Probeklausur der Tutoren

INFORMATIK II SS2005

Lösungsvorschlag

Aufgabe 1: Verständnis- und Wissensfragen (6 Punkte)

Kreuzen Sie an, ob die Aussage wahr ([W]) oder falsch ([F]) ist.

Hinweis: Jedes korrekte Kreuz zählt 0,5 Punkte, jedes falsche Kreuz bewirkt 0,5 Punkte Abzug! Die Teilaufgabe wird mindestens mit 0 Punkten bewertet.

W	**F**	Bei der Konstruktion eines Huffmanbaums vereinigt der Greedy-Algorithmus zuerst die großen Häufigkeiten, damit diese dann weit oben im Baum stehen.
W	F	Es gibt prädikatenlogische Formeln, die keine äquivalente Formel in konjunktiver Normalform besitzen.
W	**F**	Zu jeder prädikatenlogischen Formel gibt es eine äquivalente Formel in Skolemnormalform.
W	**F**	Zu jeder Häufigkeitsverteilung eines Alphabets, ist der Huffmanbaum und sind damit die Huffmancodeworte eindeutig bestimmt.
W	**F**	Für alle Graphen G gilt: G ist antisymmetrisch $\Leftrightarrow \neg(G$ ist symmetrisch$)$
W	F	Für alle Graphen G gilt: G ist zyklisch $\Leftrightarrow \neg(G$ ist azyklisch$)$
W	**F**	Beim O-Kalkül gilt: $g(n) \in O(f(n)) \Rightarrow f(n) \in O(g(n))$
W	F	Monte-Carlo-Algorithmen terminieren immer.
W	**F**	Der Algorithmus von Pollard-Rho berechnet die Primfaktorzerlegung einer natürlichen Zahl.
W	**F**	Es gilt: $O(0) = O(1)$
W	**F**	Rot-Schwarz-Bäume sind immer perfekt ausbalanciert.
W	**F**	$\Theta(f(n)) = \Theta(g(n)) \iff f(n) \in o(g(n)) \wedge g(n) \in \omega(f(n))$.

Lösung:

(a) Der Algorithmus fasst immer Zeichengruppen mit den kleinsten Wahrscheinlichkeiten zusammen.

(b) Formeln mit Quantoren können nicht in KNF gebracht werden.

(c) Durch das Einfügen von Skolemfunktionen erhält man i. a. nur erfüllbarkeitsäquivalente Formeln.

(d) Der Huffmancode ist optimal, die zugehörigen Worte allerdings nicht eindeutig, wenn gleiche Wahrscheinlichkeiten im Verfahren auftreten.

(e) Die Gleicheit als Relation erfüllt beide positiven Aussagen.

(f) Siehe Definition.

(g) Gegenbeispiel: $n \in O(n^2)$ aber $n^2 \notin O(n)$.

(h) Ein Monte-Carlo-Algorithmus terminiert immer; er kann aber falsche Ergebnisse liefern.

(i) Der Algorithmus von Pollard-Rho bestimmt einen Primfaktor, keine komplette Zerlegung.

(j) $O(0)$ enthält nur die Nullfunktion; $O(1)$ enthält aber alle konstanten Funktionen.

(k) Es wird lediglich garantiert, daß die Höhe eines Rot-Schwarz-Baum nie größer als das Doppelte der Höhe eines optimalen binären Suchbaums ist.

(l) Gegenbeispiel: $f(n) = g(n) = 1$. Dann gilt: $\Theta(1) = \Theta(1)$, jedoch $\nexists x_0$ mit $\forall x > x_0 : 1 < \frac{1}{2} \cdot 1$ und somit $1 \notin o(1)$.

Aufgabe 2: Relationen & Graphen (3 + 1 + 5 Punkte)

(a) Wieviele Kanten kann ein *irreflexiver* gerichteter Graph mit n Knoten höchstens haben. Beweisen Sie ihre Vermutung mittels Induktion.

Hinweis: Irreflexiv ist nicht dasselbe wie *nicht reflexiv*.

(b) Wieviele Kanten kann ein gerichteter Graph mit n Knoten höchstens haben, wenn er weder *reflexiv* noch *transitiv* ist.

(c) *Defintion:* Ein binärer Baum heißt Bruder-Baum, wenn

(i) jeder innere Knoten 1 oder 2 Nachfolger hat,
(ii) jeder unäre Knoten einen binären Bruder hat,
(iii) alle Blätter dieselbe Tiefe haben.

Wie viele Blatt-Knoten hat ein Bruder-Baum der Höhe 4 (Wurzel: Höhe 0, Blätter werden nicht gezählt), falls er eine minimale Anzahl von Blatt-Knoten hat?

Lösung:

(a) $n^2 - n = n(n-1)$

Behauptung: Ein irreflexiver Graph mit n Knoten hat höchsten $n^2 - n$ Kanten.

IA: $n = 1$. Der Graph kann keine Kanten haben. $\Rightarrow 0 = 1^2 - 1$.
$n = 2$. Zwei Kanten: eine in jede Richtung. $\Rightarrow 2 = 2^2 - 2$.
IV: Die Behauptung gilt für einen Graph mit n Knoten und $n^2 - n$ Kanten.
IS: Wir erweitern den Graph um einen neuen Knoten. Dann verbinden wir diesen Knoten mit allen alten Knoten, und fügen so $2n$ neue Kanten ein (n Kanten in Richtung der alten Knoten und n in Richting des neuen Knoten). Es können keine weiteren Kanten von oder zu dem neuen Knoten gezogen werden, da die einzige noch nicht gezogene Kante die reflexive ist. Zusammen mit der IV folgt, dass die Kantenanzahl maximal ist. Dann ist $|E'| = |E| + 2n = n^2 - n + 2n = n^2 + 2n + 1 - n - 1 = (n+1)^2 - (n+1)$ □

(b) $n^2 - 1$ für $n \in \mathbb{N}_{>1}$, da erst ab 2 Knoten eine Nicht-Transitivität möglich ist. In dem Fall gilt für einen fast vollständigen Graphen (ohne reflexive Kante bei Knoten 1): $(1,2) \in E$ und $(2,1) \in E$, aber $(1,1) \notin E$.

(c) 13 Blätter. Der Baum ist ein Fibonacci-Baum.

Aufgabe 3: O-Kalkül (6 + 4 Punkte)

(a) Prüfen Sie, in welche der angeführten Komplexitätsklassen die *worst-case* Laufzeitkomplexität der folgendenen Algorithmen fällt.

Hinweis: Tragen Sie in jedes Kästchen entweder ein ✓ falls der Algorithmus in der Klasse ist, oder ein ✘ falls dieser nicht in der Klasse enthalten ist. Jeder korrekte Haken oder Kreuz zählt 0,25 Punkte, jeder falsche Haken oder Kreuz bewirkt 0,5 Punkte Abzug! Jeder der Algorithmen wird mindestens mit 0 Punkten und maximal mit 1 Punkt bewertet.

	$\omega(1)$	$\Theta(\log n)$	$O(n)$	$o(n \log n)$	$O(n^2)$	$\Omega(n^2)$	$O(n^3)$
Insertionsort	✓	✘	✘	✘	✓	✓	✓
Mergesort	✓	✘	✘	✘	✓	✘	✓
Quicksort	✓	✘	✘	✘	✓	✓	✓
Floyd-Warshall	✓	✘	✘	✘	✘	✓	✓
Rot-Schwarz-Baum-Insert	✓	✓	✓	✓	✓	✘	✓
Lookup in einem Hashtable	✓	✘	✓	✓	✓	✘	✓

(b) Beweisen Sie für zwei Funktionen $f, g : \mathbb{N} \to \mathbb{N}$ anhand der Definition des O-Kalküls: $\max\{f, g\} \in \Theta(f + g)$.

Lösung:

(a) s.o.

(b) Da für alle $n \in \mathbb{N}$ die Ungleichung

$$\frac{f(n) + g(n)}{2} \leq \max\{f(n), g(n)\} \leq f(n) + g(n)$$

gilt, ist die Bedingung

$$\exists c_1, c_2 > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 : \; c_1(f(n) + g(n)) \leq \max\{f(n), g(n)\} \leq c_2(f(n) + g(n))$$

mit $c_1 = \frac{1}{2}$, $c_2 = 1$ und $n_0 = 1$ erfüllt.

Aufgabe 4: Haskell (5 + 7 Punkte)

(a) Gegeben sei eine duplikatfreie Liste `l :: [a]`. Wir bezeichnen die Liste aller möglichen Teillisten mit Auslassungen als Kombinationen `c :: [[a]]` von `l`. Zum Beispiel sind die Kombinationen von `[1, 2, 3]` gerade `[ [1,2,3], [1,2], [1,3], [2,3], [1], [2], [3], [] ]`. Die relative Position der Kombination ist dabei unerheblich.

Schreiben Sie eine Funktion `combs :: [a] -> [[a]]`, welche die Kombinationen ihres Arguments berechnet.

Hinweis: Sie können dabei beliebige Funktionen der Standardbibliothek verwenden.

(b) Der ADT Menge modelliert eine Menge im mathematischen Sinne. Vervollständigen Sie das Modul Menge indem Sie die Operationen implementieren.

Hinweis: Eine Menge im mathematischen Sinne darf insbesondere keine zwei gleichen Elemente enthalten!

```
module Menge where

type Menge a = [a]

leereMenge    :: (Eq a) => Menge a                        -- liefert eine leereMenge
istLeer       :: (Eq a) => Menge a -> Bool                -- Menge leer?
hatElement    :: (Eq a) => Menge a -> a -> Bool           -- Element in der Menge enthalte
einfuegen     :: (Eq a) => Menge a -> a -> Menge a        -- fügt Element in eine Menge ei
vereinigung   :: (Eq a) => Menge a -> Menge a -> Menge a  -- vereinigt zwei Mengen
schnitt       :: (Eq a) => Menge a -> Menge a -> Menge a  -- schneidet zwei Mengen
```

Lösung:

(a) Varianten:

- –
```
combs []     = [[]]
combs (x:xs) = (map (x:) co) ++ co
               where co = combs xs
```
  – Gleicher Ansatz in Mengenschreibweise:
```
combs []     = [[]]
combs (x:xs) = [x:ys|ys<-zs]++zs
               where zs = combs xs
```
- Schrittweises Auslassen (erzeugt Doubletten):
```
combs xs = xs:[zs | ys <- auslassungen xs, zs <- combs ys]
auslassungen []     = []
auslassungen (x:xs) = xs:[(x:ys) | ys <- (auslassungen xs)]
```
- Aufbau über die Potenzmenge:
```
combs xs = [comb i xs | i<- [0..max]]
where max=2^(length xs)-1
comb _ [] = []
comb i (x:xs) | r==0 = comb q xs
              | r==1 = x : (comb q xs)
              where (q,r) = divMod i 2
```

(b)
```
leereMenge = []

istLeer [] = True
istLeer _  = False

einfuegen xs e | hatElement xs e = xs
               | otherwise       = e:xs

hatElement [] _     = False
hatElement (x:xs) e | x == e    = True
                    | otherwise = hatElement xs e

vereinigung xs []     = xs
vereinigung xs (y:ys) = vereinigung (einfuegen xs y) ys

schnitt xs []     = []
schnitt xs (y:ys) | hatElement xs y = einfuegen (schnitt xs ys) y
                  | otherwise       = schnitt xs ys
```

Aufgabe 5: Prädikatenlogik (5 + 4 Punkte)

(a) Gegeben sei die prädikatenlogische Formel $F = \neg\Big(\exists z\big(P(z) \wedge \forall y(Q(y) \rightarrow \forall x\ R(x,y,z))\big)\Big)$. Bereinigen Sie zunächst die Operatoren. Stellen Sie dann die bereinigte Pränexform her und erstellen Sie die Skolemform von F.

(b) Über den natürlichen Zahlen (ohne Null) seien folgende Prädikate definiert:

$$P(x,y) := \{(x,y) \mid x \text{ teilt } y \text{ ohne Rest}\}$$
$$Q(x,y) := \{(x,y) \mid x = y\}$$
$$R(x,y) := \{(x,y) \mid x < y\}$$

Formulieren Sie die folgenden Aussagen in Prädikatenlogik.

- x ist eine Primzahl
- x ist eine gerade Zahl
- x ist der ggT von y und z
- x und y sind teilerfremd

Lösung:

(a)
$$\begin{aligned}
F &\equiv \neg\Big(\exists z\big(P(z) \wedge \forall y(Q(y) \rightarrow \forall x\ R(x,y,z))\big)\Big) \\
&\equiv \forall z\,\neg\big(P(z) \wedge \forall y(Q(y) \rightarrow \forall x\ R(x,y,z))\big) \\
&\equiv \forall z\big(\neg P(z) \vee \exists y\,\neg(Q(y) \rightarrow \forall x\ R(x,y,z))\big) \\
&\equiv \forall z\big(\neg P(z) \vee \exists y(Q(y) \wedge \exists x\,\neg R(x,y,z))\big) && \text{(bereinigt)} \\
&\equiv \forall z\,\exists y\,\exists x\big(\neg P(z) \vee (Q(y) \wedge \neg R(x,y,z))\big) && \text{(Pränexform)} \\
&\equiv \forall z\big(\neg P(z) \vee (Q(f(z)) \wedge \neg R(g(z),f(z),z))\big) && \text{(Skolemform)}
\end{aligned}$$

(b) Primzahl: $\forall t(\neg P(t,x) \vee Q(t,1) \vee Q(t,x))$
Gerade Zahl: $P(2,x)$
ggT: $P(x,y) \wedge P(x,z) \wedge \forall t(P(t,y) \wedge P(t,z) \rightarrow Q(t,x) \vee R(t,x))$
teilerfremd: $\forall t(P(t,x) \wedge P(t,y) \rightarrow Q(t,1))$

Aufgabe 6: Rekurrenzrelationen (6 + 4 + 4 Punkte)

(a) Gegeben sei folgende Haskellfunktion (1,5+1+1,5+2 Punkte):

```
s [x] = [x]
s xs  = min:s(o min xs)
        where min = minimum xs
              o x (y:ys) | x==y      = ys
                         | otherwise = y:(o x ys)
```

(i) Was macht die obige Funktion?

(ii) Welche Eingabe stellt den Worst-Case bezogen auf die Anzahl der Vergleiche dar?

(iii) Leiten Sie eine Rekurrenz für die Anzahl der Vergleiche her.
Hinweis: Nehmen Sie an, `minimum` habe einen Aufwand von n bei einer Eingabelänge von $n+1$.

(iv) Lösen Sie die Rekurrenz.

(b) Gegeben seien

- die ersten k Glieder einer Rekurrenz: $f_0,\ f_1, \ldots, f_{k-1}$,
- die Differenzen dieser Glieder: $d_i = f_i - f_{i-1}$

- und deren Differenzen $D_i = d_i - d_{i-1}$.

Weiterhin gelte: $D_{i+1} = d_i$ für alle $i \geq 0$. In welcher Komplexitätsklasse liegt f? Begründen Sie Ihre Meinung.

(c) Finden Sie aus den Angaben von Teil (b) eine rekursive Formel für die Rekurrenz.

Lösung:

(a) (i) SelectionSort

(ii) Eingabe $(a_1 : a_2 : .. : a_n)$ mit $a_1 > a_2 > ... > a_n$

(iii) $T(1) = 0$, $T(n) = T(n-1) + \underbrace{n-1}_{\text{für } \texttt{minimum}} + \underbrace{n}_{\text{für o(hne)}}$

(iv) Herumprobieren führt zu

IA: $T(1) = 0$, $T(2) = 0 + 1 + 2 = 3$, $T(3) = 3 + 2 + 3 = 8$, $T(4) = 8 + 3 + 4 = 15$
$\Rightarrow$ Behauptung: $T(n) = n^2 - 1$

IV: Die Behauptung gelte für ein $n \in \mathbb{N}$.

IS: $n \hookrightarrow n + 1$.

$$\begin{aligned} T(n+1) &= T(n) + n + (n+1) \\ &= n^2 - 1 + 2n + 1 \\ &= (n^2 + 2n + 1) - 1 \\ &= (n+1)^2 - 1 \end{aligned}$$

(b) Da die Differenzen jeweils die „Ableitung“ darstellen und die „Ableitung“ sich immer wiederholt, d.h. die Ableitung gleich der ihrer ursprüglichen Funktion ist, kann auf jeden Fall geschlossen werden, dass die Rekurrenz exponentiell wächst $\Rightarrow O(x^n)$.

Weiter kann man schließen: $D_{i+1} = d_{i+1} - d_i = d_i \Rightarrow d_{i+1} = 2d_i$. Das heißt

$$\begin{aligned} f_1 &= f_0 + d_1, \\ f_2 &= f_1 + d_2 = (f_0 + d_1) + (2d_1) = f_0 + 3d_1, \\ f_3 &= f_2 + d_3 = f_2 + 2d_2 = (f_0 + 3d_1) + 4d_1 = f_0 + 7d_1. \end{aligned}$$

Daraus kann man erkennen, dass f_n in $O(2^n)$ liegen muss.

(c) Wir müssen zuerst die Sonderfälle f_1, f_0 und f_i für $i \leq 0$ betrachten. Zunächst legen wir fest, dass für alle $i \leq 0$ gelte: $f_i = 0$. Für f_1 gilt nun:

$$f_1 = 3f_0 - 2f_{-1} = 3f_0 \neq f_0 + d_1.$$

Deswegen führen wir den Ausdruck $(d_1 - 2f_0)[n = 1]$ ein, wobei $[n = 1]$ wieder eins ergibt, wenn $n = 1$, und sonst null. Für f_0 müssen wir analog noch den Ausruck $f_0[n = 0]$ hinzunehmen. Damit lautet die Formel für die Rekurrenz:

$$f_n = 3f_{n-1} - 2f_{n-2} + (d_1 - 2f_0)[n = 1] + f_0[n = 0].$$