4.4 Perfektes Hashing

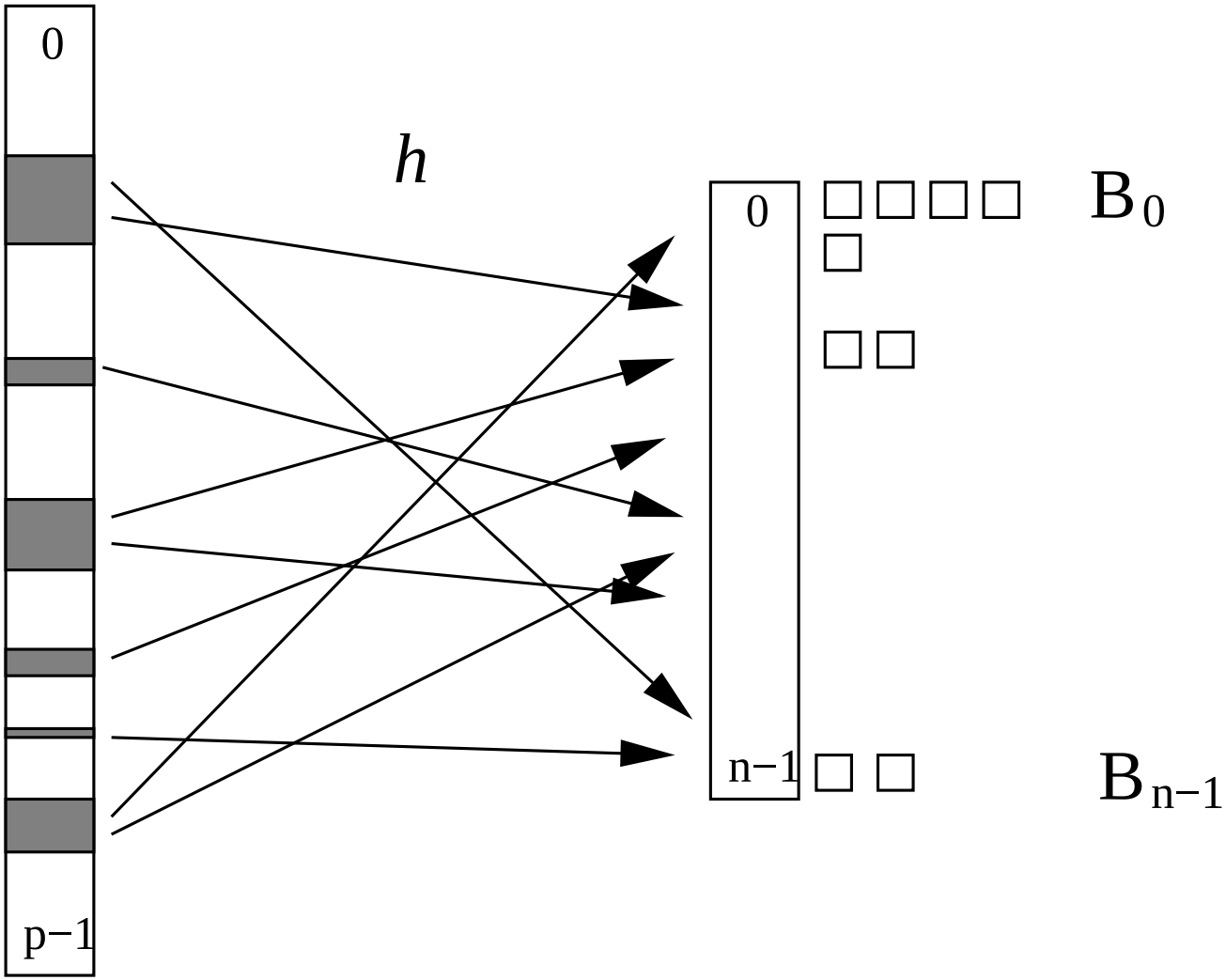
Das Ziel des perfekten Hashings ist es, für eine Schlüsselmenge eine Hashfunktion zu finden, so dass keine Kollisionen auftreten. Die Größe der Hashtabelle soll dabei natürlich möglichst klein sein.

4.4.1 Statisches perfektes Hashing

Sei $U = \{0, 1, \ldots, p-1\}$, p prim, das Universum, $n \in \mathbb{N}$ die Größe des Bildbereichs $\{0, 1, \ldots, n-1\}$ der Hashfunktionen und $S \subseteq U$, $|S| = m \leq n$, eine Menge von Schlüsseln.

Eine Hashfunktion $h : U \to \{0, 1, \ldots, n-1\}$ partitioniert S in „Buckets“

$$B_i = \{x \in S;\ h(x) = i\},\ \text{für } i = 0, 1, \ldots, n-1\,.$$



Hashfunktion h mit Buckets B_i

Definition 33

$\mathcal{H} = \mathcal{H}_{2,n}$ bezeichne die Klasse aller Funktionen

$$h_{a,b} : U \to \{0, 1, \ldots, n-1\}$$

mit

$$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod n \text{ für alle } x \in U\,,$$

wobei $0 < a < p$ und $0 \leq b < p$.

Lemma 34

$\mathcal{H}$ ist universell, d.h. für alle $x, y \in U$ mit $x \neq y$ gilt

$$\Pr[h(x) = h(y)] \leq \frac{1}{n}\,,$$

wenn h zufällig und gleichverteilt aus $\mathcal{H}$ gewählt wird.

Beweis:

Sei $h_{a,b}(x) = h_{a,b}(y) = i$. Dann ist

$$i = \underbrace{(ax + b) \bmod p}_{\alpha} = \underbrace{(ay + b) \bmod p}_{\beta} \pmod{n}$$

Sei $\alpha \in \{0, \ldots, p-1\}$ fest. Dann gibt es in der obigen Kongruenz $\lceil p/n \rceil - 1$ Möglichkeiten für β, nämlich

$$\beta \in \{i, i+n, i+2n, \ldots\} \setminus \{\alpha\},$$

da $\alpha \neq \beta$ und $x \neq y$ gilt.

Beweis:

Also gibt es höchstens

$$p \cdot \left(\left\lceil \frac{p}{n} \right\rceil - 1 \right) = p \cdot \left(\left(\left\lfloor \frac{p-1}{n} \right\rfloor + 1 \right) - 1 \right) \leq \frac{p(p-1)}{n}$$

Möglichkeiten für das Paar (α, β). Jedes Paar (α, β) bestimmt aber genau ein Paar (a, b), da $\mathbb{Z}_p$ ein Körper ist.

Weil es insgesamt $p(p-1)$ Paare (a, b) gibt und h uniform zufällig aus $\mathcal{H}$ ausgewählt wird, folgt

$$\Pr[h(x) = h(y)] \leq \frac{p(p-1)/n}{p(p-1)} = \frac{1}{n}$$

für jedes Paar $x, y \in U$ mit $x \neq y$. □

Lemma 35

Sei $S \subseteq U$, $|S| = m$. Dann gilt:

1. $$\mathbb{E}\left[\sum_{i=0}^{n-1} \binom{|B_i|}{2}\right] \leq \frac{m(m-1)}{2n}$$

2. $$\mathbb{E}[\sum_{i=0}^{n-1} |B_i|^2] \leq \frac{m(m-1)}{n} + m$$

3. $$\Pr[h_{a,b} \textit{ ist injektiv auf } S] \geq 1 - \frac{m(m-1)}{2n}$$

4. $$\Pr[\sum_{i=0}^{n-1} |B_i|^2 < 4m] > \frac{1}{2}, \textit{ falls } m \leq n$$

Beweis:

Definiere die Zufallsvariablen $X_{\{x,y\}}$ für alle $\{x,y\} \subseteq S$ gemäß

$$X_{\{x,y\}} = \begin{cases} 1 & \text{falls } h(x) = h(y)\,, \\ 0 & \text{sonst}\,. \end{cases}$$

Wegen Lemma 34 gilt $\mathbb{E}[X_{\{x,y\}}] = \Pr[h(x) = h(y)] \leq 1/n$ für alle Paare $\{x,y\} \subseteq S$. Weiter ist

$$\mathbb{E}\left[\sum_{i=0}^{n-1} \binom{|B_i|}{2}\right] = |\{\{x,y\} \subseteq S;\ h(x) = h(y)\}|$$
$$\leq \binom{m}{2} \cdot \frac{1}{n}\,.$$

Beweis (Forts.):

Da $x^2 = 2 \cdot \binom{x}{2} + x$ für alle $x \in \mathbb{N}$, folgt

$$\mathbb{E}[\sum_{i=0}^{n-1} |B_i|^2] = \mathbb{E}\left[\sum_{i=0}^{n-1} \left(2 \cdot \binom{|B_i|}{2} + |B_i|\right)\right]$$
$$\stackrel{(1)}{\leq} 2 \cdot \frac{m(m-1)}{2n} + m\,.$$

Aus der Markov-Ungleichung ($\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$ für alle $t > 0$) folgt

$$\Pr[h_{a,b} \text{ nicht injektiv auf } S] = \Pr\left[\sum_{i=0}^{n-1} \binom{|B_i|}{2} \geq 1\right]$$
$$\stackrel{(1)}{\leq} \frac{m(m-1)}{2n}\,.$$

Beweis (Forts.):

Für $m \leq n$ folgt aus (2), dass

$$\mathbb{E}[\sum_{i=0}^{n-1} |B_i|^2] \leq m + m = 2m\,.$$

Also folgt, wiederum mit Hilfe der Markov-Ungleichung, dass

$$\Pr\left[\sum_{i=0}^{n-1} |B_i|^2 > 4m\right] \leq \frac{1}{4m} \cdot 2m = \frac{1}{2}\,.$$

□

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)

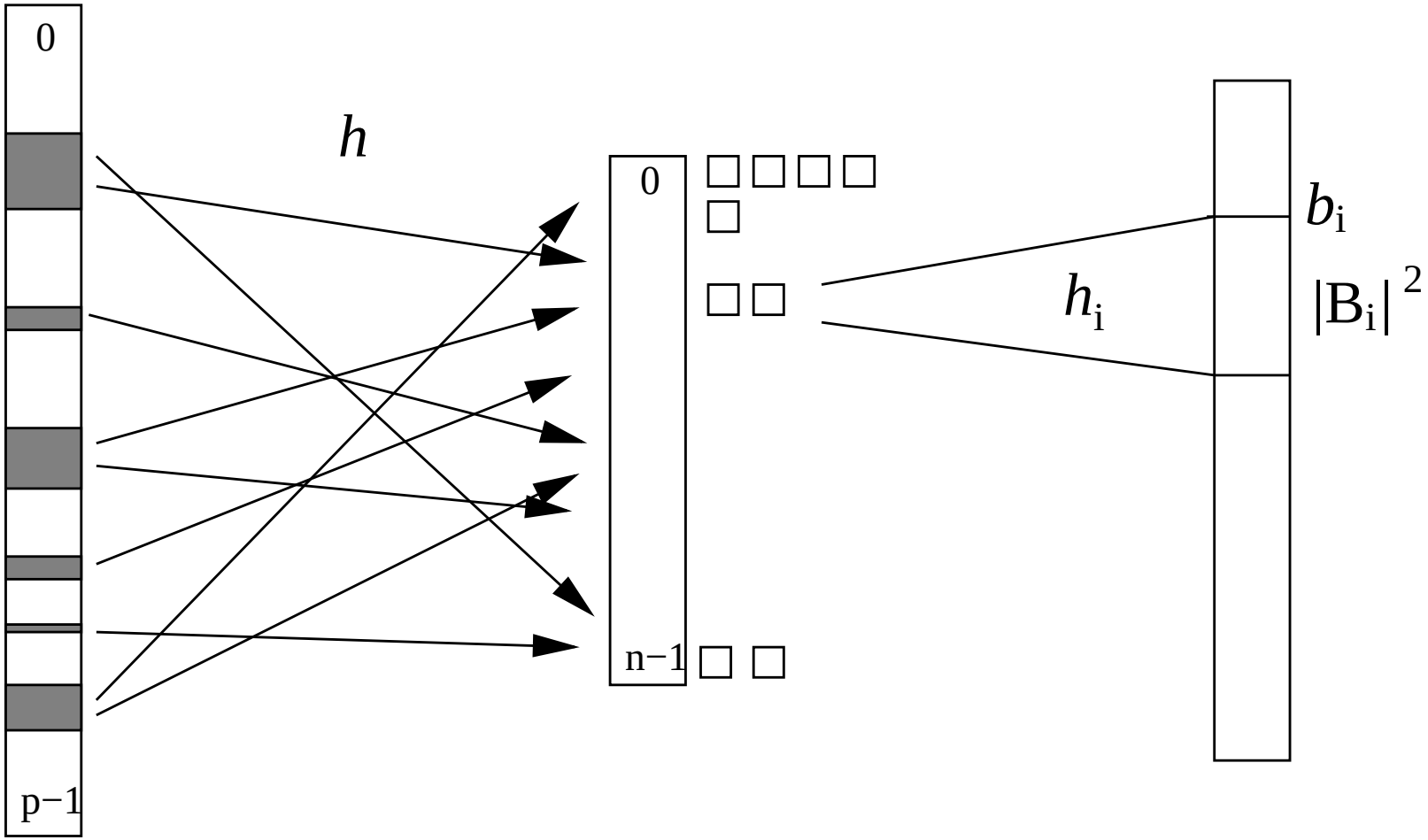
verwendet ein zweistufiges Hashverfahren.

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 35 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 $i \in \{0, \ldots, n-1\}$ gilt, ebenfalls gemäß Lemma 35, 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 $U = \{0, \ldots, p-1\}$ 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

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

□

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)