Informatik II

Übung 7

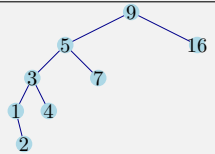
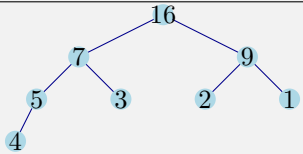
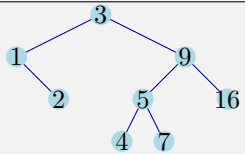
Andreas Bärtschi, Andreea Ciuprina, Felix Friedrich, Patrick Gruntz,
Hermann Lehner, Max Rossmannek, Chris Wendler

FS 2018

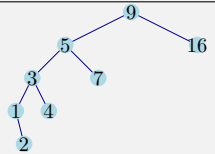
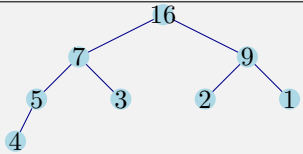
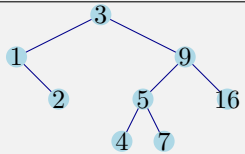
Heutiges Programm

1. Rekapitulation binäre Bäume
2. Wiederholung Vorlesung
3. String-Hashing und Modulo-Rechnen
4. In-Class-Exercises: Sliding Window

Vergleich binärer Bäume

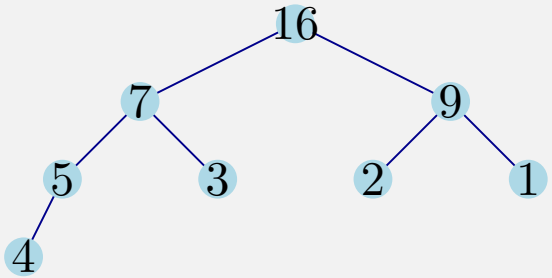
	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in Java:		PriorityQueue	TreeSet
			
Einfügen	$\mathcal{O}(h(T))$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Suchen	$\mathcal{O}(h(T))$	$\mathcal{O}(n)$ (!!)	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(h(T))$	Suchen + $\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Vergleich binärer Bäume

	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in Java:		PriorityQueue	TreeSet
			
Einfügen	$\mathcal{O}(h(T))$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Suchen	$\mathcal{O}(h(T))$	$\mathcal{O}(n)$ (!!)	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(h(T))$	Suchen + $\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Bemerke: $\mathcal{O}(\log n) \ll \mathcal{O}(h(T)) \ll \mathcal{O}(n)$

Letzte Woche: Pre- / In- / Post- Order

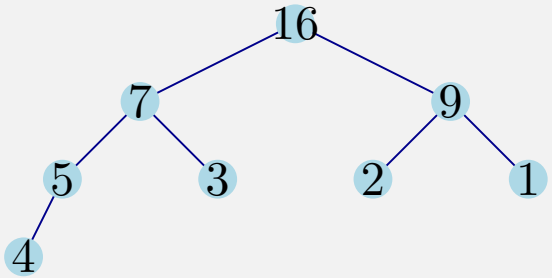


Pre-order:

In-order:

Post-order:

Letzte Woche: Pre- / In- / Post- Order

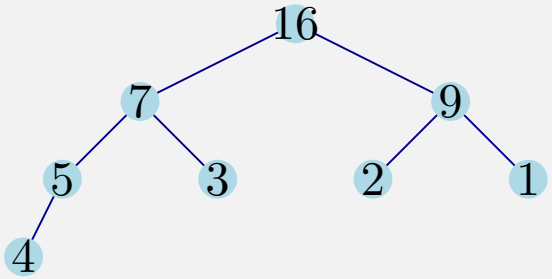


Pre-order: 16 7 5 4 3 9 2 1

In-order:

Post-order:

Letzte Woche: Pre- / In- / Post- Order

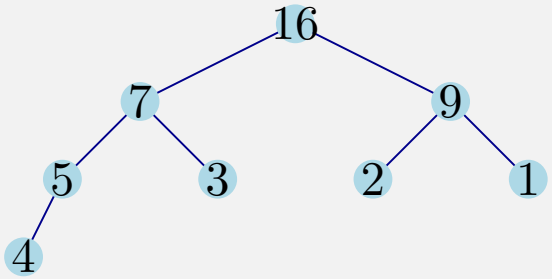


Pre-order: 16 7 5 4 3 9 2 1

In-order: 4 5 7 3 16 2 9 1

Post-order:

Letzte Woche: Pre- / In- / Post- Order



Pre-order: 16 7 5 4 3 9 2 1

In-order: 4 5 7 3 16 2 9 1

Post-order: 4 5 3 7 2 1 9 16

Gutes Hashing

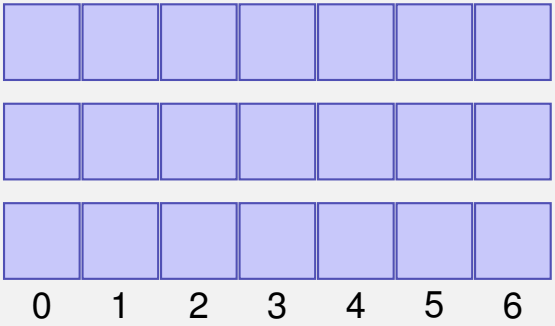
Gutes Hashing. . .

- verteilt die Menge der Schlüssel möglichst gleichmässig auf die Positionen der Hashtabelle.
- vermeidet beim Sondieren möglichst ein Ablaufen langer belegter Bereiche (siehe primäre Häufung).
- vermeidet, Schlüssel mit gleichem Hashwert auch noch gleich zu sondieren (siehe sekundäre Häufung).

Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

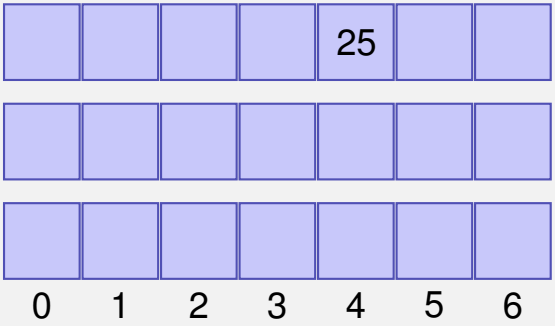
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

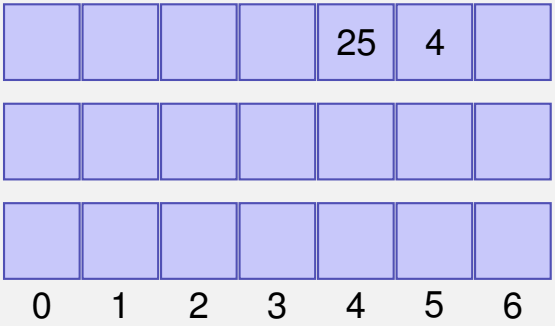
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

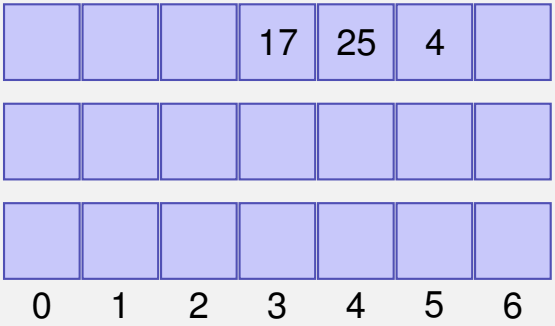
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

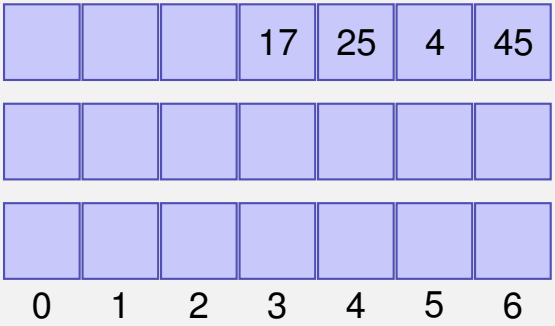
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

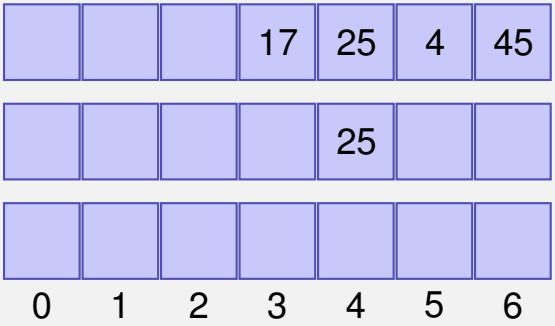
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

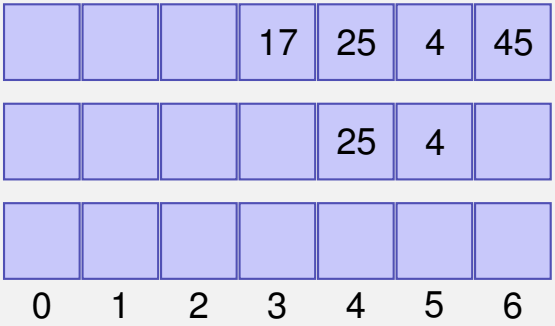
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

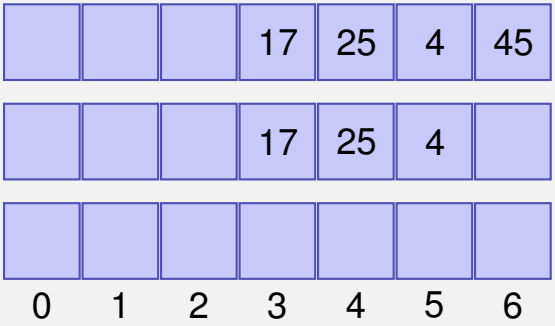
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

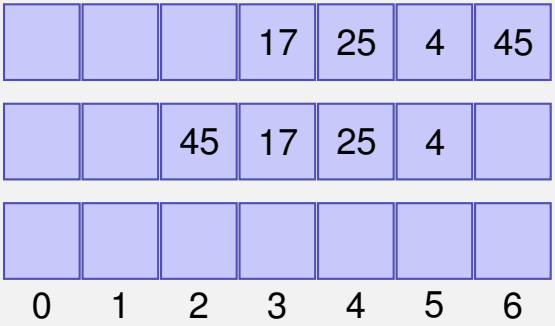
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

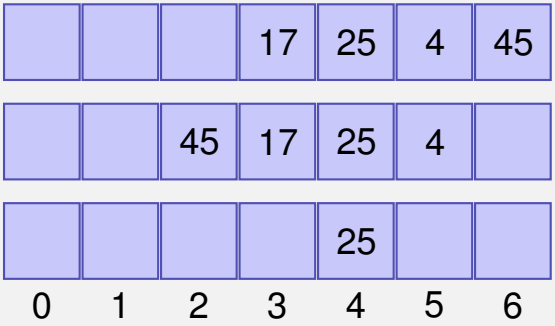
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

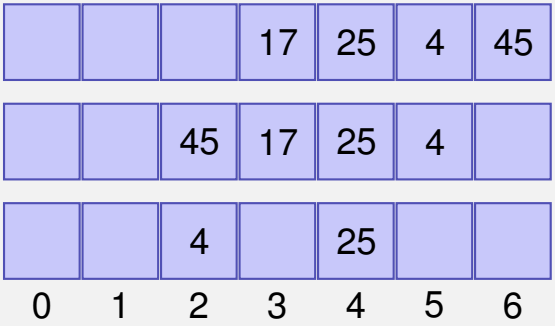
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

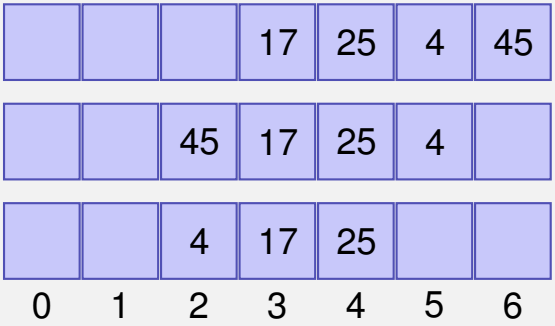
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

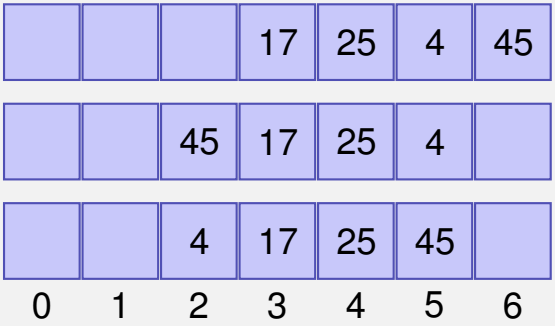
- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Beispiele Hashing

Füge die Schlüssel $25, 4, 17, 45$ in die Hashtabelle ein. Verwende dabei $h(k) = k \bmod 7$ und sondiere nach rechts, $h(k) + s(j, k)$:

- Lineares Sondieren, $s(j, k) = j$.
- Quadratisches Sondieren, $s(j, k) = (-1)^{j+1} \lceil j/2 \rceil^2$.
- Double Hashing, $s(j, k) = j \cdot (1 + (k \bmod 5))$.



Rechenregeln Modulo

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$
$$(a - b) \bmod m = ((a \bmod m) - (b \bmod m) + m) \bmod m$$
$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

Aufgabe: Berechnen Sie

$$12746357 \bmod 11$$

Rechenregeln Modulo

Aufgabe: Berechnen Sie

$$12746357 \bmod 11$$

Rechenregeln Modulo

Aufgabe: Berechnen Sie

$$
\begin{aligned}
&12746357 \bmod 11 \\
&= (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11
\end{aligned}
$$

Rechenregeln Modulo

Aufgabe: Berechnen Sie

$$
\begin{aligned}
&12746357 \bmod 11 \\
&= (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11 \\
&= (7 + 50 + 3 + 60 + 4 + 70 + 2 + 10) \bmod 11
\end{aligned}
$$

Für die zweite Gleichheit haben wir verwendet, dass $10^2 \bmod 11 = 1$.

Rechenregeln Modulo

Aufgabe: Berechnen Sie

$$
\begin{aligned}
&12746357 \bmod 11 \\
&= (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11 \\
&= (7 + 50 + 3 + 60 + 4 + 70 + 2 + 10) \bmod 11 \\
&= (7 + 6 + 3 + 5 + 4 + 4 + 2 + 10) \bmod 11
\end{aligned}
$$

Für die zweite Gleichheit haben wir verwendet, dass $10^2 \bmod 11 = 1$.

Rechenregeln Modulo

Aufgabe: Berechnen Sie

$$
\begin{aligned}
&12746357 \bmod 11 \\
&= (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11 \\
&= (7 + 50 + 3 + 60 + 4 + 70 + 2 + 10) \bmod 11 \\
&= (7 + 6 + 3 + 5 + 4 + 4 + 2 + 10) \bmod 11 \\
&= 8 \bmod 11.
\end{aligned}
$$

Für die zweite Gleichheit haben wir verwendet, dass $10^2 \bmod 11 = 1$.

Implementation Hash(String) in Java

$$h_{c,m}(s) = \left(\sum_{i=0}^{k-1} s_i \cdot c^{k-1-i} \right) \bmod m$$

```
int ComputeHash(int C, int M, String s) {
  int hash = 0;
  for (int i = 0; i < s.length(); ++i){
    hash = (C * hash % M + s.charAt(i)) % M;
  }
  return hash;
}
```

In-Class-Exercises: Sliding Window

Gegeben sei ein String `text` der Länge n.
Wir suchen den kürzesten Substring `text`$[l, r]$, welcher die Buchstaben ‘a’, ‘b’, und ‘c’ je mindestens einmal enthält.

In-Class-Exercises: Sliding Window

Gegeben sei ein String `text` der Länge n.
Wir suchen den kürzesten Substring `text`$[l, r]$, welcher die Buchstaben ‘a’, ‘b’, und ‘c’ je mindestens einmal enthält.

- Brute-Force: Alle $\frac{n \cdot (n-1)}{2}$ Substrings durchtesten dauert $\mathcal{O}(n^3)$.

In-Class-Exercises: Sliding Window

Gegeben sei ein String `text` der Länge n.
Wir suchen den kürzesten Substring `text`$[l, r]$, welcher die Buchstaben ‘a’, ‘b’, und ‘c’ je mindestens einmal enthält.

- Brute-Force: Alle $\frac{n \cdot (n-1)}{2}$ Substrings durchtesten dauert $\mathcal{O}(n^3)$.
- Idee: Betrachte einen Substring `text`$[l, r]$:
  - Noch nicht alle drei Buchstaben $\rightarrow$ Substring vergrössern.
  - Alle drei Buchstaben enthalten $\rightarrow$ Substring verkleinern.

In-Class-Exercises: Sliding Window

Gegeben sei ein String `text` der Länge n.
Wir suchen den kürzesten Substring `text`$[l, r]$, welcher die Buchstaben ‘a’, ‘b’, und ‘c’ je mindestens einmal enthält.

- Brute-Force: Alle $\frac{n \cdot (n-1)}{2}$ Substrings durchtesten dauert $\mathcal{O}(n^3)$.
- Idee: Betrachte einen Substring `text`$[l, r]$:
  - Noch nicht alle drei Buchstaben $\rightarrow$ Substring vergrössern.
  - Alle drei Buchstaben enthalten $\rightarrow$ Substring verkleinern.
- Sliding Window Approach.

In-Class-Exercises: Sliding Window

Sliding Window Approach:
`https://codeboard.io/projects/79469`

In-Class-Exercises: Sliding Window

Sliding Window Approach:
`https://codeboard.io/projects/79469`

Laufzeit: $\mathcal{O}(n)$.

- In jedem Schritt wird das Sliding Window entweder nach rechts verlängert oder links verkürzt. Es gibt also höchstens $2n$ Schritte.
- Da wir nur eine konstante Anzahl Buchstaben hashen, erfolgen HashMap Operationen in $\mathcal{O}(1)$.

Vergleich mit Aufgabe 7.3

Aufgabe 7.3: Hier suchen wir einen ganz bestimmten Substring “abc”, und nicht nur dessen Buchstaben ‘a’, ‘b’, ‘c’!

- Einfacher, da unser Sliding Window immer dieselbe Länge hat!
- Aber auch schwieriger, weil die Reihenfolge der Buchstaben nun eine Rolle spielt!

Fragen oder Anregungen?