Algorithmen und Berechenbarkeit: Vorlesung 03 - Mitschrift

Letztes Update: 2018/03/11 - 20:39 Uhr

Landau-Symbole und Laufzeiten

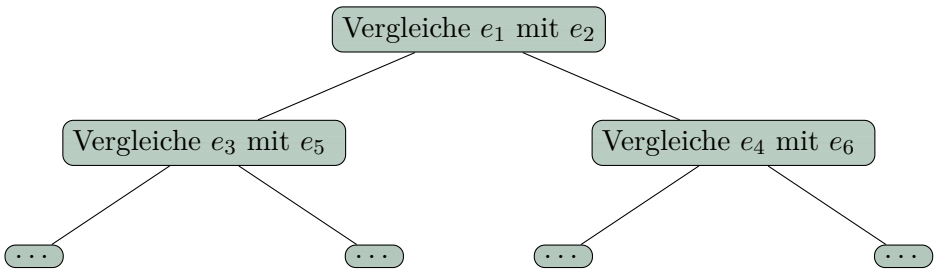
- $\mathcal{O}(n^2)$ Die Menge aller Funktionen, die (asymptotisch) nicht schneller wachsen als n^2. $(\leq)$
- $o(n^2)$ Die Menge aller Funktionen, die (asymptotisch) echt langsamer wachsen als n^2. $(<)$
- $\Omega(n^2)$ Die Menge aller Funktionen, die (asymptotisch) mindestens so schnell wachsen wie n^2. $(\geq)$
- $\omega(n^2)$ Die Menge aller Funktionen, die (asymptotisch) echt schneller wachsen als n^2. $(>)$
- $\Theta(n^2)$ Die Menge aller Funktionen, die (asymptotisch) genau so schnell wachsen wie n^2. $(=)$

Einschub: Vergleichsbasiertes Sortieren

Beim *vergleichsbasierten Sortieren* von n Objekten, wobei die Elemente nur verglichen werden dürfen, liefert jeder Algorithmus eine obere Schranke für $T(n)$ (Laufzeit für n Elemente). Zum Beispiel ist die obere Schranke von **Bubblesort** $\in \mathcal{O}(n^2)$ und die von **Mergesort** $\in \mathcal{O}(n \cdot \log(n))$.

Satz: *Es kann bewiesen werden, dass* $T(n) \in \Omega(n \cdot \log(n)) \Rightarrow T(n) \in \Theta(n \cdot \log(n))$.

Beweis: Sei A ein beliebiger Algorithmus zum Sortieren. A vergleicht e_i mit e_j. Die Vergleiche können exemplarisch als Binärbaum skizziert werden.



Trifft der Algorithmus auf ein Blatt des Binärbaums, so hat der Algorithmus fertig sortiert, bzw. der Algorithmus hat *herausgefunden*, was die Permutation der Eingabe war. Es lässt sich nun festhalten

1. Der Baum muss $n!$ Blätter haben.
2. Die Worst-Case-Laufzeit des Algorithmus A entspricht genau der Tiefe des Baums.

Nun lässt sich die minimale Tiefe eines Binärbaums, der $n!$ Blätter hat, berechnen. Sei

$$2^n \approx n! \approx \left(\frac{n}{e}\right)^{\frac{n}{e}}$$

Sei x die minimale Tiefe. Dann gilt

$$\begin{aligned} x &= \log_2 \left(\frac{n}{e}\right)^{\frac{n}{e}} \\ &= n \cdot \log(n) \end{aligned}$$

□

Randomisierte Algorithmen: Monte-Carlo und Las-Vegas

In Vorlesung 01 und 02 wurden zwei verschiedene Arten von randomisierten Algorithmen angeschnitten: Las-Vegas- bzw. Monte-Carlo-Algorithmen.

- **Las-Vegas:**
  - *Charakterisierung:* Die Laufzeit hängt vom Zufall ab, die Korrektheit nicht.
  - *Beispiel*: Randomisierter Closest-Pair-Algorithmus
  - *Umwandlung in Monte-Carlo-Algorithmus:* Die Umwandlung von Las-Vegas- in Monte-Carlo-Algorithmen ist möglich. Sei A_L ein Las-Vegas-Algorithmus. Man lässt A eine bestimmte Anzahl an Schritten laufen und bricht dann ab. War A bis dahin fertig, so muss das Ergebnis korrekt sein. War A bis dahin nicht fertig, dann gibt es auch kein korrektes Ergebnis.

    Sei $f(n)$ die erwartete Laufzeit von A. A darf nun für maximal $\alpha \cdot f(n)$ (mit $\alpha \geq 1$) Schritte laufen. Es gilt: A hat immer eine Laufzeit von $< \alpha \cdot f(n)$. Die Wahrscheinlichkeit, dass A ein falsches Resultat liefert, entspricht genau der Wahrscheinlichkeit, dass A länger als $\alpha \cdot f(n)$ Zeit benötigt.
- **Monte-Carlo:**
  - *Charakterisierung:* Die Laufzeit hängt nicht vom Zufall ab, die Korrektheit schon.
  - *Beispiel*: Kargers Min-Cut-Algorithmus
  - *Umwandlung in Las-Vegas-Algorithmus:* Die Umwandlung ist nicht für alle Monte-Carlo-Algorithmen können ohne Weiteres möglich. Für manche Probleme ist die Verifikation des Ergebnisses einfacher als die Berechnung:
    - Sortieren: $\mathcal{O}(n \cdot \log(n))$ vs. $O(n)$
    - Kürzeste Wege: $\mathcal{O}(n \cdot \log(n+m))$ vs. $O(m)$

    Das Überführen von Monte-Carlo- in Las-Vegas-Algorithmen ist möglich, wenn ein effizienter *Checker* existiert. Sei B ein Monte-Carlo-Algorithmus mit Laufzeit $f(n)$, sei C ein Checker mit Laufzeit $f(n)$, die Erfolgswahrscheinlichkeit von B sei $p(n)$.

Ein Las-Vegas-Algorithmus kann nun auf folgende Weise erzeugt werden:

```
lasse B laufen
if B.result ist korrekt then
    return B.result
else
    starte neu
end if
```

Für die erwartete Laufzeit R_B ergibt sich

$$\begin{aligned} R_B &= p(n) \cdot (f(n) + g(n)) \\ &\quad + ((1 - p(n)) \cdot p(n) \cdot (f(n) + g(n))) \cdot 2 \\ &\quad + ((1 - p(n))^2 \cdot p(n) \cdot (f(n) + g(n))) \cdot 3 + \ \ldots \\ &= (f(n) + g(n)) \cdot p(n) \cdot \sum_{i=1}^{\infty} (1 - p(n))^i \\ &< \frac{f(n) + g(n)}{p(n)} \cdot (i + 1) \end{aligned}$$

Anhang

Markov-Ungleichung

Sei X eine nicht negative Zufallsvariable mit $E[X] = \mu$. Dann gilt:

$$P(X \geq \alpha \cdot \mu) \leq \frac{1}{\alpha}$$