

# Hashfunktionen

ALEXANDER KRAHL

HTWK Leipzig

alexander.krahl@stud.htwk-leipzig.de

## Zusammenfassung

*Hashfunktionen besitzen einen breiten Anwendungsbereich. In diesem Paper gehe ich auf die Nutzung von Hashfunktionen zum effizienten Durchsuchen von Daten mithilfe einer Hashliste ein. Außerdem betrachte ich die Nutzung von Hashfunktionen im Kryptographischen Bereich zur Signierung von Nachrichten und Speichern von Passwörtern. Ich begründe den Einsatz von Hashfunktionen in diesen Bereichen anhand ihrer Eigenschaften.*

## I. EINFÜHRUNG

**Hashfunktionen** sind eine surjektive mathematische Funktionsklasse. Injektivität ist von der genutzten Hashfunktion abhängig. Aufgrund dieser Eigenschaften eignen sich Hashfunktionen zur Generierung von Schlüsseln oder einer ein-weg-Verschlüsselung.

## II. HASHLISTEN

Bei **Hashlisten** handelt es sich um eine Datenstruktur, in welcher die Speicheradresse von Datensätzen durch eine Hashfunktion berechnet wird. Die Hashfunktion ist für diesen Fall definiert als

$$h : \mathcal{K} \rightarrow \{0, \dots, m-1\} [4, S.191]$$

Wobei  $\mathcal{K}$  die Menge aller Schlüssel ist und  $m$  die Länge des zugrunde liegenden Arrays.

Ziel ist es, Suchoperationen in  $\theta(1)$  durchzuführen.[1] Hashlisten sind nicht geordnet, das Einfügen und Löschen von Elementen erfordert die Suche nach dem Element, da jedes Element einzigartig sein muss. Hashlisten sind geeignet für Verwendungen in einer Datenmenge, hauptsächlich welche durchsucht wird. Schlüssel  $k, k'$  sind **Synonyme**, wenn gilt  $h(k) = h(k')$ . Wenn zwei synonyme Elemente in die gleiche Hashliste eingefügt werden, kommt es zu **Kollisionen**, beide Elemente würden an der gleichen Stelle gespeichert. Kollisionen

müssen behandelt werden, da es sonst zu Datenverlust durch Überschreiben kommen würde. Aus diesem Grund sollten Hashfunktionen zur Nutzung für eine Hashliste möglichst injektiv sein, um Kollisionen zu vermeiden.

Hashlisten sind geeignet für Verwendungen, in denen eine Datenmenge hauptsächlich durchsucht wird. Alle Operationen sind im Worst Case  $\mathcal{O}(n)$ . Dieser Fall tritt ein, wenn alle Elemente den gleichen Hash zugeordnet bekommen, also alle kollidieren. Die Suche ist jedoch im Average case  $\theta(1)$  für ein  $m$ , welches groß genug ist.[4, S. 192][1]

Es sind Hashfunktionen gefordert, welche wenige Kollisionen erzeugen sowie ein Verfahren, um diese effizient aufzulösen. Die Hashfunktion sollte Eingangsdaten gleichmäßig über das Feld verteilen. Dies verhindert die **Häufung** von Elementen, welche die Rate an Kollisionen erhöht.[1]

Eine Methode zum Erzeugen einer Hashfunktion ist die Divisions-Rest-Methode.

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

$m$  sollte nicht  $r^i \pm j$ , für kleine positive Ganzzahlen  $i, j$  sein. Ein Wert für  $m$  mit geringer Kollisionsrate ist eine Primzahl und wird nicht durch  $r^i \pm j$  geteilt. Durch diese Wahl wird die Anzahl der Kollisionen minimiert.[4, 1, S. 193f] Für den Fall  $|K| \leq m, K \subset \mathcal{K}$  kann eine **perfekte Hashfunktion** durch injektive Abbildung gebildet werden. Da diese keine Kollisionen erzeugt, ist die Suche immer in  $\theta(1)$

möglich.[4, S. 194f.]

Eine Möglichkeit zur Behandlung von Kollisionen ist das **indirekte** Verfahren. Bei diesem werden Synonyme Elemente in einer Liste abgelegt. ein Elemente in der Hashliste zeigt auf die Liste seiner Synonyme. Alternativ dazu ist das **direkte** Verfahrenen. Bei diesem hält die Hashliste nur Zeiger auf die Kollisionslisten. Dies beeinträchtigt die Laufzeit der Operationen negativ da im Worst Case alle Elemente einer Liste durchsucht werden müssen.[4, S. 199f]

Eine andere Möglichkeit wird als **offenes Hashverfahren** bezeichnet. Im Fall einer Kollision wird für das Element eine andere Position in der Hashliste mit Hilfe einer Sondierungs-funktion berechnet. Eine **Sondierungs-funktion** erzeugt eine Permutation aller Hashadressen welche **Sondierungsfolge** genannt wird. Entfernte Elemente werde als entfernt markiert, bei der Suche werden sie als vorhandene Elemente betrachtet, bei dem Einfügen werden sie als nicht vorhandene Elemente angesehen. Dies ist notwendig da das löschen eines Elements die Sondierungsfolge unterbricht indem ein freier Speicherplatz vorhanden ist. Somit wahren alle Synonyme welche nach dem gelöschten Element eingefügt wurden nicht mehr Adressierbar.[4, S. 203f.]

Verschiedene Sondierungs-funktionen erzeugen verschiedene Sondierungsfolgen:

1. Lineares Sondieren:

$$h(k), h(k)-1, h(k)-2, \dots, 0, m-1, \dots, h(k)+1$$

2. Quadratisches Sondieren:

$$h(k), h(k)-1, h(k)+1, h(k)-4, h(k)+4, \dots$$

[4, S. 205f.]

Jedoch führen beide Möglichkeiten zu einer Häufung. Umso mehre Plätze in der Hashliste belegt sind umso länger dauerten auch alle Operationen da die Sondierungsfolge welche durchlaufen werden muss länger ist.[4, S. 208] Eine effektiver Methode des Sondierens ist das random Probing. Beim random Probing wird abhängig von  $k$  ein zufälliger Platz gewählt. Da für Synonyme nicht die gleichen Sondierungsfolgen generiert werden wird vermutet das die Anzahl der Kollisionen minimal sind.[4,

S. 208f.]

### III. KRYPTOGRAPHISCHE HASHFUNKTIONEN

Hashfunktionen werden sie im Bereich der Kryptographie zur Authentifikation oder als Einwegfunktion zur Verschlüsselung von Daten eingesetzt. Aus diesem Grund sollten kryptographische Hashfunktionen im Gegensatz zu Hashfunktionen welche für eine Hashliste genutzt werden nicht injektiv sein. Kryptographische Hashfunktion haben die Form

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

[2, S. 48] Die Daten  $\mathcal{S}$  werden in eine Bitfolge fester Länge umgewandelt. Die Funktion  $h$  sollte folgende Eigenschaften besitzen:

1. Die Laufzeit der Hashfunktion ist gering
2. Preimage-resistance: Es ist schwer die Daten  $k$  für einen gegebenen Hash  $y$  zu finden das gilt  $h(k) = y$
3. 2nd-preimage resistance: 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  $h(k) = h(k')$

(Vgl.[2, S. 48]) Schwer bedeutet hier das eine Lösung es Problems möglich ist, jedoch diese durch die Anzahl der Berechnungen nicht in absehbarer Zeit berechenbar ist. Aus 2. folgt das eine Veränderung der Daten eine Änderung des Hashes mit sich zieht. Collision resistance zieht 2nd-preimage-resistance mit sich. Diese Eigenschaften sind notwendig im die Sicherheit einer Kryptographischen Hashfunktion zu gewährleisten.

Kryptographische Hashfunktionen werden genutzt um verschlüsselte Kommunikation abzusichern. Kommunizieren zwei Personen Bob und Alice mithilfe einer Verschlüsselung wird durch die Operation

$$m_e \oplus h(k)$$

die Nachricht signiert. Wobei  $m_e$  die verschlüsselte Nachricht,  $h$  eine Kryptographische Hashfunktion,  $k$  ein beiden bekannter Schlüssel ist.

Aus der Definition der XOR Operation folgt das eine erneute Anwendung der Operation mit gleichem Operanden, hier der Hashfunktion, die ursprünglichen Eingangsdaten  $m_e$  wiederhergestellt werden können. Dadurch ist die Kommunikation nicht anfällig für plaintext-chipertext Attacken, bei dieser kann ein Angreifer aus einem Stück unverschlüsselten(plain) und verschlüsselten(cipher) Text den geheimen Schlüssel welcher zur Verschlüsselung genutzt wurde bekommen. Durch die Signierung ist der geheime Schlüssel  $k$  sicher auch wenn ein Angreifer teile der verschlüsselten und unverschlüsselten Nachricht besitzt.[2, S.48,49]

Aus den Eigenschaften einer Hashfunktion folgt die Unumkehrbarkeit dieser. Deshalb werden Hashfunktionen zur Speicherung von Passwörtern genutzt. Indem nur der gehashte Werte das Passworts in der Datenbank gespeichert wird ist dieses sicher im Fall eines leaks.[2, S. 53 S.231][3]

#### IV. FAZIT

Hashfunktionen ermöglichen die effizient Durchsuchung einer Datenmenge, sowie die Sicherheit im Internet durch Signierung von Nachrichten und zur Speicherung von Passwörtern. Auch in Zukunft werden besonders im Bereich der Kryptographie neue Hashfunktionen benötigt, welche alte verwundbare ablösen oder noch auch auf moderner Hardware auch noch sicher sind.

#### LITERATUR

- [1] R. K. Bhullar, L. Pawar, V. Kumar & Anjali. A novel prime numbers based hashing technique for minimizing collisions. In: *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, S. 522–527. 2016.
- [2] B. Gilbert, F. Benjamin, K. Martin & R. Gerhard. *A Course in Mathematical Cryptography*. Berlin/Boston: De Gruyter, 2015.
- [3] R. Sanek. Password Storage in Databases: Best Practices. In: C. Barr, C. Dobson, A. Hill, P. Michael, R. Pipan & E. Rankins

(Hrsg.), *Auburn University Journal of Undergraduate Scholarship*, Band 4, S. 33–39. Office of Undergraduate Research 206 Carter Hall Auburn University, Alabama 36849 USA: Auburn University, 2015.

- [4] O. Thomas & W. Peter. *Algorithmen und Datenstrukturen*. Berlin: Springer, 6 Aufl., 2017.