1 Perfektes Hashing

In diesem Kapitel werden wir Verfahren zum perfekten Hashing vorstellen. Das Ziel im perfekten Hashing ist es, eine Abbildung einer Schlüsselmenge auf eine Hashtabelle zu finden, so dass es keine Kollisionen zwischen den Schlüsseln gibt, d.h. jeder Schlüssel einer anderen Position in der Hashtabelle zugewiesen wird. Zunächst werden wir ein Verfahren für statisches perfektes Hashing vorführen, d.h. die Schlüsselmenge ist fest vorgegeben, und nur Lookup Anfragen sind erlaubt, und danach werden wir ein Verfahren für dynamisches perfektes Hashing vorstellen, d.h. es sind auch Insert und Delete Anfragen erlaubt.

1.1 Statisches perfektes Hashing

Wir starten mit einigen Vereinbarungen danach werden wir einige Hilfssätze formulieren, die für die Analyse der perfekten Hashtabelle wichtig sind. Im folgenden bezeichnet stets

- $U = \{0, 1, \ldots, p-1\}$ (p Primzahl) das *Universum*, d.h. die Menge aller möglichen Schlüsselwerte,
- $x, y, \ldots \in U$: Schlüssel,
- $s \in \mathbb{N}$ die Größe des Bildbereichs $\{0, \ldots, s-1\}$ einer Hashfunktion, und
- $S \subseteq U$, $|S| = n$, eine Schlüsselmenge.

Eine Hashfunktion $h : U \to \{0, \ldots, s-1\}$ zerlegt S in "Buckets" $B_i = \{x \in S \mid h(x) = i\}$, $0 \leq i < s$.

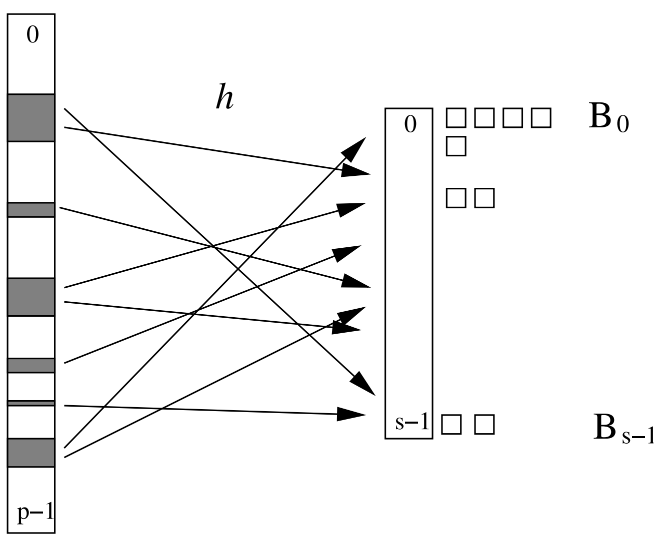

Abbildung 1: Veranschaulichung einer Hashfunktion h mit Buckets B_i.

Definition 1.1 *$\mathcal{H}_s$ bezeichnet die Klasse aller Funktionen*

$$h_{a,b} : U \to \{0, \dots, s-1\}$$

mit

$$h_{a,b}(x) = ((a \cdot x + b) \textit{ mod } p) \textit{ mod } s \quad \textit{für alle } x \in U$$

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

Lemma 1.2 *$\mathcal{H}_s$ ist* universell*, d.h. für alle $x, y \in U$ mit $x \neq y$ gilt*

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

für eine zufällig as $\mathcal{H}_s$ ausgewählte Hashfunktion h.

Beweis. Angenommen, $i = h_{a,b}(x) = h_{a,b}(y)$, so ist

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

Wir zählen nun die Anzahl der Möglichkeiten dafür, dass diese Kongruenz gilt. Sei $\alpha \in \{0, \dots, p-1\}$ fest. So gibt es noch $\lceil p/s \rceil - 1$ Möglichkeiten für β ($\beta \in \{i, s+i, 2s+i, \dots\} \setminus \{\alpha\}$), da $\alpha \neq \beta$ und $x \neq y$ gilt. Also gibt es höchstens

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

viele Möglichkeiten für das Paar (α, β). Jedes Paar (α, β) bestimmt genau ein Paar (a, b), das obige Kongruenz erfüllt, weil $\{0, \dots, p-1\}$ mit $+$ und $\cdot$ (mod p) ein Körper ist. Da es aber insgesamt $p(p-1)$ viele Paare (a, b) gibt und h uniform zufällig aus $\mathcal{H}_s$ ausgewählt ist, folgt, dass

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

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

Lemma 1.3 *Sei $S \subseteq U$ mit $|S| = n$, dann gilt:*

(a) $\mathrm{E}[\sum_{i=0}^{s-1} \binom{|B_i|}{s}] \leq \frac{n(n-1)}{2s}$

(b) $\mathrm{E}[\sum_{i=0}^{s-1} |B_i|^2] \leq \frac{n(n-1)}{s} + n$

(c) $\Pr[h_{a,b}$ *ist injektiv auf* $S] \geq 1 - \frac{n(n-1)}{2s}$

(d) $\Pr[\sum_{i=0}^{s-1} |B_i|^2 < 4n] > \frac{1}{2}$ *falls* $n \leq s$

Beweis. (a): Definiere die Zufallsvariable $X_{\{x,y\}}$ für alle $\{x,y\} \subseteq S$ als

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

Wegen Lemma 1.2 gilt $\mathrm{E}[X_\{x,y\}] = \Pr[h(x) = h(y)] \leq 1/s$ für alle $\{x,y\} \subseteq S$. Weiterhin ist

$$\begin{aligned} \sum_{i=0}^{s-1} \binom{|B_i|^2}{2} &= |\{\{x,y\} \subseteq S \mid h(x) = h(y)\}| \\ &= \sum_{\{x,y\} \in S} X_{\{x,y\}} \end{aligned}$$

Aus der Linearität des Erwartungswertes folgt daher

$$\begin{aligned} \mathrm{E}\left[\sum_{i=0}^{s-1} \binom{|B_i|^2}{2}\right] &= \mathrm{E}\left[\sum_{\{x,y\} \in S} X_{\{x,y\}}\right] = \sum_{\{x,y\} \in S} \mathrm{E}[X_{\{x,y\}}] \\ &\leq \binom{n}{2} \cdot \frac{1}{s} \end{aligned}$$

(b): Da $x^2 = 2 \cdot \binom{x}{2} + x$ für jedes $x \in \mathbb{N}$, gilt

$$\begin{aligned} \mathrm{E}\left[\sum_{i=0}^{s-1} |B_i|^2\right] &= \mathrm{E}\left[\sum_{i=0}^{s-1} \left(2 \cdot \binom{|B_i|^2}{2} + |B_i|\right)\right] \\ &\overset{(a)}{\leq} 2 \cdot \frac{n(n-1)}{2s} + n \end{aligned}$$

(c): Wegen der Markov Ungleichung ($\Pr[X \geq t] \leq \frac{1}{t}\mathrm{E}[X]$ für alle $t \geq 0$) gilt

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

(d): Für $n \leq s$ folgt aus (b), dass $\mathrm{E}[\sum |B_i|^2] \leq n + n = 2n$. Also gilt wegen der Markov Ungleichung, dass

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

□

Nun sind wir soweit, den Aufbau der perfekten Hashtabelle nach Fredman, Komlos und Szemeredi (Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM* 31(3), 1984, Seiten 538–549) vorzustellen. Die Idee bei der Tabelle ist es, ein zweistufiges Hashverfahren zu verwenden. Für einen gegebenen Schlüssel x wird zunächst $i = h(x)$ berechnet, um über den Tabellenplatz $T[i]$ b_i, $|B_i|$ und $h_i \in \mathcal{H}_{|B_i|^2}$ zu ermitteln, und danach wird im Tabellenplatz $T'[b_i + h_i(x)]$ nachgeschaut, ob x darin ist. Falls das der Fall ist, wird *true* ausgegeben und sonst *false*. Falls $\sum |B_i|^2 < 4n$ ist, so wird nur $O(n)$ Platz verwendet. (Siehe auch Abb. 2)

Der Algorithmus zur Aufbau der Hashtabelle arbeitet wie folgt:

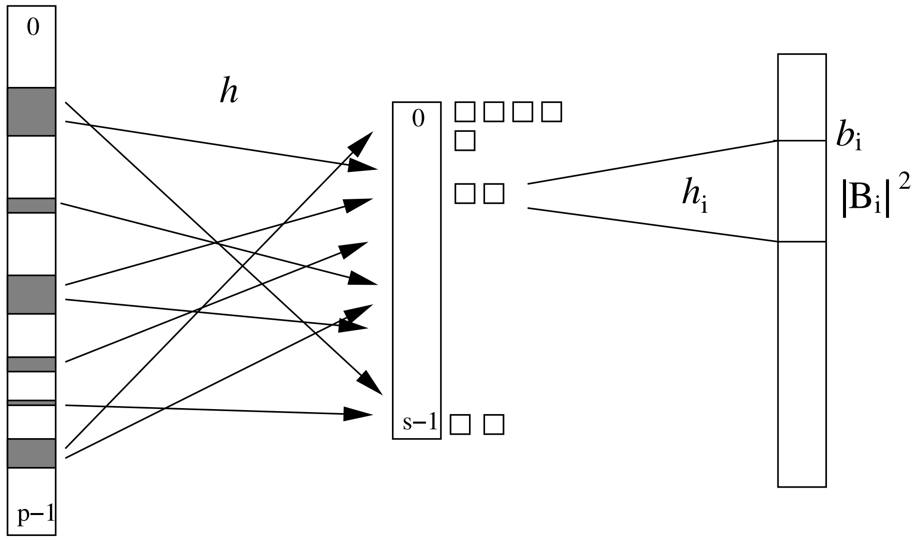

Abbildung 2: Aufbau der zweistufigen Hashtabelle von Fredman, Komlos und Szemeredi.

- Eingabe: $S \subseteq U$, $|S| = n$
- Ausgabe: Hashtabelle nach Abb. 2
- Methode:
  1. Wähle $h \in \mathcal{H}_s$ zufällig. Berechne $h(x)$ für alle $x \in S$.
  2. Falls $\sum_i |B_i|^2 \geq 4n$, dann wiederhole 1.
  3. Kontruierte die Mengen B_i für alle $0 \leq i < s$.
  4. Für $i = 0$ bis $s - 1$ tue
     (a) Wähle $h_i \in \mathcal{H}_{|B_i|^2}$ zufällig.
     (b) Falls $h_i|_{B_i}$ nicht injektiv ist, wiederhole (a).

Es ist einfach zu sehen, dass wenn der Algorithmus terminiert, er eine Hashtabelle mit $O(n)$ Platz konstruiert. Die Frage ist also nur, wie lange der Algorithmus braucht, um zu terminieren. Schauen wir uns zunächst die (1-2)-Scheife an. Ein einmaliger Durchlauf dieser Schleife kostet $O(n)$ Zeit. Weiterhin ist nach Lemma 1.3(d) die Wahrscheinlichkeit dafür, dass Schritt 1 wiederholt werden muss, höchstens $1/2$ für jedes neue h. Also ist

$$\Pr[\text{(1-2)-Scheife wird} > k\text{-mal durchlaufen}] \leq \left(\frac{1}{2}\right)^k$$

Da für eine Zufallsvariable X auf den natürlichen Zahlen gilt

$$\mathrm{E}[X] = \sum_{i=1}^{\infty} i \cdot \Pr[X = i] = \sum_{i=1}^{\infty} \Pr[X \geq i]$$

folgt

$$\mathrm{E}[\text{\# (1-2)-Scheifendurchläufe}] \leq \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = 2$$

Also ist die erwartete Zeit für die (1-2)-Scheife $O(n)$. Schritt 3 kostet offensichtlich $O(n)$ Zeit. Für jedes i mit $0 \leq i < s$ gilt weiterhin nach Lemma 1.3, 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 folgt wie für die (1-2)-Scheife, dass die erwartete Anzahl der Durchläufe der (a-b)-Scheife höchstens 2 ist und damit

$$\mathrm{E}[\text{Zeit für } B_i] = O(|B_i|^2)$$

ist. Daraus ergibt sich, dass

$$\mathrm{E}[\text{Gesamtzeit für Schritt 4}] = O(\sum_i |B_i|^2) = O(n)$$

1.2 Dynamisches perfektes Hashing

Bevor wir eine Methode zum dynamischen Hashing vorstellen, brauchen wir einige mathematische Grundlagen. Zur Erinnerung: $U = \{0, \ldots, p-1\}$ für eine Primuzahl p.

Definition 1.4 *$\mathcal{H}_s^k$ bezeichne die Klasse aller Polynome $h_{\vec{a}}$ vom Grad höchstens $k-1$ mit*

$$h_{\vec{a}}(x) = \left(\left(\sum_{j=0}^{k-1} a_j x^j \right) \textit{ mod } p \right) \textit{ mod } s \quad \textit{für jedes } x \in U$$

und $\vec{a} = (a_0, \ldots, a_{k-1}) \in U^k$.

Definition 1.5 *Eine Klasse $\mathcal{H}$ von Funktionen von U nach $\{0, \ldots, s-1\}$ heit (c,k)*-universell*, falls für alle paarweise verschiedene $x_0, \ldots, x_{k-1} \in U$ und für alle $i_0, \ldots, i_{k-1} \in \{0, \ldots, s-1\}$ gilt, dass*

$$\Pr[h(x_0) = i_0 \;\wedge \ldots \wedge\; h(x_{k-1}) = i_{k-1}] \leq \frac{c}{s^k}$$

Theorem 1.6 *$\mathcal{H}_s^k$ ist (c,k)-universell mit $c = (1 + \frac{s}{p})^k$*

Beweis. Da U mit $+$ und $\cdot$ (mod p) ein Körper ist, gibt es für jedes Tupel $(y_0, \ldots, y_{k-1}) \in U^k$ genau ein Tupel $(a_0, \ldots, a_{k-1}) \in U^k$ mit

$$\sum_{j=0}^{k-1} a_j x_r^j \equiv y_r \quad (\bmod p)$$

für alle $0 \leq r < k$. Daraus folgt, dass

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

da jedes y_r die Werte $\{i_r, i_r+s, i_r+2s, \ldots\}$ annehmen kann. Da es insgesamt p^k viele Möglichkeiten für $\vec{a}$ gibt, folgt daraus

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

□

Daraus lässt sich auch einfach folgern, dass $\mathcal{H}_s^k$ (c, k')-universell mit $c = (1 + \frac{s}{p})^{k'}$ für alle $1 \leq k' \leq k$ ist.

Jetzt sind wir soweit, ein dynamisches perfektes Hashverfahren vorzustellen, dass Kuckuck Hashing heißt. Kuckuck Hashing arbeitet mit zwei Hashtabellen, T_1 und T_2, die je aus den Positionen $\{0, \ldots, s-1\}$ bestehen. Weiterhin benötigt es zwei $(1+\delta, O(\log n))$-universelle Hashfunktionen h_1 und h_2 für ein genügend kleines $\delta > 0$, die die Schlüsselmenge U auf $\{0, \ldots, s-1\}$ abbilden. Jeder Schlüssel $x \in S$ wird entweder in Position $h_1(x)$ in T_1 oder Position $h_2(x)$ in T_2 gespeichert, aber nicht beiden. In diesem Fall ist die Lookup(x) Operation sehr einfach:

- return $T_1[h_1(x)] = x \;\vee\; T_2[h_2(x)] = x$

Die Insert Operation verwendet nun das Kuckucksprinzip, um neue Schlüssel einzufügen. Gegeben ein Schlüssel x, so wird zunächst versucht, x in $T_1[h_1(x)]$ unterzubringen. Ist das erfolgreich, sind wir fertig. Falls aber $T_1[h_1(x)]$ bereits durch einen 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 fertig. Falls $T_2[h_2(y)]$ bereits durch einen Schlüssel z besetzt ist, nehmen wir z heraus und fügen stattdessen y in $T_2[h_2(y)]$. 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 Lookup Operation wie folgt:

1. if Lookup(x) then return
2. loop MaxLoop times
   (a) $x \leftrightarrow T_1[h_1(x)]$
   (b) if $x = \perp$ then return
   (c) $x \leftrightarrow T_2[h_2(x)]$
   (d) if $x = \perp$ then return
   end loop
3. rehash(); Insert(x)

Die Frage ist natürlich, ob diese Insert Operation (mit hoher Wahrscheinlichkeit) terminiert, und wenn ja, wieviel Zeit sie benötigt. Dazu wollen wir die Wahrscheinlichkeit berechnen, dass die Insert-Scheife mindestens t-mal durchlaufen wird (wobei wir annehmen, dass $t <$MaxLoop). Dazu müssen wir zwei Fälle betrachten:

1. Die Insert Operation formt eine Endlosschleife während der ersten t Runden, d.h., x_ℓ in Abb. 3 wurde zu einer bereits besuchten Position verschoben mit $\ell \leq 2t$.

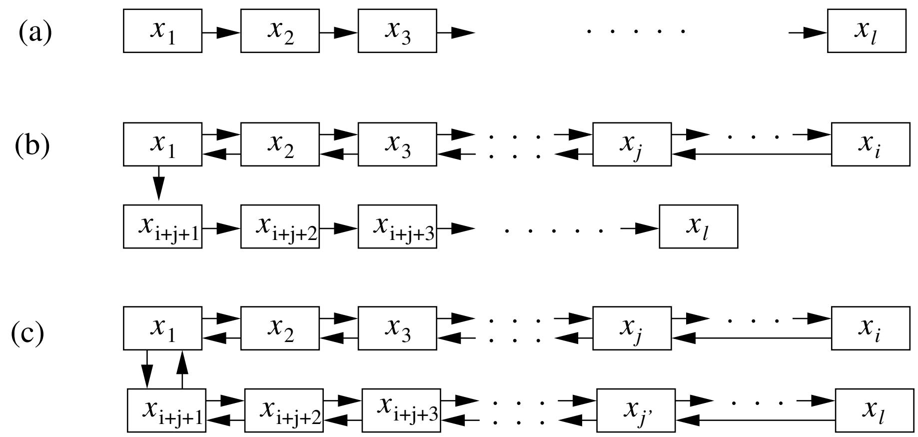

Abbildung 3: Drei Fälle für den Ausgang der Insert Operation. (a): Es wird keine Position zweimal besucht. (b): x_i besucht die Position von x_j, was dazu führt, dass alle Schlüssel $x_j, \dots, x_2$ wieder in ihre Ausgangspositionen zurückgeschoben werden. x_1 versucht daraufhin die andere Alternativposition, und hier terminiert die Kuckucksregel in x_ℓ. (c): x_i besucht die Position von x_j und x_ℓ besucht die Position von $x_{j'}$, was zu einer Endlosschleife führt.

2. Die Insert Operation formt keine Endlosschleife wärend der ersten t Runden.

Wir untersuchen zunächst den ersten Fall. Sei $v \leq \ell$ 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 in Abb. 3 gibt, s^{v-1} viele Möglichkeiten für die Positionen der Schlüssel gibt, und n^{v-1} viele Möglichkeiten für die Schlüssel außer x_1 gibt. Angenommen, wir haben $(1, v)$-universelle Hashfunktionen, dann passiert jede Möglichkeit nur mit einer Wahrscheinlichkeit von s^{-2v}. Falls nun $s \geq (1+\epsilon)n$ für eine Konstante $\epsilon > 0$, dann ist die Wahrscheinlichkeit für den Fall 1 höchstens

$$\sum_{v=3}^{\ell} v^3 s^{v-1} n^{v-1} s^{-2v} \leq \frac{1}{sn} \sum_{v=3}^{\infty} v^3 (n/s)^v = O(1/n^2)$$

Für den zweiten Fall benötigen wir das folgende Lemma.

Lemma 1.7 *Angenommen, die* Insert *Operation formt keine Endlosschleife nach ℓ besuchten Schlüsseln. Dann gibt es eine Schlüsselfolge in der Länge mindestens $\ell/3$ in $x_1, \dots, x_\ell$, in der alle Schlüssel verschieden sind.*

Beweis. Falls die Insert Operation niemals zu einer bereits besuchten Position zurückkehrt, die das Lemma wahr. Nehmen wir also an, dass die Operation zu einer bereits besuchten Position zurückkehrt, und seien i und j so definiert wie in Abb. 3. Falls $\ell < i+j$, dann bilden die ersten $j-1 \geq (i+j-1)/2 \geq \ell/2$ Schlüssel die gesuchte Folge. Für $\ell \geq i+j$ muss eine der Folgen $x_1, \dots x_{j-1}$ und $x_{i+j-1}, \dots, x_\ell$ die Länge mindestens $\ell/3$ haben. $\square$

Konzentrieren wir uns also auf eine Folge $x'_1, \ldots, x'_v$ 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

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

Gegeben x'_1, so gibt es n^{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 Positionsübereinstimmungen oben 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 erfüllt ist, höchstens

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

Diese Wahrscheinlichkeit ist polynomiell klein in n, falls $t = \Omega(\log n)$ ist. D.h. es reicht die Verwendung von $(1+\delta, O(\log n))$-universellen Hashfunktionen für ein genügend kleines $\delta > 0$, um mit hoher Wahrscheinlichkeit eine Terminierung der Insert Operation ohne ein Rehashing sicherzustellen.

Fügt man die beiden Fälle oben zusammen, so ergibt sich eine erwartete Laufzeit der Insert Operation von

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

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