RWTH Aachen **WS 2012/13**
Lehrgebiet Theoretische Informatik Übungsblatt 8
Reidl–Ries–Rossmanith–Sanchez–Tönnis 3.12.2012

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T18

a) Wiederholen Sie die Definition der O-Notation.

b) Welche der folgenden Aussagen sind wahr?

1. $n^2 = O(2^n)$
2. $\sqrt{n} = O(n/\log n)$
3. $\sqrt{n-1} = \sqrt{n} + O(1)$
4. $\displaystyle\int_0^{O(n)} O(t)\,dt = O(n^2)$
5. $(\log n)^n = O\big(n^{\log n}\big)$

Lösungsvorschlag.

1. $n^2 = O(2^n)$
   Wahr.
2. $\sqrt{n} = O\left(\frac{n}{\log n}\right)$
   Wahr.
3. $\sqrt{n-1} = \sqrt{n} + O(1)$
   Wahr.
4. $\int_0^{O(n)} O(t)dt = O(n^2)$
   Wahr.
5. $(\log n)^n = O\left(\left(n^{\log n}\right)\right)$
   Falsch.

Aufgabe T19

Wir bezeichnen mit p_N die N-te Primzahl.
Zu gegebener Eingabe $N \in \mathbb{N}$ sei die kleinste Primzahl gesucht, welche größer als N ist.
Ist diese Aufgabe in polynomieller Zeit lösbar, falls

a) N in unärer Kodierung vorliegt,

b) N in binärer Kodierung vorliegt?

Hinweis zu Teilaufgabe b: Cramérs Vermutung von 1936 besagt, daß die Primzahllücke $g(p_n) = p_{n+1} - p_n$ zwischen der n-ten und $(n+1)$-ten Primzahl durch

$$g(p_n) = O\big((\ln p_n)^2\big)$$

beschränkt ist.

Lösungsvorschlag.
Zunächst erinnern wir daran, daß Zahlen auf Primalität zu testen in polynomieller Zeit möglich ist.

a) In der unären Kodierung reicht es auszunutzen dass zwischen N und $2N$ immer eine Primzahl zu finden ist. Die $(N+1)$-te Primzahl befindet sich also in diesem Intervall. Testen wir jede Zahl dazwischen, finden wir nach höchstens $O(N)$ Schritten die nächste Primzahl. Damit ergibt sich eine Laufzeit von $O(N \cdot P)$, wobei P die Laufzeit des Primzahltestes darstellen soll.

b) Wendet man hier die Lösung aus Aufgabenteil a) an, ergibt sich eine exponentielle Laufzeit. Dies liegt daran, dass wir N Zahlen testen müssen, was im logarithmischen Kostenmaß exponentiell in der Eingabelänge ist. Nutzen wir jedoch Cramérs Vermutung, ergibt sich das nur logarithmisch viele Zahlen getestet werden müssen. Was wieder zu einer polynomieller Laufzeit führt. Damit erhalten wir einen Algorithmus, der entweder in polynomieller Zeit läuft oder aber eine wichtige mathematische Vermutung widerlegt.

Aufgabe H18 (5 Punkte)
Für einen beliebigen Algorithmus sei $t(n) \geq n$ die Laufzeit im uniformen Kostenmaß und $t'(n)$ die Laufzeit im logarithmischen Kostenmaß.
Finden Sie eine möglichst interessante Beziehung zwischen $t(n)$ und $t'(n)$.

Lösungsvorschlag.
Ein einfaches Beispiel für einen Algorithmus der im uniformen Kostenmaß Laufzeit $O(n)$ hat und im logarithmischen Kostenmaß $O(2^n)$ ist der folgende:

Eingabe: $n \in \mathbb{N}$
    $x = 2$
    `For` $i = 1, ..., n$`Do`
        $x = x^2$
    `End For`
Ausgabe: x

Hier wird $x = 2^{2^n}$ berechnet was als Ausgabe für sich genommen schon einen Platzbedarf von 2^n hat. Im uniformen Kostenmaß wird die Schleife allerdings nur n mal gezählt und die Ausgabe in einem Schritt ausgegeben. Das logarithmische Kostenmaß ist hier viel realistischer, da hier jeder Schleifendurchlauf mit bis zu 2^n eingeht.

Aufgabe H19 (10 Punkte)

Welche Funktion berechnet der folgende Algorithmus?

Analysieren Sie seine Laufzeit sowohl im uniformen als auch im logarithmischen Kostenmaß.

Eingabe: $a \in \mathbb{N}$
    $b = 2$
    while $a \neq 0$
        $b = b \cdot b$
        $a = a - 1$
    end while
    $c = 2$
    while $b \neq 0$
        $c = c \cdot c$
        $b = b - 1$
    end while
Ausgabe: c

Lösungsvorschlag.

Nach der ersten Schleife hat b den Wert $b = 2^{2^a}$. Nach Durchlauf der zweiten Schleife ist, analog, $c = 2^{2^b} = 2^{2^{2^{2^a}}}$.

uniform: Unter der Voraussetzung das multiplizieren lediglich eine Zeiteinheit kostet, wird die Laufzeit des Algorithmus durch die zweite Schleife dominiert: insgesamt benötigt er $O(2^{2^a})$ Operationen.

log.: Im letzten Durchlauf der zweiten Schleife wird die Zahl $2^{2^{b-1}}$ mit sich selbst multipliziert, diese Operation dominiert die Laufzeit des Algorithmus: im logarithmischen Kostenmaß kostet sie nämlich $\log 2^{2^{b-1}} = 2^{2^{2^a}-1}$ Zeit. Damit ist die Laufzeit des obigen Algorithmus $O(2^{2^{2^a}})$.

Aufgabe H20 (8 Punkte)

Als Eingabe sei eine Menge von Intervallen

$$\left\{ \left[\frac{a_1}{b_1}, \frac{c_1}{d_1}\right], \left[\frac{a_2}{b_2}, \frac{c_2}{d_2}\right], \left[\frac{a_3}{b_3}, \frac{c_3}{d_3}\right], \ldots, \left[\frac{a_n}{b_n}, \frac{c_n}{d_n}\right] \right\}$$

gegeben.

Die Anfangs- und Endpunkte jedes Intervalls sind rationale Zahlen, welche durch die Binärdarstellung ihres Zählers und Nenners kodiert werden.

Ist in polynomieller Zeit berechenbar, ob der Schnitt zwischen allen gegebenen Intervallen leer ist?

Falls Sie der Meinung sind, daß die Antwort *ja* ist, dann geben Sie einen Algorithmus an, der die Aufgabe in polynomieller Zeit löst. Wie groß ist die Laufzeit Ihres Algorithmus?

Lösungsvorschlag.
Gehe die Intervalle durch und merke das Maximum über die Anfangspunkte a sowie das Minimum über die Endpunkte e. Falls $a \leq e$ dann ist der Schnitt nicht leer. Jetzt liegt nämlich genau das Intervall $[a, e]$ in allen gegebenen Intervallen. Ansonsten ist der Schnitt leer.
Die Laufzeit im logarithmischen Kostenmaß ist in $O(n \log n)$, da Vergleiche zwischen zwei Intervallgrenzen in $O(\log n)$ machbar sind und alle Intervalle nur einmal aufgezählt werden.

Aufgabe H21 (Bonusaufgabe)
Gegeben sind zwei Mengen von natürlichen Zahlen $\{k_1, \ldots, k_n\}$ und $\{l_1, \ldots, l_n\}$, die jeweils als durch Trennzeichen separierte Binärdarstellungen als Eingabestring repräsentiert werden.
Kann die folgende Frage in polynomieller Zeit beantwortet werden?

$$\sqrt{k_1} + \cdots + \sqrt{k_n} < \sqrt{l_1} + \cdots + \sqrt{l_n}$$

Hinweis: Vorsicht! Dies ist eine seit vielen Jahren offene Frage, deren Beantwortung weitreichende Konsequenzen hätte. Es ist heute noch nicht einmal bekannt, ob dieses Problem in *NP* liegt.