Script generated by TTT

Title: Täubig: GAD (22.05.2012)

Date: Tue May 22 14:33:28 CEST 2012

Duration: 88:25 min

Pages: 32

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
   $\Rightarrow$ 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

Vorteil:

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

$\Rightarrow$ Cache-Effizienz!

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

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

Hashing with Linear Probing

Problem: Löschen von Elementen

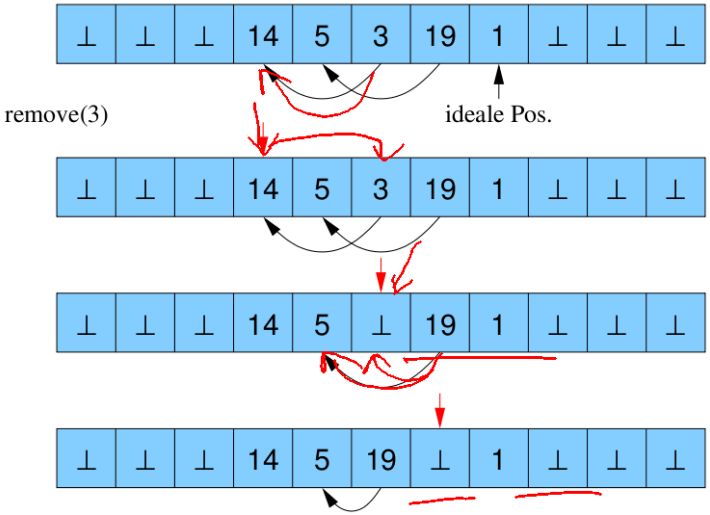
1. Löschen verbieten
2. Markiere Positionen als gelöscht
   (mit speziellem Zeichen $\neq \perp$)
   Suche endet bei $\perp$, aber nicht bei markierten Zellen

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

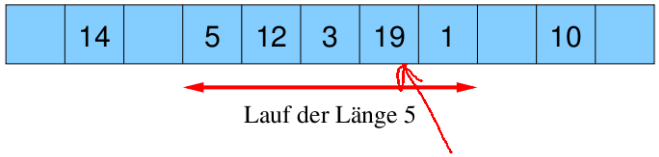
   Für jedes $e \in S$ mit idealer Position $i = h(key(e))$ und aktueller Position j gilt:
   T[i], T[i+1],. . . ,T[j] sind besetzt

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)

Hashing with Linear Probing

Beweis.

Vorbetrachtung:

$$\frac{k^k}{k!} < \sum_{i \geq 0} \frac{k^i}{i!} = e^k$$

$$\Rightarrow \qquad k! > \left(\frac{k}{e}\right)^k$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}$$

$$\leq \frac{n^k}{k!} < \left(\frac{en}{k}\right)^k$$

Hashing with Linear Probing

Beweis.

- n: Anzahl der Elemente
- $m > 2en$: Größe der Hashtabelle
- Lauf der Länge k: k aufeinanderfolgende besetzte Zellen
- Anzahl Möglichkeiten zur Wahl von k Elementen: $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$
- Wahrscheinlichkeit, dass k Hashwerte genau einen Lauf der Länge k an einer bestimmten Stelle ergeben: $\frac{k^k}{m^k} < \left(\frac{k}{m}\right)^k$
- Es gibt aber k verschiedene Anfangspositionen des Laufs der Länge k, so dass $T[i]$ darin liegt.

$\Rightarrow p_k = \Pr[T[i] \text{ in Lauf der Länge } k] < \left(\frac{en}{k}\right)^k k \left(\frac{k}{m}\right)^k < k\left(\frac{1}{2}\right)^k$

Hashing with Linear Probing

Beweis.

- Erwartete Länge des Laufs:

$$\mathbb{E}[\text{Länge des Laufs um } T[i]] = \sum_{k\geq 1} k \cdot p_k < \sum_{k\geq 1} k^2 \left(\frac{1}{2}\right)^k = O(1)$$

D.h. also die erwartete Laufzeit der Operationen insert, remove und find ist eine Konstante.

□

Übersicht

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 $\left((k+1)^3 - k^3\right)/j = O(k^2/j)$

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 $z \le (k+1)^3$, die nicht prim ist, muss einen Teiler $t \le \sqrt{(k+1)^3} = (k+1)^{3/2}$ haben.
- Für eine gewünschte ungefähre Tabellengröße m' (evt. nicht prim) bestimme k so, dass $k^3 \le m' \le (k+1)^3$
- 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 $\left((k+1)^3 - k^3\right)/j = O(k^2/j)$

Dynamisches Wörterbuch

- Hilfsmittel: Wachstum der harmonischen Reihe

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

- insgesamt:

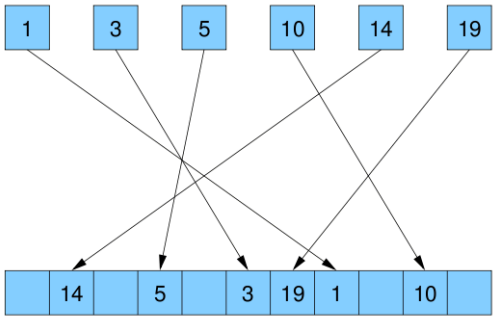
$$\begin{aligned} \sum_{j \le (k+1)^{3/2}} O\left(\frac{k^2}{j}\right) &= k^2 \sum_{j \le (k+1)^{3/2}} O\left(\frac{1}{j}\right) \\ &\in k^2 \cdot O\left(1 + \ln\left((k+1)^{3/2}\right)\right) \\ &\in O(k^2 \ln k) \\ &\in o(m) \end{aligned}$$

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

Übersicht

Perfektes Hashing für statisches Wörterbuch

- bisher: konstante *erwartete* Laufzeit
  (nicht ausreichend für Real Time Scenario)
- Ziel: konstante Laufzeit im worst case für find()
  durch perfekte Hashtabelle ohne Kollisionen
- Annahme: statische Menge S von n Elementen



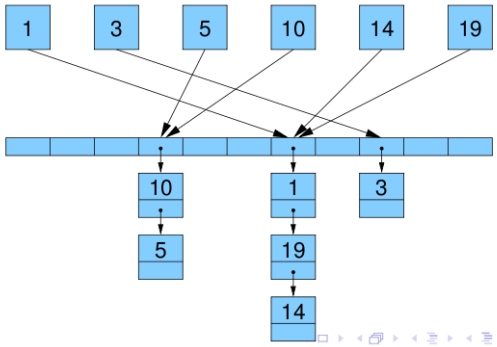
Statisches Wörterbuch

- S: feste Menge von n Elementen mit Schlüsseln k_1 bis k_n
- H_m: c-universelle Familie von Hashfunktionen auf $\{0,\ldots,m-1\}$
  (Hinweis: 2-universelle Familien existieren für alle m)
- $C(h)$ für $h \in H_m$: Anzahl Kollisionen in S für h, d.h.

$$C(h) = \left|\{(x,y) : x,y \in S,\ x \neq y,\ h(x) = h(y)\}\right|$$

Beispiel:

$C(h) = 2 + 6 = 8$



Statisches Wörterbuch

Lemma

Es gilt

$$\mathbb{E}[C(h)] \leq cn(n-1)/m$$

Beweis.

- Definiere $n(n-1)$ Indikator-Zufallsvariablen $X_{ij}(h)$:
  Für $i \neq j$ sei $X_{ij}(h) = 1 \Leftrightarrow h(k_i) = h(k_j)$.
- Dann ist $C(h) = \sum_{i \neq j} X_{ij}(h)$

$$\mathbb{E}[C] = \mathbb{E}\left[\sum_{i \neq j} X_{ij}\right] = \sum_{i \neq j} \mathbb{E}\left[X_{ij}\right] = \sum_{i \neq j} \Pr\left[X_{ij} = 1\right] \leq n(n-1) \cdot c/m$$

$\Rightarrow$ Für quadratische Tabellengröße ist die erwartete Anzahl von Kollisionen (und damit die erwartete worst-case-Laufzeit für find) eine Konstante. □

Statisches Wörterbuch

Lemma

Es gilt

$$\mathbb{E}[C(h)] \leq cn(n-1)/m$$

Beweis.

- Definiere $n(n-1)$ Indikator-Zufallsvariablen $X_{ij}(h)$:
  Für $i \neq j$ sei $X_{ij}(h) = 1 \Leftrightarrow h(k_i) = h(k_j)$.
- Dann ist $C(h) = \sum_{i \neq j} X_{ij}(h)$

$$\mathbb{E}[C] = \mathbb{E}\left[\sum_{i \neq j} X_{ij}\right] = \sum_{i \neq j} \mathbb{E}\left[X_{ij}\right] = \sum_{i \neq j} \Pr\left[X_{ij} = 1\right] \leq n(n-1) \cdot c/m$$

$\Rightarrow$ Für quadratische Tabellengröße ist die erwartete Anzahl von Kollisionen (und damit die erwartete worst-case-Laufzeit für find) eine Konstante. □

Markov-Ungleichung

Satz (Markov-Ungleichung)

Für jede nichtnegative Zufallsvariable X und Konstante k gilt:

$$\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$

Beweis.

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{z \in \mathbb{R}} z \cdot \Pr[X = z] \\
&\geq \sum_{z \geq k \cdot \mathbb{E}[X]} z \cdot \Pr[X = z] \geq \sum_{z \geq k \cdot \mathbb{E}[X]} k \cdot \mathbb{E}[X] \cdot \Pr[X = z] \\
&\geq k \cdot \mathbb{E}[X] \cdot \Pr[X \geq k \cdot \mathbb{E}[X]]
\end{aligned}$$

□

Markov-Ungleichung

Satz (Markov-Ungleichung)

Für jede nichtnegative Zufallsvariable X und Konstante k gilt:

$$\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$

Beweis.

$$\begin{aligned}\mathbb{E}[X] &= \sum_{z \in \mathbb{R}} z \cdot \Pr[X = z] \\ &\geq \sum_{z \geq k \cdot \mathbb{E}[X]} z \cdot \Pr[X = z] \geq \sum_{z \geq k \cdot \mathbb{E}[X]} k \cdot \mathbb{E}[X] \cdot \Pr[X = z] \\ &\geq k \cdot \mathbb{E}[X] \cdot \Pr[X \geq k \cdot \mathbb{E}[X]]\end{aligned}$$

□

Statisches Wörterbuch

Lemma

Für mindestens die Hälfte der Funktionen $h \in H_m$ gilt:

$$C(h) \leq 2cn(n-1)/m$$

Beweis.

- Aus Markov-Ungleichung $\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$ folgt:

$$\Pr[C(h) \geq 2cn(n-1)/m] \leq \Pr[C(h) \geq 2\mathbb{E}[C(h)]] \leq \frac{1}{2}$$

⇒ Für höchstens die Hälfte der Funktionen ist $C(h) \geq 2cn(n-1)/m$

⇒ Für mindestens die Hälfte der Funktionen ist $C(h) \leq 2cn(n-1)/m$

□

Statisches Wörterbuch

Lemma

Für mindestens die Hälfte der Funktionen $h \in H_m$ gilt:

$$C(h) \leq 2cn(n-1)/m$$

Beweis.

- Aus Markov-Ungleichung $\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$ folgt:

$$\Pr[C(h) \geq 2cn(n-1)/m] \leq \Pr[C(h) \geq 2\mathbb{E}[C(h)]] \leq \frac{1}{2}$$

⇒ Für höchstens die Hälfte der Funktionen ist $C(h) \geq 2cn(n-1)/m$

⇒ Für mindestens die Hälfte der Funktionen ist $C(h) \leq 2cn(n-1)/m$

□

Statisches Wörterbuch

Lemma

Wenn $m \geq cn(n-1)+1$, dann bildet mindestens die Hälfte der Funktionen $h \in H_m$ die Schlüssel injektiv in die Indexmenge der Hashtabelle ab.

Beweis.

- Für mindestens die Hälfte der Funktionen $h \in H_m$ gilt $C(h) < 2$.
- Da $C(h)$ immer eine gerade Zahl sein muss, folgt aus $C(h) < 2$ direkt $C(h) = 0$

⇒ keine Kollisionen (bzw. injektive Abbildung)

□

- Wähle zufällig $h \in H_m$ mit $m \geq cn(n-1)+1$
- Prüfe, ob injektiv ⇒ behalten oder erneut wählen

⇒ Nach durchschnittlich 2 Versuchen erfolgreich

Statisches Wörterbuch

Lemma

Für mindestens die Hälfte der Funktionen $h \in H_m$ gilt:

$$C(h) \leq 2cn(n-1)/m$$

Beweis.

- Aus Markov-Ungleichung $\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$ folgt:

$$\Pr[C(h) \geq 2cn(n-1)/m] \quad \leq \quad \Pr[C(h) \geq 2\mathbb{E}[C(h)]] \quad \leq \quad \frac{1}{2}$$

$\Rightarrow$ Für höchstens die Hälfte der Funktionen ist $C(h) \geq 2cn(n-1)/m$

$\Rightarrow$ Für mindestens die Hälfte der Funktionen ist $C(h) \leq 2cn(n-1)/m$

□

Statisches Wörterbuch

Lemma

Wenn $m \geq cn(n-1)+1$, dann bildet mindestens die Hälfte der Funktionen $h \in H_m$ die Schlüssel injektiv in die Indexmenge der Hashtabelle ab.

Beweis.

- Für mindestens die Hälfte der Funktionen $h \in H_m$ gilt $C(h) < 2$.
- Da $C(h)$ immer eine gerade Zahl sein muss, folgt aus $C(h) < 2$ direkt $C(h) = 0$

$\Rightarrow$ keine Kollisionen (bzw. injektive Abbildung)

□

- Wähle zufällig $h \in H_m$ mit $m \geq cn(n-1)+1$
- Prüfe, ob injektiv $\Rightarrow$ behalten oder erneut wählen

$\Rightarrow$ Nach durchschnittlich 2 Versuchen erfolgreich

Statisches Wörterbuch

Ziel: lineare Tabellengröße

Idee: zweistufige Abbildung der Schlüssel

- 1. Stufe bildet Schlüssel auf Buckets von konstanter durchschnittlicher Größe ab
- 2. Stufe benutzt quadratisch viel Platz für jedes Bucket
- Benutze Information über $C(h)$, um die Anzahl Schlüssel zu beschränken, die auf jede Tabellenposition abgebildet werden

Statisches Wörterbuch

- B_ℓ^h: Menge der Elemente in S, die h auf ℓ abbildet, $\ell \in \{0, \ldots, m-1\}$ und $h \in H_m$
- b_ℓ^h: Kardinalität von B_ℓ^h, also $b_\ell^h := \left|B_\ell^h\right|$
- Für jedes ℓ führen die Schlüssel in B_ℓ^h zu $b_\ell^h(b_\ell^h - 1)$ Kollisionen
- Also gilt:

$$C(h) = \sum_{\ell \in \{0,\ldots,m-1\}} b_\ell^h(b_\ell^h - 1)$$

Perfektes statisches Hashing: 1. Stufe

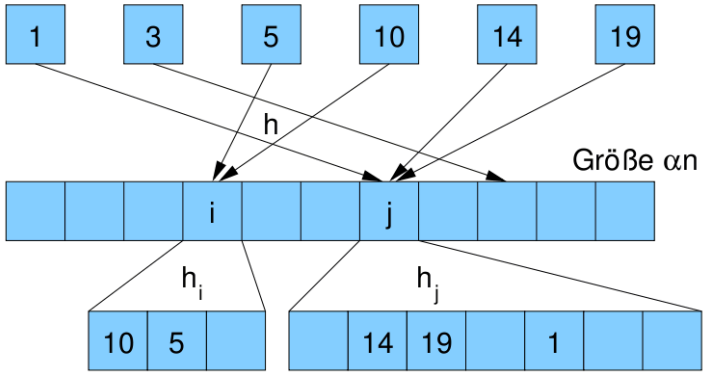
Konstruktion der Hashtabelle:

- Sei α eine (später festzulegende) Konstante
- Wähle Funktion $h \in H_{\lceil \alpha n \rceil}$, um S in Teilmengen B_ℓ aufzuspalten.
- Wähle h dabei aus dem Teil von $H_{\lceil \alpha n \rceil}$, für den gilt

$$C(h) \leq \frac{2cn(n-1)}{\lceil \alpha n \rceil} \leq \frac{2cn}{\alpha} \qquad \text{(vorletztes Lemma)}$$

- Für jedes $\ell \in \{0, \ldots, \lceil \alpha n \rceil - 1\}$ seien B_ℓ die Elemente, die auf ℓ abgebildet werden und $b_\ell = |B_\ell|$ deren Anzahl

Perfektes statisches Hashing



Perfektes statisches Hashing: 2. Stufe

- Für jedes B_ℓ:

  Sei $m_\ell = cb_\ell(b_\ell - 1) + 1$

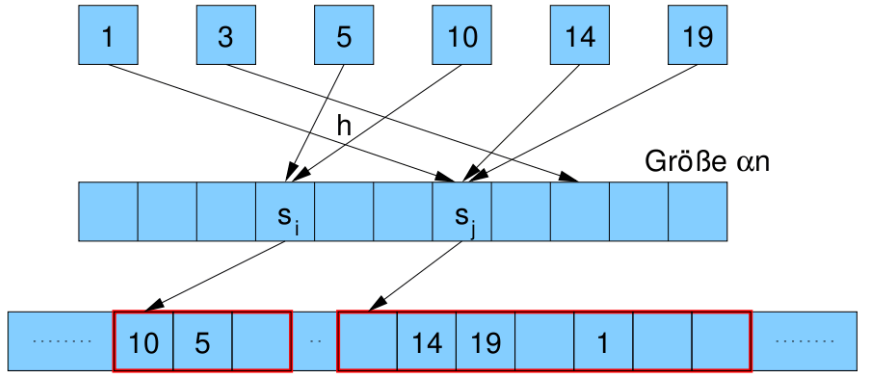
  Wähle eine Funktion $h_\ell \in H_{m_\ell}$, die B_ℓ injektiv in $\{0, \ldots, m_\ell - 1\}$ abbildet
  (mindestens die Hälfte der Funktionen in H_{m_ℓ} tun das)

- Hintereinanderreihung der einzelnen Tabellen ergibt eine Gesamtgröße der Tabelle von $\sum_\ell m_\ell$
- Die Teiltabelle für B_ℓ beginnt dabei an Position $s_\ell = m_0 + m_1 + \ldots + m_{\ell-1}$
- Die Anweisungen

$$\ell = h(x); \quad \text{return } s_\ell + h_\ell(x);$$

  berechnen dann eine injektive Funktion auf S.

Perfektes statisches Hashing



Perfektes statisches Hashing

- Die Funktion ist begrenzt durch:

$$\begin{aligned}\sum_{\ell=0}^{\lceil \alpha n \rceil - 1} m_\ell &= \sum_{\ell=0}^{\lceil \alpha n \rceil - 1} (c \cdot b_\ell (b_\ell - 1) + 1) \qquad \text{(siehe Def. der } m_\ell\text{'s)} \\ &\leq \lceil \alpha n \rceil + c \cdot C(h) \\ &\leq 1 + \alpha n + c \cdot 2cn/\alpha \\ &\leq 1 + (\alpha + 2c^2/\alpha) n\end{aligned}$$

- $\alpha = \sqrt{2}c$ minimiert diese Schranke
- Für $c = 1$ ergibt sich $2\sqrt{2}n$ als größter Adresswert

Satz

Für eine beliebige Menge von n Schlüsseln kann eine perfekte Hashfunktion mit Zielmenge $\{0, \ldots, 2\sqrt{2}n\}$ in linearer erwarteter Laufzeit konstruiert werden.