Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 14. April 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

1. Übung

Abgabe der Lösungen: 2. Mai 2000

Aufgabe 1: **2+1 Punkte**

Für ein bestimmtes Problem stehen vier unterschiedliche Algorithmen mit unterschiedlichen Laufzeitkomplexitäten zur Verfügung:

$$\text{A: } \log_2 n \qquad \text{B: } n \qquad \text{C: } n^2 \qquad \text{D: } 2^n$$

Auf dem vorhandenen Rechner kann mit jedem dieser Algorithmen ein Problem der Größe $n = 1000$ innerhalb einer Stunde gelöst werden.

(a) Nun soll ein neuer Rechner angeschafft werden, der in gleicher Zeit doppelt so viele Instruktionen abarbeitet. Was ist die maximale Problemgrösse n die bewältigt werden kann, wenn weiterhin nur eine Stunde zur Verfügung steht.

(b) Welchen Algorithmus würden Sie für ein kleines Problem $n = 500$ verwenden?

Aufgabe 2: **2 Punkte**

Zeigen Sie, daß für beliebige Funktionen f, g: $\mathbb{N} \to \mathbb{R}$ gilt:

$$g \in O(f) \quad \Longleftrightarrow \quad f \in \Omega(g).$$

Aufgabe 3: **2+2+2+2+1+1 Punkte**

Beantworten Sie folgende Fragen und beweisen Sie jeweils Ihre Behauptung. Verwenden Sie ggf. die folgende Definition der o-Notation („Klein-o-Notation“):

$$g \in o(f) \quad \Longleftrightarrow \quad \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$$

(a) Es sei $g(n) := 4711n^2 - 42n + 5$ und $f(n) := n^2$. Gilt $g \in o(f)$? Gilt $g \in \Theta(f)$?

(b) Es sei $g(n) := \lceil \sqrt[3]{n} \rceil$ und $f(n) := \lceil \sqrt{n} \rceil$. Gilt $g \in O(f)$? Gilt $g \in \Omega(f)$?

(c) Es sei $g(n) := \sum_{j=0}^{n}(n-j)$ und $f(n) := n^2$. Gilt $g \in o(f)$? Gilt $g \in O(f)$?

(d) Es sei $g(n) := \sum_{j=0}^{n} 2^j$ und $f(n) := 3^n$. Gilt $g \in O(f)$? Gilt $g \in \Theta(f)$?

(e) Es sei $g(n) := \sum_{j=1}^{n} i \cdot 2^i$ und $f(n) := n^2 \cdot 2^n$. Gilt $g \in O(f)$?

(f) Es sei $g(n) := \lceil \log n \rceil$ und $f(n) := \lceil \sqrt{n} \rceil$. Gilt $g \in O(f)$?

Aufgabe 4: Komplexität von Funktionen **4 Punkte**

Sortieren Sie folgende Funktionen nach ihrer Komplexitätsordnung (*Schreibweise:* $g(n) \ll f(n) \iff f \in \Omega(g)$). Teilen Sie Ihre Liste derart in Äquivalenzklassen auf, daß f und g genau dann in derselben Äquivalenzklasse sind, wenn $f \in \Theta(g)$. Begründen Sie Ihre Antwort. Verwenden Sie gegebenenfalls die Stirling'sche Näherung. Es ist $\mathrm{ld}\, n := \log_2 n$.

$$n^3 \quad n^2 \quad n! \quad 2^{(5+\mathrm{ld}\, n)} \quad n^n \quad n \quad n \cdot 2^n \quad \left(\frac{4}{3}\right)^n \quad 42 \quad 3^n \quad (\mathrm{ld}\, n)^{\mathrm{ld}\, n}$$

Aufgabe 5: Programmieraufgabe: Array-Implementierung von Listen **4 Punkte**

Implementieren Sie den abstrakten Datentyp Liste von ganzen Zahlen mit Hilfe eines Arrays. Realisieren Sie dabei die folgenden Funktionen:

- leere Liste initialisieren
  - PROCEDURE Init();
- zum ersten/letzten/nächsten/vorherigen Element der Liste
  - PROCEDURE ToFirst() ;
  - PROCEDURE ToLast() ;
  - PROCEDURE Forth() ;
  - PROCEDURE Back() ;
- testen ob Ende/Anfang der Liste erreicht
  - PROCEDURE IsAfter() : BOOLEAN ;
  - PROCEDURE IsBefore() : BOOLEAN ;
- aktuelles Element lesen/setzen
  - PROCEDURE Get() : INTEGER ;
  - PROCEDURE Set(item: INTEGER) ;
- neues Element am Ende der Liste anfügen
  PROCEDURE Append(item : INTEGER);
- neues Element vor aktuellem Element einfügen
  PROCEDURE InsertBefore(item : INTEGER);
- aktuelles Element entfernen
  PROCEDURE Delete();

Aufgabe 6: Programmieraufgabe: Türme von Hanoi **4+2+3 Punkte**

(a) Gegeben seien drei Stäbe A, B und C. Jeder Stab kann maximal n Scheiben aufnehmen, wobei niemals eine größere Scheibe auf einer kleineren Scheibe liegen darf. Zu Beginn liegen sämtliche Scheiben auf dem Stab A. Die Scheiben sollen von A nach C transportiert werden. Implementieren Sie eine rekursive Lösung des Problems.

(b) Verwenden Sie die Implementierung aus Teil a), um die Anzahl der Transportschritte zu bestimmen, die erforderlich sind, um n=3, n=7 bzw. n=10 Scheiben von A nach C zu bewegen.

(c) Formulieren und lösen Sie explizit die Rekursionsgleichung, die den Auwand für das Problem der „Türme von Hanoi“ beschreibt. Bestimmen Sie die Lösung der Gleichung für n=3, n=7 und n=10.

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 28. April 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

2. Übung

Abgabe der Lösungen: 8. Mai 2000

Aufgabe 7: Oh Oh Oh **3 Punkte**

Beweisen oder widerlegen Sie, daß für Funktionen $f, g : \mathbb{N} \to \mathbb{R}^+$ gilt:

$$f \in O(g) \wedge g \notin O(f) \quad \Rightarrow \quad f \in o(g)$$

Aufgabe 8: Doppelt verkettete Listen **4 Punkte**

Implementieren Sie nochmals die „Liste von ganzen Zahlen“ aus Aufgabe 5 aber diesmal mit Hilfe einer doppelt verketteten Liste. D.h. jeder Knoten besitzt sowohl einen Zeiger auf seinen Nachfolger als auch einen auf seinen Vorgänger. Demonstrieren Sie an einem Beispielprogramm, dass beide Implementierungen identische Ergebnisse liefern. (Eine Musterlösung zu Aufgabe 5 wird in Kürze im Web zu Verfügung stehen.)

Aufgabe 9: Binärer Suchbaum **3+4+1 Punkte**

Unter einem binären Suchbaum verstehen wir einen Baum vom Grad zwei mit folgenden Eigenschaften:

- Für die Werte der Knoten (Schlüsselelemente) ist eine totale Ordnung definiert.
- Der Schlüssel eines Knotens n ist größer oder gleich seinem linken Nachfolger n_1 und kleiner oder gleich seinem rechten Nachfolger n_2.

1. Implementieren Sie für einen binären Suchbaum mit ganzen Zahlen die folgenden Prozeduren:
   - leeren Baum initialisieren
     PROCEDURE Init();
   - Element einfügen
     PROCEDURE Insert(item: INTEGER);
   - Vorhandensein eines Werts testen
     PROCEDURE Search(item: INTEGER) : BOOLEAN;
   - Alle Elemente in aufsteigender Reihenfolge ausgeben
     PROCEDURE Write();
2. In einen anfangs leeren Suchbaum werden n zufällige Zahlen (in zufälliger Reihenfolge) eingefügt. Welche Zeitkomplexitäten haben die Funktionen Insert, Search und Write auf dem so entstandenen Baum im *best-*, *average-* und *worst-case*?
3. Verwenden Sie den Suchbaum um die unten angegebenen Beispieldaten zu sortieren. Welche Laufzeitkomplexität hat dieses Sortierverfahren im *best-*, *average-* und *worst-case*?

Aufgabe 10: Baumtraversierung **1+1+1+3 Punkte**

Implementieren Sie *nicht-rekursive* Prozeduren, die auf einem Binärbaum einen

1. Preorder-Tiefendurchlauf
2. Postorder-Tiefendurchlauf
3. Inorder-Tiefendurchlauf
4. Breitendurchlauf

ausführen. Testen Sie diese auf dem in Aufgabe 9.3 erzeugten Baum.

Aufgabe 11: Binomialkoeffizienten **3+3+4 Punkte**

Der Binomialkoeffizient $\binom{n}{m}$ mit $m, n \in \mathbb{N}$ und $n \geq m$ ist wie folgt rekursiv definiert:

$$\binom{n}{0} := 1 \qquad \binom{n}{n} := 1 \qquad \binom{n}{m} := \binom{n-1}{m-1} + \binom{n-1}{m}$$

1. Schreiben sie eine *rekursive* Modula-3 Prozedur, die Binomialkoeffizienten gemäß obiger Definition berechnet.
2. Schreiben sie eine *nicht-rekursive* Modula-3 Prozedur, die den Binomialkoeffizienten bestimmt, ohne ein Zwischenergebnis mehrfach zu berechnen.
3. Wie groß sind die Laufzeit- und Speicherplatzkomplexitäten ihrer beiden Prozeduren?

Aufgabe 12: Maximale Teilfolge **1+2+7 Punkte**

Gegeben sei eine Liste ganzer Zahlen a_k der Länge n ($k \in \{1, \ldots, n\}$). Die Teilfolgensumme von a in den Grenzen $i \leq j$ ist folgendermaßen definiert:

$$S_a(i, j) := \sum_{k=i}^{j} a_k$$

Gesucht wird die Teilfolge $a_i, \ldots, a_j$, deren Summe $S_a(i, j)$ maximal ist; wir nennen diese die *maximale Teilfolge*.

Beispiel:

Der schraffierte Bereich stellt für das gegebene Beispiel die maximale Teilfolge dar ($i = 3,\ j = 7$).

1. Wieviele Teilfolgen existieren überhaupt?
2. Geben Sie eine untere Schranke für die Komplexität eines Verfahrens zur Bestimmung der maximalen Teilfolge an.
3. Entwerfen und implementieren Sie einen Algorithmus zur Bestimmung der maximalen Teilfolge mit möglichst geringer Komplexität. Testen Sie ihren Algorithmus auf den unten angegebenen Daten.

Beispieldaten für die Aufgaben 9, 10 und 12:

18	-5	-8	-3	121	-30	100	-89	20	-100	10	105	-34	2
-105	200	300	-611	500	-222	-299	-495	-125	501	-798	-30	450	13

Übung am 2. Mai

Die nächste Übung findet am Dienstag, den 2. Mai statt. Bitte entnehmen Sie folgender Tabelle wann und wo sich Ihre Übungsgruppe trifft. Wenn es Ihnen nicht möglich ist, zu dem für Ihre Gruppe benannten Ersatztermin zu kommen, so können Sie ausnahmsweise eine andere Gruppe ihrer Wahl besuchen. Geben Sie dort bitte an, zu welcher Gruppe Sie normalerweise gehören. Wenn Sie nicht in Ihre eigene Übungsgruppe gehen, geben Sie Ihre Lösungen zum ersten Übungsblatt bitte am Ende der Vorlesung bei einem der Assistenten ab.

		regulärer Termin		Ausweichtermin am 2.5.	
Gruppe	Betreuer	Uhrzeit	Raum	Uhrzeit	Raum
1	Stefan Jacobs	08.15 - 09.45 Uhr	SG 404	08.30 - 09.45 Uhr	RS 107
2	Rüdiger Bartsch	08.15 - 09.45 Uhr	SG 13	08.30 - 09.45 Uhr	SG 12
3	Markus Klinke	12.15 - 13.45 Uhr	SG 202	08.30 - 09.45 Uhr	SG 13
4	Ingo Zander	12.15 - 13.45 Uhr	SG 203	08.30 - 09.45 Uhr	RS 108
5	Alexander Eiken	12.15 - 13.45 Uhr	SG 512	08.30 - 09.45 Uhr	001
6	Michael Evertz	12.15 - 13.45 Uhr	SFo 14	08.30 - 09.45 Uhr	004
7	Sven Johr	12.15 - 13.45 Uhr	6019	08.30 - 09.45 Uhr	004
8	Arndt Baars	12.15 - 13.45 Uhr	6124	08.30 - 09.45 Uhr	SG 12
9	Achim Lücking	15.45 - 17.15 Uhr	SG 202	13.30 - 14.45 Uhr	5056
10	Rüdiger Bartsch	15.45 - 17.15 Uhr	SG 13	14.45 - 16.00 Uhr	5056
11	Macherey / Bisani	15.45 - 17.15 Uhr	SG 413	8.30 - 09.45 Uhr	6124

Hinweis: Die Räume 001 und 004 befinden sich in der Eilfschornsteinstr. 7, der Raum 6124 am Lehrstuhl für Informatik VI, Ahornstr. 55, Erweiterungsbau E2, 1. Etage.

Die folgenden Präsenzaufgaben werden in den Übungsgruppen am Dienstag, den 2. Mai besprochen. Es werden daher keine Punkte vergeben.

Aufgabe 13: Definitionen

$$
\begin{aligned}
\Theta(g) &:= \{f \mid \exists c_1 > 0, \exists c_2 > 0, \exists\, n_0 > 0 : \quad 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \,\forall\, n \ge n_0\} \\
O(g) &:= \{f \mid \exists\, c > 0, \exists\, n_0 > 0 : \quad 0 \le f(n) \le cg(n) \,\forall\, n \ge n_0\} \\
\Omega(g) &:= \{f \mid \exists\, c > 0, \exists\, n_0 > 0 : \quad 0 \le cg(n) \le f(n) \,\forall\, n \ge n_0\} \\
o(g) &:= \{f \mid \forall c > 0\ \exists\, n_0 > 0 : \quad 0 \le f(n) < cg(n) \,\forall\, n \ge n_0\} \\
\omega(g) &:= \{f \mid \forall c > 0\ \exists\, n_0 > 0 : \quad 0 \le cg(n) < f(n) \,\forall\, n \ge n_0\}
\end{aligned}
$$

- Zeigen Sie, dass obige Definition von $\Theta(g)$ zu der in der Vorlesung gegebenen äquivalent ist.
- Zeigen Sie, dass obige Definition von $o(g)$ zu der aus Aufgabe 3 äquivalent ist.

Aufgabe 14: Wachstumsklassen als Vergleichsrelation

Beweisen Sie ($f, g, h : \mathbb{N} \to \mathbb{R}^+$):

Transitivität: $g \in \Theta(h) \Rightarrow \Theta(g) \subseteq \Theta(h)$, und analog für O, o, Ω und ω

Reflexivität: $f \in \Theta(f)$, $f \in O(f)$, $f \in \Omega(f)$

Symmetrie: $f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$

Umkehrsymmetrie: $f \in O(g) \Leftrightarrow g \in \Omega(f)$, $f \in o(g) \Leftrightarrow g \in \omega(f)$

Finden Sie ein Gegenbeispiel zu: $f \notin o(g) \wedge f \notin \omega(g) \Rightarrow f \in \Theta(g)$

Aufgabe 15: Laufzeitanalyse

Welche Laufzeit- und Speicherplatzkomplexität hat die folgende Funktion? Wie kann man den Wert von `f` effizienter berechnen?

```
PROCEDURE f(n: INTEGER) : INTEGER =
VAR i, s: INTEGER ;
BEGIN
    IF n > 0 THEN
        s := 0 ;
        FOR i := 1 TO n DO
            s := s + f(n-1) ;
        END ;
        RETURN s ;
    ELSE
        RETURN 1 ;
    END ;
END f;
```

Aufgabe 16:

Warum sagt die Behauptung „Die Laufzeit von Algorithmus A ist mindestens $O(n^2)$“ nichts aus?

Aufgabe 17:

Beweisen Sie, dass für $f, g : \mathbb{N} \to \mathbb{R}^+$ gilt:

$$\max(f, g) \in \Theta(f + g)$$

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 5. Mai 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

3. Übung

Abgabe der Lösungen: 15. Mai 2000

Achtung: Ab dem 8. Mai 2000 gelten folgende Raumänderungen

Gruppe	Betreuer	Uhrzeit	Raum
6	Michael Evertz	12.15 - 13.45 Uhr	H212 (Intzestr. 1)
8	Arndt Baars	12.15 - 13.45 Uhr	SFo 14

Master-Theorem

Sei $f(n)$ eine Funktion und $T(n)$ definiert durch die Rekursion

$$T(n) = aT(n/b) + f(n),$$

wobei $a \geq 1$ und $b > 1$ reelle positive Konstanten sind. Dann läßt sich $T(n)$ in asymptotischer Schreibweise wie folgt ausdrücken:

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$, $\varepsilon > 0$, dann ist $T(n) = \Theta(n^{\log_b a})$.
2. Falls $f(n) = \Theta(n^{\log_b a})$, dann ist $T(n) = \Theta(n^{\log_b a} \cdot \log_b n)$
3. Falls $f(n) = \Omega(n^{log_b a + \varepsilon})$, $\varepsilon > 0$, und $af(n/b) \leq cf(n)$ mit $c < 1$ und genügend großem n, dann ist $T(n) = \Theta(f(n))$.

Aufgabe 18: Rekursionsgleichungen — 2+2+2+2+2 Punkte

Schätzen Sie folgende Rekursionsgleichungen möglichst genau ab. Formulieren Sie Ihre Antwort mit Hilfe der O- bzw. Θ-Notation.

1. $T(n) = 3T(\lfloor \frac{n}{3} \rfloor + 28) + n$
2. $T(n) = T(\frac{2n}{3}) + 1$
3. $T(n) = 3T(\frac{n}{4}) + n \log n$
4. $T(n) = 3T(\lfloor \sqrt[3]{n} \rfloor) + \log_3 n$

Aufgabe 19: Rekursionsgleichungen — 2+2 Punkte

Lösen Sie folgende Rekursionsgleichungen exakt:

1. $T(0) = \alpha$, $T(1) = \beta$
   $T(n) = (1 + T(n-1))/T(n-2)$
2. $T(0) = 5$
   $2T(n) = nT(n-1) + 3(n!)$

Aufgabe 20: N-Damen Problem **5 Punkte**

Auf einem Schachbrett der Größe $N \times N$ sind N Damen so zu plazieren, daß sie sich nicht gegenseitig schlagen können. (Die Dame zieht auf ein beliebiges anderes Feld entlang der Linie, Reihe oder einer der Diagonalen, auf welcher sie steht.)

1. Implementieren Sie einen rekursiven Algorithmus, der das Problem löst.
2. Testen Sie Ihren Algorithmus für $N = 5$ und $N = 8$. Wieviele Möglichkeiten gibt es jeweils, die Damen auf dem Schachbrett unter der oben genannten Einschränkung zu plazieren?

Aufgabe 21: Rucksack-Problem **1+3+2+4+3+3 Punkte**

Einbrecher Ede und sein Komplize Greedy planen, das Museum Ludwig auszurauben. Dort sind folgende Exponate ausgestellt:

	Exponat	Gewicht [kg]	Wert [TDM]
1	Statue von Arno Bräker	8	47
2	Ming-Vase	17	100
3	Siebdruck von Andy Warhol	7	41
4	Römische Münzen	6	35
5	Venus von Milo (Leihgabe des Louvre)	25	150
6	Filz mit Fett von Joseph Beuys	10	45
7	Verhüllte Printe von Christo	6	34
8	Graffiti auf Beton	15	60
9	mittelalterlicher Meßkelch	7	40
10	roter Hering	6	12

Ede muß die Beutestücke in einem Rucksack transportieren und kann maximal 30 kg tragen. Beide Ganoven streiten darüber, welche Gegenstände mitgenommen werden sollen.

Greedy schlägt vor, die Gegenstände nach ihrem Verhältnis von Wert zu Gewicht zu ordnen und diese in absteigender Reihenfolge einzupacken bis der Rucksack voll ist.

Ede verläßt sich lieber auf eine Backtracking-Strategie: Hierbei wird jeder Gegenstand probeweise eingepackt und das Verfahren rekursiv für den verbleibenden Platz angewendet.

Da sich beide nicht auf ein Verfahren einigen können, beschließen sie, den Informatik-Studenten Karl Hacker zu fragen. Karl will das Problem mittels dynamischer Programmierung lösen und definiert dazu folgende Hilfsgröße: $W(i, g)$ ist der Wert der wertvollsten Zusammenstellung von Kunstwerken, die genau g kg wiegt und sich nur aus den Stücken $1, \ldots, i$ zusammensetzen darf.

1. Welchen Gesamtwert erzielt man mit der Greedy-Strategie?
2. Implementieren Sie Edes Backtracking-Strategie und bestimmen Sie den Wert der wertvollsten Zusammenstellung.
3. Formulieren Sie die Rekursionsgleichung für $W(i, g)$.
4. Implementieren Sie einen Algorithmus, der die Hilfsgröße rekursiv auswertet. Verbessern Sie diese Implementierung durch Memoization.
5. Lösen Sie das Problem mittels dynamischer Programmierung.
6. Vergleichen Sie ihre Programme hinsichtlich der Laufzeit. Welche Laufzeitkomplexitäten haben die vier Algorithmen in Abhängigkeit von der Zahl der Gegenstände und dem zulässigen Gesamtgewicht?

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 12. Mai 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

4. Übung

Abgabe der Lösungen: 22. Mai 2000

Aufgabe 22: Polynommultiplikation **3+1+3 Punkte**

Es seien $p(X)$ und $q(X)$ ganzzahlige Polynome vom Grad $N-1$, $N \in \mathbb{N}$ (N sei eine Zweierpotenz, d.h. $N = 2^k$, $k \in \mathbb{N}$). Die Polynome werden nun wie folgt zerlegt:

$$\begin{aligned} p(X) = p_l(X) + X^{N/2} \cdot p_r(X) \quad \text{mit} \quad & p_l(X) := a_0 + a_1 \cdot X + \ldots + a_{N/2-1} \cdot X^{N/2-1}, \\ & p_r(X) := a_{N/2} + a_{N/2+1} \cdot X + \ldots + a_{N-1} \cdot X^{N/2-1}, \\ q(X) = q_l(X) + X^{N/2} \cdot q_r(X) \quad \text{mit} \quad & q_l(X) := b_0 + b_1 \cdot X + \ldots + b_{N/2-1} \cdot X^{N/2-1}, \\ & q_r(X) := b_{N/2} + b_{N/2+1} \cdot X + \ldots + b_{N-1} \cdot X^{N/2-1} \end{aligned}$$

Für das Produkt der beiden Polynome gilt nun

$$\begin{aligned} r(x) &:= p(X) \cdot q(X) \\ &= p_l(X) \cdot q_l(X) + \Big[p_l(X) \cdot q_r(X) + p_r(X) \cdot q_l(X) \Big] \cdot X^{N/2} + p_r(X) \cdot q_r(X) \cdot X^N \end{aligned}$$

Dies liefert vier Teilprobleme der Größe $N/2$.

1. Stellen Sie die Rekursionsgleichung für die Anzahl der Koeffizientenmultiplikationen bei Problemgröße N auf und bestimmen Sie ihre Lösung.
2. Gibt es eine Verbesserung hinsichtlich der Anzahl benötigter Koeffizientenmultiplikationen gegenüber einem „naiven“ Verfahren? Begründen Sie kurz Ihre Antwort.
3. Wieviele Polynommultiplikationen sind erforderlich, wenn die folgende Zerlegung verwendet wird:

   $$p(X) \cdot q(X) = z_l(X) + \Big[z_n(X) - z_l(X) - z_r(X) \Big] \cdot X^{N/2} + z_r(X) \cdot X^N \qquad \text{mit}$$
   $$z_l(X) := p_l(X) \cdot q_l(X), \qquad z_r(X) := p_r(X) \cdot q_r(X), \qquad z_n(X) := \Big[p_l(X) + p_r(X) \Big] \cdot \Big[q_l(X) + q_r(X) \Big]$$

   Geben Sie wiederum eine Rekursionsgleichung zur Bestimmung der Anzahl benötigter Koeffizientenmultiplikationen bei Problemgröße N an und bestimmen Sie die Lösung.

Aufgabe 23: Divide and Conquer **4 Punkte**

Eine Menge $\{a_1, \ldots, a_n\}$ von n natürlichen Zahlen wird durch ein Array repräsentiert in dem die Elemente in aufsteigender Reihenfolge abgelegt sind. Entwerfen und implementieren Sie einen Algorithmus nach der *Divide-and-Conquer*-Strategie, der testet ob eine Zahl in der Menge enthalten ist. Welche Laufzeitkomplexität hat Ihr Verfahren?

Aufgabe 24: Matrixkettenprodukt **3+3+1+3 Punkte**

Gegeben seien n Matrizen A_i der Dimensionen $r_{i-1} \times r_i$. Finden Sie eine Klammerung des Kettenprodukts $A_1 \cdot A_2 \cdots A_n$, die die Anzahl der (skalaren) Multiplikationen minimiert. Benutzen Sie die folgende

Kostenfunktion:

$$m(i,j) = \begin{cases} 0 & \text{für} \quad i = j \\ \min\limits_{i \le k < j} \{m(i,k) + m(k+1,j) + r_{i-1} r_k r_j\} & \text{für} \quad i < j \end{cases}$$

1. Implementieren Sie jeweils eine rekursive Funktion mit und eine ohne Memoization, die die optimale Klammerung bestimmt.

2. Implementieren Sie ein Programm, das die optimale Klammerung mittels dynamischer Programmierung bestimmt.

3. Testen Sie Ihre Programme an folgendem Dimensionsvektor: $r = (5, 9, 3, 15, 7, 30, 6)$. Wieviele (skalare) Multiplikationen sind nötig, wenn man die Matrizen von links nach rechts miteinander mulipliziert? Wieviele im Fall der optimalen Klammerung?

4. Messen Sie die Laufzeit Ihrer Programme für unterschiedliche Werte von n und dem Dimensionsvektor $(r_1, \ldots, r_n)$ mit $r_i = \lceil 5 \sin(i/4) + 15 \rceil$. Stellen Sie Ihre Ergebnisse graphisch dar, d.h. tragen Sie die Laufzeit $T(n)$ für jeden der drei Algorithmen gegen n ab.

Aufgabe 25: Raktisch vs. Retisch **4 Punkte**

Prof. Dr. P. Raktisch hat für ein bestimmtes Problem einen Algorithmus entwickelt und implementiert, der für eine Eingabe der Größe n eine Laufzeit $O(n^4)$ hat. Prof. Dr. Theo Retisch hat für dasselbe Problem ein Divide & Conquer-Verfahren entwickelt, das eine Eingabe der Größe n in 80 Teilprobleme der Größe $\lceil \frac{n}{3} \rceil$ zerlegt. Für die Verwaltung und Zerlegung benötigt das Divide & Conquer-Verfahren darüber hinaus noch eine gewisse Anzahl von weiteren Rechenschritten. Eine Analyse zeigt, daß die Anzahl $w(n)$ der weiteren Rechenschritte für eine Eingabe der Größe n durch

$$w(n) = 26\, w\left(\left\lceil \frac{n}{3} \right\rceil \right)$$

gegeben ist. Nun ist zwischen Prof. Raktisch und Prof. Retisch ein Streit darüber entbrannt, welches das bessere Verfahren ist. Jeder der beiden behauptet, daß sein Verfahren das bessere sei. Schlichten Sie den Streit, indem Sie aufzeigen, welcher der beiden Algorithmen das hinsichtlich der Laufzeit bessere Verfahren ist.

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 19. Mai 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

5. Übung

Abgabe der Lösungen: 29. Mai 2000

Aufgabe 26: Algebraische Spezifikation abstrakter Datentypen **1+1+2 Punkte**

Versuchen Sie eine algebraische Spezifikation des abstrakten Datentyps *Queue* zu formulieren, indem Sie die Sorten (Datentypen), Operationen (zur Beschreibung der Syntax des Datentyps) und die Axiome (zur Beschreibung der Semantik des Datentyps) definieren.

Aufgabe 27: Stabilität von Sortierverfahren **2+2 Punkte**

Ein Sortieralgorithmus heißt *stabil*, wenn die relative Ordnung der Elemente mit gleichen Schlüsseln durch den Sortieralgorithmus nicht verändert wird.

a) Ist Bubblesort ein stabiler Algorithmus?

b) Ist Shellsort ein stabiler Algorithmus?
Gehen Sie hierbei von folgender Implementation aus.

```
PROCEDURE ShellSort(VAR a:  ARRAY[1..n] OF ItemType)
VAR i, j, incr:  CARDINAL;
BEGIN
   incr := n DIV 2; (* Schrittweite *)
   WHILE (incr > 0) DO
      FOR i:=incr + 1 TO n DO
         j := i - incr;
         WHILE (j > 0) DO
             IF (a[j] > a[j+incr]) THEN
                swap (a[j], a[j+incr]);
                (* swap () vertauscht die beiden Elemente *)
                j := j - incr;
             ELSE
                j := 0; (* Abbruch der WHILE-Schleife *)
             END; (* IF *)
         END; (* WHILE *)
      END; (* FOR *)
      incr := incr DIV 2;
   END; (* WHILE *)
END ShellSort;
```

Begründen Sie Ihre Antwort!

Aufgabe 28: BubbleSort **3 Punkte**

Betrachten Sie die folgende Variante von BubbleSort und bestimmen Sie die Anzahl der Bewegungen und Vergleiche für den *Best-*, *Average-* und *Worst-Case*.

```
PROCEDURE BubbleSort (VAR a : ARRAY OF ItemType) =
  VAR b : BOOLEAN;
BEGIN
  REPEAT
    b := TRUE;
    FOR i:= FIRST(a) TO LAST(a)-1 DO
      IF a[i] > a[i+1] THEN
        b := FALSE;
        swap(a[i], a[i+1]);
      END;
    END;
  UNTIL b;
END BubbleSort;
```

Gibt es irgendwelche Verbesserungen gegenüber dem in der Vorlesung vorgestellten Verfahren?

Aufgabe 29: Quicksort **4 Punkte**

Führen Sie das in der Vorlesung behandelte Sortierverfahren Quicksort manuell auf den folgenden Sequenzen von Schlüsseln durch, wobei Sie stets das letzte (rechte) Element $a[r]$ des zu sortierenden Teil-Arrays als Pivot-Element verwenden sollen. Geben Sie dabei sämtliche Zwischenschritte an.

- 16, 5, 2, 7, 4, 8, 10, 12, 11, 6, 13, 14, 15, 3, 9, 1
- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
- 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Welche Möglichkeiten gibt es, das Verhalten des Algorithmus zu verbessern?

Aufgabe 30: Insertion Sort **4 Punkte**

Implementieren Sie das Verfahren „Sortieren durch Einfügen“, wobei statt eines Arrays eine doppelt verkettete Liste verwendet werden soll. Dabei können Sie die Methoden der Schnittstelle aus Aufgabe 8 verwenden. Testen Sie Ihre Implementierung auf der folgenden Sequenz von Schlüsselelementen:

7, 21, 13, 8, 26, 4, 9, 1, 15, 8, 3, 5, 6

Aufgabe 31: Paarsuche **5 Punkte**

Gegeben sei eine sortierte Liste L von n natürlichen Zahlen und eine natürliche Zahl x. Gesucht wird ein Paar (a, b) von Elementen aus L, für das gilt: $x = a + b$.

Implementieren Sie einen Algorithmus, der diese Aufgabe in $O(n)$ Schritten löst. Dabei soll das Verfahren (bis auf die Speicherung der Liste) nur konstanten Platz benötigen. Der Algorithmus soll ein solches Paar zurückliefern, falls eines existiert.

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 26. Mai 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

6. Übung

Abgabe der Lösungen: 5. Juni 2000

Aufgabe 32: Heapsort **5 Punkte**

Führen Sie das in der Vorlesung behandelte Sortierverfahren Heapsort manuell auf der folgenden Sequenz von Schlüsseln durch. Geben Sie dabei sämtliche Zwischenschritte für beide Phasen des Algorithmus (Heapaufbau und Sortierung) an.

16, 5, 2, 7, 4, 8, 10, 12, 11, 6, 13, 14, 15, 3, 9, 1

Aufgabe 33: Verbessertes Quicksort **4+4+4 Punkte**

Die Laufzeit von QuickSort kann in der Praxis dadurch verbessert werden, in dem man den Geschwindigkeitsvorteil von Insertionsort auf „fast sortierten“ Folgen ausnutzt. Wenn Quicksort auf eine Teilfolge mit weniger als k Elementen angewendet wird, tut man einfach nichts. Anschließend (wenn der oberste QuickSort-Aufruf beendet ist) läßt man InsertionSort über die ganze Folge laufen.

1. Begründen Sie, daß dieses Verfahren eine Zeit von $O(nk + n \operatorname{ld}(n/k))$ benötigt.
2. Implementieren Sie obigen Algorithmus.
3. Testen Sie ihre Implementierung auf einer (genügend) langen Folge von (Pseudo-)Zufallszahlen, die Sie entweder mit einem bereits vorhandenen Zufallszahlengenerator oder nach der unten beschriebenen Kongruenzmethode generieren. Messen Sie die für den Sortiervorgang benötigte Zeit für unterschiedliche Werte von k und stellen Sie diese graphisch dar. Wo liegt das Optimum von k für Ihre Implementierung?

Aufgabe 34: Medianbestimmung **4+4 Punkte**

Der Median einer Folge von Zahlen $a_1, \ldots, a_n$ ist so definiert, daß es genau so viele Elemente gibt, die größer als der Median sind, wie solche die kleiner sind. Mit anderen Worten: Wenn man die Folge sortiert, steht der Median genau in der Mitte.

1. Entwerfen und implementieren Sie einen auf QuickSort basierenden Algorithmus zur Bestimmung des Medians. (Die Folge soll nicht vollständig sortiert werden.)
2. Zeigen Sie, dass Ihr Algorithmus im Mittel $O(n)$ Vergleiche benötigt. Wieviele Vergleiche braucht er im schlimmsten Fall?

Aufgabe 35: Zufallszahlen **1 Bonuspunkte**

Mit der Kongruenzmethode kann man sehr einfach eine Folge von quasi-zufälligen Zahlen erzeugen. Sie basiert auf folgender Rekursionsgleichung:

$$z_{n+1} = a \cdot z_n \mod m$$

Gute Werte für die Konstanten sind $a = 7^5 = 16807$ und $m = 2^{31} - 1 = 2147483647$. Der Startwert $z_0 > 0$ kann beliebig gewählt werden.

Implementieren Sie dieses Verfahren. Verwenden Sie dabei folgende Methode, um einen Überlauf der Variablen zur vermeiden:

$$a \cdot z \mod m = \begin{cases} (a \cdot z \mod q) - r\lfloor z/q \rfloor & \text{falls das } \geq 0 \text{ ist} \\ (a \cdot z \mod q) - r\lfloor z/q \rfloor + m & \text{sonst} \end{cases}$$

mit $q := \lfloor m/a \rfloor = 127773$ und $r := m \mod a = 2836$.

Aufgabe 36: In situ Bucketsort **7 Bonuspunkte**

Gegeben sei eine Folge von n Zahlen aus der Menge $\{1, \ldots, m\}$, wobei $m < n$ und $n, m \in \mathbb{N}$. Entwerfen und implementieren Sie eine Variante von Bucketsort mit folgenden Eigenschaften:

1. Der Algorithmus muß nicht stabil sein.
2. Er arbeitet in situ.
3. Außer dem Platz für das Histogramm und die Eingabedaten wird nur Speicher $O(1)$ benötigt.
4. Die Laufzeit beträgt wie beim Standard-Bucketsort $O(m + n)$.

Aufgabe 37: Erzeugen einer zufälligen Permutation **5 Bonuspunkte**

Entwerfen und implementieren Sie einen Algorithmus, der eine zufällige Permutation einer gegebenen Menge mit genau n Elementen erzeugt. Der Algorithmus soll eine garantierte Laufzeit von $O(n)$ haben und sicherstellen, daß jede mögliche Permutation mit der gleichen Wahrscheinlichkeit erzeugt wird. Verwenden Sie einen beliebigen Zufallszahlengenerator (z.B. aus Aufgabe 35) und gehen Sie davon aus, dass die damit erzeugten Zahlen gleichverteilt sind.

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 2. Juni 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

7. Übung

Abgabe der Lösungen: 19. Juni 2000

Aufgabe 37: Prioritätswarteschlangen — 4+2+4+1 Punkte

Unter einer Prioritätswarteschlange versteht man einen Datentyp, der folgende Operationen unterstützt:

Put(x) fügt Element x in die Warteschlange ein.

Min() gibt das Element mit dem kleinsten Schlüssel zurück.

GetMin() entfernt das Element mit dem kleinsten Schlüssel.

ReplaceMin(x) ersetzt das Element mit dem kleinsten Schlüssel durch x.

1. Welche Laufzeitkomplexitäten haben die genannten Operationen, wenn man die Prioritätswarteschlange mit den unten aufgeführten Datenstrukturen realisiert? Unterscheiden Sie gegebenenfalls zwischen worst- und best-case und begründen Sie Ihre Antworten!

   - unsortierte Liste
   - sortierte Liste
   - binärer Suchbaum
   - Heap

   Welche Datenstruktur eignet sich am besten?

2. Implementieren Sie Min und GetMin auf einem Heap.

3. Implementieren Sie Put auf einem Heap.

4. Warum sollte man ReplaceMin nicht als Hintereinanderausführung von GetMin und Put realisieren? Wie sonst?

Aufgabe 38: Interpolationssuche — 2+2+2 Punkte

Die Suche nach einem Element v in einer sortierten Liste $(a_1, \ldots, a_n)$ kann erheblich beschleunigt werden, wenn man etwas über die statistische Verteilung der Elemente weiß.

1. Schreiben Sie eine Prozedur zur Erzeugung einer sortierten Liste gleichverteilter Zufallszahlen. Greifen Sie gegebenenfalls auf Lösungen zu früheren Aufgaben zurück.

2. Modifizieren Sie die binäre Suche aus Aufgabe 23 so, daß das nächste zu betrachtende Element m nicht in der Mitte des betrachteten Bereichs $i, \ldots, j$ sondern durch *lineare* Interpolation gewählt wird:

$$m = (j - i)\frac{v - a_i}{a_j - a_i} + i$$

3. Testen Sie diesen Algorithmus für verschiedene n. Lassen Sie dabei zählen wieviele Schlüsselvergleiche ausgeführt werden. Mitteln Sie über genügend viele zufällig gewählte Listen und zu suchende Elemente und versuchen Sie auf diese Weise experimentell zu verifizieren, daß die Anzahl der Schlüsselvergleiche im average-case $O(\log \log n)$ ist.

Aufgabe 39: Multiplikative Hashfunktion **3 Punkte**

Das multiplikative Hashverfahren verwendet die folgende Hashfunktion:

$$h(x) = \lfloor m(Ax - \lfloor Ax \rfloor) \rfloor$$

Zeigen Sie, daß für $x, y \in \mathbb{N}$, $x \neq y$ und $A \in \mathbb{R} \setminus \mathbb{Q}$ gilt:

$$Ax - \lfloor Ax \rfloor \neq Ay - \lfloor Ay \rfloor$$

Aufgabe 40: Hashing **6 Punkte**

In einem kleinen Studentenkino mit $m = 19$ Plätzen wird ein Film der Filmförderung-NRW gezeigt. Um dem Ansturm auf die wenigen guten Plätze Einhalt zu gebieten, sollen die Plätze auf die in einer Schlange stehenden Studentinnen und Studenten mit einem Hashverfahren verteilt werden.

Als Schlüsselnummer werden dabei nur die beiden letzten Ziffern der Matrikelnummern der in der Schlange stehenden verwendet. Als Adressen stehen die Platznummern 0 bis 18 zur Verfügung. Welche Platznummern erhalten die folgenden in einer Schlange anstehenden Besucher

83, 52, 37, 52, 44, 91, 82, 95, 63, 12, 54, 41, 19, 01, 88, 15, 01, 17, 08

(angegben sind jeweils nur die beiden letzten Ziffern der Matrikelnummern) bei Verwendung

1) der Quersumme als Hashfunktion

2) der Division-Rest-Methode als Hashfunktion?

2) der multiplikativen Methode auf Aufgabe 39 mit $A := \frac{\sqrt{5}-1}{2}$.

Lösen Sie für alle angegebenen Hashverfahren evtl. auftretende Platzkollisionen durch

a) lineares Sondieren

b) quadratisches Sondieren

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 9. Juni 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

8. Übung

Abgabe der Lösungen: 26. Juni 2000

Aufgabe 41: Hashtabelle — 4+4 Punkte

Ein Telefonbuch soll Einträge der folgenden Form enthalten:

```
TYPE Entry = RECORD
         name   : TEXT ;
         number : INTEGER ;
     END ;
```

1. Erfinden Sie eine geeignete Hashfunktion, die einen Namen, also eine Zeichenfolge s beliebiger Länge, auf eine natürlich Zahl $h(s) \in \{0, \ldots, m-1\}$ abbildet ($m \in \mathbb{N}$).
2. Implementieren Sie die folgenden „Telefonbuch"-Operationen mit einer geschlossenen Hashtabelle:
   - `Insert(e: Entry)` fügt neuen Eintrag ein
   - `Search(n: TEXT) : INTEGER` sucht die Nummer zum Namen `n`, wird kein Eintrag gefunden, so wird -1 zurückgeliefert.
   - `Delete(n: TEXT)` entfernt den Eintrag zum Namen `n`

   Verwenden sie qudratisches Sondieren als Kollisionsstrategie und stellen Sie sicher, daß das Entfernen von Elementen die Kollisionsstrategie nicht durcheinanderbringt.

Aufgabe 42: Binärer Suchbaum: Bereichssuche — 2+1+2 Punkte

Gegeben sei ein binärer Suchbaum, sowie zwei Schlüsselwerte $i < j$. Gesucht sind alle im Baum gespeicherten Werte die in dem durch i und j begrenzten Bereich liegen.

1. Formulieren Sie einen Algorithmus, der eine solche *Bereichssuche* durchführt. Das heißt, es sollen alle Werte zwischen i und j ausgegeben werden.
2. Wie ist die Laufzeit Ihres Algorithmus?
3. Implementieren und testen Sie ihren Algorithmus. Greifen Sie gegebenenfalls auf die Lösung zu Aufgabe 9 zurück.

Aufgabe 43: Binärer Suchbaum — 6 Punkte

Wir definieren das *Gewicht* eines Baumes als die Anzahl seiner Knoten. Nehmen wir an, man fordert, daß in einem Binärenbaum für jeden Knoten das Gewicht seines einen Teilbaums höchstens doppelt so groß ist wie das Gewicht des anderen. Kann man mit dieser Forderung garantieren, daß die Höhe des Baums höchstens logarithmisch mit der Zahl der Knoten wächst? Beweisen Sie ihre Behauptung.

Aufgabe 44: Binärer Suchbaum **3+4 Punkte**

In einem „Suchbäumen mit schneller sequentieller Verarbeitung“ nutzt man die freien Nachfolgezeiger (die sonst mit `NIL` oder `z` belegt werden), um den direkten Zugriff auf die unmittelbaren Vorgänger und Nachfolger zu ermöglichen. Ein Knoten ohne linken Teilbaum enthält also in seinem linken Zeigerfeld einen Zeiger auf seinen Vorgänger, d.h. den Knoten mit dem nächstkleineren Schlüssel. Entsprechend enthält ein Knoten ohne rechten Teilbaum in seinem rechten Zeigerfeld einen Zeiger auf seinen Nachfolger, d.h. den Knoten mit dem nächstgößeren Schlüssel. Die Zweideutigkeit (Teilbaum oder Vorgänger/Nachfolger) wird durch zwei boolesche Felder `bl` und `br` aufgelöst (`TRUE`: Teilbaum, `FALSE`: Vorgänger bzw. Nachfolger).

Der Knotentyp ist folgendermaßen definiert:

```
TYPE Node = REF RECORD
               key    : INTEGER;  (* Schluessel            *)
               l, r   : Node ;    (* linker/rechter Zeiger *)
               bl, br : BOOLEAN;  (* Teilbaum vorhanden?   *)
            END;
```

Implementieren Sie die folgenden beiden Funktionen unter Verwendung dieser Definition:

1. `PROCEDURE Successor(n : Node) : Node`
   ermittelt den Nachfolger von `n`
2. `PROCEDURE Insert(VAR tree : Node ; v : INTEGER) ;`
   fügt `v` in den Baum `tree` ein

Aufgabe 45: Finden eines Duplikats **6+1+3 Bonuspunkte**

Gegeben sei eine Folge von $n > 2^d$ Zahlen aus dem Bereich $0, \ldots, 2^d - 1$, wobei $d \in \mathbb{N}$. Jede Zahl kann also mit d Binärstellen dargestellt werden. Außerdem gibt es offenbar mindestens eine Zahl, die mehrmals vorkommt.

1. Entwerfen Sie einen möglichst effizienten Algorithmus der eine Zahl *konstruiert*, die mindestens zweimal vorkommt. Hinweis: Es ist möglich, das Problem mit einem Laufzeitbedarf von $O(n \cdot d)$ und einem Speicherbedarf von $O(1)$ zu lösen.
2. Demonstrienen Sie ihren Algorithmus anhand eines kleinen Beispiels auf dem Papier.
3. Implementieren Sie ihren Algorithmus.

Lehrstuhl für Informatik VI
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 23. Juni 2000

M. Bisani, W. Macherey

Datenstrukturen und Algorithmen – Informatik I

9. Übung

Abgabe der Lösungen: 3. Juli 2000

Aufgabe 46: Optimaler Suchbaum **6+4 Punkte**

Für 5 gegebene Schlüsselelemente $k_1 < k_2 < \ldots < k_5$ soll ein optimaler Suchbaum konstruiert werden. Für das gesuchte Element x sind die folgenden Zugriffshäufigkeiten bekannt:

Ereignis:	Häufigkeit:
$x < k_1$	4
$x = k_1$	8
$k_1 < x < k_2$	1
$x = k_2$	3
$k_2 < x < k_3$	2
$x = k_3$	5
$k_3 < x < k_4$	4
$x = k_4$	10
$k_4 < x < k_5$	1
$x = k_5$	4
$k_5 < x$	3

1. Konstruieren Sie den zugehörigen optimalen Suchbaum mittels dynamischer Programmierung.
2. Schreiben Sie ein auf dynamischer Programmierung basierendes Programm, das nach Eingabe der Zugriffshäufigkeiten den optimalen Suchbaum berechnet.

Aufgabe 47: AVL-Baum **6+2 Punkte**

Die Namen der Monate seien lexikographisch geordnet, d.h. es gilt: April < August < Dezember < Februar < Januar < Juli < Juni < Mai < März < November < Oktober < September.

1. Geben Sie den AVL-Baum an, der entsteht, wenn man die Monatsnamen in der üblichen Reihenfolge (Januar, Februar, März, ...) in einen anfangs leeren Baum einfügt. Geben Sie auch die Zwischenstadien nach jeder Einfügeoperation an.
2. Entfernen Sie den Namen Juni aus dem in Teil 1 entstandenen Baum und geben Sie den neuen Baum an.

Aufgabe 48: B-Baum **4+2+4+4+4 Punkte**

Ein *B-Baum vom Grad b* (mit knotenorientierter Speicherung) hat folgende Eigenschaften (dabei ist b gerade, $b = 2a$ mit $a \in \mathbb{N}$) :

a) Jeder Knoten x speichert $n(x) \leq b - 1$ Schlüsselelemente $x.\text{key}[i]$ mit $i = 1, \ldots, n(x)$. Für jeden Knoten außer der Wurzel gilt $n(x) \geq a - 1$.

b) Die Schlüssel innerhalb eines Knotens x wachsen monoton: $x.\text{key}[i] \leq x.\text{key}[j]$ für $1 \leq i < j \leq n(x)$.

c) Jeder Knoten x hat entweder keinen oder genau $n(x) + 1$ Nachfolger.

d) Alle Knoten ohne Nachfolger haben die gleiche Tiefe.

e) Alle Schlüssel k, die im i-ten Teilbaum eines Knotens x gespeichert sind, liegen zwischen dem $i-1$-ten und i-ten Schlüssel: $x.\text{key}[i-1] \leq k \leq x.\text{key}[i]$. Für $i = 1$ gilt nur $k \leq x.\text{key}[1]$, für $i = n(x)$ gilt nur $x.\text{key}[n(x)] \leq k$.

Beispiel für einen B-Baum vom Grad $b = 4$ (sogenannter 2-3-4-Baum):

In Modula-3 könnte man die Knoten eines solchen Baums folgendermaßen deklarieren:

```
TYPE Node = REF RECORD
                n     : [1..b-1] ;  (* Anzahl der Schluessel *)
                key   : ARRAY [1..b-1] OF KeyType ;
                child : ARRAY [1..b]   OF Node ;
            END ;
```

Aufgaben:

1. Zeigen Sie, daß für die Höhe h eines B-Baums, der insgesamt N Schlüssel enthält, gilt

$$h \leq \log_a \frac{N+1}{2}$$

   (Ein Baum mit genau einem Knoten (der Wurzel) hat Höhe $h = 0$.)

2. Beschreiben Sie einen möglichst effizienten Algorithmus zum Auffinden eines Schlüsselelements in einem solchen B-Baum.

3. Ein Knoten x heißt *voll* genau dann, wenn er $b-1$ Schlüssel hat ($n(x) = b-1$). Erläutern Sie wie ein voller Knoten, dessen Vaterknoten nicht voll ist, aufgespalten werden kann.

   Entwerfen Sie ausgehend davon einen Algorithmus zum Einfügen eines Schlüssels in einen B-Baum. Hinweis: Der Algorithmus sollte dabei nie in einen Teilbaum mit voller Wurzel eintreten.

   Beschreiben Sie wie der Baum beim Einfügen von Elementen wächst und warum Bedingung (d) dabei nie verletzt wird.

4. Ein Knoten x heißt *minimal gefüllt* genau dann, wenn er $a-1$ Schlüssel hat ($n(x) = a-1$). Erläutern Sie wie zwei minimal gefüllte Geschwisterknoten verschmolzen werden können, wenn ihr Vaterknoten nicht minimal gefüllt ist. Erläutern Sie wie man durch Verschieben von Schlüsseln einen minimal gefüllten Knoten um einen Schlüssel vergrößern kann, wenn dieser einen nicht minimal gefüllten Geschwisterknoten hat.

   Entwerfen Sie ausgehend davon einen Algorithmus zum Entfernen eines Schlüssels aus einem B-Baum. Hinweis: Der Algorithmus sollte dabei nie in einen Teilbaum mit minimal gefüllter Wurzel eintreten.

5. Demonstrieren Sie Ihre Algorithmen an einem B-Baum vom Grad $b = 4$. Fügen Sie die Buchstaben

   K, A, C, B, J, H, I, F, D, E, G

   in dieser Reihenfolge in einen anfangs leeren Baum ein. Skizzieren Sie den Baum nach jeder Einfügeoperation. Entfernen Sie nun nacheinander die Elemente

   B, C, G, F

   und skizzieren Sie jeweils den entstandenen Baum.

Datenstrukturen und Algorithmen – Informatik I

10. Übung

Abgabe der Lösungen: 10. Juli 2000

Aufgabe 49: Topologisches Sortieren **3+4+3 Punkte**

Für einen azyklischen und gerichteten Graphen $G = (V, E)$ wird durch die Kantenmenge $E \subseteq V \times V$ auf der Menge der Knoten eine Halbordnung beschrieben. Zur Erinnerung: Sei $\mathcal{M}$ eine nicht-leere Menge und $\leq \subseteq \mathcal{M} \times \mathcal{M}$ eine binäre Relation auf $\mathcal{M}$, dann, heißt $\leq$ eine *Halbordnung* genau dann, wenn $\leq$ die folgenden Eigenschaften besitzt:

- **Reflexivität:** $\forall x \in \mathcal{M}: \quad x \leq x$
- **Transitivität:** $\forall x, y, z \in \mathcal{M}: \quad x \leq y \wedge y \leq z \implies x \leq z$
- **Antisymmetrie:** $\forall x, y \in \mathcal{M}: \quad x \leq y \wedge y \leq x \implies x = y$

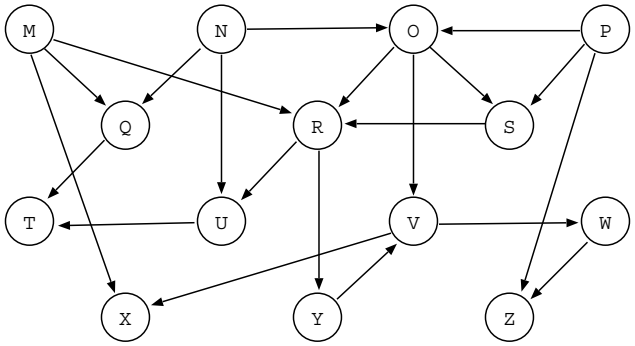
Beispielsweise kann man $(v, w) \in E$ dann interpretieren als „v ist teurer als w“.

Darüber hinaus gibt es den Begriff der *totalen Ordnung*. Eine totale Ordnung ist eine spezielle Halbordnung, die zusätzlich die Eigenschaft besitzt, daß beliebige Elemente aus der Grundmenge $\mathcal{M}$ bzgl. der Relation $\leq$ anordnenbar sind, d.h. für je zwei Elemente $x, y \in \mathcal{M}$ gilt stets

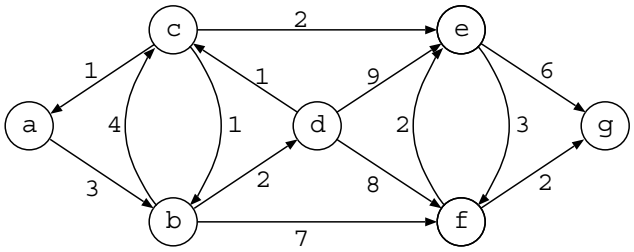
$$x < y \ \vee \ x = y \ \vee \ y < x$$

Eine topologische Sortierung eines azyklischen und gerichteten Graphen $G = (V, E)$ ist nun eine Abbildung $ord: \ V \longrightarrow \{1, \ldots, n\}$ mit $n = |V|$, so daß mit $(v, w) \in E$ auch $ord(v) < ord(w)$ gilt.

1. Beweisen Sie durch Induktion über die Knotenzahl $|V|$, daß für einen Graphen $G = (E, V)$ genau dann eine topologische Sortierung existiert, wenn G zyklenfrei ist. Hinweis: Für einen azyklischen Graphen gibt es mindestens einen Knoten v mit Eingangsgrad 0. Durch Entfernen von v entsteht ein um einen Knoten verkleinerter azyklischer Graph, an dessen topologische Sortierung v „vorne“ angefügt wird.
2. Formulieren Sie einen Algorithmus, der für einen azyklischen Graphen eine topologische Sortierung bestimmt, indem sukzessive Knoten mit Eingangsgrad 0 aus dem Graphen entfernt werden.
3. Implementieren Sie den Algorithmus und testen Sie ihn anhand des folgenden Graphen:



Implementieren Sie einen Algorithmus, der für einen gerichteten Graphen $G = (E, V)$ die transitive Hülle berechnet und zudem entscheidet, ob der Graph zusammenhängend ist. Testen Sie Ihren Algorithmus an dem folgenden Beispiel:

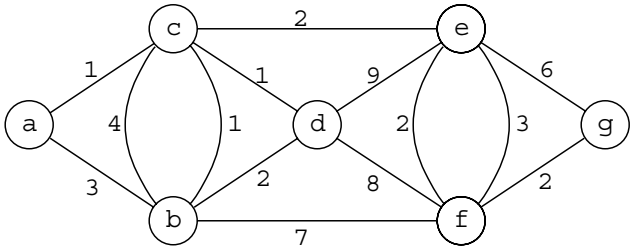


Aufgabe 51: Dijkstra-Algorithmus **4 Punkte**

Bestimmen Sie manuell ein Ranking der Knoten für den in Aufgabe 50 gegebenen Graphen durch Anwendung des Dijkstra-Algorithmus. Verwenden Sie als Startpunkt den Knoten a. Welche Kosten hat der günstigste Weg, der von a nach g führt?

Aufgabe 52: Minimaler Spannbaum **4 Punkte**

Bestimmen Sie manuell für den folgenden Graphen den minimalen Spannbaum:



Aufgabe 53: Implementierung: Dijkstra-Algorithmus **15 Bonuspunkte**

Implementieren Sie den Dijkstra-Algorithmus. Verwenden Sie Adjazenzlisten zur Repräsentierung der Kanten und eine (heap-basierte) Prioritätswarteschlange zur Verwaltung der Randknoten. In der Prioritätswarteschlange soll kein Knoten doppelt auftreten. Die dürfen Ergebnisse aus Aufgabe 37 verwenden.

Datenstrukturen und Algorithmen – Informatik I

Informationen und zusätzliche Aufgaben

Ankündigung

Die 10. Übung wird am 14. Juli (Gr, 10.00 Uhr) anstelle der Vorlesung besprochen.

Fragestunde:

In der vorlesungsfreien Zeit bieten wir 14-tägig eine **Fragestunde** an. Dabei kann alles, was klausurrelevant sein könnte, insbesondere natürlich Inhalte der Vorlesung und der Übungen, mit den verantwortlichen Assistenten diskutiert werden. Die Termine sind: 28. 7., 11. 8., 25. 8. und 1. 9. jeweils von 14.00–16.00 Uhr im Hörsaal AH VI.

Übungsschein:

Teilnehmer, die die Bedingungen für den Erhalt eines Übungscheins erfüllt haben, können diesen ab dem 24. Juli im Sekretariat des Lehrstuhls für Informatik VI abholen. Teilnehmern, die die Bedingungen knapp verfehlt haben, steht die Möglichkeit eines kurzen Prüfungsgesprächs mit den Assistenten offen. Wenden Sie sich in diesem Fall für die Terminabsprache direkt an uns.

Klausurtermin:

Datum	Prüfungsfach	Ort	Zeit
07.09.2000	Datenstrukturen (alte DPO)	Aula 2, AH IV, AH V	8.00–13.00 Uhr
07.09.2000	Informatik I (Programmierung sowie Datenstrukt. u. Algorithmen)	Aula 2, AH IV, AH V	8.00–13.00 Uhr

Zusätzliche Aufgaben

Die folgenden Aufgaben sollen eine Vorstellung vom Schwierigkeitsgrad der Vordiplomsklausur vermitteln. Die Aufgaben werden nicht korrigiert und es wird keine Musterlösung zur Verfügung gestellt.

Diese Aufgabensammlung kann keinen vollständigen Überblick über die klausurrelevanten Themen bieten. Achten Sie darauf, daß Ihnen alle bisher gestellten Übungsaufgaben vertraut sind!

Aufgabe 54: Θ-Notation

Es seien $f, g : \mathbb{N} \to \mathbb{R}^+$ mit

$$g(n) := 2n^2 + 7n + 97 \quad \text{und}$$
$$f(n) := n^2$$

Beweisen bzw. widerlegen Sie: $g \in \Theta(f)$

Prof. Dr. Theo Retisch und Prof. Dr. P. Raktisch streiten sich wieder. Diesmal hat Prof. Raktisch für ein bestimmtes Problem einen Algorithmus entwickelt und implementiert, der für eine Eingabe der Größe $n \in \mathbb{N} \setminus \{0\}$ eine Laufzeit von $\Theta(n \cdot \log n)$ hat. Prof. Theo Retisch hat für dasselbe Problem ein Divide & Conquer-Verfahren entwickelt, das eine Eingabe der Größe $n \in \mathbb{N} \setminus \{0\}$ in

$$2 \text{ Teilprobleme der Größe } \left\lceil \frac{n}{4} \right\rceil$$

zerlegt. Für die Verwaltung und Zerlegung benötigt das Divide & Conquer-Verfahren darüber hinaus noch eine gewisse Anzahl von weiteren Rechenschritten, welche für eine Eingabe der Größe n

$$7 \cdot \sqrt{n}$$

beträgt. Schlichten Sie den zwischen Prof. Retisch und Prof. Raktisch entbrannten Streit, indem Sie nachweisen, daß das Divide & Conquer-Verfahren von Prof. Retisch das (hinsichtlich der Laufzeit) bessere Verfahren ist. Weisen Sie die Verbesserung explizit nach!

Aufgabe 56: Sortieren durch Auswahl

(a) Sortieren Sie das Feld

$$84, 11, 36, 64, 99, 70, 56, 29, 44$$

mit dem in der Vorlesung vorgestellten Verfahren „Sortieren durch Auswahl“. Geben Sie hierbei auch diejenigen **Zwischenzustände** des Feldes an, die jeweils nach dem Vertauschen zweier Schlüsselelemente bei Anwendung des Sortierverfahrens entstehen.

(b) Wieviele Vergleiche zwischen Schlüsselelementen werden

(i) im Mittel

(ii) im „worst-case“ (d.h. im schlimmsten Fall)

beim Sortierverfahren „Sortieren durch Auswahl“ für eine Eingabe der Größe $n \in \mathbb{N} \setminus \{0\}$ durchgeführt? Geben Sie (ohne Beweis!) die größenordnungsmäßige Anzahl in Abhängigkeit von der Eingabegröße n an.

Aufgabe 57: AVL-Baum

Geben Sie den AVL-Baum an, der entsteht, wenn man mittels des in der Vorlesung vorgestellten Verfahrens nacheinander (in dieser Reihenfolge) die Zahlen

$$10, 35, 64, 9, 53, 47, 20, 28, 30, 15$$

in den anfangs leeren Baum einfügt. Geben Sie nach jedem Einfügen den entstandenen Zwischenbaum an. Sollte im Verlauf der Einfügeoperationen eine Rotation bzw. eine Doppelrotation durchgeführt werden, so geben Sie darüber hinaus auch den Baum vor und nach dieser Operation an.

Aufgabe 58: Komplexitätsklassen

1. Gegeben sind folgende Funktionen $f_i : \mathbb{N} \to \mathbb{R}^+$:

$$f_1(n) = n^2 + 1000n$$

$$f_2(n) = \begin{cases} n & , \text{ falls } n \text{ ungerade} \\ n^3 & , \text{ falls } n \text{ gerade} \end{cases}$$

$$f_3(n) = \begin{cases} n & , \text{ falls } n \leq 100 \\ n^3 & , \text{ falls } n > 100 \end{cases}$$

Untersuchen Sie für jedes Paar (f_i, f_j), ob $f_i(n) \in O\big(f_j(n)\big)$ und ob $f_i(n) \in \Omega\big(f_j(n)\big)$.

(a)

$$f(n) + g(n) \in \Theta\Big(\min\big(g(n), f(n)\big)\Big)$$

(b)

$$O\big(f(n)\big) + O\big(g(n)\big) \in O\big(|f(n)| + |g(n)|\big)$$

(c)

$$n! \in \Theta\Big(n^{n+1/2}\, e^{-n}\Big) \quad \text{mit } n! = \sqrt{2\pi n} \cdot \Big(\frac{n}{e}\Big)^n \cdot \big(1 + \Theta(1/n)\big)$$

Aufgabe 59: Rekursionsgleichungen

Schätzen Sie folgende Rekursionsgleichungen möglichst genau ab. Formulieren Sie ihre Antwort mit Hilfe der Θ-Notation.

1. $T(n) = T(\sqrt{n}) + 1$
2. $T(n) = 7T(n/3) + n^2$
3. $T(n) = 3T(n/2) + n$
4. $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

Aufgabe 60: Algorithmen-Analyse

Gegeben sei ein lineares Feld positiver reeller Zahlen, das in Modula-3 durch die folgenden Vereinbarungen beschrieben ist:

```
CONST   n = 500;
TYPE    feld = ARRAY[1..n] of REAL;
VAR     a: feld;
```

Gegeben seien außerdem eine Funktion $g : \mathbb{R} \to \{0, 1\}$, die als Werte 0 oder 1 liefert und die folgende Funktion `gtest`:

```
PROCEDURE gtest (li, re: CARDINAL) : CARDINAL =
  VAR m: CARDINAL;
  BEGIN
    IF (li > re) THEN
      RETURN 0;
    ELSE
      m := (li + re) DIV 2;
      RETURN gtest(li, m-1) + g(a[m]) + gtest(m+1,re);
    END
  END gtest;
```

1. Beschreiben Sie, welches Resultat die Funktion beim Aufruf `gtest(1, n)` für ein gegebenes Feld a liefert.
2. Ermitteln Sie größenordnungsmäßig die Anzahl der Additionen bei der Ausführung eines Aufrufs von `gtest(li, re)` im *worst case* in Abhängigkeit von $|\texttt{re} - \texttt{li}|$ mit Hilfe einer Rekursionsformel.
3. Geben Sie ein *iteratives* Verfahren zur Ermittlung des Funktionswertes `gtest` an. Verwenden Sie hierzu den gleichen Funktionskopf.

Sortieren Sie die Buchstabenfolge

$$\text{A, I, X, L, A, C, H, A, P, E, L, L, E}$$

mit folgenden Sortierverfahren:

1. Insertionsort
2. Quicksort
   Hinweis: Nehmen Sie jeweils das letzte Element als Pivotelement.
3. Heapsort
4. Bucketsort (Schlüsselbereich: A..Z)

Aufgabe 62: Optimaler Suchbaum

Gesucht wird ein optimaler Suchbaum mit fünf Knoten sowie den Zugriffshäufigkeiten der Knoten ($\beta_i, i = 1, ..., 5$) bzw. der "Nieten" (erfolglose Zugriffsversuche) ($\alpha_j, j = 0, ..., 5$) mit

$$\alpha_0 = 4,\ \beta_1 = 1,\ \alpha_1 = 0,\ \beta_2 = 3,\ \alpha_2 = 2,\ \beta_3 = 8,\ \alpha_3 = 5,\ \beta_4 = 10,\ \alpha_4 = 0,\ \beta_5 = 3,\ \alpha_5 = 3$$

Konstruieren Sie den optimalen Suchbaum und stellen Sie den Baum graphisch dar.

Aufgabe 63: Erweiterung binärer Suchbäume

In einem binären Suchbaum seien als Schlüsselwerte ganze Zahlen gespeichert. Es sollen Anfragen der folgenden Form unterstützt werden: „Ermitteln Sie für ein beliebiges ganzzahliges Suchintervall die Summe der darin enthaltenen Schlüsselwerte."

1. Welche zusätzliche Information muß in jedem Knoten verwaltet werden, damit Anfragen dieser Art in $O(\log n)$ Zeit durchgeführt werden können? Geben Sie den Anfragealgorithmus an!
2. Geben Sie entsprechend modifizierte Algorithmen für das Einfügen und Löschen an, die die zusätzliche Information mitändern.

Aufgabe 64: Kartesisches Produkt

Seien $L_1, ..., L_N$ Mengen von Wörtern. Gesucht ist das kartesische Produkt dieser Mengen. Nehmen Sie dazu an, daß es $p \cdot q$ Zeiteinheiten kostet, das kartesische Produkt einer Menge A der Größe p mit einer Menge B der Größe q zu bilden. Die Resultatmenge $A \times B := \{\, ab \mid a \in A, b \in B \,\}$ hat die Größe $p \cdot q$. Beispiel:

$$A = \{a, b\} \quad B = \{1, 2, 3\} \quad \mapsto \quad A \times B = \{a1, a2, a3, b1, b2, b3\}$$

Geben Sie einen effizienten Algorithmus an, der die optimale Klammerung für die Berechnung des kartesischen Produkts $L_1 \times L_2 \times \ldots \times L_N$ bestimmt.

Aufgabe 65: Min-/ Max-Suche

In einem Array a : ARRAY[1..n] OF INTEGER soll das Maximum und das Minimum gesucht werden.

1. Geben Sie einen Algorithmus an, der dieses Problem mit $O(n)$ Vergleichen löst.
2. Zeigen Sie: Gilt $n = 2m$ und besitzt a die Heap-Eigenschaft, d.h.

   $$\begin{aligned} a[i] &\geq a[2i] && \forall\ i = 1 \ldots m \quad \text{und} \\ a[i] &\geq a[2i+1] && \forall\ i = 1 \ldots m-1 \end{aligned}$$

   so kann man das Problem mit höchstens m Vergleichen lösen.

In einer Variante des binären Suchbaumes werden alle zu verwaltenden Schlüssel in den Blättern gespeichert. Die inneren Knoten dürfen beliebige Schlüsselwerte enthalten, die sich als *Wegweiser* eignen, d.h. die Suchbaumeigenschaft nicht verletzen. Die Blätter enthalten zusätzliche Zeiger auf das linke und rechte Nachbarblatt, bilden also eine doppelt verkettete Liste.

1. Geben Sie entsprechende Knotendeklarationen in einer Modula-3-ähnlichen Notation an.
2. Schreiben Sie eine Prozedur *insert* in einer Modula-3-ähnlichen Programmiersprache, die neue Elemente in den Baum einfügt.

Aufgabe 67: Suchen in Mengen

Formulieren Sie eine Prozedur in einer Modula-3-ähnlichen Programmiersprache, die testet, ob zwischen Mengen, die als geordnete Listen dargestellt werden, eine Inklusion (Teilmengenbeziehung) besteht. Die Richtung der Inklusion sei beim Aufruf nicht festgelegt. Vielmehr soll die Prozedur einen Zeiger auf die größere Menge zurückliefern, bzw. *NIL*, falls in keiner Richtung eine Inklusion besteht.

Aufgabe 68: Sortieren

Es soll ein Array der Länge n aufsteigend „*in situ*“ sortiert werden. Die Array-Elemente können nur zwei verschiedene, unbekannte Werte eines Wertebereichs annehmen, auf dem eine totale Ordnung definiert ist. Geben Sie einen Sortieralgorithmus für diesen speziellen Fall an, der eine Laufzeit von $O(n)$ hat. Das Sortierverfahren braucht nicht stabil zu sein.

Aufgabe 69: Suchbäume

Die Buchstabenfolge

I, N, F, O, R, M, A, T, J, K

soll in folgende Suchbäume eingetragen werden:

1. binärer Suchbaum
2. AVL–Baum
3. (2,4)–Baum

Aufgabe 70: Rekursionsgleichungen

a) Schätzen Sie die Lösungen folgender Rekursionsgleichungen möglichst genau ab. Formulieren Sie ihre Antwort mit Hilfe der Θ-Notation und begründen Sie sie.

(i) $T(n) = 3\,T(n-2) + 2\,, \quad T(1) = T(2) = 2$

(ii) $T(n) = 7\,T\left(\frac{n}{3}\right) + n^2\,, \quad T(1) = 1$

b) Beweisen oder widerlegen Sie:

$$f(n) + g(n) = \Theta\Big(\min\big(f(n), g(n)\big)\Big)$$

Aufgabe 71: Algorithmenentwurf: Sortieren mit Vorsortierung

Eine zu sortierende Folge M von n Elementen liege aufgespalten in k verschiedenen Dateien vor, von denen jede in sich bereits sortiert ist. Geben Sie einen Algorithmus an, der M vollständig in $O(n\ log\ k)$ Zeit sortiert. Gehen Sie davon aus, daß k der zu sortierenden Records gleichzeitig in den Hauptspeicher passen. Begründen Sie, warum Ihr Verfahren von der geforderten Laufzeitkomplexität ist.

Aufgabe 72: Suchbäume

Prof. Dr. Theo Retisch glaubt, eine interessante Eigenschaft binärer Suchbäume entdeckt zu haben und stellt die folgende Behauptung auf: Angenommen, die Suche nach einem Schlüssel k endet in einem Blatt

Suchpfad liegen, die Menge B enthält Schlüssel, die auf dem Suchpfad liegen und die Menge C enthält die Schlüssel, die rechts des Pfades liegen. Dann behauptet Prof. Retisch, daß für alle beliebigen $a \in A, b \in B$ und $c \in C$ gilt:

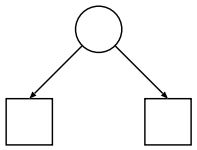
$$a \leq b \leq c$$

Beweisen oder widerlegen Sie die Aussage des Professors.

Aufgabe 73: Fibonacci-Bäume

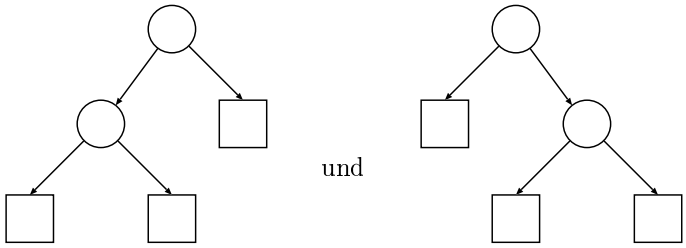
Für eine Zahl $h \in \mathbb{N}\backslash\{0\}$ ist die Menge der *Fibonacci-Bäume der Höhe h* induktiv wie folgt definiert:

- Der Baum

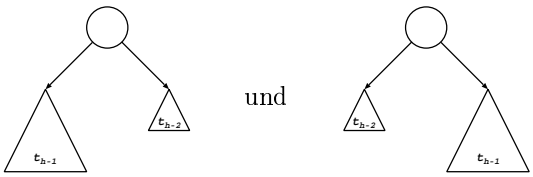
  ist ein Fibonacci-Baum der Höhe 1.

- Die Bäume

  sind Fibonacci-Bäume der Höhe 2.

- Ist für $h > 2$ der Baum t_{h-1} ein Fibonacci-Baum der Höhe $h-1$ und t_{h-2} ein Fibonacci-Baum der Höhe $h-2$, so sind auch die Bäume

  (also der Baum, dessen linker Teilbaum t_{h-1} ist und dessen rechter Teilbaum t_{h-2} beziehungsweise umgekehrt) Fibonacci-Bäume der Höhe h.

a) Geben Sie alle Fibonacci-Bäume der Höhe 3 an.

b) Zeigen Sie: Jeder Binäre Suchbaum, der ein Fibonacci-Baum ist, ist auch ein AVL-Baum.

c) Zeigen Sie durch Induktion über den Aufbau der Fibonacci-Bäume: Alle Fibonacci-Bäume der Höhe h haben genau $\text{Fib}(h+2) - 1$ innere Knoten. Hierbei bezeichnet $\text{Fib}(i)$ die i-te Fibonacci-Zahl: $\text{Fib}(0) = 0$, $\text{Fib}(1) = 1$, $\text{Fib}(2) = 1$, $\text{Fib}(i) = \text{Fib}(i-2) + \text{Fib}(i-1)$ für $i \geq 3$.

Aufgabe 74: Dynamische Programmierung

In einer Lagerhalle befinden sich N Typen von Möbelstücken, wobei jedes Möbelstück ausreichend oft vorhanden ist. Jeder Typ i hat einen Wert w_i und ein Gewicht g_i mit $g_i, w_i \in \mathbb{N}$. Ein Möbeltransporter mit einer Ladekapazität der Größe $M \in \mathbb{N}$ soll derart beladen werden, daß der Gesamtwert der zu transportierenden Möbelstücke maximal wird.

Verwenden Sie den Ansatz der dynamischen Programmierung und berechnen Sie die Werte der folgenden Hilfsgröße:

$W(i, m)$ = Wert der wertvollsten Zusammenstellung von Möbelstücken, die höchstens das Gewicht m hat und sich höchstens aus den Möbeltypen 1 bis i zusammensetzt.

a) Stellen Sie eine Rekursionsgleichung für $W(i, m)$ auf.

b) Schreiben Sie eine Prozedur in einer Modula-3-ähnlichen Programmiersprache zur möglichst effizienten Lösung des Möbeltransportproblems. Eine Lösung ist eine Sequenz von Möbelstücken, die zusammen höchstens Gewicht M haben und den größtmöglichen Gesamtwert aufweisen.

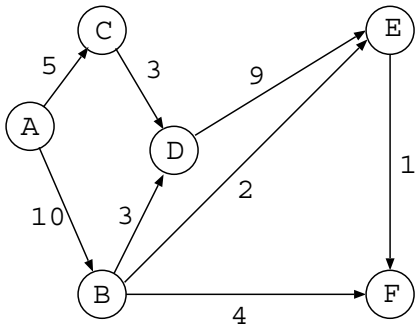
c) Geben Sie die Laufzeit Ihres Algorithmus in Abhängigkeit von M und N an.

d) Demonstrieren Sie Ihren Algorithmus anhand des folgenden Beispiels. Das maximale Ladegewicht des Transporters betrage $M = 15$ Gewichteinheiten.

Typ	i	1	2	3	4
Wert	w_i	3	4	7	11
Gewicht	g_i	2	3	5	7

Aufgabe 75: Graph-Algorithmen: Kürzeste Wege

Gegeben sei der folgende Graph G:



a) Bestimmen Sie mit einem geeigneten Verfahren aus der Vorlesung die kürzesten Wege vom Knoten A zu allen anderen Knoten. Geben Sie dabei alle Zwischenergebnisse an und konstruieren Sie explizit die besten Pfade. Erläutern Sie außerdem *kurz*, warum das Verfahren funktioniert.

b) Angenommen, Sie sollten die kürzesten Wege zwischen allen Knotenpaaren (i, j), $i, j \in \{A, B, \ldots, F\}$ bestimmen. Welches Verfahren würden Sie einsetzen, wenn die mehrfache Anwendung eines *Single-Source-Best-Path*-Algorithmus nicht erlaubt ist?

Aufgabe 76: Graph-Algorithmen: Topologisches Sortieren

Bestimmen Sie eine topologische Sortierung zu dem Graphen aus der vorherigen Aufgabe, indem Sie sukzessive Knoten mit Eingangsgrad 0 aus dem Grahpen entfernen. Geben Sie alle Zwischengraphen an.

Aufgabe 77: Heaps

a) Was ist der Unterschied zwischen der Eigenschaft „Binärer Suchbaum“ und der Heap-Eigenschaft?

Kann die Heap-Eigenschaft dazu verwendet werden, die Schlüssel eines Baums mit n Knoten in Zeit $O(n)$ in sortierter Reihenfolge auszugeben? Begründen Sie kurz Ihre Antwort.

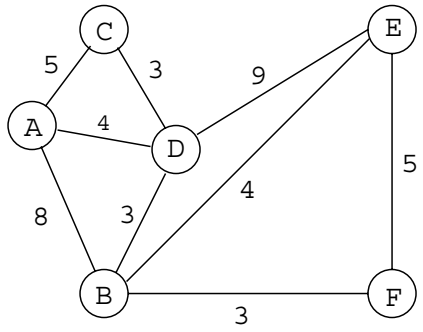
b) Zeichnen Sie die Zustände eines anfangs leeren Heaps, in den die Buchstabenfolge

H, E, A, P, S, O, R, T

dann die Heap-Eigenschaft wieder her. Wählen Sie zur graphischen Darstellung des Heaps die aus der Vorlesung bekannte Baumdarstellung.

Aufgabe 78: Graph-Algorithmen: Spannbäume

a) Gegeben sei der folgende ungerichtete Graph G:



Bestimmen Sie mit einem geeigneten Verfahren aus der Vorlesung einen minimalen Spannbaum von G. Erläutern Sie *kurz* ihr Vorgehen und geben Sie *alle* Zwischenergebnisse an.

b) Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph und T ein Spannbaum von G. Außerdem sei H ein beliebiger Teilgraph von G. Beweisen oder widerlegen Sie, daß der Schnitt von T und H ein Spannbaum von H ist.

Ein Schnitt von Graphen ist definiert als: $G \cap G' = (V \cap V', E \cap E')$, mit $G = (V, E)$ und $G' = (V', E')$

Aufgabe 79: Hashing

Fügen Sie die Schlüsselfolge $\{16, 44, 21, 5, 19, 22, 8, 33, 27, 30\}$ gemäß der Hashfunktion $h(k) = k \bmod 11$ in eine geschlossene Hashtabelle mit $b = 1$ (d.h. jeder Behälter der Hash-Tabelle kann genau ein Element aufnehmen) ein und stellen Sie das Ergebnis graphisch dar.

a) Verwenden Sie eine lineare Kollisionstrategie $h_i(k) = \big(h(k) + c \cdot i\big) \bmod 11$ mit $c = 1$.

b) Verwenden Sie nun eine quadratische Kollisionsstrategie $h_i(k) = \big(h(k) + i^2\big) \bmod 11$.

c) Diskutieren Sie *kurz* die Vor-/ und Nachteile von linearen bzw. quadratischen Kollisionstrategien. Warum ist die Wahl von i^2 in Aufgabenteil b) besonders geeignet?

d) Betrachten Sie nun die Hashfunktion $h(k) = k \bmod 14$ mit linearer Kollisionsstrategie. Geben Sie möglichst allgemein solche c an, die sich als ungeeignet erweisen.