Hashverfahren

1 Einleitung

Bei Hashverfahren versucht man, Schlüssel aus einem grossen Schlüsselbereich K in einer Hashtabelle der Grösse m zu speichern. Dazu wird eine Hashfunktion h benutzt, welche angibt, an welcher Stelle in der Hashtabelle ein Element gespeichert werden soll.

Die Hashfunktion sollte sehr schnell zu berechnen sein, und gleichzeitig die Schlüsselstruktur zerstören. Da $|K| >> m$ gilt, kommt es unweigerlich zu Kollisionen, also k_1, k_2 mit $h(k_1) = h(k_2)$. Solche k werden Synonyme genannt.

2 Hashfunktionen

- Modulare Arithmetik $h(k) = k \bmod m$

  Die Wahl von m ist von entscheidender Bedeutung. M sollte eine Primzahl sein, und zudem sollte m r^i+j für kleine i und j nicht teilen (z.B. i,j in {0,..,6})

- Multipilkatives Hashing $h(k) = \lfloor m\ (k\theta - \lfloor k\theta \rfloor) \rfloor$

  Die Wahl von θ ist hier ebenfalls ziemlich wichtig, der goldene Schnitt ist beispielsweise eine gute Wahl.

3 Kollisionen

3.1 Verketten der Überläufer

Die Überläufer können auf zwei Arten verkettet werden:

- **Separate Verkettung**: Hierbei gibt es in der Hashtabelle an jedem Ort für ein Element Platz, und zusätzlich ein Zeiger auf eine verkettete Liste, falls mehr Elemente gespeichert werden müssen.
- **Direkte Verkettung**: In der Hashtabelle selber gibt es nur Zeiger auf verkettete Listen. Der Vorteil ist offensichtlich, es braucht so keine Unterscheidung von Elementen die in der Tabelle gespeichert werden, und solchen die ausserhalb Platz finden. Dafür gibt es einen leicht grösseren Overhead, vor allem bei sehr kleinen Datensätzen.

3.2 Offene Hashverfahren

Anders als beim Verketten werden Überläufer direkt in der Tabelle abgelegt. Es gibt eine Sondierungsreihenfolge $s(k,j)$, welche festlegt, wo Synonyme gespeichert werden. Zuerst wird bei $h(k)$ geschaut, gibt es dort keinen Platz, wird fortlaufend bei $h(k) - s(k,j)$ mit aufsteigendem j = 1,2,.. nachgesehen, bis ein Platz frei wird.

3.2.1 Sondierverfahren

- Lineares Sondieren $s(k,j) = j$

Ein Problem von linearem Sondieren ist die **primäre Häufung**. Das heisst, dass Teile von bereits besetzten Bereichen immer grösser werden, und so immer länger Sondiert werden muss.

- Quadratisches Sondieren $s(k,j) = \left\lceil \frac{j}{2} \right\rceil^2 \cdot (-1)^j$
  Hier gibt es keine Probleme mit primärer Häufung, dafür spielt aber sekundäre Häufung eine Rolle: Die Sondierungsreihenfolge hängt nur von j ab, nicht aber von k
- Double Hashing $s(k,j) = j \cdot h'(k)$
  Die zweite Hashfunktion h'(k) soll natürlich unabhängig von h(x) gewählt werden. In der Praxis hat sich $s(k,j) = j \cdot (1 + (k \; mod \; (m-2)))$ bewährt, mit $h(k) = k \; mod \; m$.

4 Lineares Hashing

Lineares Hashing ist wohl das einfachste Beispiel eines dynamischen Hashverfahrens. Alle bisher betrachteten Hashverfahren haben eine fixe Tabellengrösse, was die Struktur sehr starr erscheinen lässt. Will man die Grösse der Tabelle trotzdem ändern, müssen sämtliche Einträge gemäss einer neuen Hashfunktion verteilt werden. Genau dies will man mit dynamischen Hashverfahren umgehen: Es soll möglich sein, die Grösse der Tabelle zu ändern, ohne sämtliche Einträge neu einfügen zu müssen.

Diese Verfahren sind besonders für Extern-Speicher geeignet.

4.1 Beschreibung des Verfahrens

Achtung: dieser Abschnitt ist ein ziemliches Chaos =)

Es gibt nun nicht mehr eine Hashtabelle im eigentlichen Sinn, sondern eine Hashdatei hd, welche aus Blöcken $b[1]$ bis $b[m]$ besteht. In jeden Block können mehrere Datensätze gespeichert werden, wodurch Kollisionen zu einem kleineren Problem werden. Gibt es trotzdem zu viele Synonyme (also mehr, als in einem Block Platz haben), werden diese ähnlich wie bei geschlossenen Hashverfahren durch verketten abgespeichert: Es wird ein weitere Block alloziert, welcher an den vollen Block angehängt wird. Geschieht dies zu oft, leidet natürlich die Performance, doch bei linearen Hashing kann dagegen nichts unternommen werden. Im folgenden werden solche verketten Blöcke als einen logischen Block aufgefasst. Weiter soll nicht behandelt werden, wie Datensätze innerhalb von Blocken verwaltet werden: Dies ist aufgrund der kleinen Blockgrösse mit wenig Rechenaufwand möglich.

4.1.1 Einfügen, Löschen

Will man einen Datensatz einfügen, überprüft man zuerst, ob der Belegungsfaktor einen konstanten Wert überschreitet. Ist dies der Fall, wird ein Block $b[m+1]$ an hd angehängt. Dabei wird der Block $b[pos\ ]$ aufgeteilt, wobei ein Teil der Einträge in den neuen Block verschoben werden. Dann wird pos um eins erhöht. Die globale Variable pos speichert die Position, bei welcher die nächste Teilung stattfinden wird. Weiter gibt es eine Variable l die angibt, wie viele Bits gerade mindestens für die Blockadressierung verwendet werden. Erreicht $pos\ l$, so wird pos auf 1 zurückgesetzt, und l um eins erhöht.

Analog wird beim Löschen vorgegangen, wenn der Belegungsfaktor einen gegebenen Wert unterschreitet.

4.1.2 Hashfunktion h_i

Die Hashfunktion h_i liefert die ersten i Bits des Schlüssels, rückwärts gelesen. Um nun die Blockadresse eines Schlüssels key zu finden, wird folgendermassen vorgegangen:

Ist $h_l(key)$ kleiner als pos, so ist die Adresse $h_{l+1}(key)$, andernfalls $h_l(key)$.