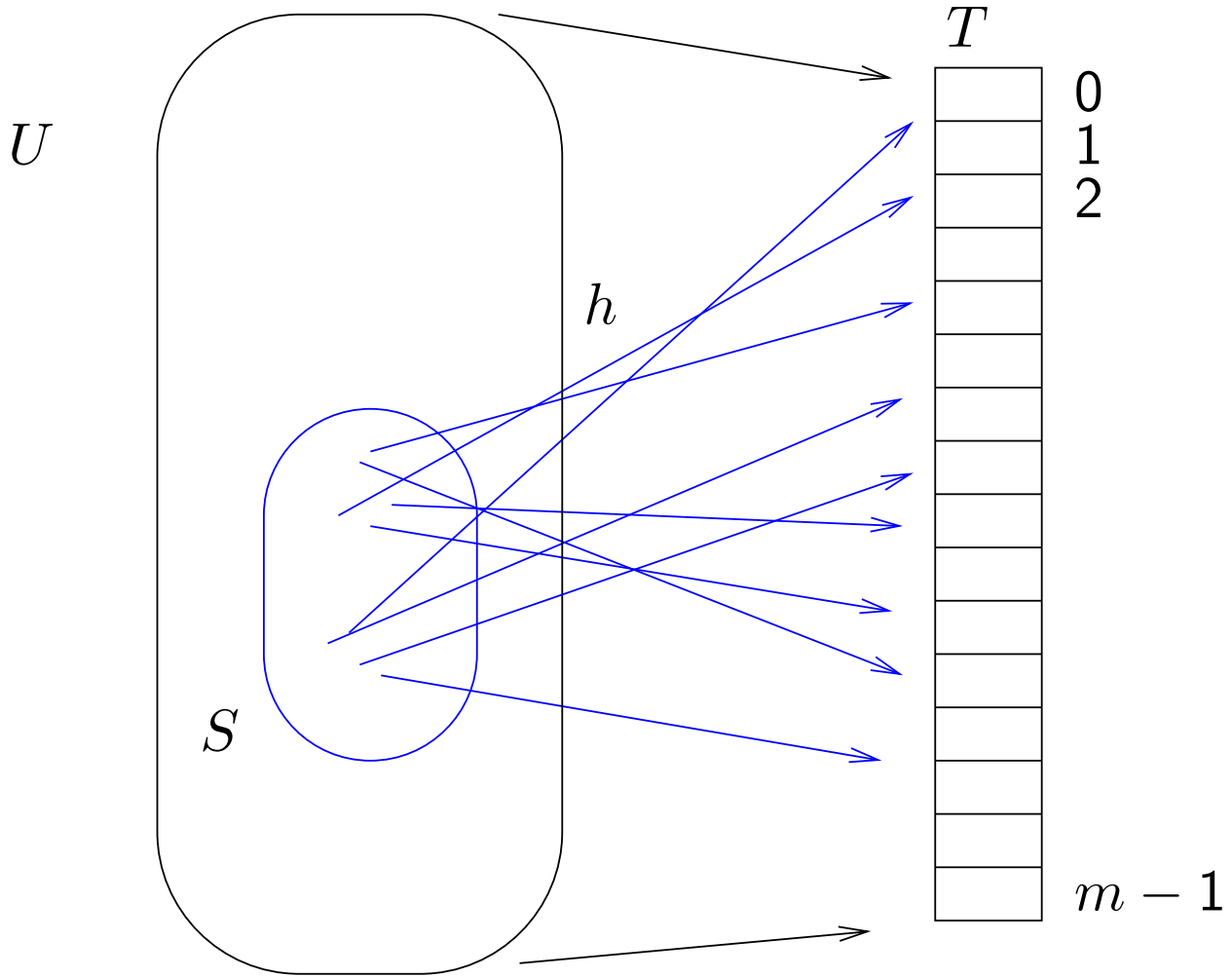
Hashing



Hashing

Hashing

- Menge U potentieller Schlüssel sehr groß, aktuelle Schlüsselmenge S jeweils nur kleine Teilmenge des Universums (im allgemeinen S nicht bekannt)
- Idee: durch Berechnung feststellen, wo Datensatz mit Schlüssel x gespeichert
- Abspeicherung der Datensätze in einem Array T mit Indizes $\{0, 1, \ldots, m-1\}$: Hashtabelle
- Hashfunktion h liefert für jeden Schlüssel $x \in U$ eine Adresse in Hashtabelle, d.h. $h : U \to \{0, 1, \ldots, m-1\}$.

Vorteil von Hashing

Sei $n := |S|$.

- Balancierte Suchbäume (AVL-Bäume, B-Bäume): dictionary operations haben Komplexität $O(\log n)$
- Hashing: für alle Operationen mittlere Komplexität $O(1)$

Anforderungen an Hashfunktionen

Gute Hashfunktionen sollten:

- surjektiv sein, d.h. den ganzen Wertebereich umfassen,
- die zu speichernden Schlüssel (möglichst) gleichmäßig verteilen, d.h. für alle Speicherplätze i und j sollte gelten $|h^{-1}(i)| \approx |h^{-1}(j)|$,
- effizient berechenbar sein.

Annahme:

- Daten unabhängig
- Prob($h(x) = j$)=$1/m$

Prob(i-tes Datum kollidiert nicht mit den ersten $i-1$ Daten, wenn diese kollisionsfrei sind)= $\frac{m-(i-1)}{m}$

Intuition:

Egal welche Speicherplätze die ersten $i-1$ Daten belegen, $m-i+1$ der m Möglichkeiten sind *gut*.

Prob(n Daten kollisionsfrei) $= \frac{m-1}{m} \cdot \frac{m-2}{m} \cdots \frac{m-n+1}{m}$

Beispiel: $m = 365$

Prob(23 Daten kollisionsfrei) ≈ 0.49

Prob(50 Daten kollisionsfrei) ≈ 0.03

Prob($2m^{1/2}$ Daten kollisionsfrei) =

$$\underbrace{\frac{m-1}{m}}\cdots\underbrace{\frac{m-m^{1/2}}{m}\cdots\frac{m-2m^{1/2}+1}{m}}$$

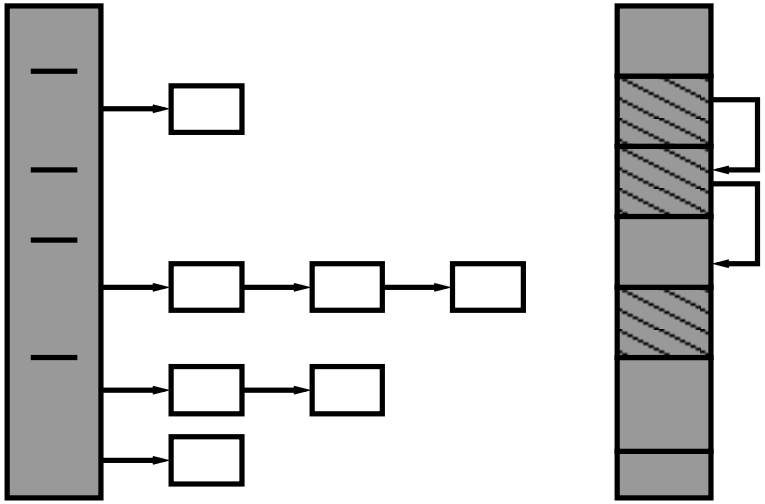
$$\leq \quad 1 \quad \cdot \quad \left(\frac{m-m^{1/2}}{m}\right)^{m^{1/2}} \quad = \left(1-\frac{1}{m^{1/2}}\right)^{m^{1/2}} \approx \frac{1}{e}$$

Hashing muss mit Kollisionen leben und benötigt Strategien zur Kollisionsbehandlung!

Kollisionsbehandlung

Verschiedene Arten der Kollisionsbehandlung:

- mittels verketteter Listen (links)
- mittels offener Adressierung (rechts)



Hashing mit Verkettung

Realisierung:

Jede Komponente der Hashtabelle enthält Zeiger auf paarweise disjunkte lineare Listen. Die i-te Liste $L(i)$ enthält alle Schlüssel $x \in S$ mit $h(x) = i$.

Vorteil: Alle Operationen werden unterstützt und $n > m$ ist möglich. (Für $n >> m$ jedoch Rehashing ratsam) Nachteil:

Hashverfahren mit offener Adressierung

Zur Erinnerung:

Im Kollisionsfall nach fester Regel alternativen freien Platz in Hashtabelle suchen (Sondierungsfolge).

Voraussetzung: Auswertung von h gilt als eine Operation.

$t(i, j)$:= Position des i-ten Versuchs zum Einfügen von Daten x mit $h(x) = j$

Hashverfahren mit offener Adressierung

Zur Erinnerung:

Im Kollisionsfall nach fester Regel alternativen freien Platz in Hashtabelle suchen (Sondierungsfolge).

Voraussetzung: Auswertung von h gilt als eine Operation.

$t(i,j)$:= Position des i-ten Versuchs zum Einfügen von Daten x mit $h(x) = j$

Anforderung an Funktion t:

- auch t in Zeit $O(1)$ berechenbar
- $t(0,j) = j$
- $t(\cdot,j) : \{0,\ldots,m-1\} \to \{0,\ldots,m-1\}$ bijektiv

Strategien zur Kollisionsbehandlung

- Lineares Sondieren
- Quadratisches Sondieren
- Multiplikatives Sondieren
- Doppeltes Hashing

Lineares Sondieren

$$t(i,j) := (i+j) \bmod m$$

Lineares Sondieren

$t(i, j) := (i + j) \bmod m$

Beispiel: $m = 19$ und $j = h(x) = 7$

Sondierungsfolge:
7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 0, 1, 2, 3, 4, 5, 6

Problem: *Clusterbildung*
Tendenz, dass immer längere zusammenhängende, belegte Abschnitte in der Hashtabelle entstehen, sogenannte *Cluster*
$\rightarrow$ erhöhte Suchzeiten

Ideales Hashing

Wunsch:

Zu jedem Zeitpunkt haben alle Positionen die gleiche Wahrscheinlichkeit, besetzt zu werden.

Beobachtung:

Das geht numerisch nicht genau, im Beispiel 11 freie Plätze, 19 mögliche Hashwerte, also alle Wahrscheinlichkeiten $k/19$, nicht $k/11$.

Modell des idealen Hashings:

Alle $\binom{m}{n}$ Möglichkeiten, die n besetzten Plätze für m Schlüssel auszuwählen, haben die gleiche Wahrscheinlichkeit.

Lineares Sondieren ist weit vom idealen Hashing entfernt.

Quadratisches Sondieren

$t(i,j) := j + (-1)^{i+1} \cdot \lfloor \frac{i+1}{2} \rfloor^2 \bmod m$

Sondierungsfolge:

$j,\ j+1^2,\ j-1^2,\ j+2^2,\ j-2^2, \ldots,\ j+(\frac{m-1}{2})^2,\ j-(\frac{m-1}{2})^2$

Beispiel: $m = 19$ und $j = h(x) = 7$

Sondierungsfolge:

$$7,\quad 8,\quad 6,\quad 11,\quad 3,\quad 16,\quad \begin{array}{r} -2 \\ = 17, \end{array}\quad \begin{array}{r} 23 \\ = 4, \end{array}\quad \begin{array}{r} -9 \\ = 10, \end{array}\quad \begin{array}{r} 32 \\ = 13, \end{array}\quad \begin{array}{r} -18 \\ = 1, \end{array}$$

$$\begin{array}{r} 43 \\ = 5, \end{array}\quad \begin{array}{r} -29 \\ = 9, \end{array}\quad \begin{array}{r} 56 \\ = 18, \end{array}\quad \begin{array}{r} -42 \\ = 15, \end{array}\quad \begin{array}{r} 71 \\ = 14, \end{array}\quad \begin{array}{r} -57 \\ = 0, \end{array}\quad \begin{array}{r} 88 \\ = 12, \end{array}\quad \begin{array}{r} -74 \\ = 2 \end{array}$$

Quadratisches Sondieren

Frage: Ist $t(\cdot, j)$ für alle j und m bijektiv?

Nein, aber immer wenn $m \equiv 3 \mod 4$ und m eine Primzahl ist

(Beweis: Zahlentheorie)

Besser als lineares Sondieren, aber für großes m sind die ersten Werte noch *nah* an j

Multiplikatives Sondieren I/II

Hier: $h(x) = x \mod (m-1)+1$ und damit in $\{1, \ldots, m-1\}$

$t(i,j) := i \cdot j \mod m$, $1 \leq i \leq m-1$

Hashwerte $1, \ldots, m-1$

Beispiel: $m = 19$ und $j = h(x) = 7$

Sondierungsfolge:

$i \cdot j$		7	14	21	28	35	42	49	56	63
$i \cdot j$	mod 19	7	14	2	9	16	4	11	18	6

$i \cdot j$		70	77	84	91	98	105	112	119	126
$i \cdot j$	mod 19	13	1	8	15	3	10	17	5	12

Multiplikatives Sondieren II/II

Frage: Ist $t(\cdot, j)$ für alle j und m bijektiv?

Nein, aber immer wenn m Primzahl und $j \neq 0$

Beweis: Durch Widerspruch!

Falls nicht, gibt es $1 \leq i_1 < i_2 \leq m-1$ mit

$$\begin{aligned} & i_1 \cdot j \equiv i_2 \cdot j \mod m \\ \Rightarrow\ & j \cdot (i_2 - i_1) \equiv 0 \mod m \\ \Rightarrow\ & j \cdot (i_2 - i_1) \text{ ist Vielfaches von } m \\ \Rightarrow\ & \text{Primfaktorzerlegung von } j \cdot (i_2 - i_1) \text{ muss } m \text{ enthalten} \end{aligned}$$

Widerspruch zu $1 \leq j \leq m-1$, $1 \leq i_2 - i_1 \leq m-1$

Doppeltes Hashing I/II

Sei $h_1(x) \equiv x \mod m$ und $h_2(x) \equiv x \mod (m-2)+1$.

i-te Position für x: $h_1(x) + i \cdot h_2(x) \mod m$, $1 \leq i \leq m-1$

Beispiel: $m = 19$ und $x = 47$

$h_1(47) \equiv 47 \mod 19 = 9$ und $h_2(47) \equiv 47 \mod 17 + 1 = 14$

Sondierungsfolge:
9, 4, 18, 13, 8, 3, 17, 12, 7, 2, 16, 11, 6, 1, 15, 10, 5, 0, 14

Doppeltes Hashing II/II

Beobachtung:
$i \cdot h_2(x)$ durchläuft für $1 \leq i \leq m-1$ die Werte $1, \ldots, m-1$ in irgendeiner Reihenfolge, ergänzt wird $0 = 0 \cdot h_2(x)$

Durch den Summanden $h_1(x)$ wird der Anfang zufällig verschoben.

Doppeltes Hashing kommt dem idealen Hashing am nächsten.

Universelles Hashing

Problem

Trotz hervorragender *Average-Case-Komplexität* von $\Theta(1)$ sehr schlechtes *Worst-Case-Verhalten* (alle aktuellen Schlüssel auf dieselbe Adresse der Hashtabelle) $\rightarrow \Theta(n)$.

Beobachtung:

Bei festgelegter Hashfunktion bestimmte Schlüsselmenge, die dieses ungünstige Verhalten hervorrufen kann.

Ausweg:

Zur Laufzeit zufällig Hashfunktion aus Klasse von Hashfunktionen (mit besonderen Eigenschaften) auswählen.

$\rightarrow$ Für jede Schlüsselmenge $S \subseteq U$ gutes Laufzeitverhalten im Mittel (bezogen auf Zufallsauswahl der Hashfunktion).

Zusammenfassung

- Auch die beste Hashfunktion kann Kollisionen nicht ganz vermeiden, deshalb sind Hashverfahren im *Worst Case* ineffiziente Realisierungen der Operationen search, insert, delete.
- Im Durchschnitt sind sie jedoch weitaus effizienter als Verfahren, die auf Schlüsselvergleichen basieren.
- Die Anzahl benötigter Schritte zum Suchen, Einfügen und Entfernen hängt (im Durchschnitt) im wesentlichen vom Belegungsfaktor, d.h. dem Verhältnis von Anzahl aktueller Schlüssel zur Größe der Hashtabelle, ab.