Algorithmen und Datenstrukturen
Tutorium
Übungsaufgaben

Michael R. Jung

Aufgabe 1

Ordnen Sie die folgenden Funktionen bezüglich ihrer asymptotischen Laufzeit!

$$n^2, \log(4^n), \log\left(2^{2^n}\right), 2^{n+1}, 4^n$$

Eine richtige Reihenfolge ist:

$$\log(4^n) = n \log 4 = 2n, n^2, \log\left(2^{2^n}\right) = 2^n, 2^{n+1}, 4^n.$$

Beweis:

1. $$\lim_{n\to\infty} \frac{2n}{n^2} = \lim_{n\to\infty} \frac{2}{n}$$
$$= 0 \qquad \left[\Rightarrow 2n \in o\left(n^2\right)\right]$$

2. $$\lim_{n\to\infty} \frac{n^2}{2^n} \overset{\text{l'Hôp}}{=} \lim_{n\to\infty} \frac{2n}{2^n \ln 2}$$
$$\overset{\text{l'Hôp}}{=} \lim_{n\to\infty} \frac{2}{2^n (\ln 2)^2}$$
$$= 0 \qquad \left[\Rightarrow n^2 \in o\left(2^n\right)\right]$$

3. $$\lim_{n\to\infty} \frac{2^n}{2^{n+1}} = \lim_{n\to\infty} \frac{1}{2}$$
$$= \frac{1}{2} \qquad \left[\Rightarrow 2^n \in \Theta\left(2^{n+1}\right)\right]$$

4. $$\lim_{n\to\infty} \frac{2^{n+1}}{4^n} = \lim_{n\to\infty} \frac{2 \cdot 2^n}{2^{2n}}$$
$$= \lim_{n\to\infty} \frac{2 \cdot 2^n}{2^n \cdot 2^n}$$
$$= \lim_{n\to\infty} \frac{2}{2^n}$$
$$= 0 \qquad \left[\Rightarrow 2^{n+1} \in o\left(4^n\right)\right]$$

Aufgabe 2

Lösen Sie die folgenden Rekurrenzen!

1. $T(n) = 8T\left(\frac{n}{4}\right) + 2n,$
   $T(1) = 5$
2. $T(n) = 16T\left(\frac{n}{4}\right) + n^2,$
   $T(4) = 3$

1)
$$
\begin{aligned}
T(n) &= 8 \cdot T\left(\frac{n}{4}\right) + 2n && (i = 1)\\
&= 8\left[8T\left(\frac{n}{16}\right) + \frac{n}{2}\right] + 2n\\
&= 64 \cdot T\left(\frac{n}{16}\right) + 6n && (i = 2)\\
&= 64\left[8T\left(\frac{n}{64}\right) + \frac{n}{8}\right] + 6n\\
&= 512 \cdot T\left(\frac{n}{64}\right) + 14n && (i = 3)\\
&= 512\left[8T\left(\frac{n}{256}\right) + \frac{n}{32}\right] + 14n\\
&= 4096 \cdot T\left(\frac{n}{64}\right) + 30n && (i = 4)\\
&= 8^i \cdot T\left(\frac{n}{4^i}\right) + 2(2^i - 1)n && (\text{Vermutung})
\end{aligned}
$$

Rekursionstiefe: $\frac{n}{4^i} = 1 \Leftrightarrow i = \log_4 n \Leftrightarrow i = \frac{1}{2}\log n$.

Es folgt:

$$\begin{aligned} T(n) &= 2^{3\cdot\frac{1}{2}\log n} \cdot 5 + 2(2^{\frac{1}{2}\log n} - 1)n \\ &= 5n^{\frac{3}{2}} + 2n^{\frac{3}{2}} - 2n \\ &= 7n^{\frac{3}{2}} - 2n \end{aligned}$$

Probe:

$$T(1) = 7 \cdot 1 - 2 \cdot 1 = 5 \checkmark$$

$$\begin{aligned} T(n) &= 8T\left(\frac{n}{4}\right) + 2n \\ &= 8\left[7\left(\frac{n}{4}\right)^{\frac{3}{2}} - 2\frac{n}{4}\right] + 2n \\ &= 8\left[\frac{7}{8}n^{\frac{3}{2}} - \frac{n}{2}\right] + 2n \\ &= 7n^{\frac{3}{2}} - 4n + 2n \\ &= 7n^{\frac{3}{2}} - 2n \checkmark \end{aligned}$$

2) $T(n) = 16\,T\left(\frac{n}{4}\right) + n^2 \qquad (i = 1)$

$= 16\left[16\,T\left(\frac{n}{16}\right) + \left(\frac{n}{4}\right)^2\right] + n^2$

$= 256\,T\left(\frac{n}{16}\right) + 2n^2 \qquad (i = 2)$

$= 256\left[16\,T\left(\frac{n}{64}\right) + \left(\frac{n}{16}\right)^2\right] + 2n^2$

$= 4.096\,T\left(\frac{n}{64}\right) + 3n^2 \qquad (i = 3)$

$= 4.096\left[16\,T\left(\frac{n}{256}\right) + \left(\frac{n}{64}\right)^2\right] + 3n^2$

$= 65.536\,T\left(\frac{n}{256}\right) + 4n^2 \qquad (i = 4)$

$= 16^i\,T\left(\frac{n}{4^i}\right) + in^2 \qquad \text{(Vermutung)}$

Rekursionstiefe: $\frac{n}{4^i} = 4 \Leftrightarrow i + 1 = \log_4 n \Leftrightarrow i = -1 + \frac{1}{2}\log n.$

Es folgt:
$$\begin{aligned} T(n) &= 16^{-1+\frac{1}{2}\log n} T(4) + \left(-1 + \frac{1}{2}\log n\right) n^2 \\ &= 2^{-4+2\log n} \cdot 3 - n^2 + \frac{1}{2} n^2 \log n \\ &= \frac{1}{2} n^2 \log n - \frac{13}{16} n^2 \end{aligned}$$

Probe:
$$\begin{aligned} T(4) &= \frac{1}{2} \cdot 16 \cdot 2 - \frac{13}{16} \cdot 16 = 16 - 13 = 3 \checkmark \\ T(n) &= 16 \left[\frac{1}{2}\left(\frac{n}{4}\right)^2 \log \frac{n}{4} - \frac{13}{16}\left(\frac{n}{4}\right)^2\right] + n^2 \\ &= 8 \cdot \frac{n^2}{16}(-2 + \log n) - 13 \cdot \frac{n^2}{16} + n^2 \\ &= \frac{n^2}{2} \log n - n^2 - \frac{13}{16} n^2 + n^2 \\ &= \frac{1}{2} n^2 \log n - \frac{13}{16} n^2 \checkmark \end{aligned}$$

Aufgabe 3

Entwerfen Sie Algorithmen, die für einen gegebenen Graphen $G = (V, E)$ testen, ob G

1. zweifärbbar ist bzw.
2. Kreise enthält!

Zeigen Sie jeweils die Korrektheit Ihres Algorithmus und analysieren Sie dessen Laufzeit!

Lösung 1:

Der Algorithmus ist hier eher trivial; man beginnt mit einem beliebigen Knoten, färbt diesen, und versucht diese Färbung in seiner Zusammenhangskomponente fortzusetzen. Sollte dies gelingen, so fährt man mit der nächsten Komponente fort, bis alle Knoten gefärbt sind. Gelingt dies zu irgendeinem Zeitpunkt nicht so, kann es keine 2-Färbung des Graphen geben und man gibt die zutreffende Antwort zurück.

Ein Pseudocode für diesen Algorithmus folgt.

Färbung(G)

```
1  Input: Graph G=(V,E)
2  Output: Graph zweifärbbar?
3  ungefärbt=V;
4  while ungefärbt≠∅ do
5    wähle v∈ungefärbt;
6    if not färbe(v,schwarz)
7      then return false;
8    endif
9  endwhile
10 return true;
```

färbe(v,f)

```
1  Input: Knoten v, Farbe f
2  Output: Zusammenhangskomponente von v zweifärbbar?
3  farbe(v)=f;
4  ungefärbt=ungefärbt\{v}
5  if f=schwarz then f=gelb else f=schwarz endif
6  foreach w∈N(v) do
7    if w∉ungefärbt
8      then if farbe(w)≠ f then return false;
9           endif
10     else
11       if not färbe(w,f)
12         then return false;
13       endif
14   endif
15 endfor
16 return true;
```

Laufzeitanalyse

Zur Laufzeit ist nur wenig zu sagen. Unter der Annahme, dass das Entfernen eines Knotens aus einer Menge in $\mathcal{O}(1)$ möglich ist, wird die Laufzeit nur durch die Schleife in `Färbung(G)` und die Schleife und die rekursiven Aufrufe innerhalb von `färbe(v,f)` bestimmt. Die äußere Schleife in `Färbung(G)` hat offensichtlich Laufzeit $\mathcal{O}(n)$, in ihr wird `färbe(v,f)` aufgerufen. Hier ist zu beobachten, dass `färbe(v,f)` höchstens einmal pro Knoten aufgerufen wird. Des weiteren gilt zusätzlich, dass die innere Schleife höchstens zweimal pro Kante aufgerufen wird.
Insgesamt hat also der Algorithmus eine Laufzeit von $\mathcal{O}(|V| + |E|)$.

Korrektheitsbeweis

Ich zeige die folgenden zwei Implikationen:

1. Algorithmus gibt `false` zurück $\Rightarrow$ G nicht zweifärbbar,
2. G nicht zweifärbbar $\Rightarrow$ Algorithmus gibt `false` zurück.

Beweis:

(1) Hier beobachten wir zunächst, wann der Algorithmus `false` zurückgibt. Es ist gut zu erkennen, dass dies genau dann geschieht, falls zu einem Zeitpunkt `färbe(v,f)` `false` zurückgibt. In letzter Konsequenz muss es dazu einen Knoten v geben, der o.B.d.A. mit `färbe(v,schwarz)` aufgerufen wird, aber schon mit `weiß` gefärbt ist (oder umgekehrt). Dies geschieht, wenn v einen `weiß` gefärbten Nachbarn w hat. Da offensichtlich bereits einmal `färbe(v,weiß)` aufgerufen wurde aber `färbe(w,schwarz)` noch nicht, muss es einen, in der bisherigen Teilfärbung alternierenden, Pfad von v zu w geben, der Algorithmus geht ja wie eine Tiefensuche vor. Auf diesem wird die Kante $\{v, w\}$ nicht benutzt. Da beide Knoten die gleiche Farbe haben, hat dieser Pfad gerade Länge.
Nehmen wir nun die Kante von v zu w hinzu, so ergibt sich insgesamt ein Kreis ungerader Länge. Folglich ist G nicht zweifärbbar.

(2) Da G nicht zweifärbbar ist, enthält er einen Kreis ungerader Länge. Sei $(v_1, \ldots, v_k, v_1)$ eins solcher, d.h. k ist ungerade. Seien o.B.d.A. $v_1, \ldots, v_{k-1}$ die ersten Knoten auf diesem Kreis, für die `färbe(v,f)` aufgerufen wird. (Sollte dies nicht bei allen diesen Knoten geschehen, so hat der Algorithmus schon abgebrochen, bevor alle diese Knoten auf diesem Kreis gefärbt werden. Dies kann allerdings nur geschen, falls der Algorithmus `false` ausgibt. In diesem Fall sind wir fertig.)

zu (2) Hier gibt es nun zwei Fälle:

1. Der Pfad $v_1, v_2, \ldots, v_{k-1}$ ist alternierend gefärbt. Folglich haben v_1 und v_{k-1} verschiedene Farben. Somit wird im weiteren Verlauf (falls der Algorithmus nicht schon vorher abbricht (s.o.)) sowohl `färbe(`v_k`,weiß)` als auch `färbe(`v_k`,schwarz)` aufgerufen. Beim zweiten Aufruf wird also `false` ausgegeben.
2. Es gibt mindestens ein Paar benachbarter Knoten v_i, v_{i+1} auf dem Kreis, die dieselbe Farbe haben. Seien sie o.B.d.A. mit `schwarz` gefärbt. Je nachdem welcher Knoten zuletzt gefärbt wurde, muss im weiteren Verlauf (sollte nicht schon vorher abgebrochen werden (s.o.)) entweder `färbe(`v_i`,weiß)` bzw. `färbe(`v_{i+1}`,weiß)` aufgerufen werden. Beide Aufrufe führen aber dazu, dass der Algorithmus `false` zurückgibt. □

Lösung 2:

Einen Kreis in einem Graphen zu finden ist mit Breiten- oder Tiefensuche möglich. Dazu wird der Graph normal traversiert, sollte man zu einem Zeitpunkt auf einen Nachbarn treffen, der nicht der direkte Vorgänger auf dem Breiten- bzw. Tiefensuchbaum ist, aber schon besucht wurde, so hat man einen Kreis gefunden.

Die Laufzeit ist mit einer Breiten- bzw. Tiefensuche identisch: $\mathcal{O}(n+m)$.

Ein Korrektheitsbeweis wäre analog zum ersten, dort wurde ja nach Kreisen ungerader Länge gesucht. Daher verzichte ich hier darauf.