

Hashing

Bei Hashing (zerhacken) handelt es sich um eine Methode für die direkte Bezugnahme auf Datensätze in einer Tabelle. Dies erfolgt mit Hilfe einer Funktion, die direkt aus dem jeweiligen Schlüssel die Adresse des zugehörigen Datensatzes errechnet.

Wenn man weiß, daß die Schlüssel nur ganze Zahlen von 1 bis N sind, kann beispielsweise der Datensatz mit dem Schlüssel 5 an der Tabellenposition 5 gespeichert werden. Hashing ist eine Verallgemeinerung dieser Methode, wenn die speziellen Kenntnisse über die Schlüsselwerte nicht vorhanden sind.

Bei der Benutzung von Hashing sind zwei Punkte zu beachten:

- 1.) Wahl einer geeigneten Hash-Funktion zur Berechnung der Tabellenadresse.
- 2.) Da die Anzahl der verfügbaren Speicherplätze in der Regel geringer ist, als die der möglichen Schlüssel, muß eine Funktion gewählt werden, die eine Doppelbelegung einer Adresse zuläßt. Die Art der Behandlung derartiger Kollisionen beeinflußt die Zugriffsdauer wesentlich.

Hashing ist ein guter Kompromiß zwischen Zeit- und Platzbedarf. Gäbe es keine Beschränkung des Speichers, könnte man jede beliebige Suche mit nur einem Zugriff auf den Speicher ausführen, wenn man den Schlüssel als Speicheradresse verwendet. Wenn es keine zeitliche Begrenzung gäbe, könnte man mit einem Minimum an Speicherplatz auskommen, indem man ein sequentielles Suchverfahren verwendet.

Hashing ermöglicht es, mit einem vertretbaren Maß an Speicherplatz als auch an Zeit auszukommen. Eine effiziente Nutzung der verfügbaren Speicherkapazität und ein schneller Zugriff auf den Speicher sind die vorrangigen Ziele jeder Hashing-Methode.

1. Berechnung von Hash-Funktionen

Benötigt wird eine Funktion, die Schlüssel (für gewöhnlich ganze Zahlen oder kurze Zeichenfolgen) in ganze Zahlen aus dem Intervall $[0 \dots M-1]$ transformiert, wobei M die Anzahl von Datensätzen ist, die in dem verfügbaren Speicher untergebracht werden kann.

Hierfür werden in der Literatur eine Vielzahl von Hash-Algorithmen vorgeschlagen. Eine der bekanntesten Funktionen ist das Divisionsrestverfahren.

Hier ist für M eine günstige Primzahl zu wählen, so daß für den beliebigen Schlüssel k der Wert des Schlüssels nach der Formel $h(k) = k \bmod M$ berechnet wird. Dies ist ein sehr einfaches Verfahren, das sich häufig leicht realisieren läßt und eine gute Verteilung der Schlüssel ergibt.

Beispiel: Gegeben ist eine Tabelle mit einer Größe $M=101$. Mit Hilfe der oben stehenden Hash-Funktion soll für den vierstelligen Schlüssel " A KEY" die Tabellenadresse berechnet werden.

Als erstes muß der Schlüssel in eine Zahl codiert werden:

Schlüssel	A	K	E	Y
Alphabetpos.	1	11	5	25
Pos. in Binärzahl	00001	01011	00101	11001

Hieraus ergibt sich die Binärzahl $00001010110010111001 = 44217$

$$44217 \bmod 101 = 80$$

Somit wird der Schlüssel A KEY an die, von der Hash-Funktion berechneten, Position 80 in der Tabelle abgebildet.

2. Methoden zur Kollisionsbehandlung

Durch die Verwendung einer Hash-Funktion kann es vorkommen, daß mehreren verschiedenen Schlüsseln ein und dieselbe Adresse zugewiesen wird. Die Behandlung solcher Kollisionen kann auf verschiedenste Art und Weise geregelt werden.

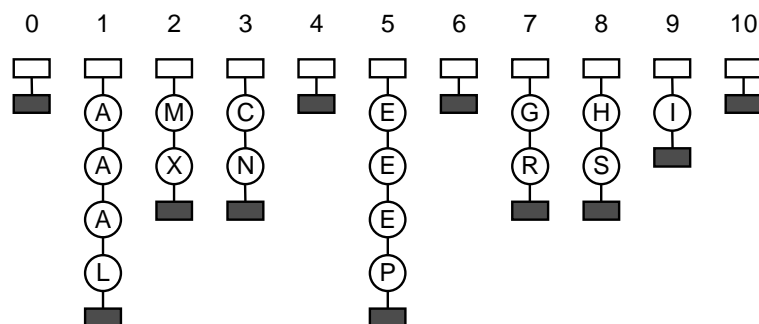
2.1 Getrennte Verkettung

Bei der getrennten Verkettung wird für jede Tabellenadresse eine verkettete Liste erzeugt, die jene Datensätze enthält, deren Schlüssel auf diese Adresse abgebildet werden.

Beispiel:

$M = 11$

Schlüssel: **A** **S** **E** **A** **R** **C** **H** **I** **N** **G** **E** **X** **A** **M** **P** **L** **E**
 Hashwerte: 1 8 5 1 7 3 8 9 3 7 5 2 1 2 5 1 5



Da die Schlüssel, die auf ein und dieselbe Tabellenposition abgebildet werden, in einer verketteten Liste abgelegt werden, können sie ebensogut geordnet gespeichert werden. Diese Methode führt zu einer Verallgemeinerung des elementaren Listensuchverfahrens. Anstatt eine einzige Liste mit einem Listenkopf zu führen, werden bei der getrennten Verkettung M Listen mit M Listenköpfen geführt.

Allgemein gilt: Eine getrennte Verkettung verringert die Anzahl der Vergleiche bei einer sequentiellen Suche durchschnittlich um den Faktor M. Es wird jedoch zusätzlicher Platz für die Verkettungen beansprucht.

Bei einer Implementation der getrennten Verkettung wird für M gewöhnlich ein relativ kleiner Wert gewählt, damit kein großer zusammenhängender Speicherbereich belegt wird. Doch es ist sicher am besten, M genügend groß zu wählen, so daß die Listen kurz genug sind, damit die sequentielle Suche zur effizientesten Methode wird.

Als Faustregel gilt, M sollte etwa einem Zehntel der zu erwarteten Schlüssel entsprechen, so daß die Listen durchschnittlich je zehn Schlüssel enthalten.

Falls mehr Schlüssel als erwartet auftreten, dauern die Suchvorgänge ein wenig länger, bei weniger Schlüssel wurde vielleicht etwas mehr Speicherplatz verwendet.

2.2 Offene Adressierung

Falls die Anzahl der Elemente, die in die Hash-Tabelle aufgenommen werden sollen, im voraus geschätzt werden kann, und falls ausreichend zusammenhängender Speicherplatz zur Verfügung steht, um alle Schlüssel aufzunehmen und noch zusätzlich Platz zu lassen, dann lohnt es sich nicht, irgendwelche Verkettungen zu verwenden.

Bei der offenen Adressierung wird im Kollisionsfall der Adreßbereich entweder in konstanten oder in quadratisch ansteigenden Abständen nach freiem Speicherplatz durchsucht.

2.2.1 Lineares Austesten

Die einfachste Methode mit offener Adressierung wird lineares Austesten genannt. Im Falle einer Kollision wird einfach die nächste Position in der Hash-Tabelle untersucht. Ist diese leer, können die Daten dort gespeichert werden. Wenn auch diese Position belegt ist, wird wieder die nächste Position untersucht, solange bis eine leere Position erreicht wurde.

Beispiel:

M = 19

Schlüssel:	A	S	E	A	R	C	H	I	N	G	E	X	A	M	P	L	E		
Hashwerte:	1	0	5	1	18	3	8	9	14	7	5	5	1	13	16	12	5		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	S	A	A	C	A	E	E	G	H	I	X	E	L	M	N		P		R

Der Umfang der Tabelle für lineares Austesten ist größer als für getrennte Verkettung, doch die Gesamtgröße des verwendeten Speicherplatzes ist geringer, da keine Verkettungen benutzt werden.

Allgemein gilt: Für eine Hash-Tabelle, die zu weniger als zwei Dritteln gefüllt ist, erfordert lineares Austesten im Durchschnitt weniger als fünf Tests.

2.2.2 Doppeltes Hashing

Da beim linearen Austesten auch andere Schlüssel untersucht werden, speziell dann, wenn sich die Tabelle zu füllen beginnt, kann das eine drastische Erhöhung der Suchzeit bewirken. Solche Anhäufungen sorgen dafür das lineares Austesten für fast volle Tabellen sehr langsam abläuft.

Mit Hilfe des doppelten Hashing kann dieses Problem praktisch beseitigt werden. Beim Hash-hash-Verfahren wird im Kollisionsfall zur Suche eines freien Speicherplatzes wieder eine Hash-Funktion verwendet, die sich von der ersten unterscheiden sollte auftreten würde. Ansonsten ist dieses Verfahren das geeignetste, um die Anzahl der Kollisionen gering zu halten.

Für das folgende Beispiel wurde folgende Funktion als zweite Hash-Funktion verwendet:
 $h_2 = 8 - (k \bmod 8)$.

Beispiel:

M = 19

Schlüssel:	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 5px;">A</td><td style="padding: 2px 5px;">S</td><td style="padding: 2px 5px;">E</td><td style="padding: 2px 5px;">A</td><td style="padding: 2px 5px;">R</td><td style="padding: 2px 5px;">C</td><td style="padding: 2px 5px;">H</td><td style="padding: 2px 5px;">I</td><td style="padding: 2px 5px;">N</td><td style="padding: 2px 5px;">G</td><td style="padding: 2px 5px;">E</td><td style="padding: 2px 5px;">X</td><td style="padding: 2px 5px;">A</td><td style="padding: 2px 5px;">M</td><td style="padding: 2px 5px;">P</td><td style="padding: 2px 5px;">L</td><td style="padding: 2px 5px;">E</td> </tr> </table>	A	S	E	A	R	C	H	I	N	G	E	X	A	M	P	L	E
A	S	E	A	R	C	H	I	N	G	E	X	A	M	P	L	E		
Hashwerte 1:	<table style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 0 10px;">1</td><td style="padding: 0 10px;">0</td><td style="padding: 0 10px;">5</td><td style="padding: 0 10px;">1</td><td style="padding: 0 10px;">18</td><td style="padding: 0 10px;">3</td><td style="padding: 0 10px;">8</td><td style="padding: 0 10px;">9</td><td style="padding: 0 10px;">14</td><td style="padding: 0 10px;">7</td><td style="padding: 0 10px;">5</td><td style="padding: 0 10px;">5</td><td style="padding: 0 10px;">1</td><td style="padding: 0 10px;">13</td><td style="padding: 0 10px;">16</td><td style="padding: 0 10px;">12</td><td style="padding: 0 10px;">5</td> </tr> </table>	1	0	5	1	18	3	8	9	14	7	5	5	1	13	16	12	5
1	0	5	1	18	3	8	9	14	7	5	5	1	13	16	12	5		
Hashwerte 2:	<table style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 0 10px;">7</td><td style="padding: 0 10px;">8</td><td style="padding: 0 10px;">3</td><td style="padding: 0 10px;">7</td><td style="padding: 0 10px;">6</td><td style="padding: 0 10px;">5</td><td style="padding: 0 10px;">8</td><td style="padding: 0 10px;">7</td><td style="padding: 0 10px;">2</td><td style="padding: 0 10px;">1</td><td style="padding: 0 10px;">3</td><td style="padding: 0 10px;">3</td><td style="padding: 0 10px;">7</td><td style="padding: 0 10px;">3</td><td style="padding: 0 10px;">8</td><td style="padding: 0 10px;">4</td><td style="padding: 0 10px;">3</td> </tr> </table>	7	8	3	7	6	5	8	7	2	1	3	3	7	3	8	4	3
7	8	3	7	6	5	8	7	2	1	3	3	7	3	8	4	3		

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
S	A	P	C	E	E		G	A	I		E	L	M	N	A	H	X	R

Allgemein gilt: Doppeltes Hashing erfordert im Durchschnitt weniger Tests als lineares Austesten.

Methoden der offenen Adressierung können in einer dynamischen Situation, bei der eine nicht vorhersagbare Anzahl von Einfüge- und Löschoperationen auszuführen sind, unzuweckmäßig sein. Es muß beim Löschen mit Vorsicht vorgegangen werden, da ein Datensatz nicht einfach aus einer Tabelle entfernt werden kann, die mit Hilfe von linearem Austesten oder doppeltem

Hashing erzeugt wurde. Es wurde eine Lücke entstehen, die bei der Suche nach einem anderen Datensatz dazu führt, daß an dieser Stelle abgebrochen wird, anstatt weiterzusuchen. Eine Lösung dieses Problems wäre, spezielle Platzhalter zu verwenden.

Zu beachten ist, daß bei der getrennten Verkettung das Löschen eines Datensatzes kein besonderes Problem darstellte.

Die Wahl der besten Hashing-Methode für eine spezielle Anwendung kann sehr kompliziert sein. Im allgemeinen besteht die beste Strategie darin, die Methode der einfachen getrennten Verkettung anzuwenden, um die Suchzeit stark zu verkürzen, wenn die Anzahl der zu verarbeitenden Datensätze nicht im voraus bekannt ist, und doppeltes Hashing zu verwenden, um eine Menge von Schlüsseln zu suchen, deren Größe im voraus grob angegeben werden kann.