Script generated by TTT

Title: Seidl: GAD (04.05.2016)

Date: Wed May 04 13:21:29 CEST 2016

Duration: 46:09 min

Pages: 14

Freiwillige Deadline 08. Mai 2016

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

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 $k-1$.

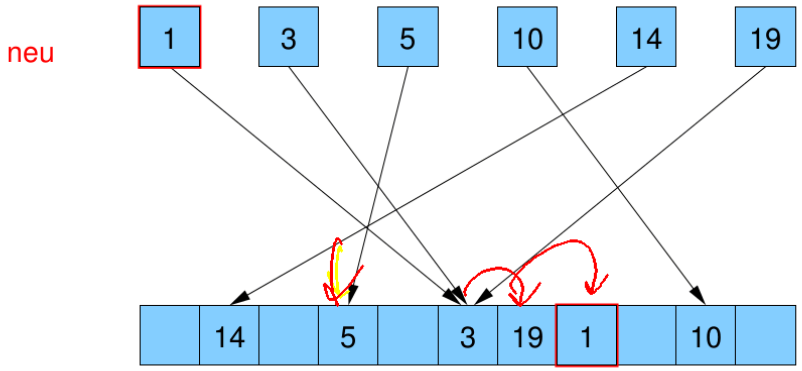
Falls $k \leq m$ können also höchstens $k-1$ von $m-1$ möglichen Werten für a zum gleichen Hashwert für $\mathbf{x}$ und $\mathbf{y}$ führen.

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. Seidl (TUM) GAD SS'16 170

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

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

Elem 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

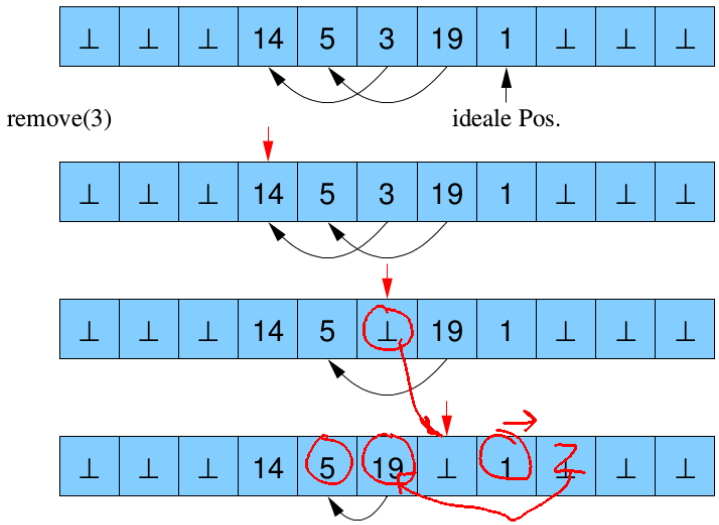
1. Löschen verbieten
2. Markiere Positionen als gelöscht
   (mit speziellem Zeichen $\neq \perp$)
   Suche endet bei $\perp$, aber nicht bei markierten Zellen

   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



Dynamisches Wörterbuch

Problem: Hashtabelle ist zu groß oder zu klein
(sollte nur um konstanten Faktor von Anzahl der Elemente abweichen)

Lösung: Reallokation

- Wähle geeignete Tabellengröße
- Wähle neue Hashfunktion
- Übertrage Elemente auf die neue Tabelle

Dynamisches Wörterbuch

Problem: Tabellengröße m sollte prim sein
(für eine gute Verteilung der Schlüssel)

Lösung:

- Für jedes k gibt es eine Primzahl in $\left[k^3, (k+1)^3\right]$
- Jede Zahl $z \leq (k+1)^3$, die nicht prim ist, muss einen Teiler $t \leq \sqrt{(k+1)^3} = (k+1)^{3/2}$ haben.
- Für eine gewünschte ungefähre Tabellengröße m' (evt. nicht prim) bestimme k so, dass $k^3 \leq m' \leq (k+1)^3$

Hashing | Anpassung der Tabellengröße

Dynamisches Wörterbuch

Problem: Tabellengröße m sollte prim sein
(für eine gute Verteilung der Schlüssel)

Lösung:

- Für jedes k gibt es eine Primzahl in $\left[k^3, (k+1)^3\right]$
- Jede Zahl $z \leq (k+1)^3$, die nicht prim ist, muss einen Teiler $t \leq \sqrt{(k+1)^3} = (k+1)^{3/2}$ haben.
- Für eine gewünschte ungefähre Tabellengröße m' (evt. nicht prim) bestimme k so, dass $k^3 \leq m' \leq (k+1)^3$
- Größe des Intervalls:
  $(k+1)^3 - k^3 + 1 = (k^3 + 3k^2 + 3k + 1) - k^3 + 1 = 3k^2 + 3k + 2$
- Für jede Zahl $j = 2, \ldots, (k+1)^{3/2}$:
  streiche die Vielfachen von j in $[k^3, (k+1)^3]$
- Für jedes j kostet das Zeit $\left((k+1)^3 - k^3 + 1\right)/j \in O(k^2/j)$

Dynamisches Wörterbuch

- Hilfsmittel: Wachstum der harmonischen Reihe

$$\ln n \quad \leq \quad H_n = \sum_{i=1}^{n} \frac{1}{i} \quad \leq \quad 1 + \ln n$$

- insgesamt:

$$\begin{aligned} \sum_{2 \leq j \leq (k+1)^{3/2}} O\left(\frac{k^2}{j}\right) &\leq k^2 \sum_{2 \leq j \leq (k+1)^{3/2}} O\left(\frac{1}{j}\right) \\ &\in k^2 \cdot O\left(\ln\left((k+1)^{3/2}\right)\right) \\ &\in O(k^2 \ln k) \\ &\in o(m) \end{aligned}$$

$\Rightarrow$ Kosten zu vernachlässigen im Vergleich zur Initialisierung der Tabelle der Größe m (denn m ist kubisch in k)