Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Holger Flier

Musterlösung
Datenstrukturen und Algorithmen

26. Januar 2012

verfasst von Holger Flier (allfällige Korrekturen bitte an hflier@inf.ethz.ch)

Aufgabe 1:

1 P a) Das Verfahren von Karatsuba/Ofman zur Multiplikation ganzer Zahlen berechnet das Produkt zweier Zahlen rekursiv anhand einer Formel, die bis auf Addition und Multiplikation mit der Basis (hier: 10) drei Produkte enthält. Geben Sie zwei Zahlen x und y an, für welche diese drei Produkte $(12 \cdot 34)$, $(56 \cdot 78)$ und $(12 \pm 78) \cdot (56 \pm 34)$ sind.

$x = \underline{1278}$, $y = \underline{3456}$

oder

$x = \underline{7812}$, $y = \underline{5634}$

1 P b) Führen Sie auf dem gegebenen Array einen Aufteilungsschritt (in-situ, d.h. ohne Hilfsarray) des Sortieralgorithmus Quicksort durch. Benutzen Sie als Pivot das am rechten Ende stehende Element im Array.

25	12	83	2	58	68	19	34	47	99	37	56	41

25	12	37	2	34	19	41	58	47	99	83	56	68

1 P c) Das untenstehende Array enthält die Elemente eines Min-Heaps in der üblichen Form gespeichert. Wie sieht das Array aus, nachdem das Minimum entfernt wurde und die Heap-Bedingung wieder hergestellt wurde?

4	24	10	38	30	72	77	75	78

1	2	3	4	5	6	7	8	9

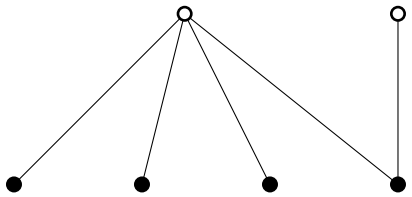
10	24	72	38	30	78	77	75

1 P d) Nehmen Sie an, die Schlüssel $1, 7, 42, 46, 35, 41, 18, 9, 30, 28, 31, 24, 19, 15, 27$ sollen in dieser Reihenfolge mittels Cuckoo-Hashing in Hashtabellen t_1 und t_2 eingefügt werden. Die Hashfunktion für Tabelle t_1 lautet $h_1(k) = k \mod 7$, die für Tabelle t_2 lautet $h_2(k) = 3k \mod 7$. Nehmen Sie ferner an, dass t_1 und t_2 jeweils die Grösse 7 haben. Wie lautet der erste Schlüssel in der oben genannten Folge, der nicht mehr eingefügt werden kann, ohne die Tabellen zu vergrössern (also eine `rehash` Operation auszuführen)?

Schlüssel: 35

1 P

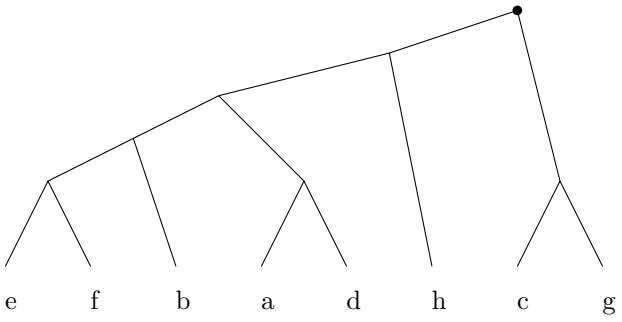
e) Zeichnen Sie einen Baum mit einer ungeraden Anzahl von Kanten und einer minimalen Anzahl von Knoten, der ein Maximum Independent Set der Grösse 4 hat.



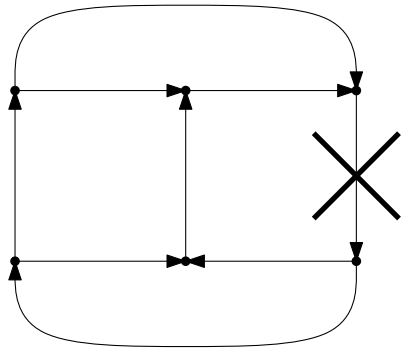
1 P

f) Gegeben sind acht Schlüssel mit der relativen Anzahl der Zugriffe. Erstellen Sie mit Hilfe des Huffman-Algorithmus einen Codierungsbaum (Trie) und zeichnen Sie diesen.

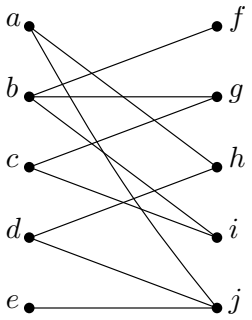
Schlüssel	a	b	c	d	e	f	g	h
Anzahl Zugriffe	31	23	60	29	13	15	68	88



1 P g) Markieren Sie in folgendem Graphen $G = (V, E)$ eine kleinstmögliche Menge S an Kanten, so dass der Graph $G' := (V, E \setminus S)$ eine topologische Sortierung hat :

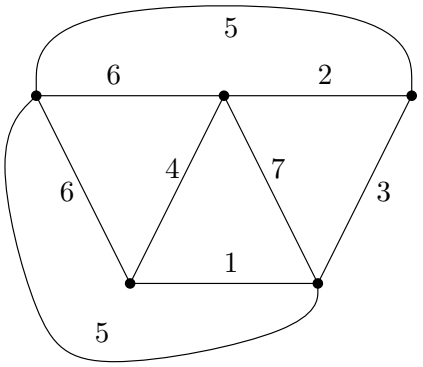


1 P h) Geben Sie eine Teilmenge der Knoten des folgenden bipartiten Graphen an, die mit dem Satz von Hall beweist, dass der Graph kein perfektes Matching hat.



Lösung: $\{b, c, f, g, i\}$ oder $\{a, d, e, h, j\}$

1 P i) Welche Kante des folgenden Graphen wird als erste bei der Berechnung eines minimalen Spannbaums (MST) verworfen, wenn man das Verfahren von Kruskal verwendet?



Lösung: Die Kante mit Gewicht 4.

Aufgabe 2:

1 P a) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, so dass folgendes gilt: Wenn Funktion f links von Funktion g steht, so gilt $f \in O(g)$.

Beispiel: Die drei Funktionen n^3, n^7, n^9 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in O(n^7)$ und $n^7 \in O(n^9)$ gilt.

Lösung:

$$20\log(n), 30\sqrt{n}, n^{\frac{2}{3}}, 10n, n\log n, 3^{\frac{n}{2}}, n!$$

3 P b) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 4 + 2T(\frac{n}{8}) & n > 1 \\ 1 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (d.h. nicht-rekursive) und möglichst einfache Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion.

Hinweise:
(1) Sie können annehmen, dass n eine Potenz von 8 ist.
(2) Für $q \neq 1$ gilt: $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

Lösung: $T = 5\sqrt[3]{n} - 4$

Induktionsvermutung durch Teleskopieren:

$$T = 4\frac{2^k - 1}{2 - 1} + 2^k = 5\sqrt[3]{n} - 4$$

Induktionsverankerung:

$$T(1) = 5\sqrt[3]{1} - 4 = 1$$

Induktionsschritt:

$$T(n) = 4 + 2T(n/8) = 4 + 2(5\sqrt[3]{n/8} - 4) = 5\frac{2}{\sqrt[3]{8}}\sqrt[3]{n} - 4 = 5\sqrt[3]{n} - 4$$

1 P

c) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für folgenden Algorithmus in Theta-Notation an (Sie müssen Ihre Antwort nicht begründen):

```
from j := 1 until j > n loop
   from k := 2 until k > n loop
      k := k * 2
      from l := 1 until l > n loop
         l := l + 10
      end
   end
   j := j + 1
end
```

Lösung: $\Theta(n^2 \log n)$

1 P

d) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für folgenden Algorithmus in Theta-Notation an (Sie müssen Ihre Antwort nicht begründen):

```
from h := 1 until h > n loop
   from j := 1 until j > n*n loop
      j := j * 3
   end
   from k := 2 until k*k > n loop
      k := k + 1
   end
   h := h + 2
end
```

Lösung: $\Theta(n^{\frac{3}{2}})$

1 P

e) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für folgenden Algorithmus in Theta-Notation an (Sie müssen Ihre Antwort nicht begründen):

```
from j := 1 until j > n loop
   from k := 1 until k > n*n*n loop
      k := k + j
   end
   j := j + 2
end
```

Lösung: $\Theta(n^3 \log n)$

Aufgabe 3:

3 P a) Wir verfolgen einen Scanline-Ansatz mit einer vertikalen Scanline, welche von links nach rechts läuft. Dazu sortieren wir die Punkte aufsteigend nach x-Koordinate und gehen alle Punkte in dieser Reihenfolge durch. In der Scanline speichern wir nur einen Punkt ab, nämlich den mit der bisher grössten y-Koordinate. Bei jedem neuen Punkt beim Scan prüft man, ob dessen y-Koordinate ein neues Maximum ist. Falls ja, ist dieser Punkt dominant und man gibt ihn aus (und merkt sich diesen Punkt als neues Maximum).

Dieser Ansatz hat eine Laufzeit von $O(n \log n)$ (für das Sortieren, der Rest geht in Linearzeit).

6 P b) Wir benutzen einen AVL-Baum als Datenstruktur, bei dem alle Knoten zusätzlich in einer Liste verkettet sind (sortiert nach x-Koordinate). Der Baum enthält per Invariante jeweils genau alle dominanten Punkte der aktuellen Punktmenge.

Die Operation `Init`() erstellt in $O(1)$ einen leeren AVL-Baum.

Die Operation `ListDominant`() traversiert einfach den AVL-Baum und gibt alle Punkte (als Liste) aus. Laufzeit: $O(k)$, wobei k die Anzahl dominanter Punkte in der aktuellen Punktmenge ist.

Die Operation `Insert`(x, y) implementiert man wie folgt: Man sucht zuerst die Stelle in der sortierten Reihenfolge, an der der Punkt eingefügt würde, sollte er nicht dominiert sein. Dann prüft man, ob der Punkt durch den Vorgänger in der Liste dominiert wird. Ist dies nicht der Fall, so fügt man den Punkt entsprechend in den AVL-Baum ein. Nun muss man noch alle Punkte löschen, die vom neu eingefügten Punkt dominiert werden. Dies können nur Punkte sein, die sich in der Liste zwischen dem neuen Punkt und dem ersten nicht dominierten Punkt, also einem Punkt (x', y') mit $y' > y$, befinden. Durch die Liste dauert dieser Schritt $O(w + \log m)$, wobei m die Anzahl aktueller Punkte vor dem Insert ist, und w die Anzahl zu entfernender Punkte.

Aufgabe 4:

5 P a) Sei $W := \sum_{i=1}^{n} w_i$ die Summe der Gewichte aller Scheiben. Die Grundidee ist, zu versuchen mit einer Teilmenge der Scheiben das Gewicht $W/2$ zu erreichen. Wenn dies möglich ist, dann kann offensichtlich ein Gleichgewicht erreicht werden. Wir nehmen im Folgenden o.B.d.A. an, dass W gerade ist (warum?).

Man benutzt ein $n \times (\frac{W}{2}+1)$-Tableau, in dem man an Position $E[i,w]$ abspeichert, ob das Gewicht w erreichbar ist, wenn man nur die Scheiben $1,2,\ldots,i$ benutzt. Die Rekursionsgleichung lautet

$$E[i,w] = E[i-1,w] \vee (w_i \leq w \wedge E[i-1, w-w_i]),$$

mit den Startbedingungen $E[1,0] = true$ und $\forall w \in \{1,\ldots,\frac{W}{2}\} : E[1,w] = (w == w_1)$.

Man füllt das Tableau aus, indem man nach der Initialisierung für alle i von 2 bis n jeweils alle w von 0 bis $W/2$ durchgeht.

Die Laufzeit beträgt $O(nW)$.

1 P b) Man kann eine Lösung rekonstruieren, indem man beim Feld $E[n, W/2]$ startet: Wenn da *true* steht, existiert eine Lösung. Startend bei $i = n$ geht man in diesem Fall wie folgt vor: Falls $E[i-1,w] = true$ ist, kommt Scheibe i nach links, und man rekonstruiert die Lösung weiter von $E[i-1,w]$ aus. Falls $E[i-1,w] = false$ ist, kommt Scheibe i nach rechts, und man rekonstruiert die Lösung weiter von $E[i-1, w-w_i]$ aus.

3 P c) Die Grundidee bleibt die selbe: Sei $W := \sum_{i=1}^{n} w_i$ die Summe der Gewichte aller Scheiben. Um ein Gleichgewicht zu erreichen, muss die Menge C das Gesamtgewicht $W/2$ haben, und die Mengen A und B je Gewicht $W/4$. Wenn W nicht durch 4 teilbar ist, ist sicher keine Lösung möglich. Falls W durch 4 teilbar ist, füllt man ein dreidimensionales Tableau aus, mit Dimensionen $n \times (W/2+1) \times (W/4+1)$: In $E[i,w,w']$ steht, ob man aus den Scheibenn $1,2,\ldots,i$ zwei Mengen bilden kann (ohne unbedingt alle zu verwenden), so dass die eine Menge Gesamtgewicht w hat und die andere Menge Gesamtgewicht w'. Nachdem dieses Tableau ausgefüllt wurde, kann die Lösung in $E[n, W/2, W/4]$ abgelesen werden (da das Gesamtgewicht aller Scheiben gleich W ist).

Die Rekursionsgleichung ist
$E[i,w,w'] = E[i-1,w,w'] \vee (w_i \leq w \wedge E[i-1, w-w_i, w']) \vee (w_i \leq w' \wedge E[i-1, w, w'-w_i])$,

mit der Initialisierung $E[1,0,0] = true$, $\forall w > 0 : E[1,w,0] = (w == w_1)$, $\forall w' > 0 : E[1,0,w'] = (w' == w_1)$.

Das Tableau füllt man nach der Initialisierung wie folgt aus:

```
from
  i := 2 until i > n loop
  from w := 0 until w > W//4 loop
    from w' := 0 until w' > W//2 loop
       E[i,w,w'] = E[i-1,w,w'] or (w_i <= w  and  E[i-1, w-w_i, w'])
                               or (w_i <= w' and  E[i-1, w, w'-w_i])
      w' := w' + 1
    end
    w := w + 1
  end
  i := i + 1
end
```

Die Laufzeit beträgt hierfür $O(nW^2)$.

Aufgabe 5:

Gegeben sind m Rechtecke und n Punkte.

a) Wir benutzen einen Scanline-Ansatz, beispielsweise einer vertikalen Scanline, welche von links nach rechts läuft. Haltepunkte sind jeweils die x-Koordinaten von Anfang und Ende der Rechtecke sowie der gegebenen Punkte. Man braucht in der Scanline eine Datenstruktur, welche Aufspiess-Abfragen beantworten kann, z.B. einen Intervall-Baum. Bei den Haltepunkten, die Rechteck-Anfang (bzw.) Ende sind, fügt man das entsprechende y-Intervall in den Intervall-Baum ein (bzw. löscht es). Bei den Haltepunkten, die einem Punkt entsprechen, führt man eine Aufspiess-Abfrage im Intervall-Baum durch. Für jedes aufgespiesste Intervall/Rechteck merken wir uns, dass es einen Punkt enthält, und entfernen es aus dem Intervallbaum (warum?). Wird ein Haltepunkt erreicht, der dem Ende eines Rechtecks entspricht, und enthält dieses Rechteck keinen Punkt, so wird es ausgegeben.

b) Der Aufbau des leeren Skeletts des Intervall-Baums benötigt $O(m)$ Zeit. Die Kosten für Einfügen und Löschen der Intervalle im Intervall–Baum betragen je Intervall $O(\log m)$, und es gibt $O(m)$ davon. Die $O(n)$ Aufspiess-Abfragen kosten $O(k + n \log m)$, wobei k die Anzahl der Rechtecke, welche Punkte enthalten, bezeichnet. Das Löschen dieser Rechtecke kostet $O(k \log m)$ Zeit. Die Ausgabe der verbleibenden Rechtecke geschieht in $O(m)$. Mit $k \leq m$ hat man insgesamt Kosten von $O(m + (m + n) \log m) = O((m + n) \log m)$.

   Das Sortieren der $2m + n$ Haltepunkte kann man in höchstens $O((m + n) \log(m + n))$ Zeit erledigen. Der Gesamtaufwand beträgt somit $O((m + n) \log(m + n))$ für diese Lösung.

c) Eine mögliche Lösung ist, alle Punkte in einem 2d-Range-Tree zu speichern, der so abgewandelt ist, dass man effizient die Anzahl Punkte in einem Rechteck bestimmen kann (man kann dazu z.B. in jedem Knoten speichern, wie viele Schlüssel der linke Teilbaum enthält). Dann kann man einfach für jedes Rechteck eine Abfrage machen, und schliesslich das Rechteck mit den meisten Punkten darin ausgeben.

   Den 2d-Range-Tree für die n Punkte baut man in $O(n \log n)$ Zeit auf.

   Ein Query kostet jeweils $O(\log^2 n)$ Zeit, wobei insgesamt m Queries ausgeführt werden. Insgesamt kommt man also auf eine Laufzeit von $O(n \log n + m \log^2 n)$.

   Mittels Fractional Cascading kann man die Laufzeit einer Query auf $O(\log n)$ verbessern, d.h. insgesamt würde die Laufzeit nur noch $O(n \log n + m \log n)$ betragen.