Vorlesungsmitschrieb

Algorithmen und Berechenbarkeit

Vorlesung 03

Letztes Update: 2017/11/18 - 12:46 Uhr

Laufzeiten

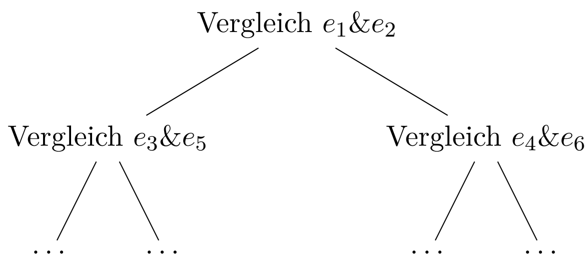
- $O(n^2)$ beschreibt die Menge aller Funktionen, die echt langsamer wachsen als n^2.
  - **In $O(n^2)$:** $n, \quad n^{1{,}99}, \quad n \cdot \log(n), \quad n \cdot \log^2(n)$
  - **Nicht in $O(n^2)$:** $n^2, \quad n^2 \cdot \log(n), \quad 2^n$
- $\Omega(n^2)$ beschreibt die Menge aller Funktionen, die mindestens so schnell wie n^2 wachsen (asymptotisch).
  - **In $\Omega(n^2)$:** $n^2, \quad n^2 \cdot \log(n), \quad n^{2{,}1}, \quad 2^n$
  - **Nicht in $\Omega(n^2)$:** $n \cdot \log(n), \quad n^{1{,}9}, \quad \sqrt{n}, \quad n \cdot \log^2(n), \quad \frac{n^2}{\log(n)}$
- $\omega(n^2)$ beschreibt die Menge aller Funktionen, die echt schneller wachsen als n^2.
  - **In $\omega(n^2)$:** $n^{2{,}1}, \quad n^2 \cdot \log(n), \quad 2^n$
  - **Nicht in $\omega(n^2)$:** $n^2, \quad n \cdot \log(n), \quad \frac{n^2}{\log(n)}, \quad \log(n)$
- $\Theta(n^2)$ beschreibt die Menge aller Funktionen, die sowohl in $O(n^2)$ als auch in $\Omega(n^2)$ enthalten sind.

Vergleichsbasiertes Sortieren

Es wird *Vergleichsbasiertes Sortieren* von n Objekten betrachtet, wobei die Elemente nur verglichen werden dürfen. Hier stellt sich die **Frage**: Was ist die Komplexität des vergleichsbasierten Sortierens?
Die ersten Überlegungen stellen klar: Jeder Sortieralgorithmus liefert eine obere Schranke für $T(n)$:

- **Bubblesort** $\Rightarrow T(n) \in O(n^2)$
- **Mergesort** $\Rightarrow T(n) \in O(n \cdot \log(n))$

Es kann bewiesen werden, dass $T(n) \in \Omega(n \cdot \log(n)) \quad \Rightarrow \quad T(n) \in \Theta(n \cdot \log(n))$
[**Beweis**] Man betrachte einen beliebigen Algorithmus $\mathcal{A}$ zum Sortieren. $\mathcal{A}$ vergleicht e_i mit e_j.



Exemplarische Vergleiche beim vergleichsbasierten Sortieren

Ein Blatt in diesem Baum heißt im Prinzip: Der Algorithmus hat fertig sortiert. Es heißt aber genauso: Der Algorithmus hat „herausgefunden“, was die „Permutation“ der Eingabe war. Daraus folgt, dass der Baum $n!$ Blätter haben muss. Es zeigt sich außerdem, dass die Worst-Case-Laufzeit des Algorithmus $\mathcal{A}$ genau der Tiefe des Baumes entspricht. Damit stellt sich die nächste **Frage**: Was ist die minimale Tiefe eines Binärbaumes, der $n!$ Blätter hat?

$$2^n = n!$$
$$\left(n! \approx \left(\frac{n}{e}\right)^{\frac{n}{e}} \Rightarrow x = \log_2 \left(\frac{n}{e}\right)^{\frac{n}{e}} \Rightarrow x = n \cdot \log(n)\right)$$

Monte-Carlo und Las-Vegas Algorithmen ineinander umwandeln

Es stellt sich die **Frage**, ob jeder Las-Vegas-Algorithmus in einen Monte-Carlo-Algorithmus umgewandelt werden kann, und andersherum.

Las-Vegas $\Rightarrow$ Monte-Carlo

Die Idee besteht aus folgender Überlegung: Lasse den Las-Vegas-Algorithmus eine bestimmte Anzahl an Schritten laufen und breche dann ab. War der Las-Vegas-Algorithmus bis dahin fertig, so muss auch das Ergebnis korrekt sein. War der Las-Vegas-Algorithmus bis dahin nicht fertig, dann gibt es auch kein korrektes Ergebnis.
Die zentrale **Frage**, die sich hier anfügt: Wie lange darf der Algorithmus laufen und was ist seine Erfolgswahrscheinlichkeit?
Sei nun $\mathcal{A}$ ein Las-Vegas-Algorithmus mit erwarteter Laufzeit $f(n)$. $\mathcal{A}$ darf nun für maximal $\alpha \cdot f(n) \; |\alpha \geq 1$ Schritte laufen. Falls der Algorithmus bis dahin fertig ist, ist das Ergebnis sicher korrekt. Ist der Algorithmus bis dahin nicht fertig, so wird *Müll* zurückgegeben.
Dieser modifizierte Algorithmus hat immer eine Laufzeit von $< \alpha \cdot f(n)$. Die Wahrscheinlichkeit, dass *Müll* zurückgegeben wird, ist gleich der Wahrscheinlichkeit, dass $\mathcal{A}$ länger als $\alpha \cdot f(n)$ Zeit zum Sortieren benötigt.

Monte-Carlo $\Rightarrow$ Las-Vegas

Nicht alle Monte-Carlo-Algorithmen können ohne Weiteres in Las-Vegas-Algorithmen umgewandelt werden.
Für manche Probleme ist die Verifikation des Ergebnisses einfacher als die Berechnung:

- Sortieren: $O(n \cdot \log(n)) \quad$ vs. $\quad O(n)$.
- Kürzeste Wege: $O(n \cdot \log(n + m)) \quad$ vs. $\quad O(m)$.

Das Überführen von Monte-Carlo- in Las-Vegas-Algorithmen ist möglich, wenn man einen effizienten *Checker* hat.

- Monte-Carlo-Algorithmus $\mathcal{A}$ hat eine Laufzeit von $f(n)$.
- Checker $\mathcal{O}$ hat eine Laufzeit von $f(n)$.
- Die Erfolgswahrscheinlichkeit von $\mathcal{A}$ sei $p(n)$.

Mit folgendem Algorithmus kann dann ein Las-Vegas-Algorithmus erzeugt werden:

```
1. Lasse Algorithmus laufen
2. Überprüfe Ergebnis mit Checker
    Falls korrekt, dann fertig
    Falls nicht korrekt, zurück zu 1.
```

Die erwartete Laufzeit $\mathcal{R}$ ist dann

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

Anhang

Markov-Ungleichung

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

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