Algorithmen und Programmierung 3, WS 2003/2004 — 1. Übungsblatt

Abgabe bis Donnerstag, 30. Oktober 2003

1. (i) (4 Punkte) Sortieren Sie die folgenden Laufzeiten aufsteigend. (Wenn $f(n) = O(g(n))$ ist, aber nicht $g(n) = O(f(n))$, dann soll $f(n)$ vor $g(n)$ kommen.)

   (a) n^3 (b) $\log_2 n$ (c) $1{,}8^n$ (d) n (e) 3^n (f) $\sqrt{n}$ (g) $n(\log_2 n)^2$ (h) n^2

   (ii) (4 Punkte) Wenn ein Programm die Laufzeit $f(n)$ aus Aufgabe (i) hat und die Computer in zehn Jahren um den Faktor 1000 schneller werden, dann kann man Probleme *welcher Größe* in derselben Zeit lösen, in der man heute Probleme der Größe (1) $n = 20$, (2) $n = 1000$ lösen kann?

2. (2 Punkte) Die Computer werden immer schneller, die Speicherelemente immer billiger. Wird der Entwurf von effizienten Algorithmen angesichts dieser Entwicklung in Zukunft an Bedeutung verlieren? Bei welchen Computeranwendungen werden Effizienzfragen eine größere/kleinere Rolle spielen?

   Diskutieren Sie diese Fragen. (mindestens 5–10 Zeilen)

3. (0 Punkte) Das folgende Programmstück ist ein Versuch, Sortieren durch Einfügen zu implementieren.

```
    void insertionSort(int[] a)
    { int x;
      for (int i = 0; i<a.length; i++)
      { † x = a[i];
        // Sortiere x zwischen a[0..i-1] ein
        // Bestimme zunächst die richtige Position j:
(*)     int j=i-1; † while (j>=0 && a[j-1]>x) j-- † ;
        // Nun verschiebe einen Teil des Feldes und füge x an dieser Stelle ein:
      † for (int k = i-1; k>=j; k--) a[k+1]=a[k];
        a[j]=x; †
      }
    }
```

   Stellen Sie dieses Programm richtig und fügen Sie an den durch † bezeichneten Stellen in Ihrem Programm Schleifeninvarianten in der Form vom Zusicherungen (assertions) ein, aus denen die Korrektheit des Programmes hervorgeht. (Ein formaler Beweis ist nicht erforderlich, aber die Zusicherungen müssen aussagekräftig sein und vor allem zutreffen.)

4. (0 Punkte) Kann man Sortieren durch Einfügen schneller machen, indem man die Einfügestelle `j` schneller findet als oben in Zeile (∗), zum Beispiel durch binäres Suchen? Wie ist es im besten und im schlechtesten Fall?

5. (0 Punkte) Zeigen Sie:

   (a) Wenn $f(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0$ ein Polynom vom Grad d mit positivem Leitkoeffizienten ($a_d > 0$) ist, dann ist
   $$f(n) = \Theta(n^d).$$

   (b) $\log_a n = \Theta(\log_b n)$ für $a, b > 1$.

   (c) Unter welchen Bedingungen folgt $g(n) = \Omega(f(n))$ aus $f(n) = O(g(n))$?

6. (0 Punkte) Welche der beiden folgenden Laufzeiten $f(n)$ und $g(n)$ ist für große Werte von n schneller, und welche für kleine n? Bei welchem Wert von n ändert sich die Antwort?

   (a) $f(n) = 10n(\log_2 n)^2$, $g(n) = 2n^{3/2}$

   (b) $f(n) = 5 \cdot 2^n$, $g(n) = 100n^2 \log_2 n$

7. (0 Punkte)

   (a) Beweisen Sie: Wenn $f(n) = O(g(n))$ ist, dann ist $f(n) + g(n) = O(g(n))$.

   (b) Beweisen Sie: $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$ für $f(n), g(n) > 0$.

   (c) Welche der folgenden Aussagen sind richtig (Begründen Sie Ihre Antworten):
   $n\sqrt{n} = O(n(\log n)^2)$; $n(\log n)^2 = O(n\sqrt{n})$; $n \log n \cdot (\log \log n) = O(n^2)$;
   $n^2 = O(n \log n \cdot (\log \log n))$; $n^2 \cdot 2^n = O(2^{n+2})$; $n^2 \cdot 2^n = O(3^n)$; $3^n = O(n^2 \cdot 2^n)$;

8. (4 Punkte) Finden Sie möglichst einfache Ausdrücke der Form $\Theta(\cdot)$ für folgende Funktionen:
   (a) $3n^2 - 4n + 32 + 27\, n \cdot \lceil \log_2 n \rceil / 2$; (b) $\max\{n\lceil \log_2 n \rceil, (\lceil \log_2 n \rceil)^4\}$; (c) $2^{2n + \lceil \log_2 n \rceil}$

9. (6 Punkte) Formulieren Sie *Sortieren durch Auswahl* in Java (als Methode wie in Aufgabe 3) oder in Haskell (als lauffähige Funktion).

10. (4 Punkte) Was passiert bei *Sortieren durch Einfügen*, wenn die gegebene Folge bereits (a) aufsteigend oder (b) absteigend sortiert ist? (c) Was passiert, wenn alle Elemente gleich sind? Ist es möglich, dass man bloß $O(n)$ Zeit benötigt?

11. (2 Punkte) Beantworten Sie die vorige Frage für *Sortieren durch Auswahl*.

12. (0 Punkte) Nehmen wir an, dass die Eingabefolge *fast sortiert* ist. Das heißt, dass jedes Element in der Ausgangsreihenfolge höchstens k Stellen von der endgültigen Position in der sortierten Reihenfolge entfernt ist. Geben Sie eine Schranke für die Laufzeit von *Sortieren durch Einfügen* in Abhängigkeit von n und k an.

13. (0 Punkte) Nehmen wir an, wir haben ein Liste mit n sortierten Elementen, gefolgt von k Elementen in beliebiger Reihenfolge. Welchens Sortierverfahren empfehlen Sie zum Sortieren der Gesamtliste für

    (a) $k = \Theta(1)$,

    (b) $k = \Theta(\log n)$,

    (c) $k = \Theta(\sqrt{n})$?

14. (0 Punkte) Wie kann man *Sortieren durch Einfügen* so anpassen, dass es für Folgen, die absteigend oder fast absteigend sortiert sind, möglichst schnell läuft?

15. (3 Punkte) Geben Sie alle möglichen topologischen Sortierungen für folgende Eingabe an: $n = 6$ und $\{(6,4),(3,5),(4,1),(3,4),(5,4),(2,1),(2,6),(3,6)\}$.

16. (0 Punkte) Was passiert beim in der Vorlesung angegebenen Algorithmus für *Topologisches Sortieren*, wenn ein Paar (i,j) in der Eingabe mehrfach auftritt? Was passiert, wenn ein Paar (i,i) auftritt?

17. (0 Punkte) Untersuchen Sie verschiedene Möglichkeiten, wie man die Liste der freien Elemente beim topologischen Sortieren verwalten kann, im Hinblick auf ihre Effizienz. Kann man auf diese Liste auch gänzlich verzichten?

    Welche Variablen oder Felder im Programm aus der Vorlesung[1] könnte man ohne Nachteil einsparen?

18. (5 Punkte) Erweitern Sie den Algorithmus zum topologischen Sortieren aus der Vorlesung, sodass bei der Existenz eines Kreises nicht einfach mit einer Meldung abgebrochen wird, sondern auch ein Kreis (als „Beweis“) ausgegeben wird.

    Beschreiben Sie Ihren erweiterten Algorithmus auf der Ebene von Pseudo-Code. (Zum Beispiel können Sie eine Schleife “für alle Elemente x ...” schreiben.)

    Erklären Sie Ihren Algorithmus *in Worten*, ohne notwendigerweise auf Details der Implementierung einzugehen. (Sie müssen nur das erklären, was gegenüber dem ursprünglichen Algorithms neu ist. Ein Korrektheitsbeweis ist nicht verlangt.)

19. (6 Punkte) Schreiben Sie ein Java-Programm für die vorige Aufgabe. Sie können das Programm aus der Vorlesung[1] erweitern.

    Ihr Programm sollte nicht mehr als $O(m + n)$ zusätzliche Zeit und nicht mehr als $O(n)$ zusätzlichen Speicher brauchen.

[1] `http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/TopoSort.java`

20. (2 Punkte) Beweisen Sie, dass man zum Finden eines Elementes in einer sortierten Liste mit n Elementen mindestens $\Omega(\log n)$ Vergleiche benötigt.

21. (a) (3 Punkte) Schreiben Sie eine Schnittstelle (`interface`) oder eine abstrakte Klasse `PWSchlange` für eine Prioritätswarteschlange, die mindestens die beiden Methoden `entferneMin` und `einfügen` enthält. Die Prioritätswarteschlange soll Objekte vom Typ `Comparable` verwalten.

    (b) (3 Punkte) Schreiben Sie eine Klasse `Halde`, die eine `PWSchlange` implementiert beziehungsweise erweitert![1]

22. (12 Punkte) Simulation einer Warteschlange in Java.
In einer Unfallambulanz müssen sich die ankommenden Patienten zunächst anmelden. Danach werden Sie in der Reihenfolge der Anmeldung von einer der $n = 2$ diensthabenden Ärztinnen befragt und untersucht. Gegebenfalls wird ein Röntgenbild angefertigt, und anschließend werden die Patienten behandelt.
Die Ankunft der Patienten ist ein Poisson-Prozess mit Rate $\lambda = 2\,\mathrm{h}^{-1}$. Das heißt, dass der Abstand von der Ankunft eines Patienten bis zur Ankunft des nächsten Patienten exponentialverteilt mit dem Mittelwert $1/\lambda = 30$ Minuten ist. Eine exponentialverteilte Zufallsvariable mit Mittelwert μ kann man mit der Formel `-`μ`*Math.log(Math.random())` erzeugen.
Die Zeit, die zur Anmeldung nötig ist, soll vernachlässigt werden. Die Dauer der ersten Untersuchung ist gleichverteilt im Intervall $[a\,..\,b]$ mit $a = 5\,\mathrm{min}$ und $b = 20\,\mathrm{min}$. Eine solche gleichverteilte Zufallsvariable kann man mit der Formel `a+(b-a)*Math.random()` erzeugen. Anschließend ist mit Wahrscheinlichkeit $p = 0{,}15$ die Behandlung beendet, und der Patient wird entlassen. Andernfalls wird der Patient zum Röntgen geschickt. Die Patienten werden dort in der Reihenfolge ihrer Ankunft an der Röntgenstation drangenommen. Die Zeit zur Erstellung eines Röntgenbildes ist gleichverteilt im Intervall $[a\,..\,b]$ mit $a = 5\,\mathrm{min}$ und $b = 10\,\mathrm{min}$ (unabhängig von der Dauer der Untersuchung). Die anschließende Behandlung ist wieder eine unabhängige Zufallsvariable im Intervall $[a\,..\,b]$ mit $a = 10\,\mathrm{min}$ und $b = 30\,\mathrm{min}$.
Untersuchen Sie nun folgende Varianten:

    (a) Jeder neue Patient wird der ersten freiwerdenden Ärztin zugeteilt. Nach dem Fertigstellen des Röntgenbildes muss der Patient von derselben Ärztin behandelt werden. Wenn eine Ärztin frei wird, nimmt sie als nächstes bevorzugt einen Patienten an, der von der Röntgenabteilung zurückkommt.

    (b) Wie oben, aber die Patienten, die vom Röntgen zurückkommen, reihen sich in dieselbe Schlange ein wie die neuen Patienten.

    Simulieren Sie diese Schlange von 7:00 bis 11:00 Uhr eines Tages. (Um 11:00 Uhr wird die Ambulanz geschlossen; es werden keine weiteren Patienten mehr angenommen, aber alle Patienten, die sich schon angemeldet haben, werden noch behandelt.)
    Bestimmen Sie für jede Variante die durchschnittliche Wartezeit eines Patienten, die maximale Wartezeit, und die Gesamtzeit, wie lange die Ärztinnen nichts zu tun hatten (von 7:00 Uhr bis zur Behandlung des jeweils letzten Patienten). Führen Sie je drei unabhängige Simulationsläufe durch.
    Schreiben Sie Ihr Programm so, dass es möglichst leicht zu ändern oder zu erweitern ist (zum Beispiel für unterschiedliche Werte von λ, p und n). Verwenden Sie ein Objekt vom Typ `PWSchlange` aus der vorigen Aufgabe zur Verwaltung der nächsten Ereignisse.
    Stellen Sie die Hierarchie aller Klassen und Schnittstellen, die Sie definiert haben, dar.

[1] siehe das Programm aus der Vorlesung:
`http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/heapsort.java`

23. (15 Punkte) Eine hash-Funktion zum Speichern ungeordneter Paare.
Finden Sie n ganze Zahlen $t_1, \ldots, t_n$, sodass alle Summen $t_i + t_j$ für $1 \leq i < j \leq n$ verschieden sind und in einem möglichst kleinen Intervall $\{a, a+1, \ldots, b\}$ enthalten sind, für $n = 5, 6, 7, 8, 9, 10$.

24. (3 Punkte) Speichern ungeordneter Paare in einem Array.
    (a) (0 Punkte) Wie kann man die Lösung der vorigen Aufgabe dazu verwenden, für eine feste Liste von n Objekten die Paare, die man aus je zwei dieser Objekte bilden kann, möglichst kompakt zu verwalten und zu speichern (wie in einer Hash-Tabelle), sodass man in konstanter Zeit auf jedes Paar $\{u, v\}$ zugreifen kann, wenn man u und v kennt?
    (b) (3 Punkte) Wie kann man dieses Problem auf andere Art mit einem Array der (optimalen) Länge $n(n+1)/2$ lösen?

25. (0 Punkte) Bestimmen des doppelten Elementes.
In einem Array $a_0, a_1, \ldots, a_n$ sind ganzzahlige Werte zwischen 1 und n gespeichert, und zwar kommt jede Zahl mindestens einmal vor. Daraus folgt, dass es genau eine Zahl geben muss, die doppelt vorkommt. Schreiben Sie ein Programm, das diese Zahl bestimmt. Das Programm soll lineare Laufzeit haben, auf das Array a nur *lesend* zugreifen, und nur konstanten zusätzlichen Speicher benötigen.[1]

26. (0 Punkte) Lösen Sie Aufgabe 23 für *geordnete* Paare: Finden Sie $2n$ ganze Zahlen $s_1, \ldots, s_n$ und $t_1, \ldots, t_n$ sodass alle Summen $s_i + t_j$ für $1 \leq i, j \leq n$ verschieden sind und in einem möglichst kleinen Intervall $\{a, a+1, \ldots, b\}$ enthalten sind.

27. Zusatzaufgabe (3 Zusatzpunkte). Finden Sie eine Formel für t_i in Aufgabe 23, die das Problem für allgemeines n löst (nicht unbedingt optimal in dem Sinn, dass das enthaltende Intervall kleinstmöglich ist). Analysieren Sie die Länge des enthaltenden Intervalls bei Ihrer Methode. (Für die beste abgegebene Formel gibt es bis zu 10 weitere Zusatzpunkte.) (Die gleiche Frage kann man natürlich auch für die geordneten Paare aus Aufgabe 26 stellen.)

28. (5 Punkte) Entfernen Sie die Endrekursion aus der Methode `zugroß` im Programm zur Verwaltung einer Halde.[2]

```
void zugroß(int i)
// a[i] ist möglicherweise größer als seine Nachfolger.
{   int kleinsterNachfolger;
    if (2*i+1 <= n) {
        if (a[2*i]<a[2*i+1]) kleinsterNachfolger = 2*i;
        else                 kleinsterNachfolger = 2*i+1;
    }
    else if (2*i <= n) kleinsterNachfolger = 2*i;
    else return;
    if (a[i] > a[kleinsterNachfolger]) {
        vertausche(i, kleinsterNachfolger);
        zugroß(kleinsterNachfolger);
    }
}
```

[1] Diese Aufgabe dient nur zur Unterhaltung und hat mit dem Stoff der Vorlesung nichts zu tun. Wenn Sie das Rätsel aus einer anderen Quelle bereits kennen, dann würde mich sehr interessieren, woher.

[2] Siehe Aufgabe 21 und das Heapsort-Programm aus der Vorlesung:
`http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/heapsort.java`

Algorithmen und Programmierung 3, WS 2003/2004 — 5. Übungsblatt

Abgabe bis Donnerstag, 27. November 2003

29. (a) (0 Punkte) In einem *vollen* binären Baum hat jeder innere Knoten zwei Kinder. Man kann einen beliebigen Binärbaum als vollen Binärbaum darstellen, indem man für jedes fehlende Kind eines Knotens ein neues Blatt (einen *externen* Knoten) einsetzt. Alle ursprünglichen Knoten werden zu inneren Knoten, und die Blätter repräsentieren die `null`-Zeiger im ursprünglichen Baum.
Beweisen Sie, dass in ein voller binärer Baum mit n Blättern $n-1$ innere Knoten hat.
    (b) (4 Punkte) Beweisen Sie, dass in einem vollen binären Baum mit n Blättern auf Tiefe $l_1, \dots, l_n$ die folgende Gleichung gilt:
$$\sum_{i=1}^{n} 2^{-l_i} = 1$$
    (c) (0 Punkte) Für jede Folge $l_1, \dots, l_n$ ganzer Zahlen, die obige Gleichung erfüllt, gibt es einen vollen binären Baum mit n Blättern auf Tiefe $l_1, \dots, l_n$.

30. (0 Punkte) In einem a-b-Baum speichert ein Knoten P mit k Kindern $k-1$ Schlüssel $v_1, \dots, v_{k-1}$, wobei v_i der kleinste Schlüssel im $(i+1)$-ten Teilbaum ist. Beweisen Sie, dass jeder Schlüssel fast genau einmal gespeichert wird. (Mit welcher Ausnahme?)

31. (a) (0 Punkte) Nehmen wir an, dass jeder innere Knoten genau 2 Kinder hat. Was können Sie aus der vorigen Aufgabe über die Beziehung zwischen der Anzahl der inneren Knoten und der Anzahl der Blätter schließen?
    (b) (0 Punkte) Welche Beziehung besteht im allgemeinen Fall (bei beliebigen Knotengraden) zwischen der Anzahl der inneren Knoten, der Anzahl der Blätter, und der Summe der Grade (Anzahlen der Kinder) der inneren Knoten? Gilt diese Beziehung auch für Bäume, die keine a-b-Bäume sind, wo also die Blätter auf verschiedenen Ebenen sein können?
Man kann die Aussage auf mehrere verschiedene Arten beweisen.

32. (0 Punkte) Konstruieren Sie für jede Höhe h einen 2-3-Baum mit Höhe h, bei dem man einen neues Element einfügen kann, sodass man den gesamten Weg zur Wurzel durchlaufen muss und sich die Höhe um 1 erhöht, und beim anschließenden Entfernen dieses Elementes der ursprünglich Zustand wiederhergestellt wird. (Bei 2-4-Bäumen kann das nicht passieren.)

33. (4 Punkte) Geben Sie ein Beispiel eines 2-3-Baums mit Höhe $h = 6$ an, bei dem man ein Element entfernen kann, sodass man mit dem Umbau des Baumes in Tiefe 4 aufhören kann, aber dennoch den gesamten Weg zur Wurzel durchlaufen muss, um die Schlüssel richtigzustellen. (Sie müssen nicht den gesamten Baum aufzeichnen, sondern nur die betroffenen Teile.)

    (Zusatzfrage, 0 Punkte.) Warum kann so etwas beim Einfügen nicht passieren?

34. (0 Punkte) Um die Schwierigkeit, die in der vorigen Aufgabe behandelt wurde, zu umgehen, kann man die vierte Bedingung bei der Definition von a-b-Bäumen abschwächen:

    > In einem a-b-Baum speichert ein innerer Knoten P mit k Kindern $k-1$ Schlüssel $v_1, \dots, v_{k-1}$, wobei v_i größer als alle Schlüssel im i-ten Teilbaum und kleiner oder gleich allen Schlüsseln im $(i+1)$-ten Teilbaum ist.

    Wie muss man das Suchen, Einfügen, und Entfernen an diese veränderten Definition anpassen?

35. (12 Punkte) Implementieren Sie 2-3-Bäume zum Speichern von Zahlen in Java oder Haskell.

36. (7 Punkte) Wir wollen eine Variante von a-b-Bäumen konstruieren, bei der weniger Speicher verschwendet wird. Es sei $b = 100$. Wie groß kann man a wählen, wenn man

    - beim Entfernen bis zu *zwei* Geschwisterknoten zum Ausborgen in Betracht zieht, und
    - beim Einfügen einen Geschwisterknoten zu Hilfe nimmt, falls dieser den Überlauf aufnehmen kann?

37. (9 Punkte) *Rot-Schwarz-Bäume* sind binäre Bäume mit folgenden Eigenschaften:[1]

    (a) Jeder innere Knoten hat zwei Kinder.
    (b) Jeder Knoten ist entweder als *rot* oder als *schwarz* gekennzeichnet.
    (c) Die Wurzel und alle Blätter sind schwarz.
    (d) Die Kinder eines roten Knotens sind schwarz.
    (e) Ein schwarzer Knoten kann höchstens ein rotes Kind haben.
    (f) Alle Wege von der Wurzel zu den Blättern enthalten gleich viele schwarzen Knoten.

    Zeichnen Sie Rot-Schwarz-Bäume mit 5, 7 und 12 Blättern. Zeigen Sie, dass man aus jedem Rot-Schwarz-Baum einen 2-3-Baum machen kann, und umgekehrt.

38. (4 Punkte) Es sei A_h die kleinstmögliche Anzahl von Blättern eines AVL-Baumes mit Höhe h. Beweisen Sie, dass die Zahlen A_h die Fibonacci-Zahlen sind.

    Was können Sie daraus für die kleinstmögliche Anzahl $n(h)$ von *inneren Knoten* eines AVL-Baumes mit Höhe h schließen?

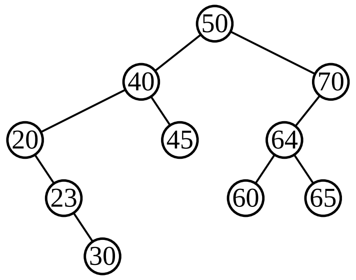
39. (0 Punkte) Wir wollen die Algorithmen zum Suchen, Einfügen und Löschen in einem binären Suchbaum so erweitern, dass man jederzeit auch das i-größte Element finden kann, und dass alle Operationen Zeit proportional zur Höhe des Baumes benötigen.

    Man könnte das lösen, indem man zu jedem Knoten den *Rang* speichert, das heißt, die Position, die dieser Schlüssel in der sortierten Reihenfolge einnimmt. Warum ist dies keine gute Idee?

40. (0 Punkte) Schreiben Sie ein Programm, das einen (nicht zu großen) binären Baum zweidimensional in übersichtlicher und schön lesbarer Form ausdruckt. Drei Vorschläge[2]:

```
      20  ----
         23  -
            30
   40  -------
      45
50  ----------
      64  -
         65
   70  ----
```

```
               50
           /----↥---------\
         40                70
  /-------↥\           /---↥
20          45         64
 ↥\                   /-↥\
   23               60    65
   ↥-\
      30
```



[1]Es gibt verschiedene andere Versionen von Rot-Schwarz-Bäumen, bei denen zum Beispiel auch die inneren Knoten Werte enthalten (im Gegensatz zu den 2-3-Bäumen, wie sie in der Vorlesung besprochen wurden).
Eine Implementierung von Rot-Schwarz-Bäumen kann man in der Klasse `TreeMap` im Paket `java.util` finden, siehe `http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/TreeMap.java`.

[2]Das dritte Beispiel ist als PostScript-Datei erstellt worden, siehe
`http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/baum.eps`.
Ihr Programm kann sich an dieser Datei als Beispiel orientieren.

41. (7 Punkte) Versuchen Sie, den Huffman-Algorithmus zur Konstruktion eines optimalen Kodes auf ein ternäres Kodealphabet (ein Alphabet mit drei Buchstaben) zu verallgemeinern. Konstruieren Sie einen optimalen ternären Kode für $n = 8$ Quellsymbole mit relativen Häufigkeiten $(p_1, \dots, p_8) = (\frac{2}{56}, \frac{3}{56}, \frac{4}{56}, \frac{6}{56}, \frac{7}{56}, \frac{10}{56}, \frac{11}{56}, \frac{13}{56})$.

42. (7 Punkte) Eine stückweise konstante Funktionen $f\colon (u, v] \to \mathbb{R}$ ist durch eine zusammenhängende Folge von Intervallen mit entsprechenden Werten $f(x) = a$ gegeben, z. B.

$$f(x) = \begin{cases} 1, & \text{für } -1 < x \leq 0 \\ -1, & \text{für } 0 < x \leq 2 \\ 6, & \text{für } 2 < x \leq 4 \end{cases}$$

Spezifizieren Sie eine Datenstruktur zur Darstellung solcher Funktionen. Nehmen Sie an, dass alle Intervalle halboffen sind, und zwar links offen und rechts abgeschlossen, wie im obigen Beispiel. Die Datenstruktur soll zumindest folgende Operationen erlauben:

    (a) Berechnung des Wertes einer solche Funktion $f(x)$ an einer gegebenen Stelle x. Falls x nicht im Definitionsbereich liegt, soll der Wert 0 zurückgegeben werden.
    (b) Addition und Multiplikation zweier stückweise konstanter Funktionen. Entscheiden Sie selbst, was der Definitionsbereich von $f + g$ beziehungsweise fg sein soll.

43. (6 Punkte) Leiten Sie aus der algebraischen Spezifikation für Mengen

$$\begin{aligned} \mathit{istenthalten}(x, \mathit{leer}) &= \mathit{falsch} && && (1) \\ \mathit{istenthalten}(x, \mathit{einfüge}(x, M)) &= \mathit{wahr} && && (2) \\ \mathit{istenthalten}(x, \mathit{einfüge}(y, M)) &= \mathit{istenthalten}(x, M), && \text{für } x \neq y && (3) \\ \mathit{istenthalten}(x, \mathit{lösche}(x, M)) &= \mathit{falsch} && && (4) \\ \mathit{istenthalten}(x, \mathit{lösche}(y, M)) &= \mathit{istenthalten}(x, M), && \text{für } x \neq y && (5) \end{aligned}$$

durch Umformungen folgende Identität her:

$$\mathit{istenthalten}(u, \mathit{einfüge}(x, \mathit{lösche}(x, M))) = \mathit{istenthalten}(u, \mathit{einfüge}(x, M)).$$

Geben Sie dabei in jedem Beweisschritt die Nummer der Gleichung an, die Sie verwenden.

44. (0 Punkte) Gegeben sei eine Funktion $f\colon \{0, \dots, n-1\} \to \{0, \dots, n-1\}$, die einen großen (aber endlichen) Bereich in sich selbst abbildet, zum Beispiel

$$f(x) = x^3 + 2x^2 + 7 \bmod 10^6.$$

Wir betrachten die Folge $a_0 = 0$, $a_{i+1} = f(a_i)$. Da der Bereich endlich ist, muss die Folge Wiederholungen enthalten. Wie kann man in $O(n)$ Zeit mit *konstantem Speicher* ein doppeltes Element finden? Wie kann man das kleinste i und das kleinste j finden, sodass $a_i = a_{i+j}$ ist?[1]

45. (0 Punkte) Der *Weihnachtsmann* ist mit seinem Rentierschlitten in die Mitte eines kreisförmigen Sees mit Umfang $100\,m$ gefallen. Am Rande des Sees befindes sich ein *Grinch*, der nicht schwimmen, aber 4,5-mal so schnell laufen kann wie die Rentiere den Schlitten im Wasser ziehen können. Auf dem Land ist der Rentierschlitten jedoch schneller als der Grinch. Kann der Weihnachtsmann die Rentiere so lenken, dass die Weihnachtsgeschenke nicht dem Grinch in die Hände fallen und er die Pakete noch rechtzeitig abliefern kann? Wenn ja, welchen Mindestabstand vom Grinch kann der Weihnachtsmann erreichen, wenn er das rettende Land erreicht? Der Einfachheit halber stelle man sich Grinch und Rentierschlitten jeweils punktförmig vor.[2]

[1] Diese Aufgabe dient nur zur Erbauung und hat mit dem Stoff der Vorlesung nicht direkt etwas zu tun. Lösungshinweise (für alle, die nicht selbst daraufkommen wollen) finden Sie auf der Netzseite der Vorlesung.

[2] Mit etwas Kreativität kann man diese Aufgabe auch mit Hilfe eines Computers lösen. Die weihnachtliche Einkleidung dieses Rätsels habe ich teilweise von `http://www.fzt86.de/Adventskalender/` übernommen.

46. (0 Punkte) Beweisen Sie folgende Gleichungen für Listen `x` und `y` durch strukturelle Rekursion aus den unten angegebenen Definitionen.

$$\texttt{x ++ [ ]} = \texttt{x} \tag{1}$$
$$\texttt{length (x ++ y)} = \texttt{(length x) + (length y)} \tag{2}$$
$$\texttt{length (reverse x)} = \texttt{(length x)} \tag{3}$$
$$\texttt{reverse (x ++ y)} = \texttt{(reverse y) ++ (reverse x)} \tag{4}$$
$$\texttt{reverse (reverse x)} = \texttt{x} \tag{5}$$

```
[ ] ++ yl = yl
(x : xr) ++ yl = x :(xr ++ yl)
```

```
length [ ] = 0
length (x : xr) = 1 + (length xr)
```

```
reverse [ ] = [ ]
reverse (x : xr) = (reverse xr) ++ [x]
```

47. (0 Punkte) Beweisen Sie (zum Beispiel durch strukturelle Induktion), dass jeder vollständig balancierten ternäre Baum der Höhe h genau $(3^{h+1} - 1)/2$ Knoten enthält. (Jeder innere Knoten hat genau 3 Kinder, und alle Blätter befinden sich auf Tiefe h.)

48. (0 Punkte) Ein binärer Suchbaum soll Einträge speichern, die außer dem Schlüssel x_i noch ein „Gewicht“ w_i enthalten. Die Algorithmen zum Suchen, Einfügen und Löschen in binären Suchbäumen sollen so erweitert werden, dass man jederzeit zu zwei Werten a und b das Gesamtgewicht aller Einträge finden kann, deren Schlüssel x_i im Intervall $a < x_i \le b$ liegt. Die Laufzeit für alle Operationen soll höchstens proportional zur Höhe des Baumes sein.

    Beschreiben Sie die Felder, die in den Knoten enthalten sind (zusammen mit einer Erklärung der Bedeutung, sofern sie nicht offensichtlich ist), und schreiben Sie ein Programmstück (Methoden in Java oder Funktionen in Haskell) für die oben beschriebene Intervallabfrage und zum Einfügen eines neuen Elementes.

49. (8 Punkte) Spezifizieren Sie den abstrakten Datentyp einer Prioritätswarteschlange, die mindestens die Operationen `entferneMin`, `einfügen` und `istLeer` enthält. Die Prioritätswarteschlange soll *Einträge* (x_i, a_i) enthalten, die einen *Schlüssel* x_i (eine Zahl) und einen *Wert* a_i enthalten. (Entscheiden Sie, wie sie mit gleichen Schlüsseln umgehen wollen.)

50. (12 Punkte) Modifizieren Sie das Programm zur Warteschlangensimulation aus Aufgabe 22 vom 3. Übungsblatt, Variante (a) folgendermaßen: Die Ankunftsrate der Patienten ist $\lambda = 5/\text{h}$, und es gibt $n = 3$ Ärztinnen. Es gibt eine zusätzliche Kategorie von *Notfallpatienten* mit Ankunftsrate $\lambda = 0{,}5/\text{h}$. Bei Ankunft eines Notfallpatienten unterbrechen alle n Ärztinnen ihre augenblickliche Tätigkeit und beschäftigen sich für einen Zeitraum, der gleichverteilt im Intervall $[a \mathinner{..} b]$ mit $a = 30\,\text{min}$ und $b = 40\,\text{min}$ ist, mit dem neuen Patienten. Danach wird der Notfallpatient zu einer anderen Abteilung geschickt und verlässt das hier untersuchte System, und die Ärztinnen fahren mit der vorher unterbrochenen Tätigkeit fort. Ein neuer Notfallpatient, der während der Behandlung eines Notfallspatienten eintrifft, kommt dran, wenn diese Behandlung abgeschlossen ist.

51. (0 Punkte) Bestimmen des doppelten Elementes.[1]
    In einem Array $a_0, a_1, \ldots, a_n$ sind ganzzahlige Werte zwischen 1 und n gespeichert. Daraus folgt, dass es mindestens eine Zahl geben muss, die doppelt vorkommt. Schreiben Sie ein Programm, das eine dieser Zahlen bestimmt. Das Programm soll lineare Laufzeit haben, auf das Array a nur *lesend* zugreifen, und nur konstanten zusätzlichen Speicher benötigen.

[1] Diese Aufgabe dient zur Entspannung und hat mit dem Stoff der Vorlesung nichts zu tun. Wenn Sie das Rätsel aus einer anderen Quelle bereits kennen, dann würde mich sehr interessieren, woher.

52. (6 Punkte) Erweitern Sie die Spezifikation für den abstrakten Datentype *Menge* (von ganzen Zahlen) aus der Vorlesung mit den Operationen Einfügen, Entfernen, Elementtest, und Erstellen einer leeren Menge um das Erstellen eines *Iterators* mit den Operationen *next* und *hasNext* (wie im Java-Interface `Iterator`). Spezifizieren Sie diese Operationen formal.

53. (0 Punkte) Erweitern Sie die algebraische Spezifikation von Mengenoperationen aus Aufgabe 43 um die Funktionen *Vereinigung* und *Durchschnitt* (von je zwei Mengen).

54. (0 Punkte) Definieren Sie in Haskell einen algebraischen Datentyp `Menge a` für Mengen von Elementen des Typs `a`, die wahlweise durch Aufzählen der Elemente oder durch Angabe einer charakteristischen Eigenschaft definiert werden können. Schreiben Sie folgende Funktionen:

```
mengeausListe    :: [a] -> Menge a           -- {x1, x2, ..., xn}
mengefürdiegilt  :: (a -> Bool) -> Menge a   -- { x | f(x) }
mengeDurchschnitt:: Menge a -> Menge a -> Menge a
mengeVereinigung :: Menge a -> Menge a -> Menge a
mengeElementvon  :: a -> Menge a -> Bool     -- x ∈ M
```

55. (0 Punkte) Schreiben Sie eine Spezifikation für *Intervallarithmetik*.
Intervallarithmetik liefert verlässliche Ergebnisse, selbst wenn die Eingabedaten mit Messfehlern behaftet sind und zwischendurch Rechenfehler auftreten. Statt mit Zahlen rechnet man mit Intervallen $[a, b]$, die durch die arithmetischen Grundoperationen, $+$, $-$, $\times$ und $:$ miteinander verknüpft und miteinander verglichen werden können (durch $<$, $>$, $\leq$, usw.) Das Ergebnis ist wieder eine Intervall.

56. (14 Punkte) Graphenalgorithmen in der künstlichen Intelligenz. Die folgende Aufgabe findet sich in einer Sammlung von Rechenaufgaben mit dem Titel *Propositiones ad acuendos juvenes* (Aufgaben zur Schärfung des Geistes der Jugend), die wahrscheinlich um das Jahr 800 am Hof Karls des Großen entstanden ist und Alkuin von York zugeschrieben wird:

    Die Aufgabe vom Wolf, der Ziege und dem Kohlkopf. Ein Mann musste einen Wolf, eine Ziege und einen Kohlkopf über einen Fluss übersetzen; er konnte aber nur ein Boot auftreiben, das gerade zwei von ihnen tragen konnte. Wie konnte er alles unversehrt hinüberbringen?[1]

    (a) Modellieren Sie diese Aufgabe durch einen Graphen. Die *Knoten* sollen den möglichen Zuständen des „Systems“ Mann–Ziege–Wolf–Kohlkopf–Boot–Fluss entsprechen, und die *Kanten* den erlaubten Übergängen: Zum Beispiel würde der Wolf die Ziege oder die Ziege den Kohlkopf fressen, wenn sie ohne Aufsichtig gelassen würden; der Mann, der das Boot rudert, kann nur einen Gegenstand oder ein Tier zusätzlich mitnehmen. Eine *Lösung* soll einem *Weg* in diesem Graphen entsprechen.

    (b) Schreiben Sie ein Programm, das diesen Graphen erstellt. Wie viele Knoten und wie viele Kanten hat der Graph? Schreiben Sie eine Programm, das mit Breitensuche einen Weg in diesem Graphen findet, der einer Lösung des Problems entspricht.

    (c) (0 Punkte) Ist die Lösung eindeutig? Wie kann man feststellen, ob es in einem Graphen nur einen einzigen Weg von s nach t gibt?

[1] PROPOSITIO DE LUPO ET CAPRA ET FASCICULO CAULI. Homo quidam debebat ultra fluvium transferre lupum et capram et fasciculum cauli, et non potuit aliam navem invenire, nisi quae duos tantum ex ipsis ferre valebat. Praeceptum itaque ei fuerat, ut omnia haec ultra omnino illaesa transferret. Dicat, qui potest, quomodo eos illaesos ultra transferre potuit. SOLUTIO. Simili namque tenore ducerem prius capram et dimitterem foris lupum et caulum. Tum deinde venirem lupumque ultra transferrem, lupoque foras misso rursus capram navi receptam ultra reducerem, capraque foras missa caulum transveherem ultra, atque iterum remigassem, capramque assumptam ultra duxissem. Sicque faciente facta erit remigatio salubris absque voragine lacerationis.

Algorithmen und Programmierung 3, WS 2003/2004 — 10. Übungsblatt

Abgabe bis Donnerstag, 29. Januar 2004

57. (0 Punkte) Eine *Multimenge* (engl. *multiset* oder *bag*) ist etwas Ähnliches wie eine Menge, außer dass Elemente auch mehrfach vorkommen dürfen. Die Reihenfolge spielt keine Rolle. Zum Beispiel ist $\{\!\{a, b\}\!\} \neq \{\!\{a, a, b\}\!\} = \{\!\{a, b, a\}\!\} \neq \{\!\{a, a, a, b\}\!\}$, für $a \neq b$.
    (a) Schreiben Sie eine Spezifikation für einen abstrakten Datentyp von Multimengen über der Grundmenge der ganzen Zahlen (`int`), die folgende Operationen unterstützt: Erzeugen einer leeren Multimenge; Einfügen und Streichen eines Elementes (dabei wird die Vielfachheit jeweils um 1 erhöht beziehungsweise erniedrigt); Feststellen der Vielfachkeit eines Elementes.
    (b) Geben Sie eine konkrete Darstellung (etwa als Java-Klasse `Multimenge`) an. Beschreiben Sie die Abstraktionsfunktion, sowie die Invarianten, die die gültigen Darstellungen charakterisieren. Geben Sie auch die Vorbedingungen für alle Operatiomen an. (Sie dürfen dabei vernünftige Einschränkungen für die verfügbaren Operationen machen.)
    (c) Implementieren Sie die Operationen. Sie können von der in der Vorlesung besprochenen Implementierung für Mengen mit bis zu 100 Elementen[1] ausgehen.
    (d) Beweisen Sie die Korrektheit Ihrer Implementierung.

58. (11 Punkte) Betrachten wir die Knoten v eines Graphen in der Reihenfolge, wie der rekursive Aufruf $T(v)$ bei der Tiefensuche *beendet* wird. Das heißt, wir fügen *am Ende* des Programms $T(v)$ folgende Zeile ein:

    ```
    num2++; T2Nummer[v] := num2;
    ```

    (a) (4 Punkte) Beweisen Sie: Wenn es eine Kante (u, v) mit $\text{T2Nummer}[v] > \text{T2Nummer}[u]$ gibt, dann enthält der Graph einen Kreis.
    (b) (4 Punkte) Wie kann man diesen Kreis bestimmen? Schreiben Sie ein Programmstück für diese Aufgabe.
    (c) (3 Punkte) Verwenden Sie die Tatsache aus Aufgabe (a), um aus der T2Nummerierung eines kreisfreien Graphen eine topologische Sortierung zu berechnen, sofern bei der Tiefensuche alle Knoten besucht werden. Beschreiben Sie den Algorithmus in Worten.

59. (0 Punkte) Bei der Tiefensuche werden möglicherweise nicht alle Knoten des Graphen besucht. Was muss man tun, damit das Verfahren der vorigen Aufgabe immer funktioniert?

60. (0 Punkte) Zeigen Sie, wie man Tiefensuche ohne Rekursion sehr einfach mit einem Stapel für noch zu bearbeitende *Kanten* implementieren kann.

61. (9 Punkte) Betrachten Sie das folgende einfache Haskell-Programm:

    ```
    camba [ ] _ = True
    camba _ [ ] = False
    camba (x:xs) (y:ys) = (x==y) && camba xs ys
    klungo [ ] _ = True
    klungo _ [ ] = False
    klungo xs (y:ys) = camba xs (y:ys) || klungo xs ys
    ```

    (a) (1 Punkt) Beschreiben Sie in Worten, was diese beiden Funktionen berechnen.
    (b) (4 Punkte) Spezifizieren Sie die beiden Funktionen mathematisch (modellierend).
    (c) (4 Punkte) Wie viele Vergleiche der Form `x==y` werden bei den folgenden Eingaben durchgeführt?

    ```
    klungo "aaaab" "aaaaaaaaaa"
    klungo "abababc" "bababababa"
    ```

[1] `http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/Menge.java`

62. (0 Punkte) Wie kann man auf einfache Art überprüfen, ob ein Graph, der mit Adjazenzlisten gespeichert ist, ein *einfacher Graph* ist (keine mehrfachen Kanten enthält)? Versuchen Sie, mit $O(m+n)$ Zeit und mit möglichst wenig zusätzlichem Speicher auszukommen.

63. (0 Punkte) Wenden die den Algorithms von Dijkstra zur Bestimmung der kürzesten Wege im folgenden gerichteten Graphen an: $V = \{1, 2, \ldots, 6\}$, der Graph ist vollständig, das heißt, von jedem Knoten gibt es eine Kante zu jedem anderen Knoten, und die Kantenlängen sind $c_{ij} = 3^{j-i}$ für $i < j$ und $c_{ij} = i - j$ für $i > j$. Der Startknoten ist der Knoten 2.

64. (6 Punkte) Konstruieren Sie einen Graphen, der auch Kanten mit negativer Länge enthält, und bei dem der Algorithmus von Dijkstra die kürzesten Wege nicht richtig bestimmt. Der Graph soll keine Kreise negativer Länge enthalten.

65. (0 Punkte) Denken Sie sich einen gierigen Algorithmus aus, der versucht, in einem Graphen einen kurzen Weg von einem Startknoten s zu einem Zielknoten t zu finden. Welche Problem können dabei auftreten?

66. (9 Punkte) Betrachten Sie die Variante einer Halde, bei der jeder innere Knoten $d \geq 2$ Kinder hat. (Für $d = 2$ ergeben sich die gewöhnlichen Halden aus der Vorlesung.) Wie verändert sich die Laufzeit für die Methoden `zugroß` beziehungsweise `zuklein` in Abhängigkeit von d? Wie wirkt sich das auf die Operationen *einfügen*, *entferneMin* und *verkleinereSchlüssel* aus?

    Wenn man eine solche „d-Halde" für den Algorithmus von Dijkstra verwendet, wie muss man dann d in Abhängigkeit von m und n wählen, dass man die optimale asymptotische Laufzeit erhält? Welche Laufzeit ergibt sich dann für das kürzeste-Wege-Problem? Was ergibt sich in den Extremfällen $m = \Theta(n)$ und $m = \Theta(n^2)$?

67. (0 Punkte) Zeigen Sie, dass der folgende *Algorithmus von Prim* einen kürzesten Spannbaum findet:

    $G := \{s\}$; $Abstand[s] := 0$;
    $N := V - \{s\}$;
    while $G \neq \emptyset$ **do**
        entferne einen Knoten v mit kleinstem $Abstand[v]$ aus G;
        füge v in B ein;
        für alle Kanten $(v, w) \in E$, die von v ausgehen, **do**
            if $w \in N$ **or** ($w \in G$ **and** $Abstand[w] > c_{vw}$)
            then $G := G \cup \{w\}$; $N := N - \{w\}$;
                $Abstand[w] := c_{vw}$;
                $Vorgänger[w] := v$;

    Was ist der Unterschied dieses Algorithms zum Algorithmus von Dijkstra für kürzeste Wege?

68. (a) (0 Punkte) Berechnen Sie die Verschiebefunktion des *Fibonacci-Wortes*

    $$abaababaabaababaababaabaababaabaab.$$

    (b) (5 Punkte) Berechnen Sie die Verschiebefunktion der *Morse-Folge*

    $$0110100110010110100101100110100110010110011010010110100110010110.$$

    (0 Punkte) Wie gehen die beiden Folgen weiter?

69. (freiwillige Zusatzaufgabe, 0 Punkte) Wie kann man n Brüche, deren Zähler und Nenner zwischen 1 und n liegen, in linearer Zeit sortieren?

70. (7 Punkte) Ein Iterator für Mengen.
    (a) (4 Punkte) Implementieren Sie den *Iterator*, den Sie in Aufgabe 52 spezifiziert haben. Sie können von der in der Vorlesung besprochenen Implementierung für Mengen mit bis zu 100 Elementen[1] ausgehen.
    (b) (3 Punkte) Geben Sie die Abstraktionsfunktion an.
    (c) (5 Zusatzpunkte) Beweisen Sie die Korrektheit Ihrer Implementierung.
71. (3 Punkte) Gegeben sei ein vollständiger[2] Graph $G = (V, E)$, dessen Knoten V Punkte in der Ebene sind, und wo die Kantenlängen den (Euklidischen) Abständen zwischen den Punkten entsprechen. Beweisen Sie:
    (a) (3 Punkte) In einem kürzesten spannenden Baum können sich nie zwei Kanten kreuzen.
    (b) (5 Zusatzpunkte) In einem kürzesten-Wege-Baum (mit einem beliebigen Startknoten) können sich nie zwei Kanten kreuzen.

    Bleiben diese Aussagen auch gültig, wenn der Graph nicht vollständig ist?
72. (0 Punkte) Für zwei Wörter x und y der Länge n ist x eine *zyklische* Verschiebung von y, wenn man $x = ab$ und $y = ba$ für zwei Wörter a und b schreiben kann. Wie kann man in linearer Zeit feststellen, ob x eine zyklische Verschiebung von y ist? (Man kann diese Frage auf das Teilwortproblem zurückführen.)
73. (5 Punkte) Wie kann man den Algorithmus von Knuth, Morris und Pratt so erweitern, dass er *alle* Vorkommen eines Musters x in einem Text w findet? (Man benötigt eventuell den Wert h_{m+1} der Verschiebefunktion, für $m = |x|$.)
74. (0 Punkte) Die verbesserte Verschiebefunktion zum Suchen von Zeichenketten ist folgendermaßen definiert:

    $$h[i] = \max \{ k \mid 1 \le k < i,\ p_1 \dots p_{k-1} = p_{i-k+1} \dots p_{i-1} \text{ und } p_k \ne p_i \} \cup \{0\}$$

    (a) Berechnen Sie die verbesserte Verschiebefunktion der Muster aus Aufgabe 68.
    (b) Zeigen Sie, dass beim Suchen mit der verbesserten Verschiebefunktion auf keinen Fall mehr Vergleiche der Form $p_i = s_j$ durchgeführt werden als mit der ursprünglichen Verschiebefunktion (ohne die Bedingung „$p_k \ne p_i$“). Gilt dies auch, wenn man den Aufwand an Vergleichen beim Berechnen der Verschiebefunktion mit berücksichtigt? Finden Sie ein Beispiel, bei dem tatsächlich weniger Vergleiche notwendig sind.
    (c) Schreiben Sie einen Algorithmus zum Berechnen der verbesserten Verschiebefunktion.
75. (0 Punkte) Wie kann man aus der Verschiebefunktion des Wortes $xw\$$ berechnen, ob x ein Teilwort von y ist? (Hier ist $ irgendein Buchstabe.)
76. (0 Punkte) Welche Bedeutung hat die Flächenformel $\frac{1}{2} \cdot |\sum_i (x_i y_{i+1} - y_i x_{i+1})|$, wenn die Folge der Punkte $(x_i; y_i)$ gar kein Polygon beschreibt, weil sich zum Beispiel Kanten kreuzen?
77. (a) (5 Punkte) Berechnen Sie die Fläche des Fünfecks[3] mit den Ecken $(-1{,}3; 10000{,}12)$, $(-0{,}253; 10000{,}47)$, $(0{,}69; 10000{,}33)$, $(1{,}529; 10002{,}12)$, $(-0{,}783; 10001{,}05)$ mit der Formel aus der vorigen Aufgabe. Berechnen Sie auch den Flächeninhalt des um den Vektor $(0; -10000)$ verschobenen Fünfecks. Welches Ergebnis halten Sie für das genauere? Begründen Sie Ihre Antwort.
    (b) (0 Punkte) Was passiert, wenn man das Fünfeck um den Vektor $(10000; -10000)$ verschiebt? Wie erklären Sie diese Ergebnisse?

[1] `http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/Menge.java`
[2] Ein Graph ist *vollständig*, wenn zwischen allen Paaren von Knoten eine Kante verläuft.
[3] `http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/5eck`