

Zusammenfassung - Kryptographie

Marc Meier

13. Oktober 2015

Korrektheit und Vollständigkeit der Informationen sind nicht gewährleistet. Macht euch eigene Notizen oder ergänzt/korrigiert meine Ausführungen!

Inhaltsverzeichnis

1 Grundlagen und Wiederholungen

1

Planung der Veranstaltung

1. Wiederholung (ca 3-4 Termine)

- Probleme, Kodierung von Problemen, Laufzeit von Algorithmen
- P, PN, NPC...
- Grundlegende Probleme & Algorithmen

2. Algorithmen für schwierige Probleme (ca. 15 Termine)

- parallele / randomisierte Algorithmen
- Approximationsalgorithmen
- parametrisierte Algorithmen

3. Kryptographie (ca 10 Termine)

- Public-Key-Kryptographie

1 Grundlagen und Wiederholungen

1.1 Probleme, deren Kodierung und Laufzeit von Algorithmen

Anhand einiger Beispiele soll die Wichtigkeit der Kodierung der Probleme bzw. der Eingabe verdeutlicht werden.

1.1.1 Sortieren

Eingabemenge: Natürliche Zahlen a_1, a_2, \dots, a_n

Ausgabe: Eingabe aufsteigend sortiert

Beim Sortieren handelt es sich um ein *Suchproblem*. Beispiele für bekannte Sortieralgorithmen und deren Laufzeitkomplexität sind Bubblesort $\mathcal{O}(n^2)$ und Mergesort $\mathcal{O}(n \log n)$ ¹.

1.1.2 Eingabelänge N

Die Länge der Eingabe entspricht *nicht* der Anzahl der zu sortierenden Elemente n . Dies ist aufgrund der Kodierung in eine maschinenlesbare Form der Fall. Für gewöhnlich verwenden heutige Computer eine Darstellung im Binärsystem. Daher gilt:

$$N := \sum_{i=1}^n \log a_i \quad (1)$$

¹Sofern nicht anders angegeben entspricht $\log n$ immer dem dualen Logarithmus von n (Basis 2)

Im Folgenden soll die Eingabe $[11, 13, 113]$ kodiert werden. Für eine Turingmaschine mit dem Eingabealphabet $\Sigma := \{0, 1, \$\}$ würde diese folgendermaßen aussehen: 1011\$1101\$1110001. Um eine Lesbarkeit für Binärrechner zu ermöglichen, werden weiterhin folgende Ersetzungen durchgeführt: $1 \rightarrow 11$ $0 \rightarrow 00$ $\$ \rightarrow 01$. Die resultierende Kodierung wäre nun: 110011110**1**111100110**1**11111100000011. Sei nun L die maximale Wortlänge².

$$L := \max_{i=1}^n \log a_i \quad (2)$$

Hier sollte ergänzt werden, wie man auf das $O(N^3)$ kommt.

1.1.3 Primzahl

Eingabemenge: Eine natürliche Zahl n

Ausgabe: Ist n eine Primzahl?

Es handelt sich um ein *Entscheidungsproblem*. Die Eingabelänge ist $N := \log n$. Ein naiver Algorithmus wird im Folgenden beschrieben.

```

1: if  $n = 2$  then
2:   return Ja
3: else
4:   for  $d := 2$  to  $n - 1$  do
5:     if  $n \bmod d = 0$  then
6:       return Nein
7:     else
8:       end if
9:   end for
10:  return Ja
11: end if
```

Die Komplexität lässt sich wie folgt herleiten: Für die Verzweigungen (Zeilen 1 und 5), sowie für die return-Statements in den Zeilen 2 und 6 ist eine beschränkte Komplexität $O(1)$ anzunehmen. Alle Statements in der for-Schleife werden $n - 2$ mal wiederholt, die Komplexität ist $O(n)$.

Die daraus resultierende Komplexität $O(n)$ ist äquivalent zu $O(2^{\log n})$. Weil $\log n$ der Eingabelänge N entspricht, ist die Komplexität entsprechend $O(2^N)$, also exponentiell. Demnach ist der Algorithmus nicht effektiv.

²An dieser Stelle habe ich mit Förmig geschnattert und weiß daher nicht, ob das so korrekt ist