Algorithmen & Datenstrukturen

Woche 3

Nicolas Wehrli, Tim Rieder, Marius Tomek

10. Oktober 2022

ETH Zürich

Organisatorisches

- Rolecall
- Peergrading Heute: Aufgabe 2.5
- Partner Feedback

Inhalt Übungsstunde Heute

- Rückblick letzte Serie
- Übungsaufgaben Induktion
- Maxsubarraysum
  - Precomputing (Präfixsumme)
  - Divide-and-Conquer
  - Resultate merken (DP)
- Laufzeiten und $\mathcal{O}$-Notation

Rückblick letzte Serie

Häufige Fehler

- Falscher Induktionsanfang (Base Case) $k = 0, 1, ...$
- Unklare Trennung der Schritte / unsaubere definition von I.H
- Beginn des Induktionsschrittes mit dem zu zeigenden

Beispiel Induktionsschritt

Sei: $a_n = 2 \cdot a_{n-1} + 1$; $a_1 = 1$ Zu Zeigen: $a_n = 2^n - 1$

I.A: $a_1 = 1 = 2^1 - 1 = 1$

I.H: Assume $a_k = 2^k - 1$ for a given k, Show the property for for $k + 1$:

Falscher Schritt:

$$a_{k+1} = 2^{k+1} - 1$$

$$\dots$$

Richtiger Schritt:

$$\begin{aligned} a_{k+1} &= 2 \cdot a_k + 1 \\ &\overset{\text{I.H}}{=} 2 \cdot (2^k - 1) - 1 \\ &= 2^{k+1} - 1 \end{aligned} \tag{1}$$

Übungsaufgaben Induktion

Aufgabe 1.3

Sei $k \in \mathbb{N}$ fixiert

a) Zeige mittels Induktion:

$$\sum_{i=1}^{n} i^k \leq n^{k+1}$$

b) Zeige mittels Induktion:

$$\sum_{i=1}^{n} i^k \geq \frac{1}{2^{k+1}} \cdot n^{k+1}$$

maxsubarraysum

Precomputing (Präfixsummen)

i	0	1	2	3	4	5
array[i]	1	6	-4	2	5	1

Wir wollen viele (z.B. q) Summen von Subarrays möglichst effizient berechnen. Wie lange dauert die naive Version (in $\mathcal{O}$-Notation)?

Precomputing (Präfixsummen)

i	0	1	2	3	4	5
array[i]	1	6	-4	2	5	1

Wir wollen viele (z.B. q) Summen von Subarrays möglichst effizient berechnen. Wie lange dauert die naive Version (in $\mathcal{O}$-Notation)?

$$\rightarrow \mathcal{O}(nq)$$

Precomputing (Präfixsummen)

i	0	1	2	3	4	5
array[i]	1	6	-4	2	5	1
pre[i]	1	7	3	5	10	11

Wie lange dauert es, `pre` zu berechnen?

Precomputing (Präfixsummen)

i	0	1	2	3	4	5
array[i]	1	6	-4	2	5	1
pre[i]	1	7	3	5	10	11

Wie lange dauert es, `pre` zu berechnen?

$$\rightarrow \mathcal{O}(n)$$

Wie lange dauert es nun q Summen von Subarrays zu berechnen?

Precomputing (Präfixsummen)

i	0	1	2	3	4	5
array[i]	1	6	-4	2	5	1
pre[i]	1	7	3	5	10	11

Wie lange dauert es, `pre` zu berechnen?

$$\rightarrow \mathcal{O}(n)$$

Wie lange dauert es nun q Summen von Subarrays zu berechnen?

$$\rightarrow \mathcal{O}(n + q)$$

Divide-and-Conquer

Probleme in Subprobleme aufzuteilen kann das Problem einfacher lösbar machen.
Divide-and-Conquer ist an sehr vielen Stellen anzutreffen.

Sich Resultate merken

Bei vielen Divide-and-Conquer-Problemen entstehen Zwischenresultate, welche wir uns sinnvollerweise merken. Dadurch müssen wir sie nicht mehrmals berechnen, sondern können sie wiederverwenden. Wir werden dies später im Semester konkreter sehen ($\rightarrow$ *Dynamische Programmierung*).

MSS - Kadane's Algorithmus

Wir schauen uns nur ein Subproblem an: Was ist das Subarray mit der grössten Summe, welches bei Position i aufhört?
Weshalb ist dies ein "gutes" Subproblem?

MSS - Kadane's Algorithmus

Wir schauen uns nur ein Subproblem an: Was ist das Subarray mit der grössten Summe, welches bei Position i aufhört?
Weshalb ist dies ein "gutes" Subproblem? $\rightarrow$ Gute Subprobleme sind einfacher lösbar und erlauben, die ursprüngliche Lösung wieder zusammenzusetzen.

MSS - Kadane's Algorithmus

```python
def max_subarray_sum(numbers):
    best_sum = 0
    current_sum = 0

    for x in numbers:
        current_sum = max(current_sum + x, 0)
        best_sum = max(best_sum, current_sum)

    return best_sum
```

Laufzeiten und $\mathcal{O}$-Notation

Laufzeit

Für einen Input von $n = 5000$ erhalten wir folgende Resultate:

	Komplexität	Zeit in ms
Naiv	$\mathcal{O}(n^3)$	52'600
Divide-and-Conquer	$\mathcal{O}(n \log n)$	17.3
Kadane	$\mathcal{O}(n)$	2.5

$\rightarrow$ $\mathcal{O}$-Notation nicht exakt, aber trotzdem aussagekräftig

(Implementation in Python, Resultate von David Zollikofer)