Algorithmen und Programmieren 3, WS 1999/2000 — 1. Übungsblatt

Abgabe bis Dienstag, 2. November 1999

---

1. (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)

2. (0 Punkte) Das folgende Programm 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.)

3. (6 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?

4. (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 $g(n) = \Theta(f(n))$ aus $f(n) = O(g(n))$?

5. (7 Punkte) Beweisen Sie:

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

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

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

   (d) Finden Sie möglichst einfache Ausdrücke der Form $O(\cdot)$ für folgende Funktionen:
   $3n^2 - 4n + 32 + 27\,n \cdot \lceil \log_2 n \rceil / 2$; $\max\{n\lceil \log_2 n\rceil, (\lceil \log_2 n \rceil)^4\}$; $2^{2n+\lceil \log_2 n \rceil}$

6. (5 Punkte) Formulieren Sie *Sortieren durch Auswahl* in einer Programmiersprache Ihrer Wahl.

Algorithmen und Programmieren 3, WS 1999/2000 — 2. Übungsblatt

Abgabe bis Dienstag, 9. November 1999

7. (3 Punkte) Was passiert bei *Sortieren durch Einfügen*, wenn die gegebene Folge bereits (a) aufsteigend oder (b) absteigend sortiert ist? Ist es möglich, dass man bloß $O(n)$ Zeit benötigt?
8. (2 Punkte) Beantworten Sie die vorige Frage für *Sortieren durch Auswahl.*
9. (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.
10. (0 Punkte) Wie kann man *Sortieren durch Einfügen* so anpassen, dass es für Folgen, die absteigend oder fast absteigend sortiert sind, möglichst schnell läuft?
11. (2 Punkte) Geben Sie alle möglichen topologischen Sortierungen für folgende Eingabe an: $n = 6$ und $\{(6,1),(3,5),(1,4),(3,1),(2,4),(2,6),(5,1),(3,6)\}$.
12. (2 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?
13. (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 könnte man ohne Nachteil einsparen?
14. (8 Punkte) Erweitern Sie das Java-Programm zum topologischen Sortieren, sodass bei der Existenz eines Kreises nicht einfach mit einer Meldung abgebrochen wird, sondern auch ein Kreis (als „Beweis“) ausgegeben wird.

    Ihr Programm sollte nicht mehr als $O(m+n)$ zusätzliche Zeit und nicht mehr als $O(n)$ zusätzlichen Speicher brauchen.
15. (8 Punkte) Es gibt eine Variante von *Sortieren durch Verschmelzen* (mergesort), die von unten nach oben (*bottom-up*) arbeitet: Zunächst werden je zwei aufeinanderfolgende Elemente zu einem Paar zusammengefasst und gegebenenfalls vertauscht, sodass eine Folge von sortierten Zweierblöcken entsteht. Dann werden je zwei benachbarte Zweierblöcke zu einem sortierten Viererblock verschmolzen, diese werden wiederum paarweise zu Achterblöcken verschmolzen, usw.

    (a) (4 Punkte) Schreiben Sie ein Programm für dieses Sortierverfahren in Haskell oder Java[1].

    (b) (4 Punkte) Analysieren Sie die Laufzeit dieses Algorithmus.

    (c) (0 Punkte) Falls Sie für Aufgabe (a) nicht Haskell gewählt haben, analysieren Sie auch den *zusätzlichen* Speicherbedarf (zusätzlich zu dem Feld, in dem die zu sortierenden Elemente am Anfang stehen). Falls Sie für Aufgabe (a) Haskell gewählt haben, erklären Sie, warum Sie die obige Frage nicht beantworten müssen.
16. (0 Punkte) Entwerfen Sie eine bottom-up-Variante von Quicksort.

[1] oder in C oder C++ oder Pascal. Weitere Programmiersprachen auf Anfrage

Algorithmen und Programmieren 3, WS 1999/2000 — 3. Übungsblatt

Abgabe bis Dienstag, 16. November 1999

17. (7 Punkte)

    (a) 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) Schreiben Sie eine Klasse `Halde`, die eine `PWSchlange` implementiert beziehungsweise erweitert.[1]

18. (13 Punkte) Simulieren Sie eine einfache Warteschlange in Java.

    In einem Supermarkt mit $n = 3$ Kassen stellen sich ankommende Kunden bei einer der Kassen an, bis sie an dieser Kasse bedient werden.

    Die Ankunft der Kunden ist ein Poisson-Prozess mit Rate $\lambda = 1{,}5\,\mathrm{min}^{-1}$. Das heißt, dass der Abstand von der Ankunft eines Kunden bis zur Ankunft des nächsten Kunden exponentialverteilt mit dem Mittelwert $1/\lambda = 40$ Sekunden ist. Eine exponentialverteilte Zufallsvariable mit Mittelwert μ kann man mit der Formel `-`μ`*Math.log(Math.random())` erzeugen.

    Die Zeit zur Bedienung eines Kunden an der Kasse ist gleichverteilt im Intervall $[a\,..\,b]$ mit $a = 10\,\mathrm{sec}$ und $b = 200\,\mathrm{sec}$. Eine solche gleichverteilte Zufallsvariable kann man mit der Formel `a+(b-a)*Math.random()` erzeugen.

    Untersuchen Sie nun folgende Fälle:

    (a) Es gibt eine gemeinsame Schlange für alle Kassen.

    (b) Es gibt eine getrennte Schlange für jede Kasse. Neu ankommende Kunden stellen sich bei einer zufälligen Kasse an, (auch wenn eine andere Kasse frei wäre), und bleiben dort stehen, bis sie bedient werden.

    (c) (freiwillige Zusatzaufgabe, 1 Punkt.) Es gibt eine getrennte Schlange für jede Kasse. Neu ankommende Kunden stellen sich bei der kürzesten Schlange an. Wenn mehrere Schlangen die gleiche Länge haben, dann wird die „erste“ Schlange (mit der kleinsten Nummer) gewählt.

    Simulieren Sie diese Schlange von 9:00 bis 19:00 Uhr eines Tages. (Um 19:00 Uhr wird der Eingang geschlossen; es stellen sich keine weiteren Kunden mehr an, aber alle Kunden, die sich schon angestellt haben, werden noch bedient.) Bestimmen Sie für jede Variante die durchschnittliche Wartezeit eines Kunden, die maximale Wartezeit, und die Gesamtzeit, wie lange die Kassiererinnen nichts zu tun hatten. 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 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 die Programme aus der Vorlesung, `http://www.inf.fu-berlin.de/~rote/Lere/alp3/heapsort.java`

Algorithmen und Programmieren 3, WS 1999/2000 — 4. Übungsblatt

Abgabe bis Dienstag, 23. November 1999

19. (0 Punkte) Definieren Sie die arithmetischen Operationen für die ganzen Zahlen, die um die Werte $\pm\infty$ und *undefiniert* erweitert wurden. Es sollten die erwarteten Rechenregeln gelten, zum Beispiel $a + (-\infty) = -\infty$, $a/(+\infty) = 0$, für $a \in \mathbb{Z}$, $(+\infty) + (-\infty) = \mathit{undef}$, usw. Verwenden Sie dazu einen algebraischen Typ, der etwa so aussehen könnte:

```
data Int' = Normal Int
     | PlusUnendlich | MinusUnendlich | Undefiniert deriving ....
```

20. (7 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. (Sie könne sich dabei an der vorigen Aufgabe orientieren.) 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
```

21. (0 Punkte) Zwei Bäume heißen *isomorph*, wenn sie dieselbe Struktur (unabhängig von den Knotenbeschriftungen) haben. Definieren Sie ein Funktion
`isomorph :: Tree a -> Tree b -> Bool`
zum Testen, ob zwei Bäume isomorph sind.

22. (0 Punkte) Ein *vollständiger* binärer Baum der Höhe h ist ein Baum, bei dem jeder Knoten mit Tiefe $\leq h-2$ genau zwei Kinder hat.
   (a) Zeichnen Sie vollständige binäre Bäume mit 12 und mit 15 Knoten.
   (b) Wieviele Knoten kann ein vollständiger binärer Baum der Höhe h (mindestens und höchstens) haben?
   (c) Welche Höhe kann ein vollständiger binärer Baum mit n Knoten haben?

23. (15 Punkte) Das *Ungleichgewicht* eines Knotens x in einem binären Baum ist definiert als

$$|H(l(x)) - H(r(x))|,$$

wobei $l(x)$ und $r(x)$ der linke und der rechte Teilbaum von x ist und $H(b)$ die Höhe eines Baumes b ist. Das Ungleichgewicht eines *binären Baumes* ist das maximale Ungleichgewicht aller Knoten des Baumes.

   (a) (3 Punkte) Schreiben Sie ein Programm in Haskell, das das Ungleichgewicht eines Baumes berechnet.
   (b) (7 Punkte) Erweitern Sie die Java-Klasse `Baum` aus der Vorlesung, sodass man das Ungleichgewicht des Baumes (mit einer neuen Methode `getUngleichgewicht`) jederzeit ablesen kann, ohne es neu zu berechnen. (Hinweis: In der neuen Klasse sollte jeder Knoten ein Feld mit seinem Ungleichgewicht enthalten, und möglicherweise noch andere Felder. Gegebenenfalls müssen Sie die Methoden `einfügen` und `entfernen` abändern.)
   (c) (1 Punkt) Wie viele Knoten kann ein binärer Baum der Höhe h haben, dessen Ungleichgewicht 0 ist?
   (d) (4 Punkte) Ein *AVL-Baum* ist ein binärer Baum, dessen Ungleichgewicht höchstens 1 ist (nach Adel'son-Velski und Landis). Wie viele Knoten kann ein AVL-Baum der Höhe h (mindestens und höchstens) haben, für $h \leq 7$?
   (e) (freiwillige Zusatzfrage, 3 Punkte) Welche Höhe kann ein AVL-Baum mit n Knoten (mindestens und höchstens) haben?

Algorithmen und Programmieren III, WS 1999/2000 — 5. Übungsblatt

Abgabe bis Dienstag, 7. Dezember 1999

24. (3 Punkte) Schreiben Sie Funktionen vom Typ `Int->T` in Haskell, die für gegebenes n (a) einen möglichst ausgeglichenen Baum, (b) einen nach links entarteten Baum, (c) einen nach rechts entarteten Baum, beziehungsweise (d) einen entarteten Zickzackbaum mit n Knoten erzeugen. Bei (a) sollen die beiden Unterbäume jedes Knotens möglichst gleich viele Knoten enthalten. Bei (b) hat kein Knoten ein rechtes Kind; der Baum besteht aus einem einzigen Pfad, der immer nach links führt, und analog für (c). Bei (d) gibt es ebenfalls auf jeder Ebene nur einen Knoten, aber der Weg von der Wurzel zum einzigen Blatt führt abwechselnd nach links und nach rechts. (Die Werte in den Knoten können beliebig sein.)

25. (11 Punkte) Laufzeitverhalten von Funktionen in Haskell. Betrachten Sie folgende 2 Funktionen `sR1` und `sR2`[1] zum Durchlaufen eines Baumes in symmetrischer Reihenfolge und zur Ausgabe aller Knoten in einer Liste:

```
sR1:: T -> [Int]
sR1 Nil = []
sR1 (Baum l w r) =
    (sR1 l) ++ [w] ++ (sR1 r)

len:: T -> Int
len Nil = 0
len (Baum l _ r) = 1+len l+len r
```

```
data T = Nil | Baum{l::T, wert::Int, r::T}
    deriving (Show,Eq)

sR2:: T -> [Int]
sR2 b = sR' b [] where
  sR':: T -> [Int] -> [Int]
  sR' Nil x = x
  sR' (Baum l w r) x = sR' l (w:(sR' r x))
```

(a) (0 Punkte) Zeigen Sie, dass `(sR' b x) ++ y = sR' b (x++y)` ist.

(b) (11 Punkte) Vergleichen Sie die Laufzeit von `sR1` und `sR2` für die Bäume (a)–(c) aus der vorigen Aufgabe, für jeweils $n = 2^i$ Knoten, $i = 3, \ldots, 11$. Wenn man im HUGS-System `:set +s` eingibt, wird nach jeder Eingabe die Anzahl der *Reduktionen* (und auch die Anzahl der verwendeten Speicherzellen) ausgegeben, die man als Maß für die Laufzeit nehmen kann. Um von der Erzeugung des Baumes und von der Zeit für die Ausgabe der Liste abzusehen, vergleichen Sie für jeden Baum `b` jeweils folgende Eingaben:
(1) `len b`, als Maß für das einfachste Durchlaufen des Baumes,
(2) `length (sR1 b)`, sowie
(3) `length (sR2 b)`.
Bilden sie jeweils die Differenz der Reduktionenzahl zwischen (2) und (1), und zwischen (3) und (2). Schätzen Sie sodann experimentell einen Wert für die Konstante c unter der Annahme, dass sich die gemessenen Laufzeitdifferenzen nach der Formel $c \cdot n$, $c \cdot n \log_2 n$, beziehungsweise $c \cdot n^2$ verhalten, sodass die Formel möglichst gut mit Ihren Daten übereinstimmt. Welche Formel halten Sie jeweils für die passende?

(c) (Zusatzfrage, 3 Punkte) Erklären Sie die Ergebnisse, die Sie erhalten haben.

26. (7 Punkte) Nehmen wir an, dass jeder Knoten eines Baumes auch einen Zeiger zur Mutter hat:

```
class Baum2 extends Baum
{ Baum2 m;
  ...
```

Der Mutterzeiger der Wurzel ist `null`. Schreiben Sie ein nichtrekursives Programm, das die Knoten des Baumes in symmetrischer Reihenfolge ausgibt.

[1] siehe `http://www.inf.fu-berlin.de/~rote/Lere/alp3/symmetrischeReihenfolge.hs`

27. (9 Punkte) Eine *Triangulierung* eines konvexen n-Ecks P_n ist eine Zerlegung in $n-2$ Dreiecke durch $n-3$ nichtkreuzende Diagonalen, die je zwei nichtbenarchbarte Ecken verbinden.

    (a) (0 Punkte) Zeigen Sie, dass es 5 verschiedene Triangulierungen des Fünfecks P_5 gibt. (Die Ecken des P_5 sind nummeriert.) Zeigen Sie, dass es 5 verschiedene binäre Bäume mit 3 Knoten gibt. Zeigen Sie, dass es genauso viele Triangulierungen des P_n wie binäre Bäume mit $n-2$ Knoten gibt.

    (b) (4 Punkte) Eine Möglichkeit, eine Triangulierung des P_n anzugeben, ist die *Gradliste* $(a_1, \ldots, a_n)$, wobei a_i die Anzahl der Diagonalen ist, die in der i-ten Ecke zusammenkommen. So hat zum Beispiel jede Triangulierung des P_5 die Gradliste $(1, 0, 2, 0, 1)$, bis auf zyklische Verschiebungen.

    Zeigen Sie, dass es immer eine Ecke vom Grad $a_i = 0$ geben muss. (Was entspricht einer solchen Ecke geometrisch? Was passiert, wenn man eine solche Ecke entfernt?)

    (c) (0 Punkte) Zeigen Sie, dass eine Triangulierung durch ihre Gradliste eindeutig bestimmt ist.

    (d) (5 Punkte) Schreiben Sie ein Programm, das testet, ob eine gegeben Folge $(a_1, \ldots, a_n)$ die Gradliste einer Triangulierung ist.

28. (4 Punkte) Zufällige Permutationen. Wie kann man eine gegebene Folge $a_1, \ldots, a_n$ von Zahlen in eine zufällige Reihenfolge bringen? Schätzen Sie die Laufzeit und den Speicherplatz Ihres Algorithmus ab. Wie viele Zufallszahlen benötigen Sie? (Für einen Linearzeitalgorithmus gibt es einen Zusatzpunkt.)

29. (-2 Punkte) Ändern Sie die Warteschlangensimulation aus Aufgabe 18(a) wie folgt.

    Bestimmen Sie die durchschnittliche Länge ℓ der Warteschlange im Zeitraum zwischen 12:00 und 13:00 Uhr, sowie den Zeitpunkt t, wann der letzte Kunde fertig ist. Führen Sie 100 unabhängige Simulationsläufe durch, und bestimmen Sie dem maximalen, den minimalen, und den Mittelwert von ℓ und t.

30. (0 Punkte) Verschmelzen. Addition stückweise linearer Funktionen.
    Eine stückweise lineare Funktionen $f\colon [u, v] \to \mathbb{R}$ ist durch eine zusammenhängende Folge von Intervallen mit entsprechenden linearen Stücken $f(x) = ax + b$ gegeben, zum Beispiel:

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

    (a) Überlegen Sie sich eine Datenstruktur (Haskell-Datentyp/Java-Klasse) zur Darstellung solcher Funktionen. Schreiben Sie eine Funktion/Methode, die den Wert einer solche Funktion $f(x)$ an einer gegebenen Stelle x ausrechnet.

    (b) Schreiben Sie eine Funktion zur Addition zweier stückweise linearer Funktionen. Der Definitionsbereich von $f + g$ soll dabei der Durchschnitt der Definitionsbereiche von f und g sein.

    (c) Schreiben Sie eine Funktion/Methode zur Integration einer stückweise *konstanten* Funktion $f\colon [u, v] \to \mathbb{R}$. Das heißt, es soll die stückweise lineare und stetige Stammfunktion F von f berechnet werden:

$$F\colon [u, v] \to \mathbb{R}, \qquad F(x) = \int_u^x f(x)\, dx$$

31. (0 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.

32. (5 Punkte) Erweitern Sie die Algorithmen zum Suchen, Einfügen und Löschen in binären Suchbäumen so, dass man jederzeit auch das i-größte Element finden kann, und dass alle Operationen Zeit proportional zur Höhe des Baumes benötigen.

33. (0 Punkte) Gegeben sind n Güter (Sand, Gold, Salz, usw.). Von jedem Gut ist eine gewisse Menge x_i vorhanden; weiters ist der Wert w_i (pro Gewichtseinheit) bekannt. Welche Güter soll man in welcher Menge auswählen, wenn man insgesamt nur ein Gesamtgewicht von b Einheiten nehmen kann und den größten Wert erzielen möchte? (Welche Güter soll man zum Beispiel bei einem Feuer in Sicherheit bringen?)
    Wie würden Sie dieses Problem (unter Zuhilfenahme binärer Suchbäume) lösen, wenn man für verschiedene Werte von b jeweils schnell eine Antwort bekommen möchte, und wenn sich die Menge der Güter möglicherweise ändert?

34. (5 Punkte) Schreiben Sie ein Programm, das einen (nicht zu großen) binären Baum zweidimensional in übersichtlicher und schön lesbarer Form ausdruckt. Zwei Vorschläge:

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

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

35. (5 Punkte) Implementieren Sie für binäre Suchbäume eine Methode, die einen *Iterator* mit den Methoden `next()` und `hasNext()` erzeugt. (Sie dürfen annehmen, dass die Knoten Mutterzeiger wie in Aufgabe 26 haben.) Verwenden Sie dabei *nicht* das Verfahren aus der Vorlesung, das eine Kopie der Daten in einem linearen Feld anlegt, oder eine andere Methode, die mehr als konstant viel zusätzlichen Speicher benötigt.

36. (a) (0 Punkte) In einem binären Suchbaum, der ganze Zahlen speichert, sollen die Zahlen x, die im Intervall $\{\, x \mid a \le x \le b \,\}$ liegen, ausgegeben werden. Schreiben Sie ein Programm, das diese Aufgabe in $O(h+k)$ Zeit erledigt, wobei h die Höhe des Baumes und k die Länge der Ausgabe ist.

    (b) (0 Punkte) Wie kann man die in den Knoten des Baumes gespeicherte Informationen so erweitern, dass man die *Summe* der Zahlen x im Intervall $\{\, x \mid a \le x \le b \,\}$ in $O(h)$ Zeit bestimmen kann (und trotzdem noch in $O(h)$ Zeit einfügen und entfernen kann)?

[1] Es gibt verschiedene andere Versionen von Rot-Schwarz-Bäumen, bei denen zum Beispiel auch die inneren Knoten Werte enthalten (im Gegensatz zu 2-3-Bäumen).
Eine Implementierung von Rot-Schwarz-Bäumen kann man zum Beispiel in der Klasse `TreeMap` im Paket `java.util` finden, siehe `http://www.inf.fu-berlin.de/~rote/Lere/alp3/TreeMap.java`.

Algorithmen und Programmieren III, WS 1999/2000 — 7. Übungsblatt

Abgabe: Aufgabe 37 bis Dienstag, 4. Jänner, Aufgaben 39–40 bis Dienstag, 11. Jänner 2000.

---

37. (9 Punkte) Ändern Sie die Warteschlangensimulation aus Aufgabe 18(a) wie folgt.

    Die mittlere Zeit zwischen zwei Kunden sinkt auf $1/\lambda = 10$ Sekunden, dafür gibt es nun 12 Kassen. Bestimmen Sie zusätzlich die durchschnittliche Länge ℓ der Warteschlange im Zeitraum zwischen 12:00 und 13:00 Uhr, sowie den Zeitpunkt t, wann der letzte Kunde fertig ist. Führen Sie 100 unabhängige Simulationsläufe durch, und bestimmen Sie dem maximalen, den minimalen, und den Mittelwert von ℓ und t.

    Sie können Ihre eigene Lösung von Aufgabe 18(a) oder die Musterlösung `Simulation.java`[1] modifizieren. Markieren Sie in Ihrem Programmausdruck alle Teile, die Sie unverändert übernehmen konnten, mit einem grünen Strich am Rand.

38. (0 Punkte) Modifizieren Sie das Programm `Simulation.java`[1] wie folgt: Die Klassen `Kassen` und `Kasse` sollen aus der Klasse `Simulation` herausgenommen werden. Die Methoden und Felder aus `Kassen` sollen durch entsprechende statische Methoden und Felder der Klasse `Kasse` ersetzt werden, sodass die Klasse `Kassen` überflüssig wird.

39. (10 Punkte) Der Computer SUMO-0 verfügt über 10 Eingaberegister, die von `E0` bis `E9` nummeriert sind, sowie 26 Akkumulatoren, die von `Va` bis `Vz` bezeichnet sind. Ein Programm für diesen Rechner ist in der Assemblersprache SUMATRA geschrieben und sieht zum Beispiel wie das nebenstehende Programm aus.

    ```
    Vf=E0+E1
    Vc=Vf-E2
    Vu=Vf+Vf
    Vv=Vu*Vc
    Vk=Vf/E0
    Vl=Vk+E9
    ...
    ```

    In jeder Zeile wird das Ergebnis einer zweistelligen arithmetischen Operation der Variablen auf der linken Seite zugewiesen. Jede Variable darf höchstens *einmal* auf der linken Seite einer Zuweisung auftreten. Als Operand einer arithmetischen Operation kann entweder ein Eingaberegister `E`i oder Inhalt einer Variablen `V`x, deren Wert in einer früheren Zeile definiert wurde, auftreten.

    Die Programmzeilen werden der Reihe nach ausgeführt. Schleifen und Verzweigungen sind unbekannt.

    (a) (10 Punkte) Schreiben Sie ein Programm, das ein gültiges SUMATRA-Programm einliest und die Abhängigkeit der Ergebnisse von den Operanden als Graph abspeichert.

    Das Programm soll folgende Fragen beantworten können: Welche Variablen hängen (möglicherweise[2]) vom Wert des Eingaberegisters `E`i ab? Von welchen Eingaberegistern hängt der Wert der Variablen `V`x ab?

    (b) (0 Punkte) Bei gewissen Eingaben stellt sich heraus, dass das Programm mit einem Laufzeitfehler abbricht, weil durch 0 dividiert wurde. Der SUMATRA-Programmierer möchte den Laufzeitfehler dadurch vermeiden, dass er den Wert eines Eingaberegisters verändert (und alle anderen Eingaberegister gleich lässt). Ihr Programm soll feststellen, welche Eingaberegister überhaupt prinzipiell einen Einfluss auf das Auftreten einer Division durch Null haben können.

40. (5 Punkte) Dem primitiven Charakter der Sprache entsprechend, werden die Zeilen eines SUMATRA-Programmes auf *Lochkarten* gestanzt. Nehmen wir an, dass der Stapel von Lochkarten eines Programmes zu Boden gefallen ist und die Zeilen durcheinandergemischt worden sind. Wie kann man die Programmzeilen wieder in eine gute Reihenfolge bringen? (Sie brauchen kein lauffähiges Programm für diese Aufgabe zu schreiben.)

---

[1] siehe `http://www.inf.fu-berlin.de/~rote/Lere/alp3/Simulation.java`

[2] Dass zum Beispiel nach der Anweisung `Vy=Vx-Vx` die Variable `Vy` in Wirklichkeit nicht von `Vx` abhängt, brauchen Sie nicht zu berücksichtigen.

Algorithmen und Programmieren III, WS 1999/2000 — 8. Übungsblatt

Abgabe bis Dienstag, 18. Jänner 2000.

41. (13 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?

42. (0 Punkte) Das 14-15-Spiel von Sam Loyd besteht aus 15 nummerierten quadratischen Blöcken, die auf einem 4×4-Spielfeld beweglich angeordnet sind. Eines der 16 Felder ist frei. Ein Zug besteht darin, dass man einen Stein von einem benachbarten Feld auf das freie Feld verschiebt. Das Ziel ist, von einer gegebenen Ausgangsposition ausgehend, die Steine in die sortierte Reihenfolge zu bringen, sodass das freie Feld am Ende rechts unten ist.

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

$\rightarrow$

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

$\rightarrow \cdots \rightarrow$

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

    Diskutieren Sie, wie man dieses Problem als Wegeproblem in einem Graphen modellieren könnte. Wie viele Knoten und wie viele Kanten hätte der Graph? Welche Schwierigkeiten können bei diesem Problem auftreten?

43. (0 Punkte) Konstruieren Sie ein Beispiel eines Graphen mit negativen Kantenlängen, bei dem der Algorithmus von Dijkstra nicht den kürzesten Weg findet.

---

[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 Programmieren III, WS 1999/2000 — 9. Übungsblatt

Abgabe bis Dienstag, 25. Januar 2000

44. (3 Punkte)

    (a) (3 Punkte) Berechnen Sie die Verschiebefunktion des *Fibonacci-Wortes*

    $$abaababaabaababaababaabaababaabaab.$$

    (b) (0 Punkte) Berechnen Sie die Verschiebefunktion der *Morse-Folge*

    $$0110100110010110100101100110100110010110011010010110100110010110.$$

45. (0 Punkte) Die verbesserte Verschiebefunktion zum Suchen von Zeichenketten ist folgendermaßen definiert:

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

    (a) Berechnen Sie die verbesserte Verschiebefunktion der Muster aus der vorigen Aufgabe.

    (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 „$p_k \neq p_i$“). 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.

46. (5 Punkte) Erweitern Sie den Knuth–Morris–Pratt-Algorithmus so, dass er *alle* Stellen bestimmt, an denen das Muster im Text vorkommt.

47. (8 Punkte) Eine *Multimenge* ist etwas Ähnliches wie eine Menge, außer dass Elemente auch mehrfach vorkommen dürfen. Die Reihenfolge spielt keine Rolle. Zum Beispiel ist $\{\!\{a, b\}\!\} \neq \{\!\{a, a, b\}\!\} = \{\!\{a, b, a\}\!\} \neq \{\!\{a, a, a, b\}\!\}$.

    (a) (4 Punkte) 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) (4 Punkte) 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) (0 Punkte) Implementieren Sie die Operationen.

    (d) (0 Punkte) Beweisen Sie die Korrektheit Ihrer Implementierung.

48. (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.

49. (12 Punkte) Betrachten Sie die folgende Haskell-Funktion:

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

   (a) (5 Punkte) Geben Sie (formal oder umgangssprachlich, aber präzise) an, was die Funktion `istTeil` berechnet.

   (b) (0 Punkte) Bestimmen Sie experimentell oder durch Analyse die Laufzeit der Funktion in Abhängigkeit von der Länge der Argumente. Für welche Argumente dauert die Berechnung der Funktion am längsten?

   (c) (0 Punkte) Untersuchen Sie die vorige Frage auch für die Variante, wo die beiden Teile der Disjunktion (`||`) in der ersten Zeile vertauscht sind.

   (d) (3 Punkte) Beweisen Sie die folgenden logischen Implikationen:

```
     istTeil xs ys  ==>  istTeil xs (y:ys)
istTeil (x:xs) ys  ==>  istTeil xs ys
```

   (Für die zweite Aussage können Sie unter anderem Induktion nach der Länge der Liste im zweiten Argument verwenden.)

   (e) (4 Punkte) Die folgende Variante der ersten Zeile führt auf eine äquivalente Funktion:

```
istTeil (x:xs) (y:ys)
     | x==y  = istTeil xs ys
     | x/=y  = istTeil (x:xs) ys
```

   Erklären Sie diesen Sachverhalt (in Worten), und beweisen Sie ihn formal. (Die Aussagen aus der vorigen Aufgabe könnten dabei hilfreich sein.)

   (f) (0 Punkte) Untersuchen Sie diese letzte Variante auf ihre Effizienz.

50. (7 Punkte) Der Computer SUMO-1 ist eine verbesserte Variante von SUMO-0 aus Aufgabe 39. Die wichtigste Neuerung ist, dass eine Variable auch *mehrmals* auf der linken Seite einer Zuweisung auftreten kann. Weiters gibt es eine Druck-Anweisung Px, die den Inhalt der Variablen Vx ausdruckt und das Programm beendet.

   Die Programmierung in SUMATRA soll durch einen Plausibilitätstest unterstützt werden. Dazu sollen Anweisungen, die das gedruckte Endergebnis nicht beeinflussen können, identifiziert werden. Schreiben Sie ein Programm für diese Aufgabe.

51. (5 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.

52. (0 Punkte) Leiten Sie aus der algebraischen Spezifikation für Mengen

$$\begin{aligned}
istenthalten(x, leer) &= falsch \\
istenthalten(x, einfüge(x, M)) &= wahr \\
istenthalten(x, einfüge(y, M)) &= istenthalten(x, M), \qquad \text{für } x \neq y \\
istenthalten(x, lösche(x, M)) &= falsch \\
istenthalten(x, lösche(y, M)) &= istenthalten(x, M), \qquad \text{für } x \neq y
\end{aligned}$$

   (mit den Signaturen von *einfüge*, *lösche*, *leer*, und *istenthalten* wie in der Vorlesung) durch Umformungen folgende Identität her: Für $x \neq y$ gilt

$$istenthalten(u, lösche(x, einfüge(y, M))) = istenthalten(u, einfüge(y, lösche(x, M))).$$

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

54. (6 Punkte) Triangulierungen eines konvexen n-Ecks; binäre Bäume mit $n-2$ Knoten.
   (a) (6 Punkte) Schreiben Sie ein Programm, das alle möglichen Gradlisten der Triangulierungen eines konvexen n-Ecks P_n erzeugt und abzählt. (Dies ist die Umkehrung zu Aufgabe 27d.) Dokumentieren Sie die Ausgabe für $n = 3, 4, 5, 6, 7, 8$. Bestimmen Sie für $n = 9$ nur die Anzahl.

   Ein möglicher Zugang ist, aus den Gradlisten für $(n-1)$-Ecke die Gradlisten für n-Ecke zu erzeugen. Achten Sie darauf, dass Sie Gradlisten nicht mehrfach erzeugen. Dabei kann es hilfreich sein, die Gradlisten in einen digitalen Suchbaum (*trie*) zu speichern.
   (b) (0 Punkte) Man kann Triangulierungen, die durch Drehungen ineinander übergeführt werden, als äquivalent betrachten. (Dies entspricht einer zyklischen Verschiebung der Gradliste.) Es soll nun aus jeder Äquivalenzklasse nur ein Repräsentant erzeugt werden. (Bei $n = 5$ gibt es also nur eine Lösung.)
   (c) (0 Punkte) Die *Gradfolge* ist die sortierte Gradliste. So hat zum Beispiel jede Triangulierung des P_5 die Gradfolge $(0, 0, 1, 1, 2)$. Erzeugen Sie alle möglichen Gradfolgen der Triangulierungen des n-Ecks.
   (d) (∞ Punkte) Schreiben Sie ein effizientes Programm, das testet, ob eine gegebene Folge $(b_1, \ldots, b_n)$ die Gradfolge einer Triangulierung ist.

55. (5 Punkte) Ein Iterator für Mengen.
   (a) (3 Punkte) Implementieren Sie den *Iterator*, den Sie in Aufgabe 51 spezifiziert haben. Sie können von der in der Vorlesung besprochenen Implementierung für Mengen mit bis zu 100 Elementen[1] ausgehen.
   (b) (2 Punkte) Geben Sie die Abstraktionsfunktion an.
   (c) (5 Zusatzpunkte) Beweisen Sie die Korrektheit Ihrer Implementierung.

56. (0 Punkte) In einem Baum sei n_i die Anzahl der Knoten mit i Kindern, das heißt, n_0 ist die Anzahl der Blätter, usw. Die Gesamtzahl der Knoten ist $n = n_0 + n_1 + n_2 + n_3 + \cdots$. Beweisen Sie, dass $n \le 2(n_0 + n_1) - 1$ ist.

57. (a) (0 Punkte) Konstruieren Sie einen optimalen binären Code-Baum (Huffman-Baum) für $n = 15$ Knoten mit den Häufigkeiten $p_i = 1/i$, $(i = 1, \ldots, 15)$.
   (b) (5 Punkte) Wie kann man einen optimalen binären Code (Huffman-Code) in *linearer* Zeit konstruieren, wenn die Häufigkeiten p_i in *sortierter* Reihenfolge gegeben sind?

58. (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?

59. (a) (3 Punkte) Berechnen Sie die Fläche des Fünfecks[2] 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? (Bergünden 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/alp3/Menge.java`
[2] `http://www.inf.fu-berlin.de/~rote/Lere/alp3/5eck`

60. (0 Punkte) Verbesserung des Streckenschnittproblems.
    (a) Erläutern Sie, wie man den in der Vorlesung besprochenen plane-sweep-Algorithmus zum Ausgeben aller k Schnittpunkte von n Strecken so verbessern kann, dass er nur $O(n)$ Speicher benötigt. (Man muss dazu aus der Ereigniswarteschlange auch zwischendurch Ereignisse löschen.)
    (b) Wie muss man den Algorithmus modifizieren, dass er auch bei mehrfachen Schnitten (mehr als zwei Strecken gehen durch einen Punkt) funktioniert?
    (c) Erläutern Sie, welche zusätzlichen entarteten Fälle außer den eben erwähnten Mehrfachschnitten auftreten können.

61. (0 Punkte) Triangulierung eines ebenen Polygons.
    Triangulierungen eines konvexen n-Ecks sind in Aufgabe 27 definiert worden. Triangulierungen eines beliebigen[3] (auch nicht-konvexen) n-Ecks sind genauso definiert. Eine *Trapezoidierung* eines n-Ecks ist eine Zerlegung der Polygons in Dreiecke und Trapezoide,[4] die dadurch entsteht, dass man von jedem Knoten eine vertikale Strecke so weit nach oben und nach unten zieht, bis sie eine andere Polygonseite trifft. Die entstandenen Strecken, die außerhalb des Polygons liegen, werden gelöscht.
    (a) Zeigen Sie, dass auf diese Art wirklich eine Trapezoidierung entsteht.
    (b) Entwerfen Sie einen Algorithmus zur Bestimmung einer Trapezoidierung. (Sie können zum Beispiel dem plane-sweep-Muster folgen.)
    (c) (5 Zusatzpunkte) Beweisen Sie, dass jedes Polygon trianguliert werden kann.
    (d) (5 Zusatzpunkte, alternativ zu (c)) Wie kann man aus einer Trapezoidierung eine Triangulierung gewinnen?

62. (4 Punkte) Bestimmen Sie einen kürzesten spannenden Baum im vollständigen[5] Graphen mit den Knoten $V = \{5, 6, \ldots, 15\}$, wo die Länge der Kante zwischen den Knoten i und j gleich das kleinste gemeinsame Vielfache von i und j ist. Ist die Lösung eindeutig?

63. (0 Punkte) Zeigen Sie, dass der folgende Algorithmus einen kürzesten spannenden Baum T in einem zusammenhängenden Graphen (V, E) mit Kantengewichten w_{ij} berechnet:

    $T := \emptyset$; $S := V$; $u :=$ irgendein Knoten aus V;
    $d_u := 0$; $d_v := \infty$ für alle Knoten $v \neq u$, $v \in V$;
    while $S \neq \emptyset$ **do**
        wähle ein $i \in S$ mit kleinstem Wert d_i;
        if $i \neq u$ **then** $T := T \cup \{(i, f_i)\}$; **end if**;
        für alle Kanten (i, j), die von i ausgehen **do**
            if $w_{ij} < d_j$ **then** $d_j := w_{ij}$; $f_j := i$; **end if**;
        end für;
    end while;

    Worin unterscheidet sich dieser Algorithmus (der übrigens auch von E. W. Dijkstra stammt) vom Dijkstra-Algorithmus für kürzeste Wege?

64. (0 Punkte) Gegeben sei ein vollständiger[5] 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) 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?

---

[3] Wir betrachten nur *einfache* Polygone; das sind solche, deren Rand eine einzelne geschlossene Kurve ohne Selbstüberschneidungen (eine Jordan-Kurve) ist.

[4] Ein Trapezoid ist ein Viereck mit zwei parallelen Seiten.

[5] Ein Graph ist *vollständig*, wenn zwischen allen Paaren von Knoten eine Kante verläuft.