Script generated by TTT

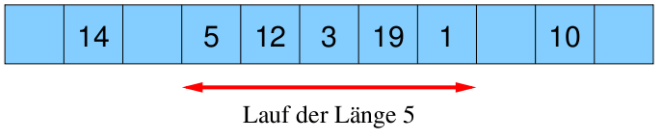
Title: Täubig: GAD (24.05.2012)

Date: Thu May 24 12:30:04 CEST 2012

Duration: 45:44 min

Pages: 20

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:

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

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

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

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

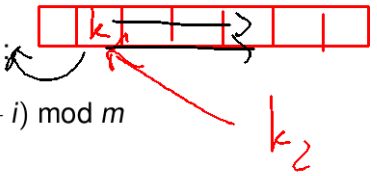
Übersicht

Übersicht

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):

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

- $h(k,i)$ 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:

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

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):

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

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

- $h(k,i)$ 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:

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

Probleme beim quadratischen Sondieren

- Quadratisches Sondieren (Quadratic Probing):

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

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

- $h(k,i)$ 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:

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

Probleme beim quadratischen Sondieren

- Quadratisches Sondieren (Quadratic Probing):

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

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

- $h(k,i)$ 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,

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

für Primzahl m

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

Verteiltes Wörterbuch / konsistentes Hashing

- Hashing kann für verteiltes Speichern von Daten benutzt werden (z.B. auf mehreren Festplatten oder Knoten in einem Netzwerk)
- Problem: Änderung bei Speichermedien (Erweiterungen, Ausfälle)

⇒ Konsistentes Hashing

- benutze eine zufällige Hashfunktion, um die Schlüssel auf eine Zahl im Intervall $[0,1)$ abzubilden
- benutze eine zweite zufällige Hashfunktion, um jedem Speichermedium ein Intervall in $[0,1)$ zuzuordnen, für das dieser Speicher dann zuständig ist

Übersicht

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

Übersicht

SelectionSort

Sortieren durch Auswählen

```
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], a[j]);
}
```

Zeitaufwand:

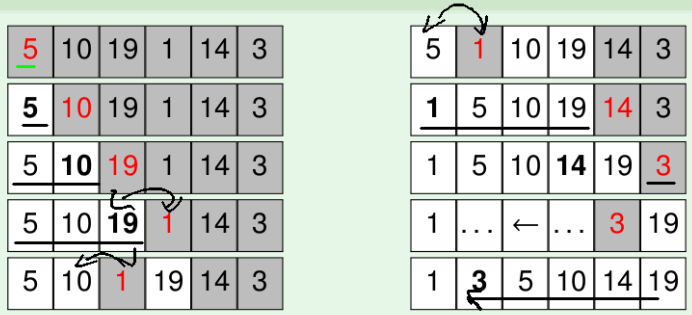
- Minimumsuche in Feld der Größe i: $\Theta(i)$
- Gesamtzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
- Vorteil: einfach zu implementieren
- Nachteil: quadratische Laufzeit

InsertionSort

Sortieren durch Einfügen

Nimm ein Element aus der Eingabesequenz und füge es an der richtigen Stelle in die Ausgabesequenz ein

Beispiel



InsertionSort

Sortieren durch Einfügen

```
insertionSort(Element[ ] a, int n) {
  for (int i = 1; i < n; i++)
    // verschiebe a_i an die richtige Stelle
    for (int j = i − 1; j ≥ 0; j − −)
      if (a[j] > a[j + 1])
        swap(a[j], a[j + 1]);
}
```

Zeitaufwand:

- Einfügung des i-ten Elements an richtiger Stelle: $O(i)$
- Gesamtzeit: $\sum_{i=1}^{n} O(i) = O(n^2)$
- Vorteil: einfach zu implementieren
- Nachteil: quadratische Laufzeit

Einfache Verfahren

SelectionSort

- mit besserer Minimumstrategie worst case Laufzeit $O(n \log n)$ erreichbar
  (mehr dazu in einer späteren Vorlesung)

InsertionSort

- mit besserer Einfügestrategie worst case Laufzeit $O(n \log^2 n)$ erreichbar
  ($\rightarrow$ ShellSort)