Hashfunktionen

Zusammenfassung

Hashfunktionen besitzen eine breiten Anwendungsbereich. In diesem Paper gehe ich auf die Nutzung von Hashfunktionen zum effizienten Suchen 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 Eingenschaft.

I. Einführung

Hashfunktionen sind eine mathematische Funktionsklasse welche surjektiv sind. Injektivität ist von der genutzten Hashfunktion abhängig. Aus Grund diese Eigenschaften eignen sich Hashfunktion zur Generierung von Schlüsseln oder einer einweg Verschlüsselung.

II. Hashlisten

Bei **Hashlisten** handelt es sich um eine Datenstrukturen in welcher die Speicheraddresse von Datensätzen durch eine Hashfunktion berechnet wird. Die Hashfunktion ist für diesen fall definiert als

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

Wobei $\mathcal{K}$ die Menge aller Schlüssel ist und m die Länge des zugrunde liegen Arrays.
Ziel ist 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 denen eine Datenmenge hauptsächlich durchsucht wird. Schlüssel k, k' sind **Synonyme** wenn gilt $h(k) = h(k')$. Wenn zwei synonyme Elemente in die gleiche Hashliste einfügt werden kommt es zu **Kollisionen**, beide Elemente würden an der gleichen Stelle gespeichert. Kollisionen müssen behandelt werden, sonst kann es zu Datenverlust kommen. Aus diesem Grund sollten Hashfunktionen zur Nutzung für eine Hashliste möglichst injektiv sein.
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 eine 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 denn 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 abge-

legt. 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 Sondierungsfunktion berechnet. Eine **Sondierungsfunktion** erzeugt eine Permutation aller Hashaddressen 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 währen alle Synonyme welche nach dem gelöschten Element eingefügt wurden nicht mehr Adressierbar[4, S. 203f.].
Verschiedene Sondierungsfunktionen erzeugen verschiedene Sondierungsfolgen:

1. Lineares Sondieren:

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

2. Quadratisches Sondieren:

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

[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 die 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 in einer Hashliste genutzt werden nicht injektiv sein. Kryptographische Hashfunktion haben die Form

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

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

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. Collison resistant: Es ist schwer zwei Daten k, k' zu finden sodass gilt $h(k) = h(k')$

(Vgl.[2, S. 48]) Schwer bedeutet in diesem fall das eine Lösung es Problems möglich ist, jedoch diese durch die Anzahl der Berechnungen nicht in absehbarer Zeit berechenbar it. Aus 2. folgt das eine Veränderung der Daten eine Änderung des Hashes mit sich zieht. Collision resistance zieht 2nd-preimage-resistance mit sich. Diese Eingenschaften 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 durch 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 Key welcher zur Verschlüsselung genutzt wurde bekommen. Durch die Signierung ist der geheime Key k sicher auch wenn ein Angreifer teile der verschlüsselten und unverschlüsselten Nachricht besitzt[2, S.48,49]. Weiterhin werden Kryptographische Hashfunktionen zur Authentifizierung von Nachrichten sowie zum speichern von Passwörtern genutzt. [2, S. 53 S.231][3]

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 Cater Hall Auburn University, Alabama 36849 USA: Auburn University, 2015.

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