TCS Self-Deceit Sheet

Von wüsten Behauptungen und kreativen Konventionen

Zusammengetragen von Übungsleitern und Assistenten
der Vorlesung EAA aus den Jahren 2011 – 2013

Ausgabe vom 28.10.2013

Aus Fehlern soll man bekanntlich lernen – es müssen aber nicht immer die eigenen sein! Unten stehen die kreativsten „Lösungen“, die uns über die Jahre in Klausuren untergekommen sind. Wir laden herzlich ein, über die einzelnen Fälle nachzudenken, die Trugschlüsse im Detail festzunageln und ähnliche Fehler vermeiden zu lernen.

Grundverrechenarten

- $\frac{(2n)!}{(n!)^2} = \frac{2}{n!}$
- $(n!)^2 = (n^2)! = 2(n!)$
- $b_n = \binom{n+1}{n} + b_{n-1} \implies b_{n+1} = \binom{n}{n+1} + b_n$
- $\frac{x \cdot y \cdot z}{y} = \frac{x}{y} \cdot \frac{y}{y} \cdot \frac{z}{y}$
- $\frac{x}{y} + \frac{y}{y} = \frac{xy}{y}$
- $\frac{n+1}{n} = \frac{1 + \frac{1}{n}}{n}$
- $\frac{n+1}{n} = n \cdot \frac{1 + \frac{1}{n}}{1}$
- $\frac{e^{n+1}}{e^n} = 1$
- $\frac{c^{n+1}}{c^n} = \frac{1}{c}$
- $\frac{n^\alpha}{n^{\alpha+\varepsilon}} = \frac{n^\alpha}{n^\alpha n} = \frac{1}{n}$

Fortgeschrittene Verrechenregeln

- $a^{\frac{1}{n}} = a^{-n}$
- $a^{\frac{1}{n-1}} = \frac{a^{\frac{1}{n}}}{a}$
- $a^{\frac{1}{n}} - a^{\frac{1}{n-1}} = a^{\frac{1}{n} - \frac{1}{n-1}}$
- $\frac{e^n}{e^{n \cdot f(n)}} = \frac{1}{e^{f(n)}}$
- $\prod_{i=1}^{n} 4i = 4 \cdot \prod_{i=1}^{n} i$

Grenzüberschreitungen

- $\lim_{n\to\infty} (n^2 - n^2) = 1$
- $\lim_{n\to\infty} a^{\frac{1}{n}} = 0$
- $\lim_{n\to\infty} n^2 = 0$
- $\lim_{n\to\infty} n + 1 = n + 1 < \infty$
- $\lim_{n\to\infty} c = \infty$
- $\lim_{n\to\infty} \frac{c^n}{c^n} = \infty$
- Für $c > 1$ gilt stets

  $$\lim_{n\to\infty} \frac{c^{n+1}}{c^n} = \infty = c\ ;$$

  es ist bekannt, dass höhere Potenz echt schneller wächst.
- $\lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n = \lim_{n\to\infty} \left(\lim_{n\to\infty} 1 + \frac{1}{n}\right)^n = \lim_{n\to\infty} 1^n = 1$
- $\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i} = 2$
- $\lim_{n\to\infty} e^n > \lim_{n\to\infty} 2^n$
- $\left[\forall\, i \in \mathbb{N}.\, a_i < b_i\right] \implies \frac{\prod_{i=1}^{n} a_i}{\prod_{i=1}^{n} b_i} \xrightarrow{n\to\infty} 0$
- $[\forall^{\infty}\, n \in \mathbb{N}.\, f(n) > g(n)] \implies \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$
- $\lim_{n\to\infty} \frac{n^n}{n^n + n} = \lim_{n\to\infty} \frac{1}{n}$

  Aus „Fachdidaktik Mathematik“: Diesen Fehler machen Schüler nur selten.
- $\lim_{n\to\infty} \frac{n^n}{n(n+1)^{n-1}} = \lim_{n\to\infty} \frac{n^n}{nn^{n-1}} = 1$

Undifferenziertes Differenzieren

- $\frac{\mathrm{d}}{\mathrm{d}n}\, n! = (n-1)!$
- $\frac{\mathrm{d}}{\mathrm{d}n}\, 4^n = n \cdot 4^{n-1}$
- $\frac{f'(n)}{g'(n)} = \left(\frac{f(n)}{g(n)}\right)'$
- $\frac{\mathrm{d}}{\mathrm{d}n}\, (\ln n)^n = n \cdot \left(\frac{1}{n}\right)^{n-1}$
- $\frac{\mathrm{d}}{\mathrm{d}n}\, n^n = n \cdot n^{n-1}$
- $\frac{\mathrm{d}}{\mathrm{d}n}\, 2^n = n \cdot \ln 2 \cdot 2^n$
- $\frac{\mathrm{d}^n}{\mathrm{d}n^n}\, n^n = n!$

Verschätzungen & Identitätskrisen

- $e \approx 1,7$
- $\frac{\ln n}{2} < 1$
- $\sum_{i=0}^{n} F_i = F_{n+1}$
- $F_n = F_{n-1} + F_{n-2} + 1$
- $\sum_{i=0}^{n} F_i^2 = \frac{F_n(F_n+1)(F_{2n}+1)}{6}$
- $n!$ ist Polynom.
- $-2 + B = 1 \implies B = -1$
- $\sum_{i=1}^{k} c_i w_i = \sum_{i=1}^{k} i = n$

Phantastische Asymptotiken

- $c^n \in \mathcal{O}(n)$ nach Definition.
- $c^{n+1} = \Theta(1)$
- $c^{n+1} = c^n \cdot c \in \Theta(c^n) \cdot c \in \Theta(c^{n+1}) \notin \Theta(c^n)$
- $F_n^2 = \Theta(F_n)$, ein solches $c > 0$ findet man immer.
- $\frac{2}{6} \ln^2(a) \sim \ln(a)$
- $f(n) = \Theta(g(n))$ genau dann, wenn ein $c > 0$ und $n_0 \geq 0$ existieren, so dass für alle $n > n_0$ $f(n) = c \cdot g(n)$ gilt.
- $\ln(n)$ wächst offensichtlich schneller als n.

- $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} \neq \infty \implies f \in o(g)$
- $n! \in o(2^n)$, denn Exponentialfunktionen schlagen alles tot.
- $n^n \in o(n!)$, denn $n!$ wächst schneller als alles andere.
- $\Theta(n \log n) \in o(nk)$.
  Anm.: n und k waren hier unabhängige Variablen.
- $(n+1)! \leq (n+2) \cdot n! \implies (n+1)! \in \mathcal{O}(n)$
- $\mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(\frac{n}{2}) \in \mathcal{O}(n^n)$, also mindestens exponentielles Wachstum.
- $2 \cdot \binom{n}{2} = o(n^2)$, weil
  $$\lim_{n\to\infty} \frac{2 \cdot \binom{n}{2}}{n^2} = \lim_{n\to\infty} \frac{\frac{n!}{(n-2)!}}{n^2} = \frac{1}{\infty} = 0 \,.$$
- Nach Definition gilt $(n+1)! \in \mathcal{O}(n!)$ genau dann, wenn
  $$\exists c > 0.\, \exists n_0 \in \mathbb{N}.\, \forall n \geq n_0.\, (n+1)! \leq c \cdot n!$$
  $$\iff (n+1) \cdot n! \leq c \cdot n! \,;$$
  mit beliebigem $c \geq n+1$ folgt also, dass $(n+1)! \in \mathcal{O}(n!)$.

Laufzeitverschätzungen

- Für $n \leq 3$: $\Theta(1)$
- Alles nur konstante Laufzeit, da keine Schleifen.
  Anm.: Angesichts eines rekursiven Algorithmus geäußert.
- Sei die [lineare] Liste zum Beispiel als Baum gegeben. Dann kann man sie in Zeit $\Theta(n \log n)$ traversieren und so eine normale Liste daraus machen.
  Anm.: Stimmt natürlich, der Ansatz wirft jedoch Fragen der Sinnhaftigkeit auf.
- Da Mergesort in $\Theta(n \log n)$ läuft, läuft `merge` in $\Theta(\log n)$.
- Der Algorithmus

  ```
  for i = 1 to n {
    i := i div a
  }
  ```

  hat Laufzeit $\Theta(\log n)$.
- ```
  a = x
  while ( a <= n ) {
    a = x div a
  }
  ```

  Anm.: Vermuteter Zweck: Zeit $\Theta(\log n)$ brauchen.
- Matrixmultiplikation geht in $O(n^2)$
  Anm.: Das ist nicht zwingend falsch, aber definitiv nicht bekannt.
- Falls Duplikate vorhanden sind, liegt die erwartete Laufzeit von Quicksort in $\mathcal{O}(3^n)$.
  Anm.: Auch das ist natürlich nicht *falsch*.
- $T_{\text{AC}} = \dfrac{T_{\text{BC}} + T_{\text{WC}}}{2}$
- Konstante Zeit; die Schleife wird $n-1$ mal durchlaufen.
- [...], da die erste Schleife (`if`) ...
  ```

Speicherzerwürfnisse

- [Speicherbedarf] $\omega(n)$ trifft nicht zu, da kein unendlicher Speicher benötigt wird.
- Es werden im Mittel Listen der Länge $\frac{n}{2}$ erzeugt. Diese benötigen also $\mathcal{O}(\log n)$ Platz.
- [Algorithmus braucht Speicher] $\mathcal{O}(n)$ und $\omega(n)$.
- Da Laufzeit in $\Theta(n^{\frac{3}{2}})$, muss Speicherbedarf in $\mathcal{O}(n)$ sein.
- Der Algorithmus hat drei rekursive Aufrufe, läuft also in Zeit $\mathcal{O}(3^n)$. Damit ist gezeigt, dass er exponentielle Zeit braucht.
- Quicksort sortiert mit $\mathcal{O}(1)$ Speicher, wenn man die rekursiven Aufrufe streicht.

Andere Analügen

- Quicksort tauscht nur, siehe also Analyse von Bubblesort.
- Bubblesort $\in \mathcal{O}(1)$.
- [In Mergesort wird] `merge` $\mathcal{O}(\log n)$ oft aufgerufen.
- [Bellman-Ford] ist im Prinzip eine etwas umständliche Umschreibung von Dijkstra.
- [Um die Rekursionsgleichung von Bellman-Ford zu berechnen,] eignet sich der Dijkstra-Algorithmus.
- [Um die Rekursionsgleichung von Bellman-Ford zu berechnen,] eignet sich der Kruskal-Algorithmus.
- [Der Shannon-Fano Algorithmus] ist stets optimal, da gleiches Verfahren wie in ReSy 1.
- Es reicht, die Summe bis m zu betrachten, da der angegebene Algorithmus

```
procedure e(n) {
  result = 1
  for i = 0 to n-1 {
    result = result + i + e(i)
  }
  return result
}
```

  auch nur eine Näherung von $e(n)$ berechnet.
- [Bei Shannon-Fano] wird der Präfixcode optimal, da immer entweder eine 0 oder eine 1 drangehangen wird.

Komplexitätshysterie

- Longest Path ist in $\mathcal{NP}$, da [...] exponentielle Laufzeit erforderlich ist.
  Anm.: Die Aussage stimmt, die Begründung nicht.

Epilog

Zu guter Letzt noch eine besondere Perle. Ausnahmsweise ist hier alles richtig, die Geradlinigkeit des Rechenwegs kann aber noch verbessert werden:

$$\frac{2^n}{e^n} = \frac{2^n}{\left(2 \cdot \frac{e}{2}\right)^n} = \frac{2^n}{2^n \cdot \left(\frac{e}{2}\right)^n} = \frac{1}{\left(\frac{e}{2}\right)^n} = \left(\frac{e}{2}\right)^{-n} = \left(\left(\frac{e}{2}\right)^{-1}\right)^n = \left(\frac{2}{e}\right)^n \xrightarrow{n\to\infty} 0, \text{ da } \frac{2}{e} < 1$$