Datenstrukturen und Algorithmen: Übungsblatt #2

Abgabe am 26. April 2018

Finn Hess (378104), Jan Knichel (377779), Paul Orschau (381085)

22.04.2018

 Finn Hess (378104), Jan Knichel (377779), Paul Orschau (381085)

Aufgabe 1

Wir definieren die Quasiordnung $\sqsubseteq$ auf Funktionen als
$f \sqsubseteq g \Leftrightarrow f \in \mathcal{O}(g)$.

a) Beweisen Sie, dass $\sqsubseteq$ eine Quasiordnung ist, das heißt dass $\sqsubseteq$ reflexiv und transitiv ist.

b) Sortieren Sie einige Funktionen in aufsteigender Reihenfolge bezüglich der Quasiordnung $\sqsubseteq$.

Lösung

Teil a)

$\sqsubseteq$ ist reflexiv, denn $\forall f$ gilt:
$f \sqsubseteq f \Leftrightarrow f \in \mathcal{O}(f) \Leftrightarrow \limsup_{n\to\infty} \frac{f(n)}{f(n)} = 1 < \infty$ □

$\sqsubseteq$ ist transitiv, denn $\forall f, g, h$ mit $f \sqsubseteq g$ und $g \sqsubseteq h$ gilt:
$f \sqsubseteq h \Leftrightarrow \limsup_{n\to\infty} \frac{f(n)}{h(n)} = \limsup_{n\to\infty} [\frac{f(n)}{g(n)} * \frac{g(n)}{h(n)}] \leq \limsup_{n\to\infty} \frac{f(n)}{g(n)} * \limsup_{n\to\infty} \frac{g(n)}{h(n)} \overset{f\sqsubseteq g \atop g\sqsubseteq h}{=} c_1 * c_2 < \infty$ □

Teil b)

$$0 \sqsubseteq 4 \sqsubseteq 2^{9000} \sqsubseteq \log n \overset{1)}{\sqsubseteq} n \log n \overset{2)}{\sqsubseteq} n\sqrt{n} \overset{3)}{\sqsubseteq} n^2 \overset{4)}{\sqsubseteq} \sum_{i=0}^{n} \frac{14i^2}{1+i} \sqsubseteq n^2 \log n \sqsubseteq n^3 \sqsubseteq \frac{n^3}{2} \sqsubseteq 2^n \sqsubseteq n! \sqsubseteq n^n$$

1. $\limsup_{n\to\infty} \frac{\log n}{n \log n} \overset{\text{l'Hopital}}{=} \limsup_{n\to\infty} \frac{\frac{1}{n}}{n*\frac{1}{n}+1*\log n} = \limsup_{n\to\infty} \frac{1}{n+n*\log n} = 0 < \infty$ □
2. $\limsup_{n\to\infty} \frac{n \log n}{n\sqrt{n}} = \limsup_{n\to\infty} \frac{\log n}{\sqrt{n}} \overset{\text{l'Hopital}}{=} \limsup_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \limsup_{n\to\infty} \frac{2\sqrt{n}}{n} = \limsup_{n\to\infty} \frac{2}{\sqrt{n}} = 0 < \infty$ □
3. $\limsup_{n\to\infty} \frac{n\sqrt{n}}{n^2} = \limsup_{n\to\infty} \frac{\sqrt{n}}{n} = \limsup_{n\to\infty} \frac{1}{\sqrt{n}} = 0 < \infty$ □
4. $\limsup_{n\to\infty} \frac{n^2}{\sum_{i=0}^{n} \frac{14i^2}{1+i}} = \limsup_{n\to\infty} \frac{n^2}{14 \sum_{i=0}^{n} [i-1+\frac{1}{i+1}]} = \frac{1}{14} * \limsup_{n\to\infty} \frac{n^2}{\frac{n(n+1)}{2}-n+\sum_{i=0}^{n} \frac{1}{i+1}} = \frac{1}{14} * \limsup_{n\to\infty} \frac{n^2}{\frac{n^2-n}{2}+\sum_{i=0}^{n} \frac{1}{i+1}}$
   $\leq \frac{1}{7} * \limsup_{n\to\infty} \frac{n^2}{n^2-n} = \frac{1}{7} * \limsup_{n\to\infty} \frac{1}{1-\frac{1}{n}} = \frac{1}{7} < \infty$ □

Aufgabe 2

Beweisen oder wiederlegen Sie folgende Aussagen:

a) $\frac{1}{4}n^3 - 7n + 17 \in \mathcal{O}(n^3)$

b) $n^4 \in \mathcal{O}(2n^4 + 3n^2 + 42)$

c) $\log(n) \in \mathcal{O}(n)$

d) $\forall \epsilon > 0 : \log(n) \in \mathcal{O}(n^\epsilon)$

e) $a^n \in \Theta(b^n)$ für zwei beliebige Konstanten $a, b > 1$

Lösung

Teil a)

Aussage wahr, da

$\limsup_{n\to\infty} \frac{\frac{1}{4}n^3-7n+17}{n^3} = \limsup_{n\to\infty}[\frac{1}{4} - \frac{7}{n^2} + \frac{17}{n^3}] = \frac{1}{4} < \infty$ □

Teil b)

Aussage wahr, da

$\limsup_{n\to\infty} \frac{n^4}{2n^4+3n^2+42} = \limsup_{n\to\infty} \frac{1}{2+\frac{3}{n^2}+\frac{42}{n^4}} = \frac{1}{2} < \infty$ □

Teil c)

Aussage wahr, da

$\limsup_{n\to\infty} \frac{\log n}{n} \overset{\text{l'Hopital}}{=} \limsup_{n\to\infty} \frac{\frac{1}{n}}{1} = \limsup_{n\to\infty} \frac{1}{n} = 0 < \infty$ □

Teil d)

Aussage wahr, da

$\limsup_{n\to\infty} \frac{\log n}{n^\epsilon} \overset{\text{l'Hopital}}{=} \limsup_{n\to\infty} \frac{\frac{1}{n}}{\epsilon n^{\epsilon-1}} = \limsup_{n\to\infty} \frac{1}{\epsilon n^\epsilon} \overset{(*)}{=} 0 < \infty$ □

$(*)$ Wir zeigen Konvergenz gegen 0:

$\forall \epsilon > 0 : \exists n_0(\epsilon) \in \mathbb{N} : |\frac{1}{an^a}| < \epsilon \Leftrightarrow ... \Leftrightarrow \sqrt[a]{\frac{1}{a\epsilon}} < n$

Teil e)

Aussage falsch, seien z.B. $a = e^3$ und $b = e^2$, dann gilt

$\lim_{n\to\infty} \frac{a^n}{b^n} = \lim_{n\to\infty} \frac{e^{3n}}{e^{2n}} = \lim_{n\to\infty} e^{3n-2n} = \lim_{n\to\infty} e^n = \infty$ □

Aufgabe 3

a) Zeigen oder wiederlegen Sie:

$$o(g(n)) \cap \Theta(g(n)) = \emptyset$$

b) Zeigen oder widerlegen Sie:

$$f(n) \in \Omega(g(n)) \wedge f(n) \in \mathcal{O}(h(n)) \implies g \in \Theta(h(n))$$

Lösung

Teil a)

Sei $f(n) \in o(g(n))$ beliebig. Das heißt es gilt $\forall c > 0, \exists n_0$ mit $\forall n \geq n_0 : 0 \leq f(n) < c * g(n)$.
Nun zeigen wir, dass $f(n)$ nicht in $\Theta(g(n))$ liegen kann.
$f(n) \in \Theta(g(n)) \Leftrightarrow \exists c_1, c_2 > 0, n_0$ mit $\forall n \geq n_0 : c_1 * g(n) \leq f(n) \leq c_2 * g(n)$.
Da $f \in o(g(n))$ gibt es auch zu jedem c_1 ein n_0, so dass $f(n) < c_1 * g(n)$.
Daher kann kein c_1 existieren, so dass $c_1 * g(n) \leq f(n)$ ab einem Index n_0 gilt. Somit kann dieses f nicht in $\Theta(g(n))$ liegen. Da f beliebig war, ist die Aussage ist gezeigt. □

Teil b)

Gegenbeispiel: Sei $f(n) = n$, $g(n) = 1$ und $h(n) = n^2$. Dann gilt zwar $f(n) \in \Omega(g(n)) \wedge f(n) \in \mathcal{O}(h(n))$, allerdings nicht $g \in \Theta(h(n))$. □

Aufgabe 4

Gegeben sei ein Algorithmus, der für ein Array von Booleans überprüft, ob alle Einträge wahr sind:

```
1  int allTrue(bool [] E) {
2    if (E.length < 1) {
3      return -1;
4    }
5    int m = E.length;
6    int i = 0;
7    while (i<E.length) {
8      if (E[i] == true) {
9        m = m - 1;
10       if (m == 0) {
11         return 1;
12       }
13     }
14     i = i + 1;
15   }
16   return 0;
17 }
```

Bei Betrachtung der Laufzeit wird angenommen, dass Vergleiche (z.B. $x < y$ oder $b == 0$) jeweils eine Zeiteinheit benötigen. Die Laufzeit aller anderen Operationen wird vernachlässigt.
Sei n die Länge des Arrays E.

a) Bestimmen Sie in Abhängigkeit von n die Best-case Laufzeit $B(n)$.

b) Bestimmen Sie in Abhängigkeit von n die Worst-case Laufzeit $W(n)$.

c) Bestimmen Sie in Abhängigkeit von n die Average-case Laufzeit $A(n)$. Hierzu nehmen wir eine uniforme Verteilung der möglichen Eingaben D_n an, d.h. jede beliebige Eingabe $E \in D_n$ tritt mit Wahrscheinlichkeit $1/|D_n|$ auf.

d) Geben Sie einen äquivalenten Algorithmus an, dessen Average-case Laufzeit ab einer gewissen Eingabelänge kleiner ist. Hierzu nehmen wir wieder eine uniforme Verteilung der möglichen Eingaben an. Begründen Sie Ihre Antwort kurz. Sie müssen nicht die Average-case Laufzeit ihres Algorithmus berechnen.

e) Gibt es einen äquivaltenen Algorithmus, dessen Worst-case Laufzeit in $o(n)$ liegt? Begründen Sie ihre Antwort.

Lösung

Zeilen im Code, welche Zeiteinheiten benötigen: **2, 7, 8, 10**.
Falls das Array **leer** ist ($n = 0$), hat der Algorithmus immer eine Laufzeit von 1 (nur Zeile 2).
Das heißt $B(0) = W(0) = A(0) = 1$.
Falls das Array **höchstens n-1** "true"-Werte enthält, ist die Laufzeit stets
$X(n,a) = \underbrace{1}_{2} + \underbrace{(n+1)}_{7} + \underbrace{n}_{8} + \underbrace{a}_{10} = 2n + 2 + a$
wobei $a \in [0, n]$ der Anzahl von "true"-Werten im Array entspricht. Das liegt daran, dass der Algorithmus den ganzen Array abläuft ... statt bei dem ersten gefundenen "false"-Wert abzubrechen. Für jeden gefundenen "true"-Wert kommt ein weiterer Vergleich in Zeile 10 hinzu.

Aufgabe 4 wird auf nächster Seite fortgesetzt...

Falls das Array **genau n** "true"-Werte enthält, wird Zeile 7 einmal weniger ausgeführt, da der Algorithmus bereits in Zeile 11 terminiert. Die Laufzeit entspricht dann
$X(n,n) = \underbrace{1}_{2} + \underbrace{n}_{7} + \underbrace{n}_{8} + \underbrace{n}_{10} = 3n+1$

Teil a)

Best-case Laufzeit: $B(n) = X(n,0) = 2n+2$

Teil b)

Worst-case Laufzeit: $W(n) = X(n,n) = X(n,n-1) = 3n+1$

Teil c)

Average-case Laufzeit: $A(n) = \sum_{I \in D_n} Pr(I) * t(I) = \sum_{I \in D_n} \frac{1}{|D_n|} * t(I) = \frac{1}{2^n} \sum_{I \in D_n} t(I) = \frac{1}{2^n} \sum_{i=0}^{n} \binom{n}{i} X(n,i) =$
$\frac{1}{2^n}[\sum_{i=0}^{n-1} \binom{n}{i} X(n,i) + X(n,n)] = \frac{1}{2^n}[\sum_{i=0}^{n-1} \binom{n}{i}(2n+2+i) + (3n+1)]$

Teil d)

```
1  int allTrue(bool [] E) {
2    if (E.length < 1) {
3      return -1;
4    }
5    int m = E.length;
6    int i = 0;
7    while (i<E.length) {
8      if (E[i] == false) {
9        return 0;
10     }
11     i = i + 1;
12   }
13   return 1;
14 }
```

Teil e)

Einen äquivalenten Algorithmus, dessen Worst-case Laufzeit in $o(n)$ liegt, kann es nicht geben, denn im schlimmsten Fall muss der Algorithmus alle Werte des Arrays mindestens einmal überprüfen. Der Algorithmus kann keine "Annahmen" über einzelne Werte machen, bzw. man kann nicht auf den Wert eines Eintrag des Arrays schließen, ohne ihn wirklich betrachtet zu haben. Da das Array n Werte hat, muss die Worst-case Laufzeit des besten Algorithmus in $\mathcal{O}(n)$ liegen!