

Lösungen und Lösungshinweise zum Buch Weicker & Weicker: »Algorithmen und Datenstrukturen«

Kapitel 1

Aufgabe 1.1: Mengenproblem

Die Menge ist $\{3,4,5,7,9\}$ und es sollten insgesamt drei Operationen zu einem Fehler führen.

Aufgabe 1.2: Sortierproblem

$\pi = (4, 2, 1, 6, 3, 5)$

Aufgabe 1.3: Stabiles Sortieren

Genau 1 Permutation sortiert stabil. $n! - 1$ Permutationen sortieren nicht stabil.

Aufgabe 1.4: Ordnungsverträgliches Sortieren

Die Inversionszahl ist 6.

Aufgabe 1.5: Wege und Zyklen

- a) Es gibt 9 knotendisjunkte Zyklen.
- b) Es gibt 7 knotendisjunkten Wege von v_1 nach v_6 .

Aufgabe 1.6: Kürzeste Wege

Die kürzesten Wege sind

v_1, v_4, v_2 v_1, v_3 v_1, v_4 v_1, v_4, v_5 v_1, v_3, v_6

Aufgabe 1.7: Rundreise

Die kürzeste Rundreise hat die Kosten 17.

Aufgabe 1.8: Dynamischer maximaler Fluss

Es können maximal 17 Einheiten von v_1 nach v_5 gebracht werden.

Aufgabe 1.9: Liegenbleibende Daten im Flussproblem

Zum Zeitpunkt $t = 6$ bleiben 6 Einheiten am Knoten v_2 liegen, falls von v_1 bereits alle Einheiten in das Netz geschickt wurden.

Kapitel 2

Aufgabe 2.1: Textsuche

Der Vergleich von zwei Zeichen ist im Beispiel 12 Mal erfolgreich und 7 Mal erfolglos. Das Muster wird zweimal im Text gefunden.

Aufgabe 2.2: Exakte Laufzeit

- a) In LINKERALG finden $n^2 + 1$ Zuweisungen zu *sum* statt. In RECHTERALG sind es $n + 1$ Zuweisungen.
- b) LINKERALG: Für $i = 1$ wird $1 + 2 + \dots + n$ summiert, für $i = 2$ sind es die Summanden $2 + 3 + \dots + n$ etc. D.h. der Faktor n wird n -mal, $n - 1$ entsprechend $(n - 1)$ -mal bis zum Faktor 1, der einmal addiert wird. Dies entspricht genau dem Ergebnis von RECHTERALG: $\sum_{k=1}^n k^2$.
- c) LINKERALG entspricht fast exakt WEITERES-LAUFZEITBEISPIEL (Algorithmus 2.3) – es ist lediglich eine Return-Anweisungen hinzugefügt. Dadurch ist die Laufzeit gemäß Beispiel 2.11: $T_1(n) = 4 + \frac{15}{2} \cdot n + \frac{5}{2} \cdot n^2$. Für RECHTERALG ergibt sich die exakte Laufzeit

$$T_2(n) = 2 + (2 + \sum_{k=1}^n (3 + 3)) = 4 + 6 \cdot n$$

Aufgabe 2.3: Konstante Faktoren

Zu a): 1 Stunde entspricht $3,6 \cdot 10^9$ Mikrosekunden.

Laufzeit	a) $n =$	b)
$5000 \cdot \log_2 n$	$3,95 \cdot 10^{216\,741}$	ab 100
$500 \cdot n$	7 200 000	nie
$50 \cdot n \cdot \log_2 n$	3 428 571	50, ..., 100
$5 \cdot n^2$	8 485	10, ..., 49
2^{n-1}	32	2, ..., 9

Aufgabe 2.4: Beweis der Lemmata

- Zweite Aussage von Lemma 2.22: Analog zum Beweis der ersten Aussage müssen Konstanten c', c'', n'_0 und n''_0 existieren, sodass $f(n) \geq c' \cdot g(n)$ für $n \geq n'_0$ und $g(n) \geq c'' \cdot h(n)$ für $n \geq n''_0$ gilt. Es kann wieder $n'''_0 = \max\{n'_0, n''_0\}$ gewählt werden, womit die gewünschte Gleichung für $c''' = c' \cdot c''$ gilt.
- Dritte Aussage von Lemma 2.22: Laut Definition ist $f(n) \in \Theta(g(n))$ äquivalent zu $f(n) \in O(g(n))$ und $f(n) \in \Omega(g(n))$. Ebenso folgt aus den Voraussetzungen $g(n) \in O(h(n))$ und $g(n) \in \Omega(h(n))$. Nun kann man die ersten beiden Aussagen des Lemmas anwenden, woraus $f(n) \in O(h(n))$ und $f(n) \in \Omega(h(n))$ folgt. Nach der Definition von Θ gilt sofort $f(n) \in \Theta(h(n))$.
- Erste Aussage von Lemma 2.25: Es existieren laut Voraussetzung Konstanten c und n_0 mit $f(n) \leq c \cdot g(n)$ für $n \geq n_0$. Um die Gleichheit der beiden Funktionsmengen $O((f + g)(n))$ und $O(g(n))$ zu zeigen, muss bewiesen werden, dass für jedes $h \in O((f + g)(n))$ auch $h \in O(g(n))$ gilt und umgekehrt. Sei zunächst $h \in O((f + g)(n))$. Dann existieren Konstanten c' und n'_0 , sodass für alle $n \geq n'_0$ gilt: $h(n) \leq c' \cdot (f(n) + g(n))$. Dann gilt aber auch mit obiger Voraussetzung für alle $n \geq \max\{n_0, n'_0\}$:

$$h(n) \leq c' \cdot (f(n) + g(n)) \leq c' \cdot (c \cdot g(n) + g(n)).$$

Damit gilt die Definition von $h \in O(g(n))$ mit $n_0'' = \max\{n_0, n_0'\}$ und $c'' = c' \cdot (c + 1)$. Die Rückrichtung lässt sich als Beweis per Widerspruch beweisen.

- Zweite Aussage von Lemma 2.25: analog zum ersten Teil.
- Dritte Aussage von Lemma 2.25 lässt sich aus den ersten beiden Aussagen ableiten, wenn man berücksichtigt, dass aus $g(n) \in \Theta(f(n))$ folgende Aussagen direkt folgen: $g(n) \in \Omega(f(n))$, $f(n) \in \Omega(g(n))$, $g(n) \in O(f(n))$ und $f(n) \in O(g(n))$.

Aufgabe 2.5: Asymptotische Komplexität

Die Aussagen werden mit der Berechnungsregel in Lemma 2.28 gezeigt. Bei Ω wird auch Lemma 2.19 benutzt.

$$\begin{aligned}
 \text{a) } \lim_{n \rightarrow \infty} \frac{n}{20 \cdot n \cdot \log n} &= \lim_{n \rightarrow \infty} \frac{1}{20 \cdot \log n} = 0 \\
 \text{b) } \lim_{n \rightarrow \infty} \frac{40 \cdot n \cdot \log n}{n^2} &= \lim_{n \rightarrow \infty} \frac{40 \cdot \log n}{n} \stackrel{\text{L'Hôpital}}{=} \lim_{n \rightarrow \infty} \frac{40 \cdot \frac{1}{n}}{1} = \lim_{n \rightarrow \infty} \frac{40}{n} = 0 \\
 \text{c) } \lim_{n \rightarrow \infty} \frac{7 \cdot \log n}{\sqrt{20 \cdot n}} &\stackrel{\text{L'Hôpital}}{=} \lim_{n \rightarrow \infty} \frac{7 \cdot \frac{1}{n}}{\frac{20}{\sqrt{20 \cdot n}}} = \lim_{n \rightarrow \infty} \frac{7 \cdot \sqrt{20 \cdot n}}{20 \cdot n} \stackrel{\text{L'Hôpital}}{=} \lim_{n \rightarrow \infty} \frac{\frac{140}{\sqrt{20 \cdot n}}}{20} \\
 &= \lim_{n \rightarrow \infty} \frac{7}{\sqrt{20 \cdot n}} = 0
 \end{aligned}$$

Für Θ in d) müssen beide Richtungen gezeigt werden.

Aufgabe 2.6: Asymptotische Komplexität und Logarithmen

Der Faktor $\frac{1}{\log_b b'}$ aus $\log_{b'} n = \frac{\log_b n}{\log_b b'}$ kann in der Definition des O in der Konstante c berücksichtigt werden. Die Aussage dieser Übungsaufgabe ist die Begründung dafür, dass im Buch an allen Stellen der asymptotischen Komplexität ein Logarithmus ohne Angabe der Basis benutzt wird.

Aufgabe 2.7: Vergleich von Laufzeiten

Bis einschließlich $n = 15$ ist der Algorithmus B aufgrund der kleineren Konstante schneller als Algorithmus A . Aber bereits bei $n = 20$ ist B mehr als 13 mal so langsam wie A .

Kapitel 3

Aufgabe 3.1: Verkettete Listen vereinigen

VEREINIGEN-LISTEN(Liste ℓ_1 , Liste ℓ_2)

Rückgabewert: nichts; Seiteneffekt: alle Elemente in Liste ℓ_1

- 1 $el \rightarrow \ell_2.anker$
- 2 **while** $el \neq \text{NULL}$
- 3 **do** $\ell_1.anker \rightarrow \ell_2.anker.nächstes$
- 4 $el.nächstes \rightarrow \ell_1.anker$
- 5 $\ell_1.anker \rightarrow el$
- 6 $\ell_1.anker \rightarrow \ell_2.anker$

Die asymptotische Laufzeit ist $O(n_2)$, wenn n_1 Elemente in ℓ_1 und n_2 Elemente in ℓ_2 sind.

Aufgabe 3.2: Bubblesort anwenden

Es werden 49 Vergleiche und 28 Schreibzugriffe durchgeführt. Dabei werden die letzten sieben Vergleiche auf einem bereits sortierten Feld durchgeführt, um festzustellen, dass das Feld sortiert ist (und daher die Repeat-Until-Schleife abbricht).

Aufgabe 3.3: Bubblesort verbessern

BESSERES-BUBBLESORT(Feld A)

Rückgabewert: nichts; Seiteneffekt: A ist sortiert

```

1  for k ← A.länge - 1, ..., 1
2  do  for i ← 1, ..., k
3      do  if A[i].wert > A[i + 1].wert
4          then  VERTAUSCHE(A, i, i + 1)
```

Der Algorithmus prüft nun nicht mehr Elemente, die bereits im hinteren Teil des Feldes an der richtigen Stelle stehen. Die Anzahl der Schlüsselvergleiche beträgt im Worst-Case $\sum_{i=1}^{n-1} i = \frac{(n-1) \cdot n}{2}$. Der ursprüngliche Algorithmus benötigt im Worst-Case $n \cdot (n-1)$ Schlüsselvergleiche. Der verbesserte Algorithmus ist also etwa doppelt so schnell.

Aufgabe 3.4: Dynamische Felder

Für VARIANTE1 wird das Feld dreimal vergrößert (auf die Längen 12, 18 und 27). Es wird jeweils das alte Feld kopiert, was $8 + 12 + 18$ Schreibzugriffe für das Kopieren ergibt. Hinzu kommen die 20 Zugriffe für das Einfügen der Elemente. Das macht insgesamt 58 Schreibzugriffe.

In VARIANTE2 muss das Feld siebenmal vergrößert werden. Da jedes Mal komplett kopiert wird, fallen hierbei $8 + 12 + 18 + 27 + 41 + 62 + 93$ Schreibzugriffe zuzüglich der 10 Zugriffe für das Einfügen der Elemente an. Es ergeben sich 271 Schreibzugriffe.

Aufgabe 3.5: Programmierung: Liste implementieren

In einer einfach programmierten Version bekomme ich die folgenden Meßergebnisse (die natürlich nur relativ zueinander interessant sind).

Anzahl der Elemente	100	1 000	10 000	100 000
rekursives Suchen	46 192 ns	441 193 ns	2 168 285 ns	StackOverflowError
iteratives Suchen	35 068 ns	181 635 ns	1 603 570 ns	1 066 879 ns

Man erkennt deutlich, dass die iterativen Algorithmen um einen erheblichen Faktor schneller sind. Zudem benötigt die Rekursion zusätzlichen Speicherplatz wodurch es eine Obergrenze für die Anzahl der Elemente gibt.

Aufgabe 3.6: Stack mit verketteter Liste

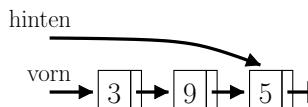
Die verkettete Liste kann direkt den Stack realisieren, indem am *anker* neue Elemente per Push auf den Stack gelegt werden und dort auch wieder per Pop entfernt werden. Auf die Elemente weiter hinten in der Liste wird nicht zugegriffen.

Aufgabe 3.7: Stack mit dynamischem Feld

Die Liste muss von vorn nach hinten gefüllt werden. Neue per Push eingefügte Elemente werden hinten angefügt. Pop entfernt jeweils das letzte Element im Feld. Da der Vergrößerungsmechanismus des dynamischen Felds lediglich als konstanter Faktor in Laufzeiten eingeht, kann auch hier jedes Push und jedes Pop mit konstanter Laufzeit realisiert werden.

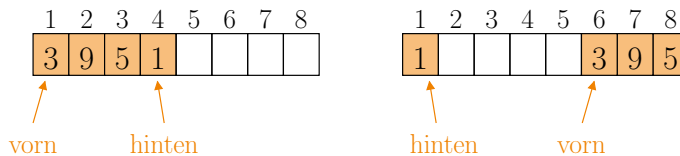
Aufgabe 3.8: Warteschlange mit verketteten Listen

Die Liste wird mit einem zusätzlichen Zeiger auf das letzte Element der Liste ausgestattet. Neue Elemente werden per Push am Ende eingefügt. Pop entfernt das erste Element der Liste.



Aufgabe 3.9: Warteschlange mit dynamischem Feld

Das Feld wird ebenfalls grundsätzlich so organisiert, dass Elemente hinten im Feld angefügt werden und vorn entfernt werden. Da der vorn im Feld frei werdende Platz wieder benutzt und nicht der gesamte Inhalt der Warteschlange umkopiert werden soll, ist es ratsam das Feld weiter von vorn zu befüllen, wenn das Ende der Liste erreicht wird. Vergrößert wird das Feld nur dann, wenn kein freier Platz mehr zur Verfügung steht.



Aufgabe 3.10: Warteschlange mit dynamischem Feld

Es wird ein Feld *farbe* benutzt, in welchem für jeden Knoten die aktuelle Farbe eingetragen wird. Die Anzahl n der Knoten sei wie die Adjazenzinformation *Adj* global bekannt. Ein möglicher Algorithmus ist der folgende.

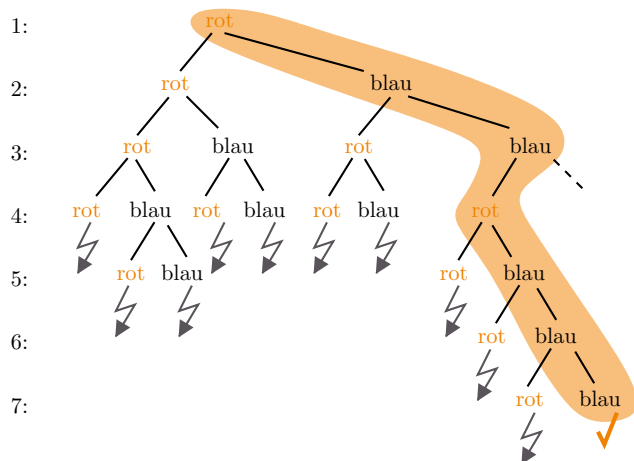
FÄRBE(Knoten i)

Rückgabewert: eine gültige Färbung

```

1  if  $i > n$ 
2  then  $\square$  return farbe
3  else  $\ulcorner$  for  $f \in \{\text{rot, blau}\}$ 
4      do  $\ulcorner$  farbe[ $i$ ]  $\leftarrow f$ 
5          gültig  $\leftarrow$  true
6          for  $k \in \text{Adj}(i)$ 
7              do  $\ulcorner$  if  $k < i$  und farbe[ $i$ ] = farbe[ $k$ ]
8                   $\sqsubset$  then  $\square$  gültig  $\leftarrow$  false
9              if gültig
10          $\sqsubset$   $\sqsubset$  then  $\square$  FÄRBE( $i + 1$ )
  
```

Dann ergibt sich der folgende Entscheidungsbaum.



Aufgabe 3.11: Entwurf eines rekursiven Algorithmus

LISTE-UMDREHEN()

Rückgabewert: –; Seiteneffekt: Liste ist umgedreht

1 $anker \rightarrow \text{LISTE-UMDREHEN-R}(anker, \text{NULL})$

LISTE-UMDREHEN-R(Restliste el , neue Liste el')

Rückgabewert: Zeiger auf die neue Liste

1 **if** $el = \text{NULL}$

2 **then** $\sqsubset$ **return** el'

3 **else** $\sqsupset$ $aktuell \rightarrow el$

4 $el \rightarrow el.nächstes$

5 $aktuell.nächstes \rightarrow el'$

6 $\sqsubset$ **return** $\text{LISTE-UMDREHEN-R}(el, aktuell)$

Aufgabe 3.12: Alle Permutationen aufzählen

Der Algorithmus erzeugt die Permutationen in der folgenden Reihenfolge: $\langle 1,2,3 \rangle$, $\langle 2,1,3 \rangle$, $\langle 3,1,2 \rangle$, $\langle 1,3,2 \rangle$, $\langle 2,3,1 \rangle$ und $\langle 3,2,1 \rangle$.

Kapitel 4

Aufgabe 4.1: Binäre Suche

Betrachtete Elemente bei der Suche nach der 4: 13, 4

Suche nach der 8: 13, 4, 7, 8

Suche nach der 22: 13, 17, 22

Aufgabe 4.2: Suchen in Feld und Liste

Bei 8 Elementen ist die Liste lediglich bei der Suche nach der 1 schneller. Das Feld ist ab dem vierten Element schneller. Die Elemente 2 und 3 werden mit gleich viel Vergleichen gefunden.

Bei 16 Elementen ist die Liste bei den Elementen 1, 2, 3 und 5 schneller. Das Feld bei den Elementen 4 und 6–16.

Aufgabe 4.3: Rekursive binäre Suche

BINÄRE-SUCHE-REK(Schlüssel *gesucht*)

Rückgabewert: gesuchte Daten bzw. Fehler falls nicht enthalten

1 **return** **BINÄRE-SUCHE-R**(*gesucht*, 1, *belegteFelder*)

BINÄRE-SUCHE-R(Schlüssel *gesucht*, Indizes *links*, *rechts*)

Rückgabewert: gesuchte Daten bzw. Fehler falls nicht enthalten

```

1  if links ≤ rechts
2  then  $\ulcorner$  mitte ←  $\lfloor \frac{\textit{links} + \textit{rechts}}{2} \rfloor$ 
3      switch
4      case A[mitte].wert = gesucht :
5          return A[mitte].daten
6      case A[mitte].wert > gesucht :
7          return BINÄRE-SUCHE-R(gesucht, links, mitte − 1)
8      case A[mitte].wert < gesucht :
9           $\llcorner$  return BINÄRE-SUCHE-R(gesucht, mitte + 1, rechts)
10 error "Element nicht gefunden"
```

Aufgabe 4.4: Intuitives Insertionsort

INTUITIVES-INSERTIONSORT(Feld *A*)

Rückgabewert: sortiertes Feld

```

1  B → allokiere Feld der Länge A.länge
2  for i ← 1, ..., A.länge
3  do  $\llcorner$  B.EINFÜGEN-SORTFELD(A[i].wert, A[i].daten)
4  return B
```

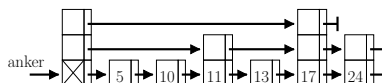
Ein sortiertes Feld hat den Aufwand $\Theta(n^2)$, da bei jedem einzufügenden Element die bereits vorhandene Liste komplett durchlaufen wird.

Aufgabe 4.5: Insertionsort

Es werden 33 Schlüsselvergleiche und 32 Schreibzugriffe benötigt.

Aufgabe 4.6: Skipliste

Zwischenergebnis nach dem Einfügen:



Nach dem Löschen fehlen die entsprechenden Knoten.

Aufgabe 4.7: Shellsort

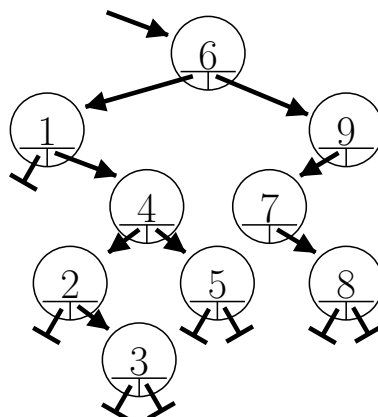
Mit Schrittweite 7 werden 3 Vergleiche und 4 Schreibzugriffe durchgeführt. Bei Schrittweite 3 sind es 9 Vergleiche und 7 Schreibzugriffe. Beim abschließenden Insertionsort (Schrittweite 1) fallen 16 Vergleiche und 14 Schreibzugriffe an.

Aufgabe 4.8: Shellsort mit anderen Schrittweiten

Obwohl mit den Schrittweiten 8 und 4 nur 9 Schlüsselvergleiche durchgeführt wurden, sind zwei Paare (8 und 10 sowie 2 und 3) zweimal verglichen worden. Dies ist bei den 12 Schlüsselvergleichen der Schrittweiten 7 und 3 nicht der Fall. Der Grund ist darin zu sehen, dass die sortierten Folgen der Schrittweite 8 genau in den Folgen der Schrittweite 4 enthalten sind.

Aufgabe 4.9: Binäre Suchbäume

a) Der resultierende Suchbaum ist:



Nach dem ersten Löschen steht die 7 in der Wurzel.

b) Bei dieser Teilaufgabe sind viele richtige Antworten möglich. Eine korrekte Reihenfolge ist: Löschen(30), Einfügen(35), Löschen(20), Einfügen(25) und Löschen(50).

Aufgabe 4.10: Ersatzknoten beim Löschen im Baum

Es müssen für einen Algorithmus *SUCHE-VORGÄNGER* im gegebenen Algorithmus *SUCHE-NACHFOLGER* lediglich jedes *links* durch *rechts* und jedes *rechts* durch *links* ersetzt werden.

Aufgabe 4.11: Breitensuche

Die Knoten werden in der folgenden Reihenfolge (und mit der in Klammern angegebenen resultierenden Distanz vom Startknoten) bearbeitet: 1(0), 4(1), 8(1), 2(2), 7(2), 6(2), 5(3) und 3(3).

Aufgabe 4.12: Modifizierte Breitensuche

Die Knoten werden in der folgenden Reihenfolge bearbeitet: 1, 8, 7, 2, 5, 3, 6 und 4. An

den Knoten 1 und 8 stehen jeweils mehrere noch nicht bearbeitete Knoten zur Wahl: Die kleinere Knotennummer wird zuerst auf den Stack gelegt und damit erst am Ende bearbeitet, wenn alle anderen erreichbaren Knoten besucht wurden.

Aufgabe 4.13: Tiefensuche

Resultierende Zeitmarken an den Knoten:

Knoten	1	2	3	4	5	6
<i>entdeckt</i>	1	2	11	8	3	5
<i>fertig</i>	10	7	12	9	4	6

Baumkanten: (1,2), (1,4), (2,5), (2,6)

Vorwärtskante: (1,5)

Rückwärtskante: (6,1)

Querkante: (3,5), (3,6), (4,2), (4,6), (6,5)

Aufgabe 4.14: Kantenkategorie

- Graph mit den Knoten $V = \{1,2\}$ und den Kanten (1,2) und (2,1).
- Graph mit den Knoten $V = \{1,2,3\}$ und den Kanten (1,2), (1,3), (2,1) und (2,3). Die betroffene Kante ist (1,3) bei den Startknoten 1 bzw. 2.
- Graph mit den Knoten $v = \{1,2,3\}$ und den Kanten (2,1), (2,3) und (3,1). Die betroffene Kante ist (3,1) bei den Startknoten 2 bzw. 3.

Aufgabe 4.15: Durchmesser eines Graphen

DURCHMESSER(Knoten V , Kanten E)

Rückgabewert: Durchmesser

```

1  durchmesser  $\leftarrow -\infty$ 
2  for  $v \in V$ 
3  do  $\lceil$  BREITENSUCHE( $V, E, v$ )
4      maximum  $\leftarrow -\infty$ 
5      for  $u \in V$ 
6      do if abstand[ $u$ ] > maximum
7           $\lfloor$  then  $\lceil$  maximum  $\leftarrow$  abstand[ $u$ ]
8      if maximum > durchmesser
9       $\lfloor$  then  $\lceil$  durchmesser  $\leftarrow$  maximum
```

Aufgabe 4.16: Topologische Sortierung

DURCHMESSER(Knoten V , Kanten E)

Rückgabewert: Feld mit Knoten in topologischer Sortierung

```

1  TIEFENSUCHE( $V, E$ )
2  ergebnis  $\rightarrow$  allokiere Feld der Länge  $|V|$ 
```

```
3  for  $index \leftarrow 1, \dots, |V|$ 
4  do  $\lceil maximum \leftarrow -\infty$ 
5      for  $v \in V$ 
6      do if  $entdeckt[v] > maximum$ 
7          then  $\lceil maximum \leftarrow entdeckte[v]$ 
8               $\lfloor knoten \leftarrow v$ 
9       $ergebnis[index] \leftarrow knoten$ 
10   $\lfloor entdeckte[knoten] \leftarrow -\infty$ 
11  return  $ergebnis$ 
```

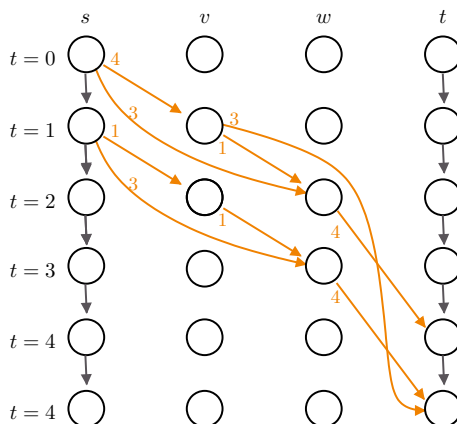
Als topologische Sortierung ergibt sich im Beispiel: 3, 5, 2, 1, 4 und 6.

Aufgabe 4.17: Level-Order-Baumausgabe

Der Algorithmus entspricht der Breitensuche, die im Wurzelknoten startet. Dabei ist darauf zu achten, dass immer zuerst der Wurzelknoten des linken Unterbaums und danach der des rechten Unterbaums in die Warteschlange geschrieben wird.

Aufgabe 4.18: Maximales Flussproblem umwandeln

Das folgende Bild zeigt lediglich diejenigen Kanten, die von der Lösung des maximalen Flussproblems benutzt werden. Die Zahlwerte geben die Einheiten an, die verschickt werden. Es resultiert ein Gesamtfluss von 11.



Aufgabe 4.19: Programmierung: Speicherung von Graphen

b) Für drei verschiedene Graphgrößen wurden zunächst alle möglichen Kanten eines Graphen in einer zufälligen Reihenfolge abgefragt. In unseren Experimenten resultierten die folgenden Laufzeiten:

$ V =$	10	100	1 000
Adjazenzmatrix	3 534 ns	501 810 ns	33 380 190 ns
Adjazenzliste	7 005 ns	2 507 550 ns	9 312 321 503 ns

Offensichtlich ist bei der wahlfreien Abfrage einzelner Kanten die Adjazenzliste eindeutig langsamer als die Adjazenzmatrix.

Werden jetzt in Graphen mit jeweils nur drei ausgehenden Kanten pro Knoten alle Kanten abgeprüft, zeigt die Adjazenliste hingegen ein wesentlich besseres Laufzeitverhalten:

$ V =$	100	1 000	10 000
Adjazenzmatrix	137 800 ns	852 687 973 ns	852 687 973 ns
Adjazenliste	71 655 ns	324 999 ns	677 369 ns

Kapitel 5

Aufgabe 5.1: Selectionsort

Es werden 66 Schlüsselvergleiche und 12 Schreibzugriffe benötigt.

Aufgabe 5.2: Dijkstra-Algorithmus

- a) Der Kürzeste-Wege-Baum enthält gemäß der nachfolgenden Tabelle die Kanten (1,4), (2,6), (5,1), (5,2), (5,9) und (6,8). Es wird in der Tabelle jeweils der Abstand und in Klammern der zugehörige Vorgängerknoten angezeigt.

Iteration	Knoten							
	1	2	3	4	5	6	7	8
1	$\infty (-)$	$\infty (-)$	$\infty (-)$	$\infty (-)$	0 (-)	$\infty (-)$	$\infty (-)$	$\infty (-)$
2	1 (5)	3 (5)	4 (5)	$\infty (-)$		$\infty (-)$	$\infty (-)$	2 (5)
3		3 (5)	4 (5)	2 (1)		$\infty (-)$	9 (1)	2 (5)
4		3 (5)	4 (5)			$\infty (-)$	9 (1)	2 (5)
5		3 (5)	3 (8)			9 (8)	9 (1)	
6			3 (8)			4 (2)	9 (1)	
7						4 (2)	9 (1)	
8							8 (6)	

- b) Ein Beispiel ist der Graph mit den Knoten {1,2,3}, den Kanten (1,2), (1,3) und (2,3) sowie den Gewichten $\gamma((1,2)) = 4$, $\gamma((1,3)) = 3$ und $\gamma((2,3)) = -2$.

Aufgabe 5.3: Kürzeste Wege: Spezialfall

Wenn die Knoten in ihrer topologischen Sortierung bearbeitet werden, wurden alle möglichen Vorgängerknoten (und damit die eingehenden Kanten) bereits betrachtet, wenn die ausgehenden Kanten berücksichtigt werden.

KÜRZESTE-WEGE-AZYKLISCH(Knoten V , Kanten $E \subset V \times V$, Kosten $\gamma : E \rightarrow \mathbb{R}$, Startknoten $s \in V$)

Rückgabewert: Distanzen *abstand*, Wege *vorgänger*

```

1  abstand[ $s$ ]  $\leftarrow$  0
2  reihenfolge  $\rightarrow$  TOPOLOGISCH-SORTIEREN( $V$ ,  $E$ )
3  for nächster  $\in$  reihenfolge
4  do  $\ulcorner$  for alle  $v \in V$  mit (nächster,  $v$ )  $\in E$ 
5      do  $\ulcorner$  if abstand[ $v$ ]  $>$  abstand[nächster] +  $\gamma$ [nächster,  $v$ ]
6          then  $\ulcorner$  abstand[ $v$ ]  $\leftarrow$  abstand[nächster] +  $\gamma$ [nächster,  $v$ ]
7       $\llcorner$   $\llcorner$  vorgänger[ $v$ ]  $\rightarrow$  nächster
```

8 **return** *abstand, vorgänger***Aufgabe 5.4: Prim-Algorithmus**

Der minimale Spannbaum enthält gemäß der nachfolgenden Tabelle die Kanten {1,10}, {2,3}, {2,10}, {3,5}, {3,8}, {4,10}, {5,9}, {6,7} und {7,9}. Es wird in der Tabelle jeweils der Abstand und in Klammern der zugehörige Vorgängerknoten angezeigt.

Iteration	Knoten									
	1	2	3	4	5	6	7	8	9	10
1	0 (–)	∞ (–)	∞ (–)	∞ (–)	∞ (–)	∞ (–)	∞ (–)	∞ (–)	∞ (–)	∞ (–)
2		∞ (–)	∞ (–)	4 (1)	∞ (–)	∞ (–)	∞ (–)	7 (1)	∞ (–)	2 (1)
3		4 (10)	∞ (–)	1 (10)	8 (10)	∞ (–)	∞ (–)	7 (1)	∞ (–)	
4		4 (10)	∞ (–)		6 (4)	∞ (–)	∞ (–)	7 (1)	∞ (–)	
5			1 (2)		3 (2)	∞ (–)	∞ (–)	7 (1)	∞ (–)	
6					2 (3)	∞ (–)	∞ (–)	1 (3)	9 (3)	
7					2 (3)	∞ (–)	4 (8)		3 (8)	
8						8 (5)	4 (8)		2 (5)	
9						7 (9)	1 (9)			
10						3 (7)				

Aufgabe 5.5: Rundreiseproblem

- Die Knoten 4, 2, 1, 3 und 5 bestimmen in dieser Reihenfolge die Rundreise. Dabei werden jeweils die minimal möglichen Kantengewichte 11, 21, 8 und 39 gewählt. Die Rückkante vom Knoten 5 zum Knoten 4 ergibt das zusätzliche Gewicht 43. So resultiert eine Rundreise der Länge 122.
- Der minimale Spannbaum mit den Kanten {4,2}, {4,3}, {3,1} und {1,5} wird in der Reihenfolge 4, 2, 3, 1 und 5 abgelaufen. Dadurch ergeben sich die Kantengewichte 11, 27, 8, 31 und 43. Die Gesamtlänge der Rundreise ist 120.

Aufgabe 5.6: Korrektheit des Primalgorithmus

Es wird immer derjenige Knoten gewählt, der von den bereits aufgenommenen Knoten über die Kante mit dem kleinsten Gewicht erreicht wird. Zum Beweis der Korrektheit, nehmen wir an, dass eine so gewählte Kante (mit eindeutigem minimalem Gewicht) nicht zum minimalen Spannbaum gehört. Dann könnte man diejenige Kante im minimalen Spannbaum, über die der Knoten alternativ erreicht wird, streichen und durch die Kante mit kleinstem Gewicht ersetzen – dann hätte man allerdings einen minimaleren Spannbaum konstruiert, was im Widerspruch zur Annahme steht.

Aufgabe 5.7: Maximaler Fluss

Es ist ein Fluss von 6 Einheiten möglich.

Aufgabe 5.8: Geldrückgabe

- a) Ein möglicher (nicht effizienter) Algorithmus ist der folgende.

RÜCKGABE-GREEDY(Betrag w)

Rückgabewert: keiner; Seiteneffekt: direkte Münzausgabe

```

1  while  $w > 0$ 
2  do  $\lceil$  switch
3      case  $25 \leq w$  : 25 Cent ausgeben
4           $w \leftarrow w - 25$ 
5      case  $10 \leq w < 25$  : 10 Cent ausgeben
6           $w \leftarrow w - 10$ 
7      case  $5 \leq w < 10$  : 5 Cent ausgeben
8           $w \leftarrow w - 5$ 
9      case  $w < 5$  : 1 Cent ausgeben
10          $w \leftarrow w - 1$ 

```

Es muss gezeigt werden, dass für alle möglichen Geldbeträge die Greedy-Ausgabe zur minimalen Münzmenge führt. Da es reicht, dies bis zum kleinsten gemeinsamen Vielfachen aller Münzwerte zu überprüfen, kann dies einfach »brute force« erledigt werden.

- b) Zum Beispiel bei der Münzwertmenge $\{1, 5, 20, 25\}$ benötigt der Greedy-Algorithmus 4 Münzen für den Rückgabebetrag 40. Allerdings wären zwei Münzen als Minimum möglich.

Kapitel 6

Aufgabe 6.1: Maximaler Fluss

Es werden zunächst 3 Einheiten über die Zwischenknoten 3 und 6 geschoben. Anschließend wird 1 Einheit über die Knoten 2, 5, 6, 4 und 7 zum Zielknoten transportiert. Zum Abschluss werden 2 Einheiten über die Zwischenknoten 2, 5, 6, 3, 4 und 7 geschoben.

Aufgabe 6.2: Einfügen in AVL-Bäume

- a) Es werden die folgenden Rotationen durchgeführt:

- beim Einfügen der 4: RL-Rotation an 1
- beim Einfügen der 3: L-Rotation an 1
- beim Einfügen der 8: LR-Rotation an 10
- beim Einfügen der 6: R-Rotation an 8
- beim Einfügen der 5: R-Rotation an 9

- b) Eine mögliche Operationenfolge ist:

- Einfügen der 15: RL-Rotation an 8
- Einfügen der 17
- Einfügen der 30: LR-Rotation an 47
- Einfügen der 4 und Einfügen der 40

Aufgabe 6.3: Extreme AVL-Bäume

Ein maximal ungleichgewichtiger AVL-Baum der Tiefe 4 erhält man, indem unter einen Wurzelknoten ein vollständiger binärer Baum der Tiefe 3 und ein Fibonacci-Baum der Tiefe 2 gehängt werden. Das Knotenverhältnis ist 7:2. Für die Tiefe 5 gilt ein Knotenverhältnis von 15:4.

Aufgabe 6.4: Löschen in AVL-Bäumen

- a) Es fallen die folgenden Rotationen an:
 - beim Löschen der 6: L-Rotation an 20 und L-Rotation an 31
 - beim Löschen der 54: L-Rotation an 69
 - beim Löschen der 66: R-Rotation an 69 (als Ersatzknoten der gelöschten Wurzel)
 - beim Löschen der 24: RL-Rotation an 31
- b) Eine mögliche Konstruktion besteht aus einem Wurzelknoten mit einem Fibonacci-Baum der Tiefe 3 und einem Fibonacci-Baum der Tiefe 4 als Unterbäume. Werden diese so angeordnet, dass die tieferen Unterbäume der Fibonacci-Bäume innen liegen, werden zwei Doppelrotationen beim Löschen des Blatts in Tiefe 3 notwendig.
- c) Eine mögliche Abfolge der Operationen ist:
 - Einfügen der 75: L-Rotation an der 70
 - Löschen der 7: R-Rotation an der 11 (als Ersatzknoten) und RL-Rotation an der 13
 - Einfügen der 76: L-Rotation an der 53
 - Löschen der 13
 - Einfügen der 20: LR-Rotation an der 22
 - Löschen der 29

Aufgabe 6.5: Gewichtsbalancierte Bäume

- a) Die Bedingung gilt für alle beiden Bäume mit 2 Knoten sowie den Baum mit je einem Knoten als rechten und linken Unterbaum.
- b) Ein gewichtsbalancierter Baum mit 5 Knoten besteht aus einem Baum mit drei Knoten im einen Unterbaum und einem einzelnen Knoten im anderen Unterbaum. Wird im tieferen Unterbaum ein Knoten eingefügt, ist die Bedingung der Gewichtsbalance nicht mehr erfüllt. Mit einer Rotation am Wurzelknoten kann sie wieder hergestellt werden.

Kapitel 7**Aufgabe 7.1: Mergesort**

Es werden 37 Schlüsselvergleiche benötigt.

Aufgabe 7.2: Mastertheorem

- a) $T(n) \in \Theta(n^3 \cdot \log n)$
- b) $T(n) \in \Theta(\sqrt{n^3}) = \Theta(n^{1.5})$
- c) $T(n) \in \Theta(n^4)$
- d) Der lokale Aufwand lässt sich nach unten durch n und nach oben durch n^2 beschränken. Da beidesmal der Gesamtaufwand $\Theta(n^3)$ resultiert, muss diese asymptotische Laufzeit auch für $T(n)$ gelten.

Aufgabe 7.3: Verbesserung von Insertionsort

Es resultieren die folgenden Rekursionsgleichungen:

- Best-Case: $T(n) = 2 \cdot T(\frac{n}{2}) + n$ mit $T(n) \in \Theta(n \cdot \log n)$
- Worst-Case: $T(n) = 2 \cdot T(\frac{n}{2}) + n^2$ mit $T(n) \in \Theta(n^2)$

Aus asymptotischer Sicht ist die Laufzeit im Worst-Case gleich geblieben, während sich die Laufzeit im Best-Case verschlechtert. Die reale Laufzeit wird in jedem Fall schlechter ausfallen.

Aufgabe 7.4: Mastertheorem mit Substitution

- a) Es ergibt sich ebenfalls die asymptotische Laufzeit $\Theta(2^n)$.
- b) Mit der Substitution ergibt sich

$$\begin{aligned} T(2^m) &= 2 \cdot T((2^m)^{\frac{1}{2}}) + m \\ &= 2 \cdot T(2^{\frac{m}{2}}) + m. \end{aligned}$$

Durch Umbenennen $S(x) = T(2^x)$ folgt die Darstellung

$$S(m) = 2 \cdot S\left(\frac{m}{2}\right) + m.$$

Das Mastertheorem ergibt direkt: $S(m) \in \Theta(m \cdot \log m)$, woraus folgende Ergebnisse für die ursprüngliche Gleichung folgen.

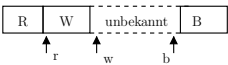
$$\begin{array}{ll} T(2^m) \in \Theta(m \cdot \log m) & \text{durch Auflösung der Umbenennung} \\ T(n) \in \Theta(\log n \cdot \log \log n) & \text{durch Rücksubstitution.} \end{array}$$

Aufgabe 7.5: Quicksort

- a) Es ergeben sich 49 Schlüsselvergleiche.
- b) Es ergeben sich 42 Schlüsselvergleiche. In der Partition mit dem Pivot-Element 12 entsteht dabei eine linke einelementige Partition mit 11 und eine rechte Partition mit 15 und 13 – das Element 12 steht bereits auf dem richtigen Platz und fällt aus den Partitionen heraus.

Aufgabe 7.6: Spezialfall für das Sortieren

Kernidee ist die folgende Invariante:



DUTCH-FLAG-PROBLEM(Feld A)

Rückgabewert: nichts; Seiteneffekt: sortiertes Feld

```
1  r ← 1
2  w ← 1
3  b ← A.länge
4  while w < b
5  do ⌈ switch
6      case A[w] = 'W' : w ← w + 1
7      case A[w] = 'B' : VERTAUSCHE(A, w, b)
8                      b ← b - 1
9      case A[w] = 'R' : VERTAUSCHE(A, w, r)
10                     r ← r + 1
11  L      w ← w + 1
```

Aufgabe 7.7: Varianten des Partitionierens

Das gegebene Feld kann mit 34 Schlüsselvergleichen sortiert werden. Dabei wirkt sich positiv aus, dass das Pivotelement immer an den richtigen Platz geschrieben wird und dadurch die Partitionen kleiner werden.

Aufgabe 7.8: Medianbestimmung

Bei der ersten Berechnung ist 12 der Median der Fünfermediane. Nach der Partitionierung entsteht links eine Partition der Größe 11 und rechts der Größe 10. Die erneute Partitionierung der linken Partition erfolgt mit dem neuen Median der Fünfermediane 8. Es resultiert eine Partition mit 3 Elementen.

Aufgabe 7.9: Vergleich von Sortiervverfahren

Wir haben die folgenden beispielhaften Laufzeiten gemessen:

Anzahl der Elemente	20	100	200	2 000	4 000
a) Insertionsort	299 ns	1 366 ns	2 115 ns	19 435 ns	37 233 ns
Quicksort	8 754 ns	53 351 ns	79 566 ns	6 318 405 ns	25 669 244 ns
b) Insertionsort	13 118 ns	13 705 ns	44 634 ns	3 968 769 ns	15 637 924 ns
Quicksort	81 452 ns	14 661 ns	30 931 ns	371 784 ns	791 251 ns
c) Insertionsort	45 149 ns	21 222 ns	81 374 ns	7 870 768 ns	31 360 847 ns
Quicksort	2 042 ns	20 058 ns	68 719 ns	5 522 960 ns	25 571 437 ns

Es fällt auf, dass Insertionsort auf kleinen Feldern effizienter als Quicksort ist. Ferner ist Quicksort auf kleinen sortierten oder umgekehrten sortierten Feldern deutlich schneller als

auf zufällig belegten Feldern – im Widerspruch zu den asymptotischen Aussagen. Dies liegt an der strukturierten Abarbeitung der sortierten Felder, die anders durch die Java-Laufzeitumgebung optimiert werden kann.

Aufgabe 7.10: Verallgemeinertes Mastertheorem

- a) Es folgt $p = 1$ direkt aus der Gleichung $\frac{2}{2^p} = 1$. Damit ergibt sich die Laufzeit als

$$\begin{aligned} T(x) &\in \Theta(x^p \cdot (1 + \int_1^x \frac{u}{u^{1+p}} du)) = \Theta(x \cdot (1 + \int_1^x \frac{1}{u} du)) \\ &= \Theta(x \cdot (1 + (\log u) \Big|_{u=1}^x)) = \Theta(x \cdot (1 + (\log x - \log 1))) \\ &= \Theta(x \cdot (1 + \log x)) = \Theta(x \cdot \log x) \end{aligned}$$

- b) Es ergibt sich $p = 2$. Für die Berechnung der Laufzeit muss $\frac{u \cdot \log u}{u^2}$ integriert werden, was mit der Stammfunktion $\frac{1}{2} \cdot (\log u)^2$ geht. So ergibt sich insgesamt die Laufzeit $T(x) \in \Theta(x^2 \cdot (\log x)^2)$.
- c) Es ergibt sich ein Wert $p = 0,9 \dots$, den wir zunächst nicht genau ausrechnen. Im Weiteren muss dann $\frac{u}{u^{1+p}} = u^{-p}$ integriert werden. Die zugehörige Stammfunktion lautet $\frac{1}{1-p} \cdot u^{1-p}$. Damit ergibt sich als Laufzeit:

$$\begin{aligned} T(x) &\in \Theta(x^p \cdot (1 + (\frac{1}{1-p} \cdot x^{1-p} - \frac{1}{1-p} \cdot 1^{1-p}))) \\ &= \Theta(x^p \cdot (1 + (\frac{1}{1-p} \cdot x^{1-p} - \frac{1}{1-p} \cdot 1^{1-p}))) \\ &= \Theta(x^p + x^p \cdot x^{1-p}) \text{ (die Konstanten wurden gestrichen)} \\ &= \Theta(x) \text{ (weil } x^p \cdot x^{1-p} = x \in \Omega(x^p)) \end{aligned}$$

Kapitel 8

Aufgabe 8.1: Floyd-Warshall-Algorithmus

Die folgenden Entfernungen sind das Ergebnis des Algorithmus:

	1	2	3	4	5
1	0	4	3	5	6
2	4	0	4	2	2
3	4	1	0	2	3
4	2	6	5	0	4
5	6	3	2	4	0

Aufgabe 8.2: Optimale Suchbäume

Es resultiert ein Suchbaum mit der mittleren Zugriffszeit 2,2. In der Wurzel steht der Wert 10, die Wurzel des linken Unterbaums ist 4 und die des rechten Unterbaums 13.

Aufgabe 8.3: Straight-Mergesort

Es werden 38 Schlüsselvergleiche benötigt. Dies ist zwar ein Vergleich mehr als beim rekursiven Mergesort. Dennoch ist mit einer besseren Laufzeit zu rechnen, da die Iteration insgesamt schneller als die Rekursion abläuft.

Aufgabe 8.4: Natürliches Mergesort

- a) Wir speichern die Anfangsindizes der bereits sortierten Teilfolgen in einer Liste.

NATÜRLICHES-MERGESORT(Feld A)

Rückgabewert: nichts; Seiteneffekt: A ist sortiert

```

1  teilfolgenListe  $\rightarrow$  NULL
2  for index  $\leftarrow A.l\ddot{a}nge - 1, \dots, 1$ 
3  do  $\ulcorner$  if  $A[index] > A[index + 1]$ 
4       $\lrcorner$  then  $\lrcorner$  teilfolgenListe  $\rightarrow$  allokiere Element(index, teilfolgenListe)
5  while teilfolgenListe.n\ddot{a}chster  $\neq$  NULL
6  do  $\ulcorner$  el  $\rightarrow$  teilfolgenListe
7      while el.n\ddot{a}chster  $\neq$  NULL
8          do  $\ulcorner$  mitte  $\leftarrow el.n\ddot{a}chster.wert - 1$  if el.n\ddot{a}chster.n\ddot{a}chster  $\neq$  NULL
9              then  $\lrcorner$  rechts  $\leftarrow el.n\ddot{a}chster.n\ddot{a}chster.wert - 1$ 
10             else  $\lrcorner$  rechts  $\leftarrow A.l\ddot{a}nge$ 
11             MISCHEN( $A, el.wert, mitte, rechts$ )
12             el.n\ddot{a}chster  $\rightarrow el.n\ddot{a}chster.n\ddot{a}chster$ 
13          $\lrcorner$   $\lrcorner$  el  $\rightarrow el.n\ddot{a}chster$ 
```

Alternativ kann der Algorithmus auch mit einem Feld realisiert werden, in dem die Anfangsindizes der Teilfolgen stehen. In dieser Fassung müsste dann in den späteren Iterationen immer nur jeder 2^i -te Index berücksichtigt werden.

- b) Das natürliche Mergesort braucht mit der Bestimmung der Teilfolgen 28 Schlüsselvergleiche. Beim Straight-Mergesort sind es 33.

Aufgabe 8.5: Geldrückgabe

Aufgabe 8.6: Bitonische Rundreisen

- a) Wird die Kante $(k, k + 1)$ benutzt, können partielle Rundreisen mit den Endpunkten $j < k$ und $k + 1$ konstruiert werden. Mit einer Kante $(j, k + 1)$ erhält man eine partielle Rundreise mit den Endpunkten k und $k + 1$.
- b) Der Index j muss minimal wie folgt gewählt werden.

$$l\ddot{a}nge[k, k + 1] = \min_{1 \leq j \leq k-1} (l\ddot{a}nge)[j, k] + \gamma_{j, k+1}$$

- c) Damit ergibt sich aus b) der folgende Algorithmus quasi automatisch. Es muss lediglich am Ende noch die fehlende Kante hinzugefügt werden, um die Rundreise zu

schließen. Wir berechnen hier nur die Länge der Rundreise; Angaben zur Reihenfolge der besuchten Knoten können selbst ergänzt werden.

BITONISCHE-RUNDREISE(Knotenzahl n , Abstände γ)

Rückgabewert: kürzeste bitonische Rundreise

```

1   $länge[1,2] \leftarrow \gamma_{1,2}$ 
2  for  $k \leftarrow 3, \dots, n$ 
3  do  $\ulcorner$  for  $j \leftarrow 1, \dots, k-2$ 
4      do  $\llcorner$   $länge[j, k] \leftarrow länge[j, k-1] + \gamma_{k-1,k}$ 
5       $minimum \leftarrow \infty$ 
6      for  $m \leftarrow 1, \dots, k-2$ 
7      do  $\ulcorner$  if  $länge[m, k-1] + \gamma_{m,k} < minimum$ 
8           $\llcorner$  then  $\llcorner$   $minimum \leftarrow länge[m, k-1] + \gamma_{m,k}$ 
9           $\llcorner$   $länge[k-1, k] \leftarrow minimum$ 
10  $gesamt \leftarrow \infty$ 
11 for  $k \leftarrow 1, \dots, n-1$ 
12 do  $\ulcorner$  if  $länge[k, n] + \gamma_{k,n} < gesamt$ 
13      $\llcorner$  then  $\llcorner$   $gesamt \leftarrow länge[k, n] + \gamma_{k,n}$ 
14 return  $gesamt$ 
```

- d) Die kürzeste bitonische Rundreise besteht aus der Hinreise über die Knoten 1, 2, 4 und 6 sowie der Rückreise vom Knoten 6 über 5 und 3 wieder zum Knoten 1. Die Kosten sind 20.
- e) Es handelt sich um einen $\Theta(n^2)$ -Algorithmus.

Kapitel 9

Aufgabe 9.1: Interpolations-Suche

- a) Bereits die erste Schätzung gibt den gewünschten Index 9 zurück.
- b) Es wird zunächst der Index 5, dann der Index 8 und in der dritten Schätzung der Index 9 errechnet.

Bei der binären Suche werden jeweils 3 Schlüsselvergleiche benötigt.

Aufgabe 9.2: Worst-Case-Laufzeit der Interpolations-Suche

- a) Es kann beispielsweise $A[1] = 1$, $A[9] = 9$ und $A[10] = 1000$ belegt werden.
- b) Das folgende Feld genügt den Anforderungen.

1	2	3	4	5	6	7	8	9	1000
---	---	---	---	---	---	---	---	---	------

- c) Eine genauere Analyse der ersten Schätzung ergibt, dass das folgende von n abhängige Feld die gewünschte Worst-Case-Laufzeit erfordert.

1	2	3	...	$n-2$	$n-1$	$2 \cdot n^2$
---	---	---	-----	-------	-------	---------------

Aufgabe 9.3: Countingsort

Das Zählerfeld C wird vor dem Sortieren mit

3	3	5	7	10
---	---	---	---	----

initialisiert und hat nach dem Sortieren die folgende Belegung.

0	3	3	5	7
---	---	---	---	---

Aufgabe 9.4: Radix-Sort

1. Iteration	100	010	110	000	101	001	011	111
2. Iteration	100	000	101	001	010	110	011	111
3. Iteration	000	001	010	011	100	101	110	111

Aufgabe 9.5: Hashing

- a) Bei der Suche nach den 9 Elementen treten 14 Kollisionen auf.

13	65	15	34				47	21	61	23	10
----	----	----	----	--	--	--	----	----	----	----	----

- b) Bei der Suche nach den 9 Elementen treten 9 Kollisionen auf.

13	10	15		65			47	21	61	23	34
----	----	----	--	----	--	--	----	----	----	----	----

- c) Bei der Suche nach den 9 Elementen treten 7 Kollisionen auf.

13		15		10		21		47	61	23	65	34
----	--	----	--	----	--	----	--	----	----	----	----	----

- d) Bei der Suche nach den 9 Elementen treten 6 Kollisionen auf.

13	34	15				21		47	61	10	65	23
----	----	----	--	--	--	----	--	----	----	----	----	----

Dabei hat die 10 beim Einfügen die 23 von ihrer Position verdrängt.

Aufgabe 9.6: Alternative Hash-Funktion

Merkur	8
Venus	9
Erde	5

Mond	7
Mars	1
Jupiter	11

Saturn	5
Uranus	9
Neptun	4

Kapitel 10**Aufgabe 10.1: Binärer Heap**

Nach dem Einfügen der Elemente sieht der Heap wie folgt aus.

10	12	11	15	14	20	19	18
----	----	----	----	----	----	----	----

Nach dem Löschen und dem Ändern der Priorität:

11	12	13	15	14	20	18	
----	----	----	----	----	----	----	--

Aufgabe 10.2: Heapsort

Beim Heapaufbau fallen 16 Schlüsselvergleiche an – davon 4 für das Absinken der 3 und 6 Schlüsselvergleiche für das Absinken der 24.

Mit 29 weiteren Schlüsselvergleichen für das eigentliche Sortieren resultieren insgesamt 45 Schlüsselvergleiche.

Kapitel 11

Aufgabe 11.1: Mehrphasen-Mergesort

13 Elemente werden auf das Band *A* und 8 Elemente auf das Band *B* geschrieben. Unter der Voraussetzung, dass die ersten Elemente der Folge in *A* stehen, enthält Band *C* nach dem ersten Mischen die folgenden Elemente.

4 · 19	2 · 28	11 · 21	15 · 25	7 · 23	12 · 14	22 · 27	1 · 20
--------	--------	---------	---------	--------	---------	---------	--------

Aufgabe 11.2: B-Baum

Beim Einfügen der 14 Elemente entsteht bei der Ordnung 1 ein Baum der Tiefe 3. In der Wurzel stehen die Elemente 4 und 8.

Beim Löschen sind die folgenden Operationen zur Reparatur des B-Baums notwendig.

- Löschen der 12: Ersatz durch das Element 13.
- Löschen der 6: Ersatz durch das Element 7. Durch Verschmelzen entsteht ein neues Blatt mit 5 und 6. In der mittleren Ebene wird das Element 10 in die Wurzel geschoben; der mittlere Knoten enthält das Element 8.
- Löschen der 9: Verschieben der 6 und der 8.
- Löschen der 10: Ersatz durch das Element 11. Durch Verschmelzen entsteht ein Blatt mit den Elementen 13 und 14. In der mittleren Ebene entsteht durch Verschmelzen ein Knoten mit 6 und 11. Die Wurzel enthält nur noch das Element 4.
- Löschen der 13 geht direkt.
- Löschen der 5: Verschmelzen ergibt ein Blatt mit den Elementen 6 und 8.
- Löschen der 1: Verschmelzen zu einem neuen Blatt mit den Elementen 2 und 3. Verschmelzen auf der nächsten Ebene ergibt einen Knoten mit den Elementen 4 und 11. Dies ist auch die neue Wurzel und der Baum hat nur noch die Tiefe 2.
- Löschen der 14: Verschmelzen ergibt die Elemente 8 und 11 im Blatt.
- Löschen der 4: Das Element 8 ersetzt die 4.
- Löschen der 2 geht direkt.
- Löschen der 11: Verschieben der Elemente 3 und 8.

Bei der Ordnung 2 ergibt sich ein B-Baum der Tiefe 2 mit den Knoten 3, 6, 9 und 12 in der Wurzel.

Die folgenden Operationen sind beim Löschen notwendig:

- Löschen der 12: Ersatz durch das Element 13. Verschmelzen ergibt ein Blatt mit 10, 11, 13 und 14.

- Löschen der 6: Ersatz durch das Element 7. Verschieben der 10 und der 9 ergibt einen Wurzelknoten mit den Elementen 3, 7 und 10.
- Löschen der 9: Verschieben der 10 und der 11 ergibt die Elemente 3, 7 und 11 in der Wurzel.
- Löschen der 10: Verschmelzen führt zum Wurzelknoten mit den Elementen 3 und 7.
- Löschen der 13 geht direkt.
- Löschen der 5: Verschieben der 7 und der 8.
- Löschen der 1: Verschmelzen ergibt ein Blatt mit 2, 3, 4 und 7. Nur noch die 8 steht in der Wurzel.
- Löschen der 14: Verschieben der 8 und der 7.
- Löschen der 4 geht direkt.
- Löschen der 2: Verschmelzen aller Elemente zu einem Knoten. Der Baum hat nur noch die Tiefe 1.
- Löschen der 11 geht direkt.

Aufgabe 11.3: Erweiterbares Hashing

Die ersten drei Einträge können in eine Tabelle mit globaler Tiefe 1 eingefügt werden. Mit dem Eintrag 000111 wird die Adresstabelle verdoppelt. Die Präfixe 10 und 11 verweisen weiterhin auf denselben Block. Auch die nächsten zwei Einträge können in der Tabelle ohne weitere Modifikationen untergebracht werden. Mit dem Eintrag 010110 muss die Adresstabelle nochmals verdoppelt werden. Jetzt verweisen 000 und 001 auf denselben Block – ebenso: 100, 110, 101 und 111. Mit dem Eintrag 001101 wird den Präfixen 000 und 001 jeweils ein eigener Block zugewiesen. Mit dem letzten Eintrag wird der letzte Block mit lokaler Tiefe 1 in zwei Blöcke mit lokaler Tiefe 2 geteilt.

Kapitel 12

Aufgabe 12.1: Selbstorganisierende Liste

- In der sortierten Liste werden 45 Schlüsselvergleiche benötigt; in der selbstorganisierenden Liste sind es 39.
- In der sortierten Liste werden 29 Schlüsselvergleiche benötigt; in der selbstorganisierenden Liste sind es 32.

Aufgabe 12.2: Evolutionärer Algorithmus

- Drei vertauschende Mutationen reichen aus. Beispielsweise kann die 6 mit der 4 vertauscht werden, dann die 6 mit der 1 und schließlich die 2 mit der 3.
- Die Permutationen können mit 2 invertierenden Mutationen ineinander überführt werden. Zunächst wird der Teil der Permutation von 6 bis einschließlich 1 invertiert; anschließend die Zahlen 4, 2 und 1.

Algorithmen und Datenstrukturen

Weicker, K.; Weicker, N.

2013, IX, 356 S. 91 Abb. in Farbe., Softcover

ISBN: 978-3-8348-1238-4