

Die Struktur der perfekten Hashtabelle nach



Michael L. Fredman, János Komlós, Endre Szemerédi:

Storing a sparse table with $\mathcal{O}(1)$ worst case access time,
Journal of the ACM **31**(3), p. 538–544 (1984)

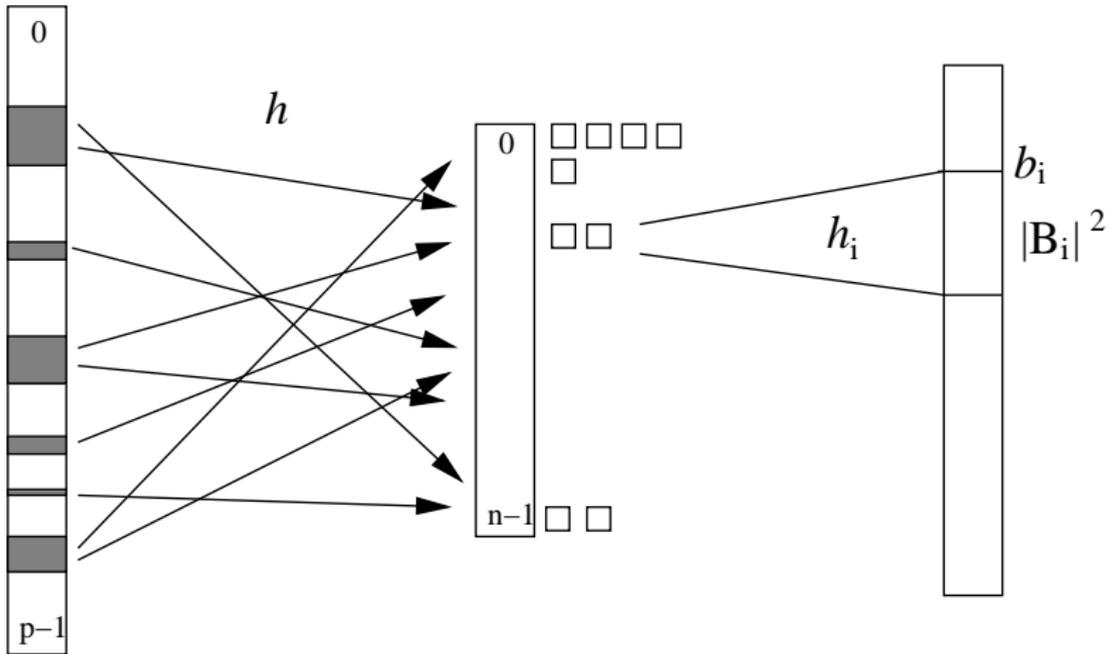
verwendet ein **zweistufiges** Hashverfahren.

Für einen gegebenen Schlüssel x wird zunächst $i = h(x)$ berechnet, um über den Tabellenplatz $T[i]$, b_i , $|B_i|$ und $h_i \in \mathcal{H}_{2,|B_i|^2}$ zu ermitteln. Dann wird im Tabellenplatz $T'[b_i + h_i(x)]$ nachgeschaut, ob x da abgespeichert ist. Falls ja, wird **true** ausgegeben und sonst **false**.

Falls

$$\sum_{i=0}^{n-1} |B_i|^2 < 4n$$

ist, so wird nur $\mathcal{O}(n)$ Platz verwendet.



Zweistufige Hashtabelle nach Fredman, Komlós und Szemerédi

Algorithmus für Hashtabelle nach FKS:

Eingabe: $S \subseteq U$, $|S| = m \leq n$

Ausgabe: Hashtabelle nach FKS

1. Wähle $h \in \mathcal{H}_{2,n}$ zufällig. Berechne $h(x)$ für alle $x \in S$.
2. Falls $\sum_i |B_i|^2 \geq 4m$, dann wiederhole 1.
3. Konstruiere die Mengen B_i für alle $0 \leq i < n$.
4. **for** $i = 0$ **to** $n - 1$ **do**
 - (a) wähle $h_i \in \mathcal{H}_{2,|B_i|^2}$ zufällig
 - (b) falls h_i auf B_i nicht injektiv ist, wiederhole (a)

Ein Durchlauf der Schleife bestehend aus den Schritten 1. und 2. benötigt Zeit $\mathcal{O}(n)$. Gemäß Lemma 34 ist die Wahrscheinlichkeit, dass Schritt 1. wiederholt werden muss, $\leq 1/2$ für jedes neue h .

Die Anzahl der Schleifendurchläufe ist also geometrisch verteilt mit Erfolgswahrscheinlichkeit $\geq 1/2$, und es ergibt sich

$$\mathbb{E}[\# \text{ Schleifendurchläufe}] \leq 2.$$

Also ist der Zeitaufwand für diese Schleife $\mathcal{O}(n)$. Schritt 3. kostet offensichtlich ebenfalls Zeit $\mathcal{O}(n)$.

Für jedes $i \in \{0, \dots, n-1\}$ gilt, ebenfalls gemäß Lemma 34, dass

$$\Pr[h_i \text{ ist auf } B_i \text{ injektiv}] \geq 1 - \frac{|B_i|(|B_i| - 1)}{2|B_i|^2} > \frac{1}{2}.$$

Damit ist auch hier die erwartete Anzahl der Schleifendurchläufe ≤ 2 und damit der erwartete Zeitaufwand

$$\mathcal{O}(|B_i|^2).$$

Insgesamt ergibt sich damit für Schritt 4. wie auch für den gesamten Algorithmus ein Zeitaufwand von

$$\mathcal{O}(n).$$

4.4.2 Dynamisches perfektes Hashing

Sei $U = \{0, \dots, p-1\}$ für eine Primzahl p . Zunächst einige mathematische Grundlagen.

Definition 35

$\mathcal{H}_{k,n}$ bezeichne in diesem Abschnitt die Klasse aller Polynome $\in \mathbb{Z}_p[x]$ vom Grad $< k$, wobei mit $\vec{a} = (a_0, \dots, a_{k-1}) \in U^k$

$$h_{\vec{a}}(x) = \left(\left(\sum_{j=0}^{k-1} a_j x^j \right) \bmod p \right) \bmod n \text{ für alle } x \in U.$$

Definition 36

Eine Klasse \mathcal{H} von Hashfunktionen von U nach $\{0, \dots, n-1\}$ heißt (c, k) -universell, falls für alle paarweise verschiedenen $x_0, x_1, \dots, x_{k-1} \in U$ und für alle $i_0, i_1, \dots, i_{k-1} \in \{0, \dots, n-1\}$ gilt, dass

$$\Pr[h(x_0) = i_0 \wedge \dots \wedge h(x_{k-1}) = i_{k-1}] \leq \frac{c}{n^k},$$

wenn $h \in \mathcal{H}$ gleichverteilt gewählt wird.

Satz 37

$\mathcal{H}_{k,n}$ ist (c, k) -universell mit $c = (1 + \frac{n}{p})^k$.

Beweis:

Da \mathbb{Z}_p ein Körper ist, gibt es für jedes Tupel $(y_0, \dots, y_{k-1}) \in U^k$ genau ein Tupel $(a_0, \dots, a_{k-1}) \in \mathbb{Z}_p^k$ mit

$$\sum_{j=0}^{k-1} a_j x_r^j = y_r \pmod{p} \quad \text{für alle } 0 \leq r < k.$$

Damit folgt, dass

$$\begin{aligned} & |\{\vec{a}; h_{\vec{a}}(x_r) = i_r \text{ für alle } 0 \leq r < k\}| \\ &= |\{(y_0, \dots, y_{k-1}) \in U^k; y_r = i_r \pmod{n} \text{ für alle } 0 \leq r < k\}| \\ &\leq \left\lceil \frac{p}{n} \right\rceil^k. \end{aligned}$$

Beweis (Forts.):

Da es insgesamt p^k Möglichkeiten für \vec{a} gibt, folgt

$$\begin{aligned}\Pr[h(x_r) = i_r \text{ für alle } 0 \leq r < k] &\leq \left[\frac{p}{n}\right]^k \cdot \frac{1}{p^k} \\ &= \left(\left[\frac{p}{n}\right] \cdot \frac{n}{p}\right)^k \cdot \frac{1}{n^k} \\ &< \left(1 + \frac{n}{p}\right)^k \cdot \frac{1}{n^k}.\end{aligned}$$



Kuckuck-Hashing für dynamisches perfektes Hashing

Kuckuck-Hashing arbeitet mit zwei Hashtabellen, T_1 und T_2 , die je aus den Positionen $\{0, \dots, n - 1\}$ bestehen. Weiterhin benötigt es zwei $(1 + \delta, \mathcal{O}(\log n))$ -universelle Hashfunktionen h_1 und h_2 für ein genügend kleines $\delta > 0$, die die Schlüsselmenge U auf $\{0, \dots, n - 1\}$ abbilden.

Jeder Schlüssel $x \in S$ wird **entweder** in Position $h_1(x)$ in T_1 **oder** in Position $h_2(x)$ in T_2 gespeichert, aber nicht beiden. Die *IsElement*-Operation prüft einfach, ob x an einer der beiden Positionen gespeichert ist.

Die *Insert*-Operation verwendet nun das Kuckucksprinzip, um neue Schlüssel einzufügen. Gegeben ein einzufügender Schlüssel x , wird zunächst versucht, x in $T_1[h_1(x)]$ abzulegen. Ist das erfolgreich, sind wir fertig.

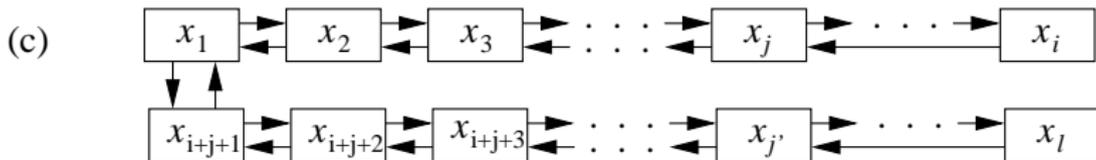
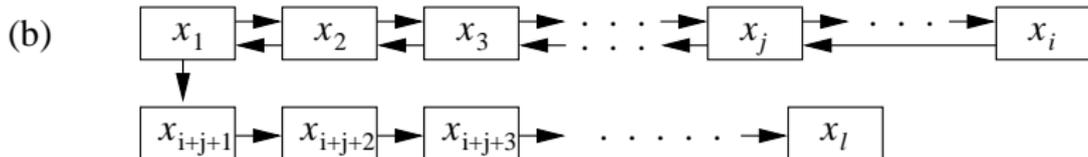
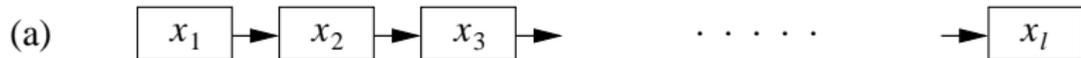
Falls aber $T_1[h_1(x)]$ bereits durch einen anderen Schlüssel y besetzt ist, nehmen wir y heraus und fügen stattdessen x in $T_1[h_1(x)]$ ein. Danach versuchen wir, y in $T_2[h_2(y)]$ unterzubringen. Gelingt das, sind wir wiederum fertig. Falls $T_2[h_2(y)]$ bereits durch einen anderen Schlüssel z besetzt ist, nehmen wir z heraus und fügen stattdessen y in $T_2[h_2(y)]$ ein. Danach versuchen wir, z in $T_1[h_1(z)]$ unterzubringen, und so weiter, bis wir endlich den zuletzt angefassten Schlüssel untergebracht haben. Formal arbeitet die Insert-Operation wie folgt:

```
if  $T_1[h_1(x)] = x$  then return fi  
repeat MaxLoop times  
    (a) exchange  $x$  und  $T_1[h_1(x)]$   
    (b) if  $x = \text{NIL}$  then return fi  
    (c) exchange  $x$  und  $T_2[h_2(x)]$   
    (d) if  $x = \text{NIL}$  then return fi  
od  
rehash(); Insert(x)
```

Für die Analyse der Zeitkomplexität nehmen wir an, dass die Schleife t -mal durchlaufen wird (wobei $t \leq \text{MaxLoop}$).

Es gilt, die folgenden zwei Fälle zu betrachten:

- 1 Die *Insert*-Operation gerät während der ersten t Runden in eine Endlosschleife
- 2 Dies ist nicht der Fall



Insert bei Kuckuck-Hashing; Endlosschleife im Fall (c)

Erster Fall: Sei $v \leq l$ die Anzahl der verschiedenen angefassten Schlüssel. Dann ist die Anzahl der Möglichkeiten, eine Endlosscheife zu formen, höchstens

$$v^3 \cdot s^{v-1} \cdot n^{v-1},$$

da es maximal v^3 Möglichkeiten für die Werte i , j und l gibt, s^{v-1} viele Möglichkeiten für die Positionen der Schlüssel, und n^{v-1} viele Möglichkeiten für die Schlüssel außer x_1 .

Angenommen, wir haben $(1, v)$ -universelle Hashfunktionen, dann passiert jede Möglichkeit nur mit einer Wahrscheinlichkeit von s^{-2v} . Falls $s \geq (1 + \delta)n$ für eine Konstante $\delta > 0$, dann ist die Wahrscheinlichkeit für den Fall 1 höchstens

$$\sum v = 3^l v^3 s^{v-1} n^{v-1} s^{-2v} 1 \leq \frac{1}{sn} \sum_{v=3}^{\infty} v^3 (n/s)^v = \mathcal{O}\left(\frac{1}{n^2}\right).$$

Zweiter Fall:

Lemma 38

Im zweiten Fall gibt es eine Schlüsselreihe der Länge mindestens $l/3$ in x_1, \dots, x_l , in der alle Schlüssel paarweise verschieden sind.

Beweis:

Nehmen wir an, dass die Operation zu einer bereits besuchten Position zurückkehrt, und seien i und j so definiert wie in der Abbildung. Falls $l \leq i + j$, dann bilden die ersten $j - 1 \geq (i + j - 1)/2 \geq l/2$ Schlüssel die gesuchte Folge.

Für $l \geq i + j$ muss eine der Folgen x_1, \dots, x_{j-1} und x_{i+j-1}, \dots, x_l die Länge mindestens $l/3$ haben.

Beweis (Forts.):

Sei also x'_1, \dots, x'_v eine solche Folge verschiedener Schlüssel in x_1, \dots, x_{2t} der Länge $v = \lceil (2t - 1)/3 \rceil$. Dann muss entweder für $(i_1, i_2) = (1, 2)$ oder für $(i_1, i_2) = (2, 1)$ gelten, dass

$$h_{i_1}(x'_1) = h_{i_1}(x'_2), h_{i_2}(x'_2) = h_{i_2}(x'_3), h_{i_1}(x'_3) = h_{i_1}(x'_4), \dots$$

Gegeben x'_1 , so gibt es n^{v-1} mögliche Folgen von Schlüsseln x'_2, \dots, x'_v . Für jede solche Folge gibt es zwei Möglichkeiten für (i_1, i_2) . Weiterhin ist die Wahrscheinlichkeit, dass die obigen Positionsübereinstimmungen gelten, höchstens $s^{-(v-1)}$, wenn die Hashfunktionen aus einer $(1, v)$ -universellen Familie stammen. Also ist die Wahrscheinlichkeit, dass es irgendeine Folge der Länge v gibt, so dass Fall 2 eintritt, höchstens

$$2(n/s)^{v-1} \leq 2(1 + \delta)^{-(2t-1)/3t+1}.$$

Diese Wahrscheinlichkeit ist polynomiell klein in n , falls $t = \Omega(\log n)$ ist. □

Beweis (Forts.):

Zusammen ergibt sich für die Laufzeit von *Insert*:

$$\begin{aligned} & 1 + \sum_{t=2}^{\text{MaxLoop}} (2(1 + \delta)^{-(2t-1)/3+1} + \mathcal{O}(1/n^2)) \\ & \leq 1 + \mathcal{O}\left(\frac{\text{MaxLoop}}{n^2}\right) + 2 \sum_{t=0}^{\infty} ((1 + \delta)^{-2/3})^t \\ & = \mathcal{O}\left(1 + \frac{1}{1 - (1 + \delta)^{-2/3}}\right) = \mathcal{O}(1 + 1/\delta). \end{aligned}$$



Beweis (Forts.):

Überschreitet m irgendwann einmal die Schranke $n/(1 + \delta)$, so wird n hochgesetzt auf $(1 + \delta)n$ und neu gehasht. Unterschreitet auf der anderen Seite m die Schranke $n/(1 + \delta)^3$, so wird n verringert auf $n/(1 + \delta)$ und neu gehasht. Auf diese Weise wird die Tabellengröße linear zur Anzahl momentan existierender Schlüssel gehalten. Der Aufwand für ein komplettes Rehashing ist $\mathcal{O}(n)$, so dass amortisiert über $\Theta(n)$ Einfügungen und Löschungen der Aufwand nur eine Konstante ist. □