Hashing

Weiterführende Literatur:

- Schneider, *Taschenbuch der Informatik*, Kapitel 6.3.3 Streutabellen (Hashing), Seite 188-189
- *Algorithmen und Datenstrukturen: Tafelübung 10, WS 2018/19*, Seite 22-39
- Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 419-439
- *Wikipedia-Artikel „Hashtabelle“*

Grundlagen

Das Ziel von Hashing ist einerseits einen *extrem großen Schlüsselraum* auf einen vernünftig *kleinen Bereich von ganzen Zahlen* abzubilden und andererseits dass *zwei Schlüssel auf die selbe Zahl* abgebildet werden, soll möglichst *unwahrscheinlich* sein.[1]

extrem großen Schlüsselraum
kleinen Bereich von ganzen Zahlen
zwei Schlüssel auf die selbe Zahl
unwahrscheinlich

Daten werden in einer *Streu(wert)tabelle* (hash table) abgelegt. Aufgrund des wahlfreien Zugriffs eignen sich Felder zum Abspeichern der Daten. Eine *Hashfunktion h* bildet ein Datenelement auf einen *Hashwert* ab. Das Datenelement benötigt dazu einen *Schlüssel* (key), der das Element eindeutig identifiziert. Der Hashwert wird als Index in dem Feld verwendet. Das Datenelement wird im entsprechenden *Bucket* der Tabelle gespeichert. Bei der *Suche* nach einem Element mit *bekanntem Schlüssel* wird der Index mittels der Hashfunktion bestimmt. Dies geschieht mit *konstantem Aufwand*. Der Aufwand des Nachschlagens an entsprechender Stelle ist abhängig von der *Organisationsform*.[2]

Streu(wert)tabelle

Hashfunktion
Hashwert
Schlüssel

Bucket
Suche
bekanntem Schlüssel
konstantem Aufwand

Kollisionen

Da Hashfunktionen im Allgemeinen *nicht eindeutig* (*injektiv*) sind, können zwei unterschiedliche Schlüssel zum selben Hash-Wert, also zum selben Feld in der Tabelle, führen. Dieses Ereignis wird als *Kollision* bezeichnet. In diesem Fall muss die Hashtabelle mehrere Werte in demselben Bucket aufnehmen.

nicht eindeutig (injektiv)

Kollision

Eine Kollision benötigt bei der Suche eine spezielle Behandlung durch das Verfahren: Zunächst wird aus einem Suchschlüssel wieder ein Hashwert berechnet, der den Bucket des gesuchten Datenobjektes bestimmt; dann muss noch durch direkten Vergleich des Suchschlüssels mit den Objekten im Bucket das gesuchte Ziel bestimmt werden.

Zur Behandlung von Kollisionen werden kollidierte Daten nach einer *Ausweichstrategie in alternativen Feldern* oder in einer *Liste* gespeichert. Schlimmstenfalls können Kollisionen zu einer *Entartung der Hashtabelle* führen, wenn wenige Hashwerte sehr vielen Objekten zugewiesen wurden, während andere Hashwerte unbenutzt bleiben.[3]

Ausweichstrategie in alternativen Feldern
Liste
Entartung der Hashtabelle

[1] Seite 4 `https://moves.rwth-aachen.de/wp-content/uploads/SS15/dsal/lec13.pdf`
[2] *Algorithmen und Datenstrukturen: Tafelübung 10, WS 2018/19*, Seite 25.
[3] *Wikipedia-Artikel „Hashtabelle“*.

Kollisionsauflösungsstrategien

Um das Kollisions-Problem zu handhaben, gibt es diverse Kollisionsauflösungsstrategien.

geschlossenes Hashing mit offener Adressierung

andere freie Stelle genommen

Wenn dabei ein Eintrag an einer schon belegten Stelle in der Tabelle abgelegt werden soll, wird stattdessen eine *andere freie Stelle genommen*. Häufig werden drei Varianten unterschieden:

lineares Sondieren

konstantes Intervall

es wird um ein *konstantes Intervall* verschoben nach einer freien Stelle gesucht. Meistens wird die Intervallgröße auf 1 festgelegt.

quadratisches Sondieren

Intervall quadriert

Nach jedem erfolglosen Suchschritt wird das *Intervall quadriert*.

Beispiele

- **nach Foliensatz der TU Braunschweig** [4]

Formel

$h(k,i) := h'(k) + (-1)^{i+1} \cdot \left\lfloor \frac{i+1}{2} \right\rfloor^2 \mod m$

$k, k+1^2, k-1^2, k+2^2, k-2^2, \ldots k+(\frac{m-1}{2})^2, k-(\frac{m-1}{2})^2 \mod m$

Werte

$m = 19$, d. h. das Feld (die Tabelle) hat die Index-Nummern 0 bis 18.
$k = h(x) = 7$

Sondierungsfolgen

i	Rechnung	Ergebnis	Index in der Tabelle
0	$7+0^2$	7	7
1	$7+1^2$	8	8
1	$7-1^2$	6	6
2	$7+2^2$	11	11
2	$7-2^2$	3	2
3	$7+3^2 = 7+9$	16	16
3	$7-3^2 = 7-9$	-2	17 $(19-2=10)$ oder $(0 \to 0, -1 \to 18, -2 \to 17)$
4	$7+4^2 = 7+16$	23	4 $(23-19=4)$ oder $(19 \to 0, 20 \to 1, 21 \to 2, 22 \to 3, 23 \to 4)$
4	$7-4^2 = 7-16$	-9	10 $(19-9=10)$ oder $(0 \to 0, -1 \to 18, -2 \to 17, \cdots, -9 \to 10)$
5	$7+5^2 = 7+25$	32	13 $(32-19=13)$
5	$7-5^2 = 7-25$	-18	1 $(19-18=1)$

[4] Seite 25 `https://www.ibr.cs.tu-bs.de/courses/ws0708/aud/skript/hash.np.pdf`

- **nach Foliensatz der RWTH Aachen** [5]

  $h'(k) = k \mod 11$

  $h(k,i) = (h'(k) + i + 3i^2) \mod 11$

  $h'(17) = 17 \mod 11 = 6$

  Sondierungsfolgen

  $h(17,0) = (17 + 0 + 3 \cdot 0^2) \mod 11 = 6$

  $h(17,1) = (17 + 1 + 3 \cdot 1^2) \mod 11 = 21 \mod 11 = 10$

  $h(17,2) = (17 + 2 + 3 \cdot 2^2) \mod 11 = 31 \mod 11 = 31 - 2 \cdot 11 = 9$

  $h(17,3) = (17 + 3 + 3 \cdot 3^2) \mod 11 = 47 \mod 11 = 47 - 4 \cdot 11 = 3$

doppeltes Hashen

eine *weitere Hash-Funktion* liefert das Intervall.

weitere Hash-Funktion

offenes Hashing mit geschlossener Adressierung

Anstelle der gesuchten Daten enthält die Hashtabelle hier *Behälter* (englisch *Buckets*), die alle Daten mit gleichem Hash-Wert aufnehmen. Es müssen die *Elemente im Behälter durchsucht werden*. Oft wird die Verkettung durch eine lineare Liste pro Behälter realisiert.[6]

Behälter

Buckets

Elemente im Behälter durchsucht werden

Die Divisionsrestmethode[7]

Die Divisionsrestmethode - auch Modulo genannt liefert eine Hashfunktion. Die Funktion lautet: $h(k) = k \bmod m$. m ist die Größe der Hashtabelle. Die Hash-Funktion kann sehr schnell berechnet werden. Die Wahl der Tabellengröße m beeinflusst die Kollisionswahrscheinlichkeit der Funktionswerte von h. Für praxisrelevante Anwendungen liefert die Wahl einer Primzahl für m.

Belegungsfaktor[8]

$$\text{Belegungsfaktor} = \frac{\text{Anzahl tatsächlich eingetragener Schlüssel}}{\text{Anzahl Hashwerte}}$$

Literatur

[1] *Algorithmen und Datenstrukturen: Tafelübung 10, WS 2018/19*. `https://www.studon.fau.de/file2619757_download.html`. FAU: Lehrstuhl für Informatik 2 (Programmiersysteme).

[5] Seite 19 `https://moves.rwth-aachen.de/wp-content/uploads/SS15/dsal/lec13.pdf`

[6] *Wikipedia-Artikel „Hashtabelle“.*

[7] *Wikipedia-Artikel „Divisionsrestmethode“.*

[8] *Algorithmen und Datenstrukturen: Tafelübung 10, WS 2018/19*, Seite 29.

[2] Gunter Saake und Kai-Uwe Sattler. *Algorithmen und Datenstrukturen. Eine Einführung in Java*. 2014.

[3] Uwe Schneider. *Taschenbuch der Informatik*. 7. Aufl. Hanser, 2012. ISBN: 9783446426382.

[4] *Wikipedia-Artikel „Divisionsrestmethode“*. `https://de.wikipedia.org/wiki/Divisionsrestmethode`.

[5] *Wikipedia-Artikel „Hashtabelle“*. `https://de.wikipedia.org/wiki/Hashtabelle`.