4.4.2 Dynamisches perfektes Hashing

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

Definition 36

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

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

Definition 37

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

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

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

Satz 38

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

Beweis:

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

$$\sum_{j=0}^{k-1} a_j x_r^j = y_r \bmod p \qquad \text{für alle } 0 \le r < k.$$

Da es sich hier um eine Vandermonde-Matrix handelt, folgt, dass

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

Beweis (Forts.):

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

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

□

Kuckuck-Hashing für dynamisches perfektes Hashing

Kuckuck-Hashing arbeitet mit zwei Hashtabellen, T_1 und T_2, die je aus den Positionen $\{0, \ldots, n-1\}$ 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 $\{0, \ldots, n-1\}$ 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 in 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.

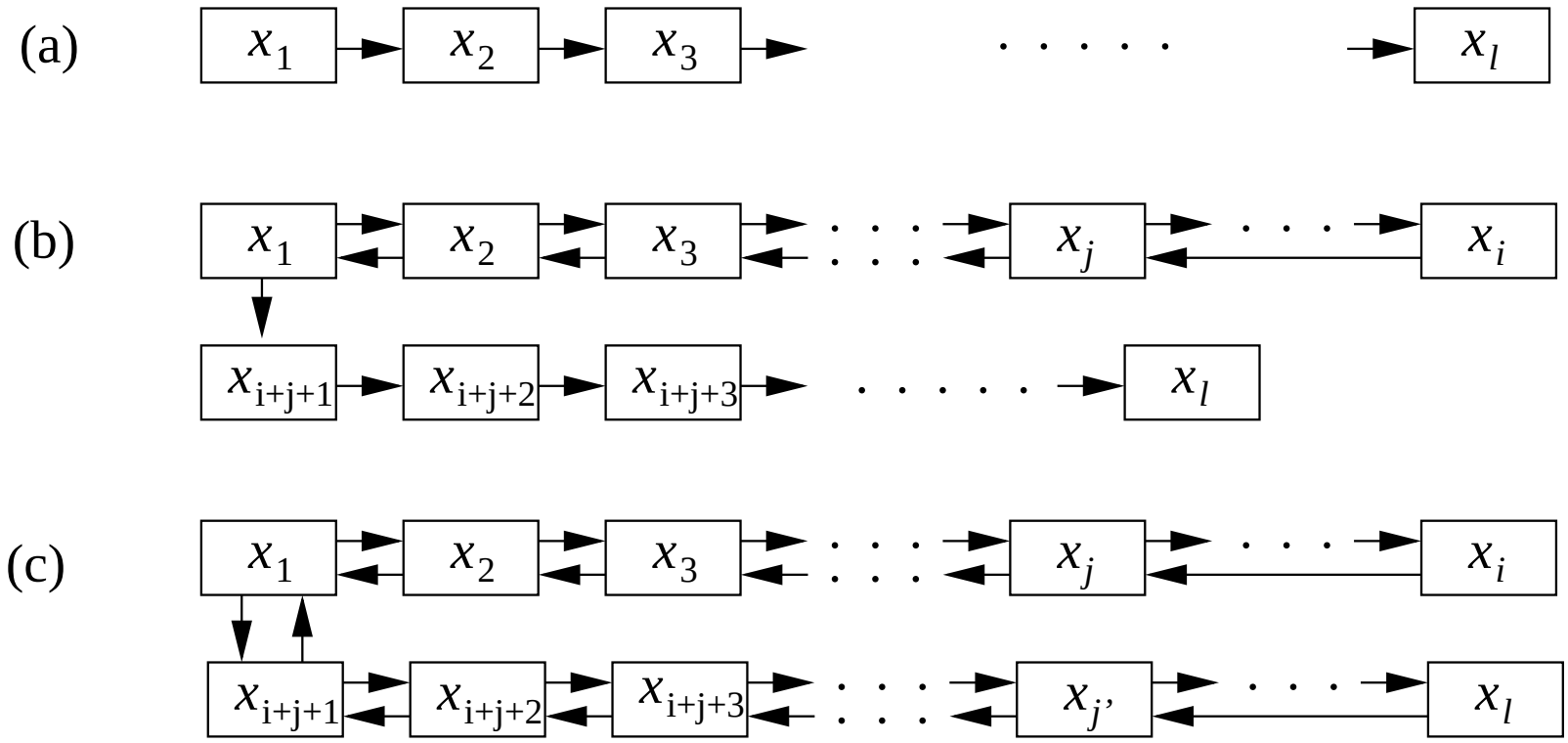
Falls aber $T_1[h_1(x)]$ bereits durch einen anderen Schlüssel y besetzt ist, nehmen wir y heraus und fügen stattdessen x in $T_1[h_1(x)]$ ein. Danach versuchen wir, y in $T_2[h_2(y)]$ unterzubringen. Gelingt das, sind wir wiederum fertig. Falls $T_2[h_2(y)]$ bereits durch einen anderen Schlüssel z besetzt 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** $x =$ NIL **then return fi**
(c) exchange x und $T_2[h_2(x)]$
(d) **if** $x =$ NIL **then return fi**
od
rehash(); *Insert*(x)

Für die Analyse der Zeitkomplexität nehmen wir an, dass die Schleife t-mal durchlaufen wird (wobei $t \leq$ 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

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

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

Angenommen, wir haben $(1, v)$-universelle Hashfunktionen h_1 und h_2, dann passiert jede Möglichkeit nur mit einer Wahrscheinlichkeit von n^{-2v}. Falls $n \geq (1+\delta)m$ für eine Konstante $\delta > 0$, dann ist die Wahrscheinlichkeit für den Fall 1 höchstens

$$\sum_{v=3}^{l} v^3 n^{v-1} m^{v-1} n^{-2v} \leq \frac{1}{nm} \sum_{v=3}^{\infty} v^3 (m/n)^v = \mathcal{O}(\frac{1}{m^2}) .$$

Zweiter Fall:

Lemma 39

Im zweiten Fall gibt es eine Schlüsselfolge der Länge mindestens $l/3$ in $x_1, \ldots, x_l$, 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.

Es muss eine der Folgen $x_1, \ldots, x_i$ bzw. $x_{i+j+1}, \ldots, x_l$ die Länge mindestens $l/3$ haben.

Beweis (Forts.):

Sei also $x'_1, \ldots, x'_v$ eine solche Folge verschiedener Schlüssel in $x_1, \ldots, x_{2t}$ 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

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

Gegeben x'_1, so gibt es m^{v-1} mögliche Folgen von Schlüsseln $x'_2, \ldots, x'_v$. Für jede solche Folge gibt es zwei Möglichkeiten für (i_1, i_2). Weiterhin ist die Wahrscheinlichkeit, dass die obigen Positionsübereinstimmungen gelten, höchstens $n^{-(v-1)}$, 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

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

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

Beweis (Forts.):

Zusammen ergibt sich für die Laufzeit von *Insert*:

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

□

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. □

Originalarbeiten zu Hashverfahren:

J. Lawrence Carter, Mark N. Wegman:
Universal Classes of Hash Functions,
Proc. STOC 1977, pp. 106–112 (1977)

Gaston H. Gonnet:
Expected Length of the Longest Probe Sequence in Hash Code Searching,
Journal of the ACM **28**(2), pp. 289–304 (1981)

Martin Dietzfelbinger et al.:
Dynamic Perfect Hashing: Upper and Lower Bounds,
SIAM J. Comput. **23**(4), pp. 738–761 (1994)

Und weiter:

Rasmus Pagh, Flemming Friche Rodler:
Cuckoo Hashing,
Proc. ESA 2001, LNCS **2161**, pp. 121–133 (2001)

Luc Devroye, Pat Morin, Alfredo Viola:
On Worst Case Robin-Hood Hasing,
McGill Univ., TR **0212** (2002)