektiv ist, wenn ψ injektiv ist.

Lösungshinweis 8.2 Wir geben den Beweis für das zweielementiges Alphabet $\Sigma = \{0, 1\}$. Erweiterungen auf andere Alphabete sind nahe liegend. Sei $w \in \Sigma^*$. Interpretiere $1w$ als Binärdarstellung von x und setze $\psi(w) := x$. Nach Aufgabe 8.1 muss nur noch gezeigt werden, dass ψ injektiv ist.

Lösungshinweis 8.3 Die Brüche in \mathbb{Q} lassen sich auch als Paare in \mathbb{N}^2 interpretieren (dabei gibt es mehrere Darstellungen von q aus \mathbb{Q} in \mathbb{N}^2 , was aber hier nicht von Belang ist.)

Nimm an, das \mathbb{R} abzählbar sei und leite einen Widerspruch her.

Lösungshinweis 8.4 a) Man nehme an, es gibt ein $k \in \mathbb{N}$. Man kann dann leicht zeigen, dass dann weder $\psi(2k) \in \varphi(k)$ noch $\psi(2k) \notin \varphi(k)$ gelten kann.

b) Das kann durchaus sein, dann muss k auf jeden Fall ungerade sein.

Lösungshinweis 8.5 Man rechnet leicht nach, dass für $\varphi(n, m) = x$ gilt:

$$n + m = \left\lceil 1/2 + \sqrt{2x + 1/4} \right\rceil.$$

Der Rest ist dann nur Rechnerei.

Lösungshinweis 8.6 Die erste ist keine, die zweite schon (welche!).

Lösungshinweis 8.7 Analog wie bei universellen Registermaschinen.

Lösungshinweis 8.8 Nein. Beispielsweise Reduktion auf das Halteproblem.

Lösungshinweis 8.9 Zum Beispiel durch Erstellen der entsprechenden Wertetabellen.

Lösungshinweis 8.10 Wir nutzen aus, dass $x \Rightarrow y$ äquivalent zu $\bar{x} \vee y$ ist. Damit lassen sich alle Klauseln aus zwei Literalen als Implikationen über zwei Literalen schreiben. Diese Implikationen übersetzen wir wie folgt in einen Graphen. Alle Literale sind Knoten. Eine gerichtete Kante von x nach y existiert genau dann, wenn $x \Rightarrow y$ als Klausel vorkommt.

Wir bestimmen dann alle starken Zusammenhangskomponenten, wobei die Literale in einer solchen starken Zusammenhangskomponente denselben logischen Wert besitzen müssen. Wir konstruieren dann den so genannten reduzierten Graphen, wobei alle Knoten einer starken Zusammenhangskomponente zu einem Superknoten zusammengefasst werden. Eine Kante von einem Superknoten W zu einem anderen Superknoten W' existiert genau dann, wenn es $w \in W$ und $w' \in W'$ gibt, so dass $(w, w') \in E$ eine Kante im ursprünglichen Graphen war. Der reduzierte Graph muss dann ein DAG sein.

Wir müssen jetzt nur noch den Superknoten logische Werte so zuordnen, dass $1 \Rightarrow 0$ nicht vorkommt. Wenn das möglich ist, ist die Formel erfüllbar, sonst nicht.

Lösungshinweis 8.11 Siehe Garey/Johnson.

Lösungshinweis 8.12 P ist nicht in \mathcal{NPO} enthalten, da weder I in \mathcal{P} ist noch μ in polynomieller Zeit berechenbar ist.

Lösungshinweis 8.13 Offensichtlich lässt sich eine potenzielle Lösung leicht in polynomieller Zeit auf ihre Zulässigkeit hin prüfen. Auch die Optimierungsfunktion μ ist in polynomieller Zeit berechenbar.

Lösungshinweis 8.14 Einfach nachrechnen.

Lösungshinweis 8.15 Es wird iterativ eine Partition (X, Y) aufgebaut. Wir betrachten einen Knoten v . Gehen von v mehr Kanten nach X als nach Y , so platzieren wir v in die Menge Y und sonst in X .

Da für jeden Knoten im Auswahlschritt mindestens die Hälfte der Kanten zwischen X und Y verläuft, ist die Approximationsgüte mindestens 2.

Lösungshinweis 8.16 Konstruiere zuerst einen minimalen Spannbaum T für G . Erzeuge dann eine Euler-Tour für T und überspringe doppelt besuchte Knoten. Dies liefert die konstruierte TSP-Tour.

Für die Approximationsgüte überlege man sich, dass das Gewicht von T höchstens so groß ist, wie die Kosten einer optimalen TSP-Tour. Die Kosten der Euler Tour sind doppelt so groß wie die des optimalen Spannbaumes, da jede Kante genau zweimal abgelaufen wird. Also sind die Kosten der Euler-Tour maximal doppelt so hoch wie die einer optimalen TSP-Tour. Beim überspringen von Knoten hilft die metrische Dreiecksungleichung, dass die Kosten nicht größer werden.

Grundlegende Algorithmen

Einführung in den Entwurf und die Analyse effizienter
Algorithmen

Heun, V.

2003, XIV, 370 S., Softcover

ISBN: 978-3-528-13140-1