Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Thomas Seidl
Anna Beer, Florian Richter

München, 26.04.2018

Algorithmen und Datenstrukturen

SS 2018

Übungsblatt Global 2: Komplexität

Aufgabe Global 2-1 *Knobelei: Euklidischer Algorithmus*

(a) Formulieren Sie den Euklidischen Algorithmus zum Finden des ggT von zwei Zahlen

Lösungsvorschlag:

```
public static int ggT(int a, int b) {
        int h;
        if (a<b) {h=a ; a=b ; b=h}
        while (b != 0) {
                h = a%b ;
                a = b ;
                b = h ;
        }
        return a;
}
```

(b) Berechnen Sie die Komplexität des Euklidischen Algorithmus.

Lösungsvorschlag:

Wir verwenden den Satz von Lamé:
Seien $a \in \mathbb{N}$ und $b \in \mathbb{N}$, so dass $a \geq b$. Benötigt der Euklidische Algorithmus zur Berechnung von $ggT(a, b)$ insgesamt n Iterationen, so gilt $b \geq fib(n)$, wobei $fib(n)$ die n-te Fibonacci-Zahl ist.
Beweis: Wir versuchen, die Ausgabe von unten nach oben zu konstruieren, s.d. für eine gegebene Anzahl an Schleifeniterationen n das minimal mögliche b und das minimal mögliche a ensteht:

- $a = 8,\ b = 5$ (n=5)
- $a = 5,\ b = 3$ (n=4)
- $a = 3,\ b = 2$ (n=3)
- $a = 2,\ b = 1$ (n=2)
- $a = 1,\ b = 1$ (n=1)
- $a = 1,\ b = 0$ (n=0)

Das kleinstmögliche a_n in Zeile n, das bei Division durch b_n in Zeile n den Rest b_{n-1} in Zeile $n - 1$ ergibt, ist:

$$a_n = b_n + b_{n-1}$$

$$b_{n+1} = b_n + b_{n-1}$$

Entspricht der Fibonacci Folge. Daraus kann man herleiten, dass $b_n \geq fib(n)$ (für n Schleifeniterationen bei der Berechnung $ggT(a_n, b_n)$)
Es ist bekannt, dass für große n gilt $fib(n) \approx \Phi^n$ (für die "Goldene Zahl" $\Phi \approx 1,618...$)
Aus $b_n \geq fib(n) \approx \Phi^n$ folgt, dass $\log_\Phi b_n \geq n$
Also $O(ggT(a, b) \in O(log(b)$

Aufgabe Global 2-2 *Komplexitätsklassen*

Vergleichen Sie die Komplexitätsklassen $O(\log n)$, $O(\sqrt{n})$, $O(\log^2 n)$, $O(\log(n^2))$, $O(n)$, $O(log(log(n)))$ und $O(\log^k n)$ miteinander. Zeigen Sie die Korrektheit der von Ihnen gefundenen Ordnung.

Lösungsvorschlag:

Da $O(\log_2 n) = O(\log_e n) = O(\log_{10}(n)$ können wir jeweils die Basis wählen für die die Rechnung am einfachsten ist. Korrekte Ordnung: $O(log(log(n))) \subseteq O(\log n) = O(\log(n^2)) \subseteq O(\log^2 n) \subseteq O(\log^k n) \subseteq O(\sqrt{n}) \subseteq O(n)$

Begründung $O(log(log(n))) \subseteq O(\log n)$:

- Alternative 1: Wir wissen $\forall n \in \mathbb{N} : log(n) < n$. Außerdem ist $log(n)$ streng monoton steigend ,also gilt auch $\forall n \in \mathbb{N} : log(log(n)) < log(n)$
- Alternative 2: Wir berechnen den Grenzwert

$$\lim_{n\to\infty} \frac{\log(\log(n))}{\log(n)} \underset{l'Hospital}{=} \lim_{n\to\infty} \frac{\frac{1}{\log(n)} \cdot \frac{1}{n}}{1/n} = \lim_{n\to\infty} \frac{1}{log(n)} = 0 < c < \infty$$

Begründung $O(\log n) = O(\log(n^2))$: Logarithmus Rechenregeln: $\log(n^k) = k * \log(n)$. Also $O(log(n^k)) = O(k * \log(n)) = O(\log(n))$

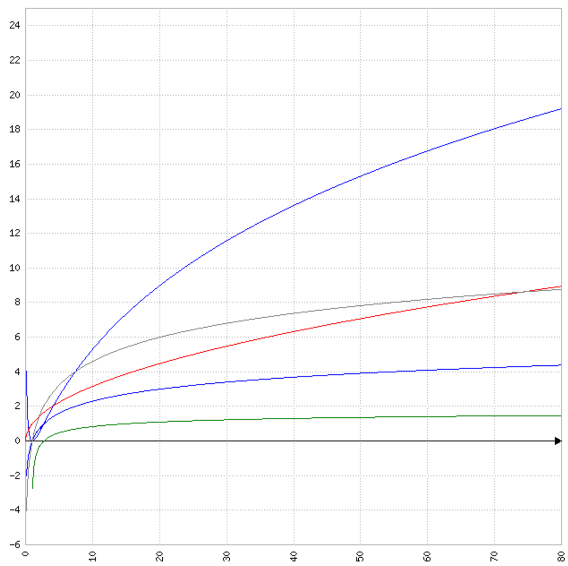
Begründung $O(\log n) \subseteq O(\log^2 n)$ Wähle $n_0 = 2$ (und $c = 1$), dann $\forall n \geq n_0 : 1 \leq log(n) \Rightarrow \forall n \geq n_0 : log(n) \leq log(n) * log(n) = log^2(n)$

Begründung $O(\log^2 n) \subseteq O(\log^k n)$ äquivalent.

Begründung $O(\log^k n) \subseteq O(\sqrt{n})$. Betrachte den Grenzwert

$$\lim_{n\to\infty} \frac{\ln^k(n)}{\sqrt{n}} \underset{l'Hospital}{=} \lim_{n\to\infty} \frac{k \ln^{k-1}(n) * 1/n}{\frac{1}{2}n^{-1/2}} = \lim_{n\to\infty} \frac{2k \ln^{k-1}(n)}{\sqrt{n}} \underset{l'Hospital}{=}$$
$$2k \lim_{n\to\infty} \frac{(k-1)\ln^{k-2}(n)}{\frac{1}{2}n^{-1/2}} = 2^2 * k * (k-1) * \lim_{n\to\infty} \frac{\ln^{k-2}(n)}{\sqrt{n}} = \cdots =$$
$$2^k k! \lim_{n\to\infty} \frac{ln^0(n)}{\sqrt{n}} = 0 < c < \infty$$

Begründung $O(\sqrt{n}) \subseteq O(n)$ Wähle $n_0 = 1$ und $c = 1$. Dann gilt $\forall n \geq n_0 : 1 \leq |\sqrt{n}|$. Daraus folgt $\forall n \geq n_0 : |\sqrt{n}| \leq c * |n|$. Also $O(\sqrt{n}) \subseteq O(n)$



Grün: $\ln(ln(n))$, Blau (unten): $\ln(n)$, Rot: $\sqrt{n}$, Grau: $\ln(n^2)$, Blau (oben): $\ln^2(n)$

Aufgabe Global 2-3 *Sieb des Eratosthenes*

(a) Schreiben sie eine Java-Klasse mit der Methode `public static boolean[] primes(int n)`, die für ein gegebenes n berechnet, welche Zahlen $p <= n$ Primzahlen sind. Orientieren Sie sich dabei am Sieb des Eratosthenes. Die Primzahlen sollen in dem zurückgegebenen Array mit `TRUE` markiert sein, alle anderen durch `FALSE`.

Lösungsvorschlag:

```
public static boolean[] primes (int n) {
        boolean[] prime = new Boolean[n+1];

        for (int i=2; i <= n; i++)
                prime[i]=true;


        for (int i=2; i <= Math.sqrt(n); i++)
                if (prime[i] == true)
                        for (int j = i+i; j <= n; j +=i)
                                prime[j] = false;
                return prime;
}
```

(b) Machen Sie eine Abschätzung, wie viel Speicher für die Berechnung der Primzahlen benötigt wird.

Lösungsvorschlag:

Für die Untersuchung der Zahlen wird jeder Zahl ein Flag zugeordnet, das angibt, ob es sich um eine Primzahl handelt oder nicht. Damit liegt der Speicheraufwand dieses Algorithmus in O(n).

(c) In welcher Komplexitätsklasse liegt das Verfahren?
Hinweis: $\sum_{i=1}^{n} \frac{1}{i} \approx \ln n * \gamma \approx \ln n * 0.58$

Lösungsvorschlag:

Initialisierungskosten des Arrays: $O(n)$
Eigentliches Verfahren:
Sei $T(2) = 1$

$$T(p) = 1 + T(p-1) + \left\{ \begin{array}{l} \lceil \frac{n}{p} \rceil, \text{falls p Primzahl} \\ 0, \text{sonst} \end{array} \right\} \leq 1 + T(p-1) + \frac{n}{p}$$

Daraus folgt:

$$T(n) \leq \sum_{i=1}^{n} (1 + n/i) \approx n + n * (\ln n * \gamma) = n + n * \ln n * \gamma \in O(n \ln n)$$

Die Abschätzung basiert auf der harmonischen Reihe $\sum_{i=1}^{n} \frac{1}{i} \approx \ln n + \gamma$, wobei es sich bei γ um die eulersche Konstante $0,57721..$ handelt. Mit Initialisierung ergibt sich: $O(n) + O(n \ln n) \subseteq O(n \ln n)$ Das nur $\sqrt{n}$-malige Ausführen der Schleife ergibt für die Summe eine neue obere Grenze:

$$T(n) \leq \sum_{i=1}^{\sqrt{n}} (1 + n/i) \approx \sqrt{n} + n * (\ln \sqrt{n} * \gamma) = \sqrt{n} + n * 1/2 \ln n * \gamma \in O(n \ln n)$$

Die Komplexitätsklasse ändert sich somit nicht durch das $\sqrt{n}$-malige Ausführen.