

Lösungsvorschlag

Aufgabe 1: Graphentheorie (2 + 1 + 2 Punkte)

Gegeben ist der folgende ungerichtete Graph $G_n = (V, E)$ mit

$$V = \bigcup_{i=0}^{n-1} V_i \quad \text{wobei} \quad V_i = \{v_{i,0}, \dots, v_{i,m_i}\}, \quad E = \{(v_{i,j}, v_{x,y}) \mid i \neq x\}.$$

Hinweis: Der Graph besteht aus n Knotengruppen, die jeweils aus $m_0, \dots, m_{n-1} > 0$ Knoten bestehen. Dabei sind die Knoten innerhalb einer Gruppe nicht miteinander verbunden, aber mit allen anderen Knoten außerhalb der Gruppe.

- Wieviele Kanten besitzt G_n abhängig von n und $m_0, \dots, m_{n-1} \in \mathbb{N}$.
- Die Kante $(x, y) \in E$ ist *transitiv*, wenn $\exists z \in V ((x, z) \in E \wedge (z, y) \in E)$. Wieviele transitiven Kanten hat G_n ?
- Skizzieren sie eine Adjazenzmatrix von G_n .

Lösung:

- Die Anzahl der Kanten ist

$$\sum_{i=0}^{n-1} m_i \sum_{j=i+1}^n m_j.$$

- Für $n = 2$ gibt es keine transitiven Kanten. Aber für $n \geq 3$ ist jede Kante transitiv, somit das gleiche Ergebnis wie oben.
- Die Adjazenzmatrix besteht aus 0- und 1- Blöcken, wobei A einen 1- und D einen 0-Block darstellt. Die Matrix hat demnach folgende Form:

$$\begin{pmatrix} D_{0,0} & A_{0,1} & \dots & A_{0,n-1} \\ A_{1,0} & D_{1,1} & \dots & A_{1,n-1} \\ \dots & & \ddots & \vdots \\ A_{n-1,0} & A_{n-1,1} & \dots & D_{n-1,n-1} \end{pmatrix}$$

Jeder Knoten ist mit jedem anderen Knoten verbunden. Daher ist die Matrix voll mit 1. Die Knoten innerhalb einer Gruppe ist allerdings kantenlos, deshalb entstehen dort die 0- Blöcke.

Da kein Knoten reflexiv sein kann, ist die Diagonale der Matrix immer leer.

Aufgabe 2: Haskell (5 Punkte)

Betrachten Sie folgende Grammatik in EBNF Form:

```
S = 'x'
  | 'a' S 'a'
  | ...
  | 'z' S 'z'.
```

Implementieren Sie eine Funktion `isValid :: String -> Bool` die überprüft ob ein Wort zur Sprache dieser Grammatik gehört. Beispiele:

```
isValid "baxab" = True
isValid "axxa"  = False
isValid "baab"  = False
isValid "xxxxx" = True
```

Hinweis: Sie dürfen die Standardbibliothek verwenden, folgende Funktionen könnten Ihnen behilflich sein:

```
head :: [a] -> a      -- Liefert das erste Element einer Liste
last  :: [a] -> a      -- Liefert das letzte Element einer Liste
tail  :: [a] -> [a]    -- Entfernt das erste Element aus einer Liste
init  :: [a] -> [a]    -- Entfernt das letzte Element aus einer Liste
```

Lösung:

```
isValid :: String -> Bool
isValid "" = False
isValid "x" = True
isValid xs | (head xs == last xs) && (isAlpha (head xs)) = isValid (tail (init xs))
           | otherwise = False

isAlpha 'a' = True
isAlpha 'b' = True
...
isAlpha 'z' = True
isAlpha _  = False
```

Aufgabe 3: Prädikatenlogik (1 + 4 + 2 Punkte)

- (a) Sei $F = \forall x_1 \forall x_2 (P_1(x_1, x_2) \wedge \neg(\forall x_3 P_2(x_1, x_2, x_3)))$ eine Prädikatenlogische Formel. Ist die Pränexnormalform von F identisch zur Skolemnormalform von F ? (1 Punkt)
- (b) Sei $F = P(x, f(y), b)$. Geben Sie den kleinstmöglichen Unifikator von F mit folgenden Formeln an.
- (i) $G_1 = P(z, f(w), b)$
 - (ii) $G_2 = P(x, f(a), b)$
 - (iii) $G_3 = P(g(z), f(a), b)$
 - (iv) $G_4 = P(c, f(a), a)$
- (c) Gegeben sei die Struktur (A, R, P) mit

$$A = \{1, 2, 3, 4, 5, 6, 7\}$$
$$R = \{(1, 2), (2, 3), (4, 5), (5, 6), (6, 7), (7, 4)\}$$
$$P = \{2, 4\}$$

Beweisen oder widerlegen Sie die folgenden Aussagen.

- (i) $\forall v_1 \exists v_2 R(v_1, v_2)$
- (ii) $\forall v_1 \left(P(v_1) \rightarrow \exists v_2 \exists v_3 (R(v_2, v_1) \wedge R(v_1, v_3)) \right)$

Lösung:

- (a) Man muss hier beachten, dass nach *identisch* und nicht *äquivalent* gefragt ist. Angesichts der Tatsache, dass F keinen Existenzquantor und keine freien Variablen besitzt, lässt sich sowas vermuten. Da aber in der Normalform keine \neg vor dem Quantoren steht, muss diese Negation aufgelöst werden. Dadurch entsteht ein Existenzquantor:

$$F \equiv \forall x_1 \forall x_2 (P_1(x_1, x_2) \wedge (\exists x_3 \neg P_2(x_1, x_2, x_3)))$$

Die Pränexnormalform ist

$$\text{PNF}(F) = \forall x_1 \forall x_2 \exists x_3 (P_1(x_1, x_2) \wedge \neg P_2(x_1, x_2, x_3))$$

wobei x_3 in der Skolemnormalform durch $f_1(x_1, x_2)$ ersetzt wird:

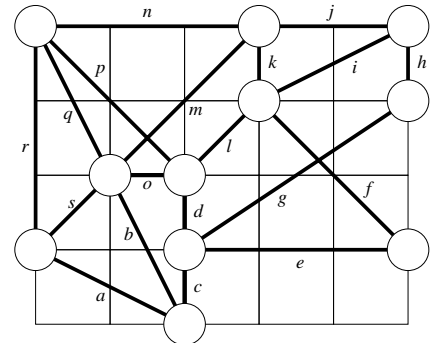
$$\text{SNF}(F) = \forall x_1 \forall x_2 (P_1(x_1, x_2) \wedge (\neg P_2(x_1, x_2, f_1(x_1, x_2))))$$

$\Rightarrow \text{PNF}(F) \neq \text{SNF}(F)$, aber $\text{PNF}(F) \equiv \text{SNF}(F)$.

- (b) (i) $s = \{x/z, y/w\}$
 (ii) $s = \{y/a\}$
 (iii) $s = \{x/g(z), y/a\}$
 (iv) Keine Unifikation möglich! a und b sind beide Konstanten.
- (c) (i) Die Behauptung gilt nicht, wähle $v_1 = 3$ als Gegenbeispiel.
 (ii) Die Aussage ist korrekt. Für $v_1 = 2$ wähle $v_2 = 1, v_3 = 3$ und für $v_1 = 4$ wähle $v_2 = 7$ und $v_3 = 5$.

Aufgabe 4: Algorithmus von Kruskal (4 + 4 Punkte)

- (a) Geben Sie in Pseudocode den Algorithmus von Kruskal zur Bestimmung des minimalen spannenden Baums in einem Graphen mit Kantengewichten an.
- (b) Geben Sie eine Reihenfolge an, in der der Algorithmus von Kruskal, angewandt auf den folgenden Graphen, die Kanten in den minimalen spannenden Baum aufnimmt. In dem Graphen entspricht die Länge einer Kante ihrem Gewicht.



Lösung:

- (a) Solange noch Kanten frei, die keinen Zyklus erzeugen, nimm kürzeste Kante (u, v) die keinen Zyklus erzeugt in den minimalen Spannbaum auf.
- (b) Länge 1: c, d, o, k, h
 Länge $\sqrt{2}$: s, l
 Länge 2: j
 Länge $\sqrt{5}$: q
 Länge $\sqrt{8}$: f
 Länge $> \sqrt{8}$: keine

Aufgabe 5: Rekurrenzrelationen (3 Punkte)

Geben Sie die Rekurrenzrelation für den Aufwand (Rechenzeit) $T(n)$ für den Aufruf `foo(x, x+n)` an. Auflösen der Rekurrenz wird *nicht* verlangt. Die Komplexität von `bar(...)` sei konstant $O(1)$.

Hinweis: Der folgende Quelltext ist nicht als Java, sondern als Pseudo-Code zu verstehen.

```
void foo(int i, int j)
{
    if (j-1 < 4) return;
    bar (1, i);
    foo (i, i+(j-i)/4);
    for (int k=0; k<=j-i; k++) bar (k, j);
    foo (i+(j-i)/4 + 1, j);
}
```

Lösung:

$$T(n) = \begin{cases} O(1), & \text{für } n < 4 \\ T(\frac{n}{4}) + T(\frac{3n}{4}) + O(n), & \text{sonst} \end{cases}$$

Aufgabe 6: Rekurrenzen (4 + 4 Punkte)

Fortsetzung von Aufgabe 6 aus der Probeklausur:

- (d) Berechnen Sie mit Hilfe der Methode der generierenden Funktion eine geschlossene Form für f_n .
- (e) Beweisen Sie ausgehend von Teil (c) mit Hilfe vollständiger Induktion, dass Ihr Ergebnis aus Teil (d) richtig ist.

Lösung:

- (d) Jetzt wird die Gleichung mit z^n multipliziert und über alle n aufsummiert.

$$G(z) := \sum_n f_n z^n = \sum_n 3f_{n-1} z^n - \sum_n 2f_{n-2} z^n + \sum_n (d_1 - 2f_0)[n=1]z^n + \sum_n f_0[n=0]z^n$$

Etwas vereinfacht:

$$G(z) := \sum_n f_n z^n = \sum_n 3f_n z^{n+1} - \sum_n 2f_n z^{n+2} + (d_1 - 2f_1)z + f_0$$

$$\text{Und nach } G(z) \text{ aufgelöst: } G(z) = \frac{(d_1 - 2f_1)z + f_0}{1 - 3z + 2z^2}$$

Diese Gleichung muss nun in eine Form gebracht werden, so dass $\frac{a}{(1-pz)^{m+1}} = \sum_{n \geq 0} \binom{n+m}{m} a p^n z^n$ anwendbar ist.

$$\text{Dazu wollen wir den Bruch von } G(z) \text{ wie folgt aufteilen: } \frac{(d_1 - 2f_1)z + f_0}{1 - 3z + 2z^2} = \frac{a}{1-pz} + \frac{b}{1-qz}$$

Die Aufteilung des Nenners ist einfach, da sich die Nullstellen $z_1 = 1$ und $z_2 = \frac{1}{2}$ leicht erraten lassen. Damit erhalten wir $p = 2$ und $q = 1$: $\frac{(d_1 - 2f_1)z + f_0}{1 - 3z + 2z^2} = \frac{a}{1-2z} + \frac{b}{1-z}$

Die Bestimmung von a und b erfolgt über einen Koeffizientenvergleich: $(d_1 - 2f_0)z + f_0 = a(1 - z) + b(1 - 2z) \Leftrightarrow (d_1 - 2f_0)z + f_0 = -(a + 2b)z + (a + b) \Rightarrow b = f_0 - d_1$ und $a = d_1$

Damit erhalten wir: $G(z) = \frac{(d_1 - 2f_1)z + f_0}{1 - 3z + 2z^2} = \frac{d_1}{1-2z} + \frac{f_0 - d_1}{1-z} = \sum_{n \geq 0} \binom{n+0}{0} d_1 2^n z^n + \sum_{n \geq 0} \binom{n+0}{0} (f_0 - d_1) 1^n z^n$
und für unsere Rekurrenz kann man die geschlossene Form ablesen: $g_n = d_1 2^n + (f_0 - d_1) 1^n = f_0 + (2^n - 1)d_1$

- (e) Zu zeigen: $g_n = f_0 + (2^n - 1)d_1 = f_n = 3f_{n-1} - 2f_{n-2}$.

- I.A.: $n = 0: g_0 = f_0 + (2^0 - 1)d_1 = f_0 \checkmark$
 $n = 1: g_1 = f_0 + (2^1 - 1)d_1 = f_0 + d_1 = f_1 \checkmark$
 $n = 2: g_2 = f_0 + (2^2 - 1)d_1 = f_0 + 3d_1 = f_2 \checkmark$
- I.V.: Für beliebiges, festes $n \geq 2$ gelte: $g_n = f_0 + (2^n - 1)d_1 = f_n$ und $g_{n-1} = f_0 + (2^{n-1} - 1)d_1 = f_{n-1}$
- I.S.: $n \hookrightarrow n + 1 \quad f_{n+1} \stackrel{\text{Def. } f_n}{=} 3f_n - 2f_{n-1} \stackrel{\text{I.V.}}{=} 3g_n - 2g_{n-1} \stackrel{\text{Def. } g_n}{=} 3(f_0 + (2^n - 1)d_1) - 2(f_0 + (2^{n-1} - 1)d_1) = 3f_0 + 3 \cdot 2^n d_1 - 3d_1 - 2f_0 - \underbrace{2 \cdot 2^{n-1}}_{2^n} d_1 + 2d_1 = f_0 + 2 \cdot 2^n d_1 - d_1$
 $= f_0 + (2^{n+1} - 1)d_1 \stackrel{\text{Def. } g_n}{=} g_{n+1} \quad \text{q.e.d}$

Eine übersichtlichere und ausführlichere Musterlösung existiert und kann unter noch bekannt-zugebender Webadresse heruntergeladen werden.

Aufgabe 7: Rekurrenzen (6 + 4 Punkte)

- (a) Gegeben sei folgende Rekurrenz: $f_0 = 0, f_n = f_{n-1} + (n - 1)$. Finden Sie für f_n eine geschlossene Form g_n und beweisen sie Ihr Ergebnis mit Hilfe vollständiger Induktion.
- (b) Schreiben Sie eine Haskell-Funktion, bei der sich die Anzahl der Additionen oder Multiplikationen nach der oben genannten Rekurrenz berechnet. Geben Sie auch an, was Ihre Funktion berechnet.

Lösung:

- (a) Es ist leicht ersichtlich, dass die geschlossene Form ein Polynom 2. Grades sein muss:

n	0	1	2	3	4
f_n	0	0	1	3	6
diff		0	1	2	3
diff ₂			1	1	1

Mit dem Ansatz $g(n) = an^2 + bn + c$ und Einsetzen der ersten 3 Glieder kommt man schnell auf:
 $g(n) = \frac{1}{2}n^2 - \frac{1}{2}n$.

Alternativ: In jedem Schritt wird $n - 1$ addiert, damit ergibt sich für $g_n: g_n = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = g(n) = \frac{1}{2}n^2 - \frac{1}{2}n$.

Induktion:

- I.A.: $n = 0: g_0 = 0 = f_0$.
 - I.V.: Für bel., festes n gelte: $g_n = \frac{1}{2}n^2 - \frac{1}{2}n = f_n$.
 - I.S.: $n \hookrightarrow n + 1 \quad f_{n+1} = f_n + n \stackrel{\text{I.V.}}{=} \frac{1}{2}n^2 - \frac{1}{2}n + n = \frac{1}{2}(n^2 + n) = \frac{1}{2}(n^2 + n + n - n + 1 - 1) = \frac{1}{2}((n + 1)^2 + (n + 1)) = \frac{1}{2}(n + 1)^2 + \frac{1}{2}(n + 1) = g_{n+1}$.
- (b) z.B.: Berechnet zu einem übergebenen n die Liste der Summen $\sum_{i=0}^k i$ für alle $k, k \in [0 \dots n]$ und damit die Liste der Werte von f_1 bis f_{n+1} .

```
f 0 = [0]
f n = (f (n-1)) ++ [sum [1..n]]
```

Die Anzahl der Additionen ist die in der Summe von 1 bis n , also $n - 1$, plus diejenigen im Rekursionsaufruf, also die Anzahl der Additionen für $n - 1$.

Aufgabe 8: Quicksort (1 + 2 Punkte)

Quicksort gilt als schnelles Sortierverfahren. Allerdings kann es in manchen Situationen nicht diesen Ansprüchen entsprechen. Die Wahl des *Pivotelements* ist für die Laufzeit des Algorithmus entscheidend.

In der Implementierung in dieser Aufgabe erhält die Sortierfunktion eine Liste, die mit Quicksort sortiert werden soll. Dabei wird als Pivotelement das erste Element der Liste gewählt.

- (a) Geben Sie eine Liste von 8 Elementen an, die Quicksort mit maximalen Aufwand sortiert (worst case).
- (b) Führen anhand ihrer Liste Quicksort durch, wobei als Pivotelement immer das erste Element in der Liste gewählt werden soll.
Mit welchem Aufwand wird die Liste sortiert (im O-Kalkül)? Es muss nicht jeder Schritt angegeben werden, aber Teilergebnisse müssen erkennbar sein.

Lösung:

- (a) Quicksort ist ein Teile- und Herrsche Algorithmus, wobei die einzelnen Elemente einer Liste ihrer Größe nach aufgeteilt werden. Wo die Grenze der Teilung ist, bestimmt das Pivotelement. Liegt dieses Element ziemlich in der mathematischen Mitte der Elemente, so wird die zu sortierende Liste in zwei relativ gleich große Hälften aufgeteilt.

Wird das Pivotelement ungünstig gewählt, dann erfolgt eine Aufteilung in zwei extrem unbalancierte Teilmengen, im schlimmsten Fall landen sogar alle Elemente in einer Teilmenge, die andere bleibt leer. Dies ist der Fall, wenn das Supremum oder Infimum der Menge gewählt wird.

Da in dieser Implementierung die Position des Pivotelements immer die erste Stelle in der Liste ist, sollte dort das Infimum oder Supremum der verbleibende Menge stehen.

Beispiele wären demnach:

[1, 2, 3, 4, 5, 6, 7, 8],
[8, 7, 6, 5, 4, 3, 2, 1] oder
[1, 8, 2, 7, 3, 6, 4, 5]

Es kann sehr viele Möglichkeiten geben. Wäre auch ne gute Aufgabe, die Anzahl der Möglichkeiten auszurechnen. Sollte eventuell 2^{n-1} sein, lege mich aber darauf nicht fest!

- (b) Durchführung:

[1, 8, 2, 7, 3, 6, 4, 5]
1 [8, 2, 7, 3, 6, 4, 5]
1 [2, 7, 3, 6, 4, 5] 8
1, 2 [7, 3, 6, 4, 5] 8
1, 2 [3, 6, 4, 5] 7, 8
1, 2, 3 [6, 4, 5] 7, 8
1, 2, 3 [4, 5] 6, 7, 8
1, 2, 3, 4 [5] 6, 7, 8

Da pro Schritt nur ein Element an seinen Platz kommt, sind $n - 1$ Durchläufe notwendig. Pro Durchlauf muss, angefangen bei n , immer ein Element weniger durchlaufen werden, woraus folgt:

$$\sum_{i=1}^n i \in O(n^2).$$