Hashfunktionen

Alexander Krahl

21.06.2018

Hashfunktionen

Alexander Krahl

21.06.2018

Hashliste

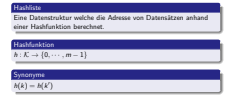
Eine Datenstruktur welche die Adresse von Datensätzen anhand einer Hashfunktion berechnet.

Hashfunktion

$h : \mathcal{K} \to \{0, \cdots, m-1\}$

Synonyme

$h(k) = h(k')$



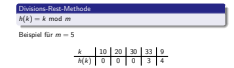
Warum? -> effiziente Datenstrukturen, verschlüsselung von Informationen
Abbildung von variabler länge auf feste länge
Hashfunktionen sind surjektiv
$k \in \mathcal{K}$

Divisions-Rest-Methode

$h(k) = k \bmod m$

Beispiel für $m = 5$

k	10	20	30	33	9
$h(k)$	0	0	0	3	4



externe Kollisionsauflösung

Speichern der Kollisionen in einer verketten Liste.

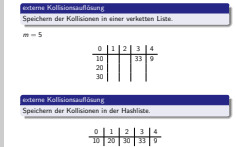
$m = 5$

0	1	2	3	4
10			33	9
20				
30				

externe Kollisionsauflösung

Speichern der Kollisionen in der Hashliste.

0	1	2	3	4
10	20	30	33	9



Hier lineares sondieren wie in der Vorlesung

k	10	20	30	33	9
$h(k)$	0	0	0	3	4

Sondierungsfunktion

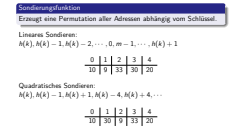
Erzeugt eine Permutation aller Adressen abhängig vom Schlüssel.

Lineares Sondieren:
$h(k), h(k)-1, h(k)-2, \cdots, 0, m-1, \cdots, h(k)+1$

0	1	2	3	4
10	9	33	30	20

Quadratisches Sondieren:
$h(k), h(k)-1, h(k)+1, h(k)-4, h(k)+4, \cdots$

0	1	2	3	4
10	30	9	33	20



Lineares Sondieren: 6 Kollisionen
Quadratisches Sondieren: 2+viele Kollisionen

k	10	20	30	33	9
$h(k)$	0	0	0	3	4

umso voller umso mehr kollisionen
Idealerweise Suchen in O(1)
Umso voller umso mehr Kollisionen->umso langsamer->mehr speicher

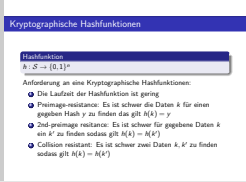
Kryptographische Hashfunktionen

Hashfunktion

$h : \mathcal{S} \rightarrow \{0,1\}^n$

Anforderung an eine Kryptographische Hashfunktionen:

1. Die Laufzeit der Hashfunktion ist gering
2. Preimage-resistance: Es ist schwer die Daten k für einen gegeben Hash y zu finden das gilt $h(k) = y$
3. 2nd-preimage resitance: Es ist schwer für gegebene Daten k ein k' zu finden sodass gilt $h(k) = h(k')$
4. Collision resistant: Es ist schwer zwei Daten k, k' zu finden sodass gilt $h(k) = h(k')$



Schwer hier->hohe laufzeit
Widerspricht 1) -> abwiegen

Speichern von Passwörtern in einer Datenbank mithilfe einer Hashfunktion und eines Saltes.

$$p_{out} = h(p_{in} + salt)$$

Speichern von Passwörtern in einer Datenbank mithilfe einer Hashfunktion und eines Saltes.

$p_{out} = h(p_{in} + salt)$

1. Ermöglichen Suchen in konstanter Zeit
2. Ermöglichen sicheres Speichern von Passwörtern

Vielen Dank fürs Zuhören

Ermöglichen Suchen in konstanter Zeit
Ermöglichen sicheres Speichern von Passwörtern
Vielen Dank fürs Zuhören

Besonders im Kryptograpischen bereich weiterentwicklung da wettrüsten