

Datenstrukturen und Algorithmen

Übung 5

FS 2019

Programm von heute

- 1 Feedback letzte Übung
- 2 Wiederholung Theorie
- 3 Programmieraufgabe

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Sei $i \in \mathbb{N}$ die Anzahl eingefügter Elemente und $n_i \in \mathbb{N}$ die Arraygrösse nachdem i eingefügt wurde.

Es gilt

$$n_i = \begin{cases} 1 & \text{falls } i = 1 \text{ [Start]} \\ 2 \cdot n_{i-1} & \text{falls } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array voll]} \\ n_{i-1} & \text{sonst} \end{cases}$$

$$n_i = 2^{\lceil \log_2 i \rceil}$$

i	n_i
1	1
2	2
3	4
4	4
5	8
6	8
...	...

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

¹Nach Aufgabenstellung: $2n$ Initialisierungen + n Kopien + neues Element

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Reale Kosten

$$t_i = \begin{cases} 1 & \text{falls } i = 1 \text{ [Start]} \\ 3n_{i-1} + 1 & \text{falls } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array voll]}^1 \\ 1 & \text{sonst} \end{cases}$$

¹Nach Aufgabenstellung: $2n$ Initialisierungen + n Kopien + neues Element

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Reale Kosten

$$t_i = \begin{cases} 1 & \text{falls } i = 1 \text{ [Start]} \\ 3n_{i-1} + 1 & \text{falls } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array voll]}^1 \\ 1 & \text{sonst} \end{cases}$$

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

¹Nach Aufgabenstellung: $2n$ Initialisierungen + n Kopien + neues Element

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\begin{aligned}\Phi_i &= 6 \cdot \text{Anzahl Elemente in der oberen Hälfte des Arrays} \\ &= 6 \cdot \left(i - \frac{n_i}{2}\right) = 6i - 3n_i\end{aligned}$$

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$\Phi_i = 6 \cdot$ Anzahl Elemente in der oberen Hälfte des Arrays

$$= 6 \cdot \left(i - \frac{n_i}{2}\right) = 6i - 3n_i$$

$$\Phi_i - \Phi_{i-1} = \begin{cases} 6 + 3n_{i-1} - 3 \widehat{n_i}^{2 \cdot n_{i-1}} & \text{falls } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array voll]} \\ 6 & \text{sonst} \end{cases}$$

Amortisierte Analyse: push_back

Strategie: verdoppeln wenn Array voll ist.

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$\begin{aligned} a_i &= t_i + \Phi_i - \Phi_{i-1} \\ &= \begin{cases} 3n_{i-1} + 1 + 6 - 3n_{i-1} & \text{falls } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array voll]} \\ 1 + 6 & \text{sonst} \end{cases} \\ &\leq 7 \quad \text{für alle } i \end{aligned}$$

Amortisierte Analyse: pop_back

Strategie: halbieren wenn Array viertel leer ist.

Amortisierte Analyse: pop_back

Strategie: halbieren wenn Array viertel leer ist.

$$t_i = \begin{cases} 1 & \text{wenn Array mehr als viertel voll ist} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4}n_{i-1} & \text{sonst, danach } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Amortisierte Analyse: pop_back

Strategie: halbieren wenn Array viertel leer ist.

$$t_i = \begin{cases} 1 & \text{wenn Array mehr als viertel voll ist} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4}n_{i-1} & \text{sonst, danach } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Amortisierte Analyse: pop_back

Strategie: halbieren wenn Array viertel leer ist.

$$t_i = \begin{cases} 1 & \text{wenn Array mehr als viertel voll ist} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4}n_{i-1} & \text{sonst, danach } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Sei k_i die Anzahl Elemente im Array im Schritt i

$$\begin{aligned} \Phi_i &= 3 \cdot \text{Anzahl leerer Elemente in der unteren Hälfte des Arrays } (1, \dots, \frac{n_i}{2}) \\ &= 3 \cdot \left(\frac{n_i}{2} - k_i \right) \end{aligned}$$

Amortisierte Analyse: pop_back

Strategie: halbieren wenn Array viertel leer ist.

$$t_i = \begin{cases} 1 & \text{wenn Array mehr als viertel voll ist} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4}n_{i-1} & \text{sonst, danach } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Sei k_i die Anzahl Elemente im Array im Schritt i

$$\begin{aligned} \Phi_i &= 3 \cdot \text{Anzahl leerer Elemente in der unteren Hälfte des Arrays } (1, \dots, \frac{n_i}{2}) \\ &= 3 \cdot \left(\frac{n_i}{2} - k_i \right) \end{aligned}$$

Amortisierte Analyse: pop_back

Strategie: halbieren wenn Array viertel leer ist. Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\Phi_i = 3 \cdot \left(\frac{n_i}{2} - k_i \right)$$

$$\Phi_i - \Phi_{i-1} = \begin{cases} 3 & \text{wenn Array mehr als viertel voll ist} \\ 3 \cdot \left(1 + \frac{n_{i-1}}{4} - \frac{n_{i-1}}{2} \right) & \text{sonst} \end{cases}$$

Amortisierte Analyse: pop_back

Strategie: halbieren wenn Array viertel leer ist. Finde Potentialfunktion so dass amortisierte Kosten konstant sind:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\Phi_i = 3 \cdot \left(\frac{n_i}{2} - k_i \right)$$

$$\Phi_i - \Phi_{i-1} = \begin{cases} 3 & \text{wenn Array mehr als viertel voll ist} \\ 3 \cdot \left(1 + \frac{n_{i-1}}{4} - \frac{n_{i-1}}{2} \right) & \text{sonst} \end{cases}$$

$$\Rightarrow 4 \geq a_i \text{ (in beiden Fällen)}$$

Amortisierte Analyse: pop und push

$$\Phi_i = 6 \cdot \text{Anzahl Elemente obere Hälfte} \\ + 3 \cdot \text{Anzahl leere Elemente untere Hälfte}$$

2. Wiederholung Theorie

Wörterbuch in C++

Assoziativer Container `std::unordered_map<>`

```
// Create an unordered_map of strings that map to strings
std::unordered_map<std::string, std::string> u = {
    {"RED", "#FF0000"}, {"GREEN", "#00FF00"}
};

u["BLUE"] = "#0000FF"; // Add

std::cout << "The HEX of color RED is: " << u["RED"] << "\n";

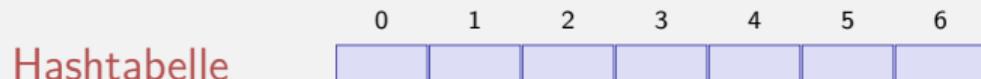
for( const auto& n : u ) // iterate over key-value pairs
    std::cout << n.first << ":" << n.second << "\n";
```

Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12

Direkte Verkettung der Überläufer



Überläufer

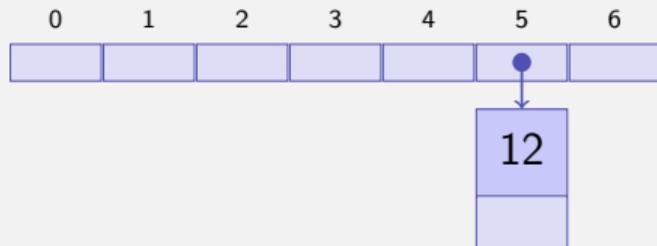
Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12, 55

Direkte Verkettung der Überläufer

Hashtabelle



Überläufer

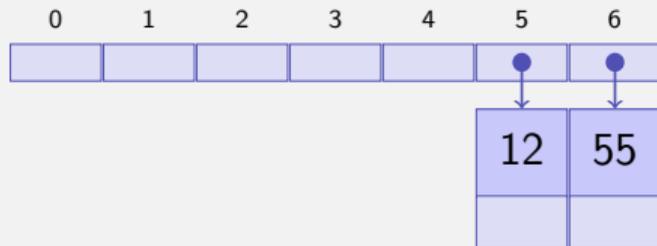
Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12, 55, 5

Direkte Verkettung der Überläufer

Hashtabelle



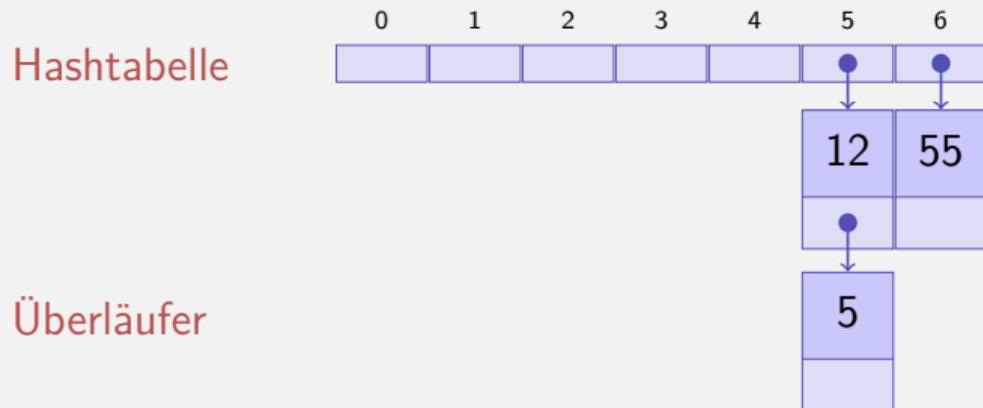
Überläufer

Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12, 55, 5, 15

Direkte Verkettung der Überläufer

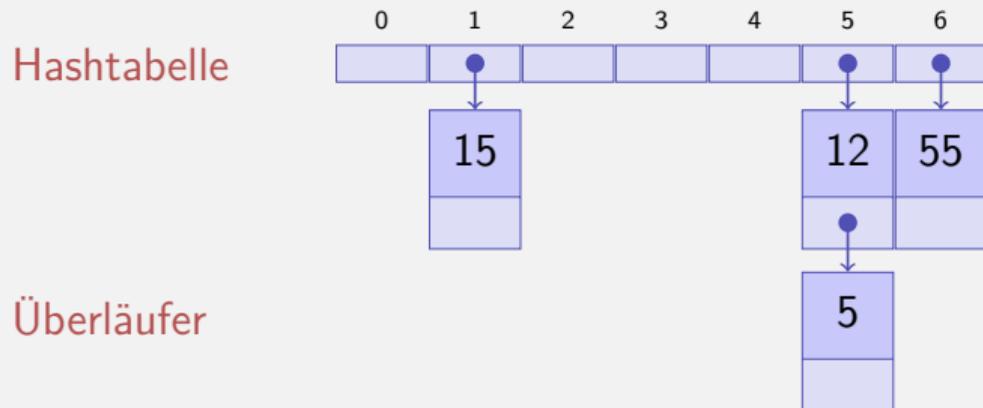


Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12, 55, 5, 15, 2

Direkte Verkettung der Überläufer



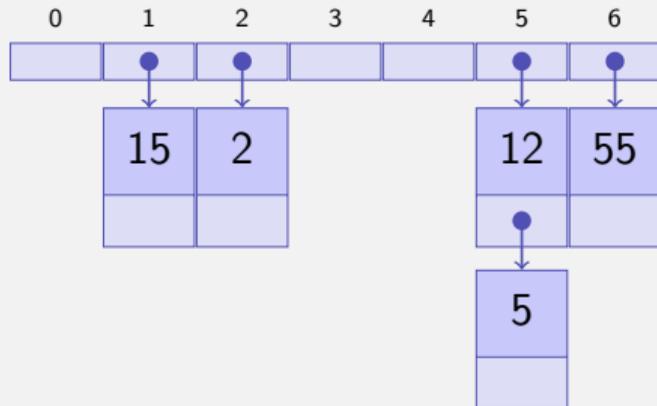
Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12, 55, 5, 15, 2, 19

Direkte Verkettung der Überläufer

Hashtabelle



Überläufer

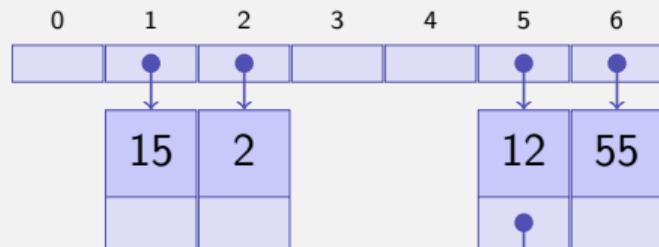
Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

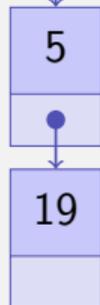
Schlüssel 12, 55, 5, 15, 2, 19, 43

Direkte Verkettung der Überläufer

Hashtabelle



Überläufer

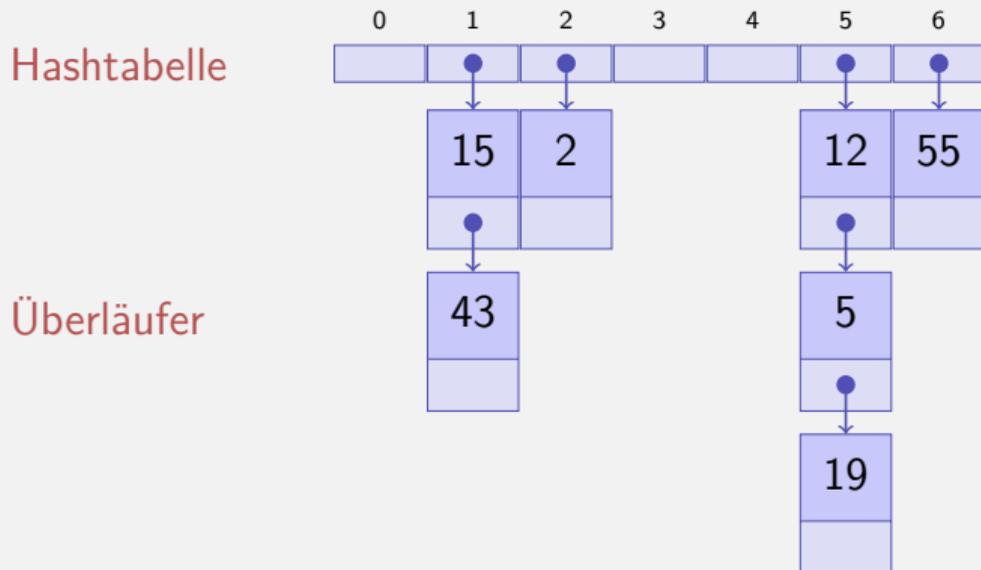


Behandlung von Kollisionen: Verkettung

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12, 55, 5, 15, 2, 19, 43

Direkte Verkettung der Überläufer



Beispiele gebräuchlicher Hashfunktionen

Divisionsmethode

$$h(k) = k \bmod m$$

- m Primzahl, nicht zu nahe² bei Potenzen von 2 oder 10

²Beispiel: sei $m = 2^k - 1$, dann ist $h(k) = h(k \cdot 2^k)$.

Beispiele gebräuchlicher Hashfunktionen

Multiplikationsmethode

$$h(k) = \lfloor (a \cdot k \bmod 2^w) / 2^{w-r} \rfloor \bmod m$$

- $m = 2^r$, w = Grösse des Maschinenworts in Bits.
- Multiplikation addiert k entlang aller Bits von a , Ganzzahldivision mit 2^{w-r} und $\bmod m$ extrahiert die oberen r Bits.
- Als Code geschrieben: `a * k >> (w-r)`
- Guter Wert für a : $\lfloor \frac{\sqrt{5}-1}{2} \cdot 2^w \rfloor$: Integer, der die ersten w Bits des gebrochenen Teils der irrationalen Zahl darstellt.

Illustration

$$\begin{array}{r} \leftarrow w \text{ bits} \rightarrow \\ \begin{array}{r} k \\ \hline 11 \quad 1 \end{array} \quad \begin{array}{l} k \\ a \end{array} \\ \times \\ \hline \begin{array}{r} k \\ + \quad k \\ + \quad k \\ = \end{array} \\ \begin{array}{r} \leftarrow r \text{ bits} \rightarrow \\ \hline \end{array} \\ \gg (w - r) \quad \begin{array}{r} 0 \quad \leftarrow r \text{ bits} \rightarrow \end{array} \end{array}$$

Offene Adressierung

Speichere die Überläufer direkt in der Hashtabelle mit einer *Sondierfunktion* $s : \mathcal{K} \times \{0, 1, \dots, m - 1\} \rightarrow \{0, 1, \dots, m - 1\}$
Tabellenposition des Schlüssels entlang der *Sondierungsfolge*

$$S(k) := (s(k, 0), s(k, 1), \dots, s(k, m - 1)) \quad \text{mod } m$$

Sondierungsfolge muss für jedes $k \in \mathcal{K}$ eine Permutation sein von $\{0, 1, \dots, m - 1\}$

Algorithmen zum Open Addressing

- **search**(k) Durchlaufe Tabelleneinträge gemäss $S(k)$. Wird k gefunden, gib **true** zurück. Ist die Sondierungsfolge zu Ende oder eine leere Position erreicht, gib **false** zurück.
- **insert**(k) Suche k in der Tabelle gemäss $S(k)$. Ist k nicht vorhanden, füge k an die erste freie Position in der Sondierungsfolge ein.³
- **delete**(k) Suche k in der Tabelle gemäss $S(k)$. Wenn k gefunden, markiere Position von k mit einem **deleted**-flag.

³Als frei gilt auch eine nicht leere Position mit **deleted** flag.

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \pmod{m}$$

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \pmod{m}$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod{m}$.

Schlüssel 12

0	1	2	3	4	5	6

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \pmod{m}$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod{m}$.

Schlüssel 12 , 55

0	1	2	3	4	5	6
					12	

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \pmod{m}$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod{m}$.

Schlüssel 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \text{ mod } m$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \text{ mod } m$.

Schlüssel 12 , 55 , 5 , 15

0	1	2	3	4	5	6
5					12	55

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \pmod{m}$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod{m}$.

Schlüssel 12, 55, 5, 15, 2

0	1	2	3	4	5	6
5	15				12	55

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \text{ mod } m$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \text{ mod } m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2			12	55

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \text{ mod } m$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \text{ mod } m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2	19		12	55

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod{m}$$

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod m$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod m$.

Schlüssel 12

0	1	2	3	4	5	6

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod{m}$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod{m}$.

Schlüssel 12 , 55

0	1	2	3	4	5	6
					12	

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod{m}$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod{m}$.

Schlüssel 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod m$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod m$.

Schlüssel 12, 55, 5, 15

0	1	2	3	4	5	6
				5	12	55

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod m$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod m$.

Schlüssel 12, 55, 5, 15, 2

0	1	2	3	4	5	6
	15			5	12	55

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod{m}$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod{m}$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
	15	2		5	12	55

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$

$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \pmod m$$

Beispiel $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod m$.

Schlüssel 12, 55, 5, 15, 2, 19

0	1	2	3	4	5	6
19	15	2		5	12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Beispiel:

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod 7$, $h'(k) = 1 + k \pmod 5$.

Schlüssel 12

0	1	2	3	4	5	6

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Beispiel:

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod 7$, $h'(k) = 1 + k \pmod 5$.

Schlüssel 12 , 55

0	1	2	3	4	5	6
					12	

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Beispiel:

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod 7$, $h'(k) = 1 + k \pmod 5$.

Schlüssel 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Beispiel:

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod 7$, $h'(k) = 1 + k \pmod 5$.

Schlüssel 12 , 55 , 5 , 15

0	1	2	3	4	5	6
5					12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Beispiel:

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod 7$, $h'(k) = 1 + k \pmod 5$.

Schlüssel 12 , 55 , 5 , 15 , 2

0	1	2	3	4	5	6
5	15				12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Beispiel:

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod 7$, $h'(k) = 1 + k \pmod 5$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2			12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.
 $S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m - 1)h'(k)) \pmod m$

Beispiel:

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \pmod 7$, $h'(k) = 1 + k \pmod 5$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2	19		12	55

3. Programmieraufgabe

Finden eines Sub-Arrays

- Gegeben: Zwei Felder $A = (a_0, \dots, a_{n-1})$ and $B = (b_0, \dots, b_{k-1})$ mit natürlichen Zahlen
- Aufgabe: Finde Position von B in A .

Finden eines Sub-Arrays

- Gegeben: Zwei Felder $A = (a_0, \dots, a_{n-1})$ and $B = (b_0, \dots, b_{k-1})$ mit natürlichen Zahlen
- Aufgabe: Finde Position von B in A .
- Naiv: Gehe durch A , vergleiche die k folgenden Einträge mit B

Finden eines Sub-Arrays

- Gegeben: Zwei Felder $A = (a_0, \dots, a_{n-1})$ and $B = (b_0, \dots, b_{k-1})$ mit natürlichen Zahlen
- Aufgabe: Finde Position von B in A .
- Naiv: Gehe durch A , vergleiche die k folgenden Einträge mit B
 - $O(nk)$ Vergleiche

Finden eines Sub-Arrays

- Gegeben: Zwei Felder $A = (a_0, \dots, a_{n-1})$ and $B = (b_0, \dots, b_{k-1})$ mit natürlichen Zahlen
- Aufgabe: Finde Position von B in A .
- Naiv: Gehe durch A , vergleiche die k folgenden Einträge mit B
 - $O(nk)$ Vergleiche
- Lösung mit Hashing: Berechne Hash $h(B)$ und vergleiche mit $h((a_i, a_{i+1}, \dots, a_{i+k-1}))$.
- Vermeide die Neuberechnung von $h((a_i, a_{i+1}, \dots, a_{i+k-1}))$ für jedes $i \implies O(n)$ erwartet

Sliding Window Hash

- Mögliche Hash-Funktion: Summe aller Elemente:
 - Kann einfach aktualisiert werden: a_i abziehen und a_{i+k} hinzufügen.
 - Aber: schlechte Hash-Funktion

Sliding Window Hash

- Mögliche Hash-Funktion: Summe aller Elemente:
 - Kann einfach aktualisiert werden: a_i abziehen und a_{i+k} hinzufügen.
 - Aber: schlechte Hash-Funktion
- Besser:

$$H_{c,m}((a_i, \dots, a_{i+k-1})) = \left(\sum_{j=0}^{k-1} a_{i+j} \cdot c^{k-j-1} \right) \bmod m$$

- $c = 1021$ Primzahl
- $m = 2^{15}$ `int`, keine Überläufe bei Berechnungen

Sliding Window Hash

```
template<typename It1, typename It2>
It1 findOccurrence(const It1 from, const It1 to,
                  const It2 begin, const It2 end)
{
    const unsigned k = end - begin;
    const unsigned M = 32768;
    const unsigned C = 1021;

    // your code here
    // ...
}
```

Sliding Window Hash

```
// elements can be compared using std::equal:  
if(std::equal(window_left, window_right, begin, end))  
    return current;  
  
// if no occurrence is found return end of array  
return to;  
}
```

Sliding Window Hash

Gehen Sie sicher, dass

- der Algorithmus c^k nur einmal berechnet,
- alle Werte modulo m berechnet werden, um Überläufe zu vermeiden (verwenden Sie die Rechenregeln für Kongruenzen), und
- alle Werte positiv sind (z.B. durch Addition von Vielfachen von m).

Fragen?