Datenstrukturen und Algorithmen

Übung 2

FS 2020

Programm von heute

1. Feedback letzte Übung
2. Wiederholung Theorie
   - Induktion
   - Analyse von Programmen
   - Lösen einfacher Rekurrenzgleichungen

Landau-Notation

- Geben Sie eine korrekte, kompakte Definition der Menge $\Theta(f)$ analog zur Definition der Mengen $\mathcal{O}(f)$ und $\Omega(f)$.

Landau-Notation

- Geben Sie eine korrekte, kompakte Definition der Menge $\Theta(f)$ analog zur Definition der Mengen $\mathcal{O}(f)$ und $\Omega(f)$.
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n)\ \forall n \geq n_0\}$

Landau-Notation

- Geben Sie eine korrekte, kompakte Definition der Menge $\Theta(f)$ analog zur Definition der Mengen $\mathcal{O}(f)$ und $\Omega(f)$.
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n)\ \forall n \geq n_0\}$
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0,\ n_0 \in \mathbb{N} : \frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n)\ \forall n \geq n_0\}$

Landau-Notation

Beweisen oder widerlegen Sie, mit $f, g : \mathbb{N} \to \mathbb{R}^+$.

(a) $f \in \mathcal{O}(g)$ if and only if $g \in \Omega(f)$.

(e) $\log_a(n) \in \Theta(\log_b(n))$ for all constants $a, b \in \mathbb{N} \setminus \{1\}$

(g) If $f_1, f_2 \in \mathcal{O}(g)$ and $f(n) := f_1(n) \cdot f_2(n)$, then $f \in \mathcal{O}(g)$.

Landau-Notation

Funktionen sortieren: wenn Funktion f links der Funktion g steht, dann $f \in \mathcal{O}(g)$.

$2^{16},\ \log(n^4),\ \log^8(n),\ \sqrt{n},\ n\log n,\ \binom{n}{3},\ n^5+n,\ \frac{2^n}{n^2},\ n!,\ n^n.$

Summe von Elementen in zweidimensionalem Feld

Probleme / Fragen?

2. Wiederholung Theorie

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.
- Induktionshypothese: Wir nehmen an, die Aussage stimmt für ein allgemeines n.

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.
- Induktionshypothese: Wir nehmen an, die Aussage stimmt für ein allgemeines n.
- Induktionsschritt $(n \rightarrow n+1)$:
  - Aus der Gültigkeit der Aussage für n (Induktionshypothese) folgt die Gültigkeit für $n+1$.
  - z.B.: $\sum_{i=1}^{n+1} i = n + 1 + \sum_{i=1}^{n} = n + 1 + \frac{n(n+1)}{2} = \frac{(n+2)(n+1)}{2}$.

Induktion: Beispiel

- Zu zeigen: $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$.

Induktion: Beispiel

- Zu zeigen: $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$.
- Induktionsanfang:
  $n = 0$: $\sum_{i=0}^{0} r^i = 1 = \frac{1-r^1}{1-r}$.

Induktion: Beispiel

- Zu zeigen: $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$.
- Induktionsanfang:
  $n = 0$: $\sum_{i=0}^{0} r^i = 1 = \frac{1-r^1}{1-r}$.
- Induktionsschritt $(n \to n+1)$:

$$\sum_{i=0}^{n+1} r^i = r^{n+1} + \sum_{i=0}^{n} r^i$$
$$= r^{n+1} + \frac{1-r^{n+1}}{1-r} = \frac{r^{n+1} - r^{n+2} + 1 + r^{n+1}}{1-r} = \frac{1-r^{n+2}}{1-r}.$$

[Übrigens ..]

Das lässt sich auch einfach direkt zeigen

$$
\begin{aligned}
\frac{r^{n+1}-1}{r-1} &\stackrel{!}{=} \sum_{i=0}^{n} r^i \\
(r-1)\cdot\sum_{i=0}^{n} r^i &= \sum_{i=0}^{n} r^{i+1} - \sum_{i=0}^{n} r^i \\
&= \sum_{i=1}^{n+1} r^i - \sum_{i=0}^{n} r^i = \sum_{i=0}^{n+1} r^i - 1 - \sum_{i=0}^{n} r^i \\
&= r^{n+1} - 1
\end{aligned}
$$

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Das Code-Fragment ruft `f()` $\Theta(n^2)$ mal auf: die äußere Schleife wird $n/9$ mal durchlaufen, und die innere Schleife ruft `f()` i mal auf.

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.
Die zweite innere Schleife ruft `f()` $\lfloor \log_2(n) \rfloor + 1$ mal auf, in Summe haben wir $\Theta(n \log(n))$ Aufrufe.

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$$T(0) = 1$$

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$
$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$
$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

n	0	1	2	3	4
$T(n)$	1	2	4	8	16

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$
$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

n	0	1	2	3	4
$T(n)$	1	2	4	8	16

Hypothese: $T(n) = 2^n$.

Induktion

Hypothese: $T(n) = 2^n$.
Induktionsschritt:

$$\begin{aligned} T(n) &= 1 + \sum_{i=0}^{n-1} 2^i \\ &= 1 + 2^n - 1 = 2^n \end{aligned}$$

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

Man sieht es auch direkt:

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

$$\Rightarrow T(n-1) = 1 + \sum_{i=0}^{n-2} T(i)$$

$$\Rightarrow T(n) = T(n-1) + T(n-1) = 2T(n-1)$$

Rekursionsgleichung

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \frac{n}{2} + 1, & n > 1 \\ 3 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (nicht rekursive), einfache Formel für $T(n)$ an und beweisen Sie diese mittels vollständiger Induktion. Gehen Sie davon aus, dass n eine Potenz von 2 ist.

Rekursionsgleichung

$$
\begin{aligned}
T(2^k) &= 2T(2^{k-1}) + 2^k/2 + 1 \\
&= 2(2(T(2^{k-2}) + 2^{k-1}/2 + 1) + 2^k/2 + 1 = \ldots \\
&= 2^k T(2^{k-k}) + \underbrace{2^k/2 + \ldots + 2^k/2}_{k} + 1 + 2 + \ldots + 2^{k-1} \\
&= 3n + \frac{n}{2}\log_2 n + n - 1
\end{aligned}
$$

$\Rightarrow$ Annahme $T(n) = 4n + \frac{n}{2}\log_2 n - 1$

Induktion

1. Hypothesis $T(n) = f(n) := 4n + \frac{n}{2}\log_2 n - 1$
2. Base Case $T(1) = 3 = f(1) = 4 - 1.$
3. Step $T(n) = f(n) \longrightarrow T(2 \cdot n) = f(2n)$ ($n = 2^k$ for some $k \in \mathbb{N}$):

$$\begin{aligned}
T(2n) &= 2T(n) + n + 1 \\
&\overset{i.h.}{=} 2(4n + \frac{n}{2}\log_2 n - 1) + n + 1 \\
&= 8n + n\log_2 n - 2 + n + 1 \\
&= 8n + n\log_2 n + n\log_2 2 - 1 \\
&= 8n + n\log_2 2n - 1 \\
&= f(2n).
\end{aligned}$$

Fragen oder Anregungen?