Script generated by TTT

Title: T?ubig: GAD (15.05.2012)

Date: Tue May 15 14:34:23 CEST 2012

Duration: 80:14 min

Pages: 35

Hashing with Chaining

Betrachte als Hashfunktionsmenge die Menge aller Funktionen, die die Schlüsselmenge (mit Kardinalität N) auf die Zahlen $0,\ldots,m-1$ abbilden.

Satz

Falls n Elemente in einer Hashtabelle T der Größe m mittels einer zufälligen Hashfunktion h gespeichert werden, dann ist die erwartete Laufzeit von remove bzw. find in $O(1+n/m)$.

Unrealistisch: es gibt m^N solche Funktionen und man braucht $\log_2(m^N) = N\log_2 m$ Bits, um eine Funktion zu spezifizieren.

$\Rightarrow$ widerspricht dem Ziel, den Speicherverbrauch von N auf n zu senken!

Hashing with Chaining

Beweis.

- Betrachte feste Position $i = h(k)$ bei remove(k) oder find(k)
- Laufzeit ist Konstante plus Zeit zum Scan der Liste also $O(1+\mathbb{E}[X])$, wobei X Zufallsvariable für Länge von $T[i]$
- Zufallsvariable $X_e \in \{0,1\}$ für jedes $e \in S$
- $X_e = 1 \Leftrightarrow h(\text{key}(e)) = i$
- Listenlänge $X = \sum_{e\in S} X_e$
- Erwartete Listenlänge $\mathbb{E}[X]$

$$= \mathbb{E}\left[\sum_{e\in S} X_e\right] = \sum_{e\in S}\mathbb{E}[X_e] = \sum_{e\in S} 0\cdot\Pr[X_e=0] + 1\cdot\Pr[X_e=1]$$
$$= \sum_{e\in S}\Pr[X_e=1] = \sum_{e\in S} 1/m = n/m$$

□

Hashing Hashing with Chaining

Hashing with Chaining

Beweis.

- Betrachte feste Position $i = h(k)$ bei remove(k) oder find(k)
- Laufzeit ist Konstante plus Zeit zum Scan der Liste also $O(1+\mathbb{E}[X])$, wobei X Zufallsvariable für Länge von $T[i]$
- Zufallsvariable $X_e \in \{0,1\}$ für jedes $e \in S$
- $X_e = 1 \Leftrightarrow h(\text{key}(e)) = i$
- Listenlänge $X = \sum_{e\in S} X_e$
- Erwartete Listenlänge $\mathbb{E}[X]$

$$= \mathbb{E}\left[\sum_{e\in S} X_e\right] = \sum_{e\in S}\mathbb{E}[X_e] = \sum_{e\in S} 0\cdot\Pr[X_e=0] + 1\cdot\Pr[X_e=1]$$
$$= \sum_{e\in S}\Pr[X_e=1] = \sum_{e\in S} 1/m = n/m$$

□

H. Täubig (TUM) GAD SS'12 172 / 631

Hashing with Chaining

Beweis.

- Betrachte feste Position $i = h(k)$ bei remove(k) oder find(k)
- Laufzeit ist Konstante plus Zeit zum Scan der Liste
  also $O(1 + \mathbb{E}[X])$, wobei X Zufallsvariable für Länge von $T[i]$
- Zufallsvariable $X_e \in \{0, 1\}$ für jedes $e \in S$
- $X_e = 1 \Leftrightarrow h(\text{key}(e)) = i$
- Listenlänge $X = \sum_{e \in S} X_e$
- Erwartete Listenlänge $\mathbb{E}[X]$

$$= \mathbb{E}\left[\sum_{e \in S} X_e\right] = \sum_{e \in S} \mathbb{E}[X_e] = \sum_{e \in S} 0 \cdot \Pr[X_e = 0] + 1 \cdot \Pr[X_e = 1]$$
$$= \sum_{e \in S} \Pr[X_e = 1] = \sum_{e \in S} 1/m = n/m$$

□

Übersicht

$\forall x, y \in \text{Key}$
$x \neq y$

c-universelle Familien von Hashfunktionen

Wie konstruiert man zufällige Hashfunktionen?

Definition

Sei c eine positive Konstante.

Eine Familie H von Hashfunktionen auf $\{0, \ldots, m-1\}$ heißt c-universell, falls für jedes Paar $x \neq y$ von Schlüsseln gilt, dass

$$\left|\{h \in H : h(x) = h(y)\}\right| \leq \frac{c}{m}|H|$$

D.h. für eine zufällige Hashfunktion $h \in H$ gilt

$$\Pr[h(x) = h(y)] \leq \frac{c}{m}$$

1-universelle Familien nennt man universell.

c-Universal Hashing with Chaining

Satz

Falls n Elemente in einer Hashtabelle der Größe m mittels einer zufälligen Hashfunktion h aus einer c-universellen Familie gespeichert werden, dann ist die erwartete Laufzeit von remove bzw. find in $O(1 + c \cdot n/m)$.

Beweis.

- Betrachte festen Schlüssel k
- Zugriffszeit X ist $O(1 +$ Länge der Liste $T[h(k)])$
- Zufallsvariable $X_e \in \{0, 1\}$ für jedes $e \in S$ zeigt an, ob e auf die gleiche Position wie k gehasht wird

c-Universal Hashing with Chaining

Satz

Falls n Elemente in einer Hashtabelle der Größe m mittels einer zufälligen Hashfunktion h aus einer c-universellen Familie gespeichert werden, dann ist die erwartete Laufzeit von remove bzw. find in $O(1 + c \cdot n/m)$.

Beweis.

- Betrachte festen Schlüssel k
- Zugriffszeit X ist $O(1 +$ Länge der Liste $T[h(k)])$
- Zufallsvariable $X_e \in \{0, 1\}$ für jedes $e \in S$ zeigt an, ob e auf die gleiche Position wie k gehasht wird

c-Universal Hashing with Chaining

Beweis.

- $X_e = 1 \Leftrightarrow h(\text{key}(e)) = h(k)$
- Listenlänge $X = \sum_{e \in S} X_e$
- Erwartete Listenlänge

$$\begin{aligned}\mathbb{E}[X] &= \mathbb{E}\left[\sum_{e \in S} X_e\right] \\ &= \sum_{e \in S} \mathbb{E}[X_e] = \sum_{e \in S} 0 \cdot \Pr[X_e = 0] + 1 \cdot \Pr[X_e = 1] \\ &= \sum_{e \in S} \Pr[X_e = 1] \leq \sum_{e \in S} c/m = n \cdot c/m\end{aligned}$$

□

Hashing Universelles Hashing

c-Universal Hashing with Chaining

Satz

Falls n Elemente in einer Hashtabelle der Größe m mittels einer zufälligen Hashfunktion h aus einer c-universellen Familie gespeichert werden, dann ist die erwartete Laufzeit von remove bzw. find in $O(1 + c \cdot n/m)$.

Beweis.

- Betrachte festen Schlüssel k
- Zugriffszeit X ist $O(1 +$ Länge der Liste $T[h(k)])$
- Zufallsvariable $X_e \in \{0, 1\}$ für jedes $e \in S$ zeigt an, ob e auf die gleiche Position wie k gehasht wird

H. Täubig (TUM) GAD SS'12 175 / 631

Hashing Universelles Hashing

c-Universal Hashing with Chaining

Beweis.

- $X_e = 1 \Leftrightarrow h(\text{key}(e)) = h(k)$
- Listenlänge $X = \sum_{e \in S} X_e$
- Erwartete Listenlänge

$$\begin{aligned}\mathbb{E}[X] &= \mathbb{E}\left[\sum_{e \in S} X_e\right] \\ &= \sum_{e \in S} \mathbb{E}[X_e] = \sum_{e \in S} 0 \cdot \Pr[X_e = 0] + 1 \cdot \Pr[X_e = 1] \\ &= \sum_{e \in S} \Pr[X_e = 1] \leq \sum_{e \in S} c/m = n \cdot c/m\end{aligned}$$

□

H. Täubig (TUM) GAD SS'12 176 / 631

Beispiele für c-universelles Hashing

Einfache c-universelle Hashfunktionen?

Annahme: Schlüssel sind Bitstrings einer bestimmten Länge

Wähle als Tabellengröße m eine Primzahl

$\Rightarrow$ dann ist der Restklassenring modulo m (also $\mathbb{Z}_m$) ein Körper, d.h. es gibt zu jedem Element außer für die Null genau ein Inverses bzgl. Multiplikation

- Sei $w = \lfloor \log_2 m \rfloor$.
- unterteile die Bitstrings der Schlüssel in Teile zu je w Bits
- Anzahl der Teile sei k
- interpretiere jeden Teil als Zahl aus dem Intervall $[0, \ldots, 2^w - 1]$
- interpretiere Schlüssel x als k-Tupel solcher Zahlen:

$$\mathbf{x} = (x_1, \ldots, x_k)$$

Familie für universelles Hashing

Definiere für jeden Vektor

$$\mathbf{a} = (a_1, \ldots, a_k) \in \{0, \ldots, m-1\}^k$$

mittels Skalarprodukt

$$\mathbf{a} \cdot \mathbf{x} = \sum_{i=1}^{k} a_i x_i$$

eine Hashfunktion von der Schlüsselmenge in die Menge der Zahlen $\{0, \ldots, m-1\}$

$$h_{\mathbf{a}}(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x} \mod m$$

Familie für universelles Hashing

Satz

$$H = \left\{ h_{\mathbf{a}} : \mathbf{a} \in \{0, \ldots, m-1\}^k \right\}$$

ist eine (1-)universelle Familie von Hashfunktionen falls m prim ist.

Oder anders:
das Skalarprodukt zwischen einer Tupeldarstellung des Schlüssels und einem Zufallsvektor modulo m definiert eine gute Hashfunktion.

Familie für universelles Hashing

Beispiel

- 32-Bit-Schlüssel, Hashtabellengröße $m = 269$

$\Rightarrow$ Schlüssel unterteilt in $k = 4$ Teile mit $w = \lfloor \log_2 m \rfloor = 8$ Bits

- Schlüssel sind also 4-Tupel von Integers aus dem Intervall $[0, 2^8 - 1] = \{0, \ldots, 255\}$, z.B. $\mathbf{x} = (11, 7, 4, 3)$
- Die Hashfunktion wird auch durch ein 4-Tupel von Integers, aber aus dem Intervall $[0, 269 - 1] = \{0, \ldots, 268\}$, spezifiziert z.B. $\mathbf{a} = (2, 4, 261, 16)$

$\Rightarrow$ Hashfunktion:

$$h_{\mathbf{a}}(\mathbf{x}) = (2x_1 + 4x_2 + 261x_3 + 16x_4) \mod 269$$

$$h_{\mathbf{a}}(\mathbf{x}) = (2 \cdot 11 + 4 \cdot 7 + 261 \cdot 4 + 16 \cdot 3) \mod 269 = 66$$

Familie für universelles Hashing

Beispiel

- 32-Bit-Schlüssel, Hashtabellengröße $m = 269$

⇒ Schlüssel unterteilt in $k = 4$ Teile mit $w = \lfloor \log_2 m \rfloor = 8$ Bits

- Schlüssel sind also 4-Tupel von Integers aus dem Intervall $[0, 2^8 - 1] = \{0, \dots, 255\}$, z.B. $\mathbf{x} = (11, 7, 4, 3)$
- Die Hashfunktion wird auch durch ein 4-Tupel von Integers, aber aus dem Intervall $[0, 269 - 1] = \{0, \dots, 268\}$, spezifiziert z.B. $\mathbf{a} = (2, 4, 261, 16)$

⇒ Hashfunktion:

$$h_{\mathbf{a}}(\mathbf{x}) = (2x_1 + 4x_2 + 261x_3 + 16x_4) \bmod 269$$

$$h_{\mathbf{a}}(\mathbf{x}) = (2 \cdot 11 + 4 \cdot 7 + 261 \cdot 4 + 16 \cdot 3) \bmod 269 = 66$$

Hashing | Universelles Hashing

Familie für universelles Hashing

Beispiel

- 32-Bit-Schlüssel, Hashtabellengröße $m = 269$

⇒ Schlüssel unterteilt in $k = 4$ Teile mit $w = \lfloor \log_2 m \rfloor = 8$ Bits

- Schlüssel sind also 4-Tupel von Integers aus dem Intervall $[0, 2^8 - 1] = \{0, \dots, 255\}$, z.B. $\mathbf{x} = (11, 7, 4, 3)$
- Die Hashfunktion wird auch durch ein 4-Tupel von Integers, aber aus dem Intervall $[0, 269 - 1] = \{0, \dots, 268\}$, spezifiziert z.B. $\mathbf{a} = (2, 4, 261, 16)$

⇒ Hashfunktion:

$$h_{\mathbf{a}}(\mathbf{x}) = (2x_1 + 4x_2 + 261x_3 + 16x_4) \bmod 269$$

$$h_{\mathbf{a}}(\mathbf{x}) = (2 \cdot 11 + 4 \cdot 7 + 261 \cdot 4 + 16 \cdot 3) \bmod 269 = 66$$

H. Täubig (TUM) GAD SS'12 180 / 631

Hashing | Universelles Hashing

Familie für universelles Hashing

Beispiel

- 32-Bit-Schlüssel, Hashtabellengröße $m = 269$

⇒ Schlüssel unterteilt in $k = 4$ Teile mit $w = \lfloor \log_2 m \rfloor = 8$ Bits

- Schlüssel sind also 4-Tupel von Integers aus dem Intervall $[0, 2^8 - 1] = \{0, \dots, 255\}$, z.B. $\mathbf{x} = (11, 7, 4, 3)$
- Die Hashfunktion wird auch durch ein 4-Tupel von Integers, aber aus dem Intervall $[0, 269 - 1] = \{0, \dots, 268\}$, spezifiziert z.B. $\mathbf{a} = (2, 4, 261, 16)$

⇒ Hashfunktion:

$$h_{\mathbf{a}}(\mathbf{x}) = (2x_1 + 4x_2 + 261x_3 + 16x_4) \bmod 269$$

$$h_{\mathbf{a}}(\mathbf{x}) = (2 \cdot 11 + 4 \cdot 7 + 261 \cdot 4 + 16 \cdot 3) \bmod 269 = 66$$

H. Täubig (TUM) GAD SS'12 180 / 631

Hashing | Universelles Hashing

Familie für universelles Hashing

Beispiel

- 32-Bit-Schlüssel, Hashtabellengröße $m = 269$

⇒ Schlüssel unterteilt in $k = 4$ Teile mit $w = \lfloor \log_2 m \rfloor = 8$ Bits

- Schlüssel sind also 4-Tupel von Integers aus dem Intervall $[0, 2^8 - 1] = \{0, \dots, 255\}$, z.B. $\mathbf{x} = (11, 7, 4, 3)$
- Die Hashfunktion wird auch durch ein 4-Tupel von Integers, aber aus dem Intervall $[0, 269 - 1] = \{0, \dots, 268\}$, spezifiziert z.B. $\mathbf{a} = (2, 4, 261, 16)$

⇒ Hashfunktion:

$$h_{\mathbf{a}}(\mathbf{x}) = (2x_1 + 4x_2 + 261x_3 + 16x_4) \bmod 269$$

$$h_{\mathbf{a}}(\mathbf{x}) = (2 \cdot 11 + 4 \cdot 7 + 261 \cdot 4 + 16 \cdot 3) \bmod 269 = 66$$

H. Täubig (TUM) GAD SS'12 180 / 631

Eindeutiges a_j

Beweis.

- Betrachte zwei beliebige verschiedene Schlüssel $\mathbf{x} = \{x_1, \ldots, x_k\}$ und $\mathbf{y} = \{y_1, \ldots, y_k\}$
- Wie groß ist $\Pr[h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})]$?
- Sei j ein Index (von evt. mehreren möglichen) mit $x_j \neq y_j$ (muss es geben, sonst wäre $\mathbf{x} = \mathbf{y}$)

$\Rightarrow$ $(x_j - y_j) \not\equiv 0 \bmod m$
d.h., es gibt genau ein multiplikatives Inverses $(x_j - y_j)^{-1}$

$\Rightarrow$ gegeben Primzahl m und Zahlen $x_j, y_j, b \in \{0, \ldots m-1\}$ hat jede Gleichung der Form

$$a_j(x_j - y_j) \equiv b \bmod m$$

eine eindeutige Lösung: $a_j \equiv (x_j - y_j)^{-1} b \bmod m$

Familie für universelles Hashing

Beispiel

- 32-Bit-Schlüssel, Hashtabellengröße $m = 269$

$\Rightarrow$ Schlüssel unterteilt in $k = 4$ Teile mit $w = \lfloor \log_2 m \rfloor = 8$ Bits

- Schlüssel sind also 4-Tupel von Integers aus dem Intervall $[0, 2^8 - 1] = \{0, \ldots, 255\}$, z.B. $\mathbf{x} = (11, 7, 4, 3)$
- Die Hashfunktion wird auch durch ein 4-Tupel von Integers, aber aus dem Intervall $[0, 269 - 1] = \{0, \ldots, 268\}$, spezifiziert z.B. $\mathbf{a} = (2, 4, 261, 16)$

$\Rightarrow$ Hashfunktion:

$$h_{\mathbf{a}}(\mathbf{x}) = (2x_1 + 4x_2 + 261x_3 + 16x_4) \bmod 269$$

$$h_{\mathbf{a}}(\mathbf{x}) = (2 \cdot 11 + 4 \cdot 7 + 261 \cdot 4 + 16 \cdot 3) \bmod 269 = 66$$

Wann wird $h(\mathbf{x}) = h(\mathbf{y})$?

Beweis.

Wenn man alle Variablen a_i außer a_j festlegt, gibt es exakt eine Wahl für a_j, so dass $h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})$, denn

$$\begin{aligned}
h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y}) \quad &\Leftrightarrow \quad \sum_{i=1}^{k} a_i x_i \equiv \sum_{i=1}^{k} a_i y_i && \bmod m \\
&\Leftrightarrow \quad a_j(x_j - y_j) \equiv \sum_{i \neq j} a_i (y_i - x_i) && \bmod m \\
&\Leftrightarrow \quad a_j \equiv (x_j - y_j)^{-1} \sum_{i \neq j} a_i (y_i - x_i) && \bmod m
\end{aligned}$$

Wie oft wird $h(\mathbf{x}) = h(\mathbf{y})$?

Beweis.

- Es gibt m^{k-1} Möglichkeiten, Werte für die Variablen a_i mit $i \neq j$ zu wählen.
- Für jede solche Wahl gibt es genau eine Wahl für a_j, so dass $h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})$.
- Für $\mathbf{a}$ gibt es insgesamt m^k Auswahlmöglichkeiten.
- Also

$$\Pr[h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})] = \frac{m^{k-1}}{m^k} = \frac{1}{m}$$

□

Familie für k-universelles Hashing

Definiere für $a \in \{1, \ldots, m-1\}$ die Hashfunktion

$$h'_a(\mathbf{x}) = \sum_{i=1}^{k} a^{i-1} x_i \bmod m$$

(mit $x_i \in \{0, \ldots, m-1\}$)

Satz

Für jede Primzahl m ist

$$H' = \{h'_a : a \in \{1, \ldots, m-1\}\}$$

eine k-universelle Familie von Hashfunktionen.

Familie für k-universelles Hashing

Beweisidee:

Für Schlüssel $\mathbf{x} \neq \mathbf{y}$ ergibt sich folgende Gleichung:

$$\begin{aligned} h'_a(\mathbf{x}) &= h'_a(\mathbf{y}) \\ h'_a(\mathbf{x}) - h'_a(\mathbf{y}) &\equiv 0 \bmod m \\ \sum_{i=1}^{k} a^{i-1}(x_i - y_i) &\equiv 0 \bmod m \end{aligned}$$

Anzahl der Nullstellen des Polynoms in a ist durch den Grad des Polynoms beschränkt (Fundamentalsatz der Algebra), also durch $k-1$.

Falls $k \leq m$ können also höchstens $k-1$ von $m-1$ möglichen Werten für a zum gleichen Hashwert für $\mathbf{x}$ und $\mathbf{y}$ führen.

Aus $\Pr[h(\mathbf{x}) = h(\mathbf{y})] \leq \frac{\min\{k-1, m-1\}}{m-1} \leq \frac{k}{m}$ folgt, dass H' k-universell ist.

Hashing Universelles Hashing

Familie für k-universelles Hashing

Definiere für $a \in \{1, \ldots, m-1\}$ die Hashfunktion

$$h'_a(\mathbf{x}) = \sum_{i=1}^{k} a^{i-1} x_i \bmod m$$

(mit $x_i \in \{0, \ldots, m-1\}$)

Satz

Für jede Primzahl m ist

$$H' = \{h'_a : a \in \{1, \ldots, m-1\}\}$$

eine k-universelle Familie von Hashfunktionen.

H. Täubig (TUM) GAD SS'12 184 / 631

Hashing Universelles Hashing

Familie für k-universelles Hashing

Beweisidee:

Für Schlüssel $\mathbf{x} \neq \mathbf{y}$ ergibt sich folgende Gleichung:

$$\begin{aligned} h'_a(\mathbf{x}) &= h'_a(\mathbf{y}) \\ h'_a(\mathbf{x}) - h'_a(\mathbf{y}) &\equiv 0 \bmod m \\ \sum_{i=1}^{k} a^{i-1}(x_i - y_i) &\equiv 0 \bmod m \end{aligned}$$

Anzahl der Nullstellen des Polynoms in a ist durch den Grad des Polynoms beschränkt (Fundamentalsatz der Algebra), also durch $k-1$.

Falls $k \leq m$ können also höchstens $k-1$ von $m-1$ möglichen Werten für a zum gleichen Hashwert für $\mathbf{x}$ und $\mathbf{y}$ führen.

Aus $\Pr[h(\mathbf{x}) = h(\mathbf{y})] \leq \frac{\min\{k-1, m-1\}}{m-1} \leq \frac{k}{m}$ folgt, dass H' k-universell ist.

H. Täubig (TUM) GAD SS'12 185 / 631

Familie für k-universelles Hashing

Definiere für $a \in \{1, \ldots, m-1\}$ die Hashfunktion

$$h'_a(\mathbf{x}) = \sum_{i=1}^{k} a^{i-1} x_i \bmod m$$

(mit $x_i \in \{0, \ldots, m-1\}$)

Satz

Für jede Primzahl m ist

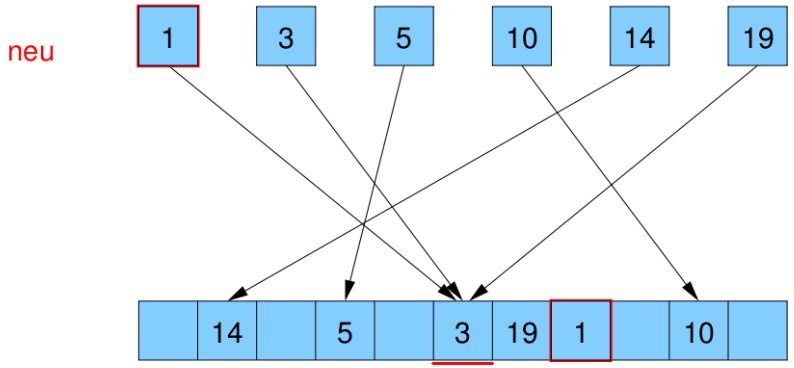
$$H' = \{h'_a : \ a \in \{1, \ldots, m-1\}\}$$

eine k-universelle Familie von Hashfunktionen.

Übersicht

Dynamisches Wörterbuch

Hashing with Linear Probing:



Speichere Element e im ersten freien
Ort T[i], T[$i+1$], T[$i+2$], ... mit $i == h(\text{key}(e))$
(Ziel: Folgen besetzter Positionen möglichst kurz)

Hashing with Linear Probing

```
Elem[m] T;                    // Feld sollte genügend groß sein

insert(Elem e) {
   i = h(key(e));
   while (T[i] ≠ null ∧ T[i] ≠ e)
      i = (i+1) % m;
   T[i] = e;
}

find(Key k) {
   i = h(k);
   while (T[i] ≠ null ∧ key(T[i]) ≠ k)
      i = (i+1) % m;
   return T[i];
}
```

Hashing with Linear Probing

Vorteil:

Es werden im Gegensatz zu Hashing with Chaining
(oder auch im Gegensatz zu anderen Probing-Varianten)
nur zusammenhängende Speicherzellen betrachtet.

⇒ Cache-Effizienz!

Hashing with Linear Probing

Problem: Löschen von Elementen

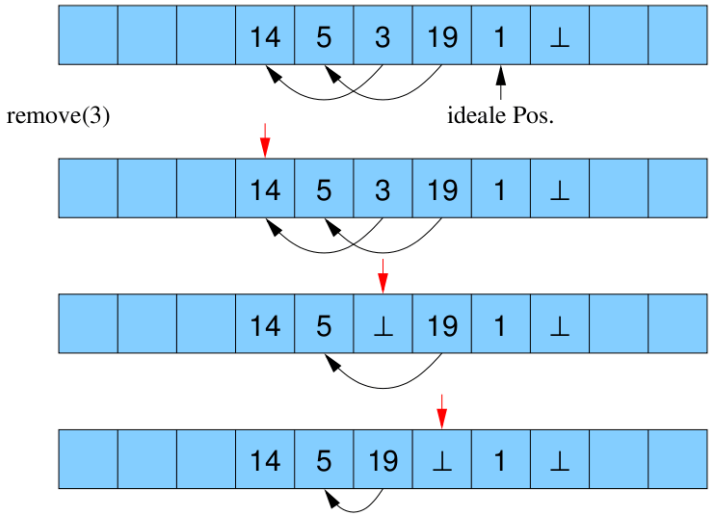
1. Löschen verbieten
2. Markiere Positionen als gelöscht
   (mit speziellem Zeichen $\neq \perp$)
   Suche endet bei $\perp$, aber nicht bei markierten Zellen

   Problem: Anzahl echt freier Zellen sinkt monoton
   ⇒ Suche wird evt. langsam oder periodische Reorganisation
3. Invariante sicherstellen:

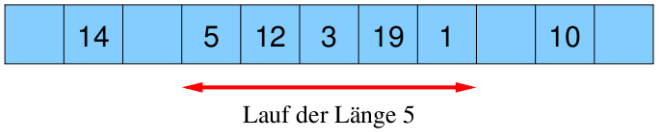
   Für jedes $e \in S$ mit idealer Position $i = h(key(e))$ und aktueller Position j gilt:
   T[i], T[i+1],. . . ,T[j] sind besetzt

Hashing with Linear Probing

Löschen / Aufrechterhaltung der Invariante



Hashing with Linear Probing



Satz

Wenn n Elemente in einer Hashtabelle T der Größe m > 2en mittels einer zufälligen Hashfunktion h gespeichert werden, dann ist für jedes T[i] die erwartete Länge eines Laufes in T, der T[i] enthält, O(1).

(e ist hier die Eulersche Zahl)