Script **generated by TTT**

Title: Seidl: GAD (10.05.2016)

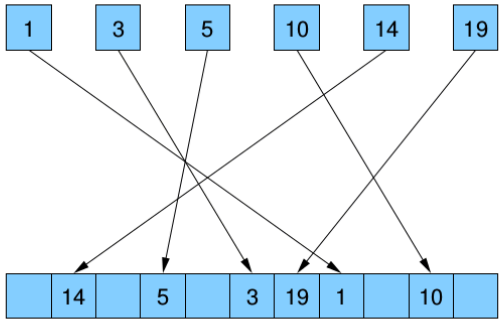
Date: Tue May 10 14:23:45 CEST 2016

Duration: 87:07 min

Pages: 37

Perfektes Hashing für statisches Wörterbuch

- bisher: konstante *erwartete* Laufzeit
  falls m im Vergleich zu n genügend groß gewählt wird
  (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



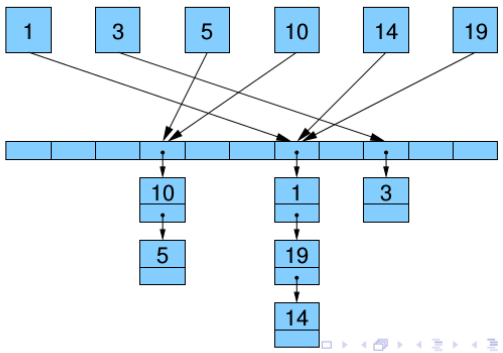
Statisches Wörterbuch

- 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:

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



Erwartete Anzahl von Kollisionen

Lemma

Für die Anzahl der Kollisionen gilt: $\mathbb{E}[C(h)] \leq cn(n-1)/m.$

Erwartete Anzahl von Kollisionen

Lemma

Für die Anzahl der Kollisionen 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. □

Hashing | Perfektes Hashing

Erwartete Anzahl von Kollisionen

Lemma

Für die Anzahl der Kollisionen 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. □

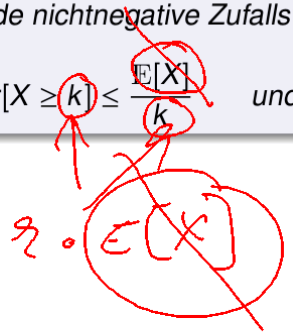
H. Seidl (TUM) GAD SS'16 181

Markov-Ungleichung

Satz (Markov-Ungleichung)

Für jede nichtnegative Zufallsvariable X und Konstante $k > 0$ gilt:

$$\Pr[X \geq k] \leq \frac{\mathbb{E}[X]}{k} \quad \text{und für } \mathbb{E}[X] > 0\text{:} \quad \Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$



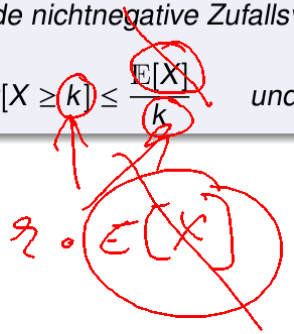
Hashing | Perfektes Hashing

Markov-Ungleichung

Satz (Markov-Ungleichung)

Für jede nichtnegative Zufallsvariable X und Konstante $k > 0$ gilt:

$$\Pr[X \geq k] \leq \frac{\mathbb{E}[X]}{k} \quad \text{und für } \mathbb{E}[X] > 0\text{:} \quad \Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$



H. Seidl (TUM) GAD SS'16 182

Markov-Ungleichung

Satz (Markov-Ungleichung)

Für jede nichtnegative Zufallsvariable X und Konstante $k > 0$ gilt:

$$\Pr[X \geq k] \leq \frac{\mathbb{E}[X]}{k} \qquad \text{und für } \mathbb{E}[X] > 0\text{:} \quad \Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$

Beweis.

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

□

Hashing | Perfektes Hashing

Markov-Ungleichung

Satz (Markov-Ungleichung)

Für jede nichtnegative Zufallsvariable X und Konstante $k > 0$ gilt:

$$\Pr[X \geq k] \leq \frac{\mathbb{E}[X]}{k} \qquad \text{und für } \mathbb{E}[X] > 0\text{:} \quad \Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$

Beweis.

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

□

H. Seidl (TUM) GAD SS'16 182

Hashfunktionen mit wenig Kollisionen

Lemma

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

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

Hashing | Perfektes Hashing

Markov-Ungleichung

Satz (Markov-Ungleichung)

Für jede nichtnegative Zufallsvariable X und Konstante $k > 0$ gilt:

$$\Pr[X \geq k] \leq \frac{\mathbb{E}[X]}{k} \qquad \text{und für } \mathbb{E}[X] > 0\text{:} \quad \Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$

Beweis.

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

□

H. Seidl (TUM) GAD SS'16 182

Hashfunktionen mit wenig Kollisionen

Lemma

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

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

Hashfunktionen mit wenig Kollisionen

Lemma

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

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

Beweis.

- Aus Lemma $\mathbb{E}[C(h)] \leq cn(n-1)/m$ und Markov-Ungleichung $\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$ folgt für $n \geq 2$:

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

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

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

□

Hashfunktionen ohne Kollisionen

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 auf Kollision $\Rightarrow$ behalten oder erneut wählen

$\Rightarrow$ Nach durchschnittlich 2 Versuchen erfolgreich

Hashfunktionen mit wenig Kollisionen

Lemma

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

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

Beweis.

- Aus Lemma $\mathbb{E}[C(h)] \leq cn(n-1)/m$ und Markov-Ungleichung $\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$ folgt für $n \geq 2$:

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

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

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

□

Hashfunktionen ohne Kollisionen

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 auf Kollision $\Rightarrow$ behalten oder erneut wählen

$\Rightarrow$ Nach durchschnittlich 2 Versuchen erfolgreich

Statisches Wörterbuch

Ziel: lineare Tabellengröße

Idee: zweistufige Abbildung der Schlüssel

- Wähle Hashfunktion h mit wenig Kollisionen (≈ 2 Versuche)

$\Rightarrow$ 1. Stufe bildet Schlüssel auf Buckets von konstanter durchschnittlicher Größe ab

- Wähle Hashfunktionen h_ℓ ohne Kollisionen (≈ 2 Versuche pro h_ℓ)

$\Rightarrow$ 2. Stufe benutzt quadratisch viel Platz für jedes Bucket, um alle Kollisionen aus der 1. Stufe aufzulösen

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 ist die Gesamtzahl der Kollisionen

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

Perfektes statisches Hashing: 1. Stufe

- 1. Stufe der Hashtabelle soll linear viel Speicher verwenden, also $\lceil \alpha n \rceil$ Adressen, wobei wir Konstante α später festlegen.
- Wähle Funktion $h \in H_{\lceil \alpha n \rceil}$, um S in Teilmengen B_ℓ aufzuspalten.

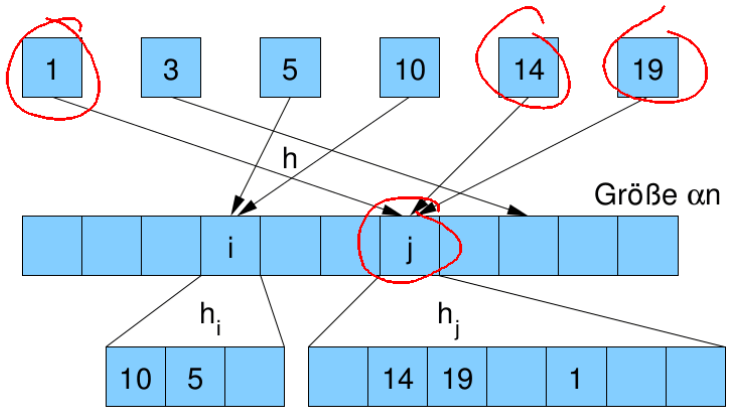
  Wähle h dabei so lange zufällig aus $H_{\lceil \alpha n \rceil}$ aus, bis gilt:

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

  Da das für mindestens die Hälfte der Hashfunktionen in $H_{\lceil \alpha n \rceil}$ gilt (vorletztes Lemma), erwarten wir dafür ≤ 2 Versuche.

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

Perfektes statisches Hashing



Perfektes statisches Hashing: 2. Stufe

- Für jedes B_ℓ:

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

  Wähle zufällig Funktion $h_\ell \in H_{m_\ell}$, bis h_ℓ die Menge B_ℓ injektiv in $\{0, \ldots, m_\ell - 1\}$ abbildet (also ohne Kollisionen).
  Mindestens die Hälfte der Funktionen in H_{m_ℓ} tut das.

- Hintereinanderreihung der einzelnen Tabellen ergibt eine Gesamtgröße der Tabelle von $\sum_\ell m_\ell$

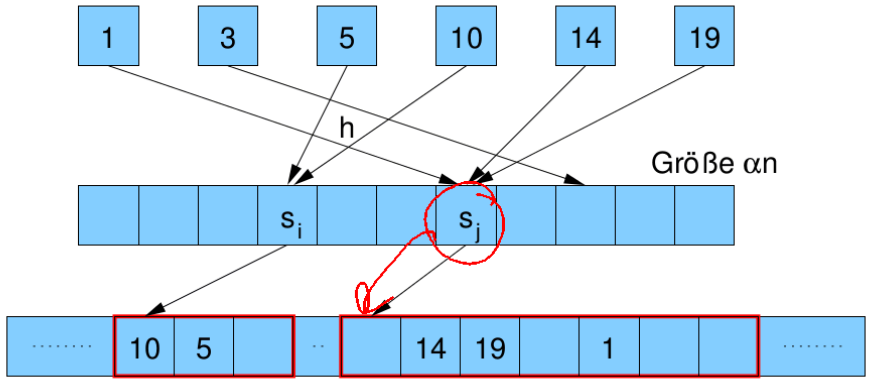
- Teiltabelle für B_ℓ beginnt an Position $s_\ell = m_0 + m_1 + \ldots + m_{\ell-1}$ und endet an Position $s_\ell + m_\ell - 1$

- Für gegebenen Schlüssel x, berechnen die Anweisungen

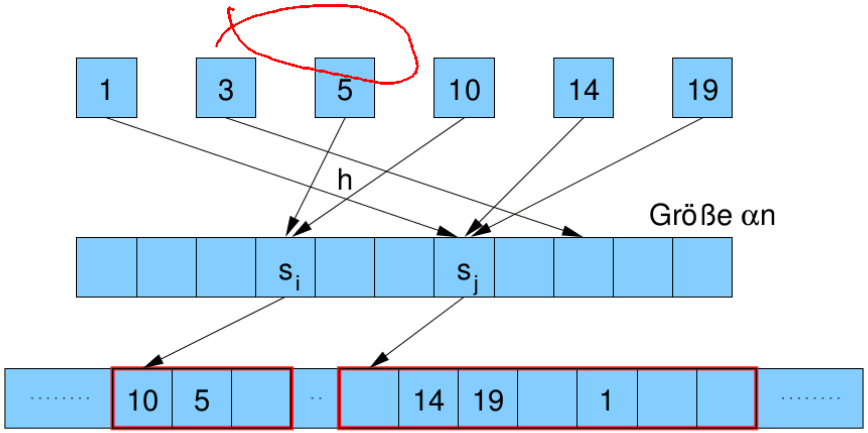
  $$\ell = h(x); \quad \text{return } s_\ell + h_\ell(x);$$

  dann eine injektive Funktion auf der Menge S.

Perfektes statisches Hashing



Perfektes statisches Hashing



Perfektes statisches Hashing

- Die Funktion ist beschränkt durch:

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

- Zur Minimierung der Schranke betrachte die Ableitung

$$\begin{aligned}f(\alpha) &= (2c^2/\alpha+\alpha)n+1\\ f'(\alpha) &= (-2c^2/\alpha^2+1)n\end{aligned}$$

$\Rightarrow f'(\alpha)=0$ liefert $\alpha=\sqrt{2}c$

$\Rightarrow$ Adressbereich: $0\ldots 2\sqrt{2}cn$

Perfektes statisches Hashing

Satz

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

- Da wir wissen, dass wir für beliebige m eine 2-universelle Familie von Hashfunktionen finden können, kann man also z.B. eine Hashfunktion mit Adressmenge $\{0,\ldots,4\sqrt{2}n\}$ in linearer erwarteter Laufzeit konstruieren. (Las Vegas-Algorithmus)
- Unsere Minimierung hat nicht den Platz berücksichtigt, der benötigt wird, um die Werte s_ℓ, sowie die ausgewählten Hashfunktionen h_ℓ zu speichern.

$\Rightarrow$ Berechnung von α sollte eigentlich angepasst werden (entsprechend Speicherplatz pro Element, pro m_ℓ und pro h_ℓ)

Perfektes dynamisches Hashing

Kann man perfekte Hashfunktionen auch dynamisch konstruieren?

ja, z.B. mit Cuckoo Hashing

- 2 Hashfunktionen h_1 und h_2
- 2 Hashtabellen T_1 und T_2
- bei find und remove jeweils in beiden Tabellen nachschauen
- bei insert abwechselnd beide Tabellen betrachten, das zu speichernde Element an die Zielposition der aktuellen Tabelle schreiben und wenn dort schon ein anderes Element stand, dieses genauso in die andere Tabelle verschieben usw.
- evt. Anzahl Verschiebungen durch $2\log n$ beschränken, um Endlosschleife zu verhindern
  (ggf. kompletter Rehash mit neuen Funktionen h_1, h_2)

Probleme beim linearen Sondieren

- Offene Hashverfahren allgemein:

  Erweiterte Hashfunktion $h(k,i)$ gibt an, auf welche Adresse ein Schlüssel k abgebildet werden soll, wenn bereits i Versuche zu einer Kollision geführt haben
- Lineares Sondieren (Linear Probing):

$$h(k,i)=(h(k)+i) \bmod m$$

Probleme beim linearen Sondieren

- Offene Hashverfahren allgemein:

  Erweiterte Hashfunktion $h(k,i)$ gibt an, auf welche Adresse ein Schlüssel k abgebildet werden soll, wenn bereits i Versuche zu einer Kollision geführt haben

- Lineares Sondieren (Linear Probing):

$$h(k,i) = (h(k) + i) \bmod m$$

- Primäre Häufung (primary clustering):
  tritt auf, wenn für Schlüssel k_1, k_2 mit unterschiedlichen Hashwerten $h(k_1) \neq h(k_2)$ ab einem bestimmten Punkt i_1 bzw. i_2 die gleiche Sondierfolge auftritt:

$$\exists i_1, i_2 \quad \forall j: \quad h(k_1, i_1 + j) = h(k_2, i_2 + j)$$

Probleme beim quadratischen Sondieren

- Quadratisches Sondieren (Quadratic Probing):

$$\begin{aligned} h(k,i) &= \left(h(k) + c_1 i + c_2 i^2\right) \bmod m \qquad (c_2 \neq 0) \\ \text{oder:} \quad h(k,i) &= \left(h(k) - (-1)^i \lceil i/2 \rceil^2\right) \bmod m \end{aligned}$$

Probleme beim quadratischen Sondieren

- Quadratisches Sondieren (Quadratic Probing):

$$\begin{aligned} h(k,i) &= \left(h(k) + c_1 i + c_2 i^2\right) \bmod m \qquad (c_2 \neq 0) \\ \text{oder:} \quad h(k,i) &= \left(h(k) - (-1)^i \lceil i/2 \rceil^2\right) \bmod m \end{aligned}$$

Probleme beim quadratischen Sondieren

- Quadratisches Sondieren (Quadratic Probing):

$$\begin{aligned} h(k,i) &= \left(h(k) + c_1 i + c_2 i^2\right) \bmod m \qquad (c_2 \neq 0) \\ \text{oder:} \quad h(k,i) &= \left(h(k) - (-1)^i \lceil i/2 \rceil^2\right) \bmod m \end{aligned}$$

- $h(k,i)$ soll möglichst surjektiv auf die Adressmenge $\{0, \ldots, m-1\}$ abbilden, um freie Positionen auch immer zu finden. Bei

$$h(k,i) = \left(h(k) - (-1)^i \lceil i/2 \rceil^2\right) \bmod m$$

  z.B. durch Wahl von m prim $\wedge$ $m \equiv 3 \bmod 4$

Probleme beim quadratischen Sondieren

- Quadratisches Sondieren (Quadratic Probing):

$$h(k,i) = \left(h(k) + c_1 i + c_2 i^2\right) \bmod m \qquad (c_2 \neq 0)$$

$$\text{oder:} \quad h(k,i) = \left(h(k) - (-1)^i \lceil i/2 \rceil^2\right) \bmod m$$

- $h(k,i)$ soll möglichst surjektiv auf die Adressmenge $\{0,\ldots,m-1\}$ abbilden, um freie Positionen auch immer zu finden. Bei

$$h(k,i) = \left(h(k) - (-1)^i \lceil i/2 \rceil^2\right) \bmod m$$

z.B. durch Wahl von m prim $\wedge$ $m \equiv 3 \bmod 4$

- Sekundäre Häufung (secondary clustering):
tritt auf, wenn für Schlüssel k_1, k_2 mit gleichem Hashwert $h(k_1) = h(k_2)$ auch die nachfolgende Sondierfolge gleich ist:

$$\forall i: \quad h(k_1,i) = h(k_2,i)$$

Double Hashing

- Auflösung der Kollisionen der Hashfunktion h
durch eine zweite Hashfunktion h':

$$h(k,i) = [h(k) + i \cdot h'(k)] \bmod m$$

wobei für alle k gelten soll, dass $h'(k)$ teilerfremd zu m ist,

$$\text{z.B.} \quad h'(k) = 1 + k \bmod m - 1$$
$$\text{oder} \quad h'(k) = 1 + k \bmod m - 2$$

für Primzahl m

- primäre und sekundäre Häufung werden weitgehend vermieden, aber nicht komplett ausgeschlossen

Statisches Wörterbuch

Lösungsmöglichkeiten:

- Perfektes Hashing
  - Vorteil: Suche in konstanter Zeit
  - Nachteil: keine Ordnung auf Elementen, d.h. Bereichsanfragen (z.B. alle Namen, die mit 'A' anfangen) teuer

- Speicherung der Daten in sortiertem Feld
  - Vorteil: Bereichsanfragen möglich
  - Nachteil: Suche teurer (logarithmische Zeit)

Sortierproblem

- gegeben: Ordnung $\leq$ auf der Menge möglicher Schlüssel

- Eingabe: Sequenz $s = \langle e_1, \ldots, e_n \rangle$

Beispiel:

5	10	19	1	14	3

- Ausgabe: Permutation $s' = \langle e'_1, \ldots, e'_n \rangle$ von s,
so dass $\text{key}(e'_i) \leq \text{key}(e'_{i+1})$ für alle $i \in \{1,\ldots,n\}$

Beispiel:

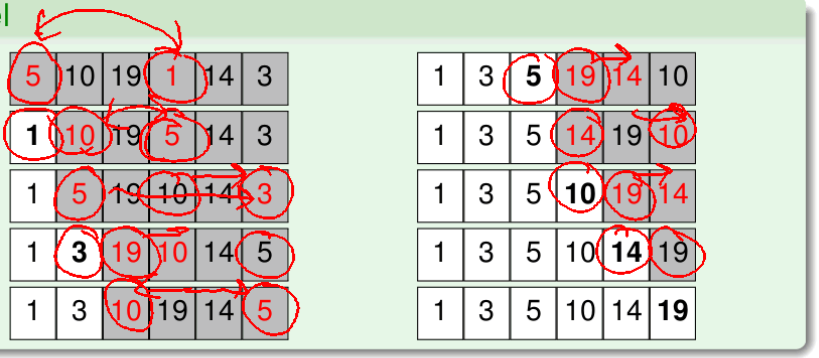
1	3	5	10	14	19

SelectionSort

Sortieren durch Auswählen

Wähle das kleinste Element aus der (verbleibenden) Eingabesequenz und verschiebe es an das Ende der Ausgabesequenz

Beispiel



SelectionSort

Sortieren durch Auswählen

```
void selectionSort(Element[ ] a, int n) {
  for (int i = 0; i < n; i++)
    // verschiebe min{a[i], ..., a[n − 1]} nach a[i]
    for (int j = i + 1; j < n; j++)
      if (a[i] > a[j])
        swap(a, i, j);
}
```

Zeitaufwand:

- Minimumsuche in Feld der Größe i: $\Theta(i)$
- Gesamtzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
- Vorteil: einfach zu implementieren
- Nachteil: quadratische Laufzeit