Zusammenfassung - Kryptographie

Marc Meier

15. Oktober 2015

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

Inhaltsverzeichnis

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 $a_1, a_2, ..., a_n$

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:

$$N := \sum_{i=1}^{n} \log a_i \tag{1}$$

[1]Sofern 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: $1 \to 11$ $0 \to 00$ $\$ \to 01$ Die resultierende Kodierung wäre nun: 11001111**01**11110011**01**11111100000011. Sei nun L die maximale Wortlänge[2].

$$L := max_{i=1}^{n} \log a_i \tag{2}$$

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 mod 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 effizient.

1.1.4 Cliquensuche im Graphen

Eingabemenge: Graph $G = (V, E)$ und $k \in N$

Ausgabe: Hat G mindestens paarweise verbundene Knoten?

Es handelt sich hier wieder um ein *Entscheidungsproblem.*

> Clique Eingabe: Graph G=(V,E), k ∈ N Frage: hat G eine Clique mit ¿=k knoten?

> Clique-Suchversion: Eingabe Graph G=(V,E) Ausgabe: Bestimme eine Clique C mit maximaler Knotenzahl

Behauptung: Clique in polynomieller Zeit lösbar =¿ Clique-Suchversion auch in polynomieller Zeit lösbar polynomielle Zeit in n = —V¿ (Knotenzahl des Graphen)

Beweis: $\Leftarrow$ Zur Eingabe (G,k) von CLIQUE bestimmen wir zunächst mit dem Alg für CLIQUE-Suchversion eine größte Clique C von G. Dann vergleiche $|C| \geq k$? Ausgabe JA, falls $|C| \geq k$ sonst NEIN

Laufzeit: Wie Laufzeit für Clique-Suchversion.

$\Rightarrow$ Betrachte folgende Zwischenversion für CLIQUE:

> CLIQUE-Zwischenversion: Eingabe: Graph G=(V,E) Aufgabe: Bestimme die maximalzahl $\omega(G)$[3] von paarweise verbundenen Knoten in G

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

Behauptung: CLIQUE in polyzeit lösbar $\Rightarrow^{(1)}$ CLIQUE-Zwischenversion in polizeit lösbar $\rightarrow^{(2)}$ CLIQUE-Suchversion in polyzeit lösbar. zu (1): Sei A ein polizeit-Algorithmus für CLIQUE. Zur Eingabe G von CLIQUE-Zwischenversion wende A auf Eingaben $(G, n), (G, n-1), ..., (G, 1)$ an! Ausgabe ist $\omega(G)$ = erstes k mit $A(G, k) = Ja$

For k:= n to 1 do if A(G,k) = 'ja' then return $omega(G) = k$

Laufzeit: $n \cdot O(n^c) = O(n^{c+1}$

(2): Sei A ein polyzeit Algorithmus für Clique-Zwischenversion. d.h. $A(G) = \omega(G)$ in $O(n^d)$ für eine Konstante d.

Zur Eingabe G von Clique-Suchversion wende folgendes an:

FOR jede Kante $e \in E$ do if $\omega(G-e) = \omega(G)$ then $G := G - e$ Return C := V(G) ohne isolierte Knoten

Graph G = (V,E), e = c – x a – b a – c b – c c – d c –x x – y

wird zu G - e = (V, E

e)

bei G - xy wird y isoliert.

Im Beispiel: $\omega(G) = 3$ G - cx -¿ $\omega = 3$ G - bc -¿ $\omega = 2$

Lösung a – b – c = C

Laufzeit: $m \cdot O(n^d) = O(n^{d+2})$

Gegenüberstellung ëinfache Formulierung"vs "formale Formulierung"PRIM: ïst n prim?"genauer: PRIM: Eingabe: Natürliche Zahl n Frage: ist n eine Primzahl?

FORMAL: $PRIM = \{bin(n)|n \in Nistprim\} \subset 0, 1^{+4}$ Venn-Diagramm $PRIM \subset \{0, 1\}^+$

In der Praxis ermittelt man die Komplexität der Einfachheit halber mit uniformen Kostenmaß. Sortierung: Eingabe: $a_1, a_2, ..., a_n$ Aufgabe: Sortiere Zahlen aufsteigend Mergesort: $O(n \cdot \log n$

Literatur

[4] bin(n) - Binärdarstellung von n