

3.3 Gewichtsbalancierte Bäume

Siehe zu diesem Thema Seite 189ff in



Mehlhorn, Kurt:

Data structures and algorithms 1: Sorting and searching

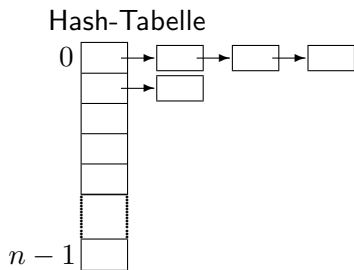
EATCS Monographs on Theoretical Computer Science

Springer Verlag: Berlin-Heidelberg-New York-Tokyo, 1984

4. Hashing

4.1 Grundlagen

- Universum U von Schlüsseln, z.B. $\subseteq \mathbb{N}_0$, $|U|$ groß
- Schlüsselmenge $S \subseteq U$, $|S| = m \leq n$
- array $T[0..n-1]$ Hashtabelle
- Hashfunktion $h : U \rightarrow [0..n-1]$



Problemstellung: Gegeben sei eine Menge M von Elementen, von denen jedes durch einen Schlüssel k aus der Menge U bezeichnet sei. Die Problemstellung ist: Wie muss die Speicherung der Elemente aus M bzw. der zugehörigen Schlüssel organisiert werden, damit jedes Element anhand seines Schlüssels möglichst schnell lokalisiert werden kann?
Gesucht ist also eine Abbildung

$$h : K \rightarrow T$$

von der Menge aller Schlüssel in den Adressraum T der Maschine. Hierbei soll jedoch eine bisher nicht beachtete Schwierigkeit berücksichtigt werden: Die Menge U der möglichen Schlüsselwerte ist wesentlich größer als der Adressraum. Folglich kann die Abbildung h nicht injektiv sein, es gibt Schlüssel k_1, k_2, \dots mit $h(k_1) = h(k_2) = \dots$

Wir werden sehen, dass aufgrund dieses Umstandes die Speicherstelle eines Elements mit Schlüssel k von einem anderen Element mit einem anderen Schlüssel l besetzt sein kann und mit einer gewissen Wahrscheinlichkeit auch sein wird: Es treten **Kollisionen** auf.

Satz 22

In einer Hashtabelle der Größe n mit m Objekten tritt mit Wahrscheinlichkeit

$$\geq 1 - e^{-\frac{m(m-1)}{2n}} \approx 1 - e^{-\frac{m^2}{2n}}$$

mindestens eine Kollision auf, wenn für jeden Schlüssel jede Hashposition gleich wahrscheinlich ist.

Beweis:

Sei A_m das Ereignis, dass unter m Schlüsseln **keine** Kollision auftritt. Dann gilt

$$\begin{aligned}\Pr[A_m] &= \prod_{j=0}^{m-1} \frac{n-j}{n} = \prod_{j=0}^{m-1} \left(1 - \frac{j}{n}\right) \\ &\leq \prod_{j=0}^{m-1} e^{-\frac{j}{n}} = e^{-\sum_{j=0}^{m-1} \frac{j}{n}} = e^{-\frac{m(m-1)}{2n}}.\end{aligned}$$

Es folgt die Behauptung. □

Korollar 23

Hat eine Hashtabelle der Größe n mindestens $\omega(\sqrt{n})$ Einträge und ist für jeden Schlüssel jede Hashposition gleich wahrscheinlich, so tritt mit Wahrscheinlichkeit $1 - o(1)$ mindestens eine Kollision auf.

Um die Kollisionszahl möglichst gering zu halten, müssen Hashfunktionen gut streuen.

Definition 24

- 1 Eine Hashfunktion

$$h : U \rightarrow \{0, 1, \dots, n - 1\}$$

heißt **perfekt** für $S \subseteq U$, wenn für alle $j, k \in S, j \neq k$ gilt

$$h(j) \neq h(k) .$$

- 2 Eine Klasse \mathcal{H} von Hashfunktionen $h : U \rightarrow \{0, 1, \dots, n - 1\}$ heißt **perfekt**, falls \mathcal{H} für jedes $S \subseteq U$ mit $|S| = n$ eine für S perfekte Hashfunktion enthält.

Grundsätzliche Fragestellungen:

- 1 Wie schwierig ist es, perfekte Hashfunktionen darzustellen (also: was ist ihre Programmgröße)?
- 2 Wie schwierig ist es, gegeben S , eine für S perfekte Hashfunktion zu finden?
- 3 Wie schwierig ist es, gegeben $k \in S$, $h(k)$ für eine für S perfekte Hashfunktion auszuwerten?

Typische „praktische“ Hashfunktionen:

$$h(k) = k \bmod n \quad (\text{Teilermethode})$$

$$h(k) = \lfloor n(ak - \lfloor ak \rfloor) \rfloor \quad \text{für } a < 1 \quad (\text{Multiplikationsmethode})$$

Wir betrachten zunächst Methoden der Kollisionsbehandlung.

4.2 Methoden zur Kollisionsauflösung

Wir unterscheiden grundsätzlich

- geschlossene und
- offene Hashverfahren.

Bei geschlossenen Hashverfahren werden Kollisionen nicht wirklich aufgelöst.

4.2.1 Geschlossene Hashverfahren (Chaining)

Die Hashtabelle ist ein Array von n linearen Listen, wobei die i -te Liste alle Schlüssel k beinhaltet, für die gilt:

$$h(k) = i.$$

Zugriff: Berechne $h(k)$ und durchsuche die Liste $T[h(k)]$.

Einfügen: Setze neues Element an den Anfang der Liste.

Sei

$$\delta_h(k_1, k_2) = \begin{cases} 1 & \text{falls } h(k_1) = h(k_2) \text{ und } k_1 \neq k_2 \\ 0 & \text{sonst} \end{cases}$$

und

$$\delta_h(k, S) = \sum_{j \in S} \delta_h(j, k), \text{ Anzahl Kollisionen von } k \text{ mit } S.$$

Die Zugriffskosten sind:

$$\mathcal{O}(1 + \delta_h(k, S))$$

Sei A eine Strategie zur Kollisionsauflösung. Wir bezeichnen im Folgenden mit

- A^+ den mittleren Zeitbedarf für eine **erfolgreiche** Suche unter Verwendung von A ;
- A^- den mittleren Zeitbedarf für eine **erfolglose** Suche unter Verwendung von A ;
- $\alpha := \frac{m}{n}$ den **Füllfaktor** der Hashtabelle.

Sondierungskomplexität für Chaining

Falls auf alle Elemente in der Hashtabelle mit gleicher Wahrscheinlichkeit zugegriffen wird, ergibt sich

- A^- : mittlere Länge der n Listen; da $\frac{m}{n} = \alpha$, folgt

$$A^- \leq 1 + \alpha.$$

- A^+ :

$$\begin{aligned} A^+ &= \frac{1}{m} \sum_{i=0}^{m-1} \left(1 + \frac{i-1}{n} + \frac{1}{n} \right) \\ &= \frac{1}{m} \sum_{i=0}^{m-1} \left(1 + \frac{i}{n} \right) \\ &= 1 + \frac{m(m-1)}{2nm} \\ &\leq 1 + \frac{m}{2n} = 1 + \frac{\alpha}{2} \end{aligned}$$

Für festen Füllfaktor α ergibt sich also im Mittel Laufzeit $\Theta(1)$.

4.2.2 Hashing mit offener Adressierung

Beispiele:

- Lineares Sondieren (linear probing)
- Quadratisches Sondieren
- Double Hashing
- Robin-Hood-Hashing
- ...

Bei dieser Methode werden die Elemente nicht in der Liste, sondern direkt in der Hash-Tabelle gespeichert. Wird bei *Insert* oder *IsElement*, angewendet auf Schlüssel k , ein Element mit Schlüssel $\neq k$ an der Adresse $h(k)$ gefunden, so wird auf deterministische Weise eine alternative Adresse berechnet. Für jeden Schlüssel $k \in U$ wird somit eine Reihenfolge (Sondierungsfolge) von Positionen in $T[]$ betrachtet, um k zu speichern bzw. zu finden.

Sondieren:

Sei $s(j, k) : [0..n - 1] \times U \rightarrow [0..n - 1]$;

Definiere $h(j, k) = (h(k) - s(j, k)) \bmod n$; $(0 \leq j \leq n - 1)$

Starte mit $h(0, k)$, dann, falls $h(0, k)$ belegt, $h(1, k)$, ...

Grundsätzliches Problem:

Sei $h(k) = h(k')$ für zwei Schlüssel $k, k' \in S$. Werde zunächst k eingefügt, dann k' , dann k gelöscht. Wie findet man k' ?

(Beachte: k' steht nicht unmittelbar an $h(k')$.)

Lösungsvorschlag: Markiere k als gelöscht, entferne es aber nicht!
Wenn Speicher gebraucht wird, k überschreiben.

Beispiele für Sondierungen

Lineares Sondieren:

Setze $s(j, k) = j$ d.h. sondiere gemäß
 $h(k), h(k) - 1, \dots, 0, n - 1, \dots, h(k) + 1$.

Es wird für *IsElement* solange rückwärts gesucht, bis entweder das Element mit Schlüssel k oder eine freie Position gefunden ist. Im letzteren Fall ist das gesuchte Element nicht in der Hash-Tabelle enthalten.

Problem: Es entstehen primäre **Häufungen** (primary clustering) um diejenigen Schlüssel herum, die beim Einfügen eine Kollision hervorgerufen haben.

Satz 25

Die durchschnittliche Anzahl der Schritte beim linearen Sondieren ist

$$\mathbb{E}[\# \text{ Sondierungsschritte}] = \begin{cases} \frac{1}{2} \left(1 + \frac{1}{(1-\alpha)} \right) & \text{erfolgreich} \\ \frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right) & \text{erfolglos} \end{cases}$$

Einige Werte:

α	erfolgreich	erfolglos
0.5	1.5	2.5
0.9	5.5	50.5
0.95	10.5	200.5

Beispiele für Sondierungen

Quadratisches Sondieren:

Setze $s(j, k) = (-1)^j \lceil \frac{j}{2} \rceil^2$, d.h. sondiere nach $h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots$

Frage: Ist das überhaupt eine Permutation von $[0..n - 1]$? Ist $s(j, k)$ geeignet, alle Positionen zu erreichen?

Man kann zeigen, dass für Primzahlen n von der Form $4i + 3$ die Sondierungsgröße $(h(k) - s(j, k)) \bmod n$ eine Permutation von $[0..n - 1]$ liefert.

Satz 26

Die durchschnittliche Anzahl der Schritte bei quadratischem Sondieren ist

$$\mathbb{E}[\# \text{ Sondierungsschritte}] = \begin{cases} 1 + \ln\left(\frac{1}{1-\alpha}\right) - \frac{\alpha}{2} & \text{erfolgreich} \\ \frac{1}{1-\alpha} - \alpha + \ln\left(\frac{1}{1-\alpha}\right) & \text{erfolglos} \end{cases}$$

Einige Werte:

α	erfolgreich	erfolglos
0.5	1.44	2.19
0.9	2.85	11.4
0.95	3.52	22.05

Beispiele für Sondierungen

Double Hashing:

Setze $s(j, k) = jh'(k)$, wobei h' eine zweite Hashfunktion ist. $h'(k)$ muss relativ prim zu n gewählt werden, damit $(h(k) - s(j, k)) \bmod n$ eine Permutation der Hashadressen wird.

Satz 27

Die durchschnittliche Anzahl der Sondierungen bei Double Hashing ist

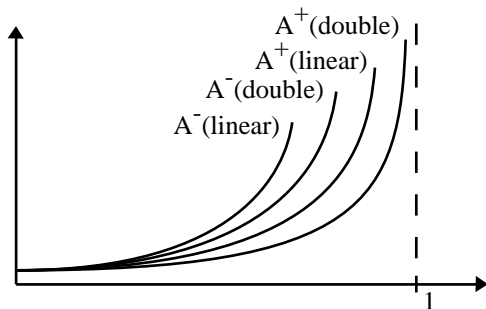
$$\mathbb{E}[\# \text{ Sondierungsschritte}] = \begin{cases} \frac{1}{\alpha} \ln \frac{1}{1-\alpha} & \text{erfolgreich} \\ \frac{1}{1-\alpha} & \text{erfolglos} \end{cases}$$

Einige Werte:

α	erfolgreich	erfolglos
0.5	1.39	2
0.9	2.55	10
0.95	3.15	20

Zum Beispiel: $h'(k) = 1 + k \bmod (n - 2)$ (mit $n > 2$ prim).

Beispiele für Sondierungen



Sondierungskomplexität