Hashverfahren

Hashing

- Hashverfahren
  - Aufgabe
  - Realisierung
- Hashfunktionen
  - Anforderungen
  - Wahl einer Hashfunktion
- Kollisionen
  - Geburtstagsparadoxon
  - Strategien zur Kollisionsbehandlung
  - Hashverfahren mit Verkettung der Überläufer
  - Offene Hashverfahren
- Ausblick
  - Universelles Hashing

Hashing

Aufgabe

Dynamische Verwaltung von Daten, wobei jeder Datensatz eindeutig durch einen Schlüssel charakterisiert ist

Viele Anwendungen benötigen nur einfache Daten-Zugriffsmechanismen (dictionary operations):

- Suche nach Datensatz bei gegebenem Schlüssel x
  search(x)
- Einfügen eines neuen Datensatzes d mit Schlüssel x
  insert(x, d) (abgekürzt insert(x))
- Entfernen eines Datensatzes bei gegebenem Schlüssel x
  delete(x)

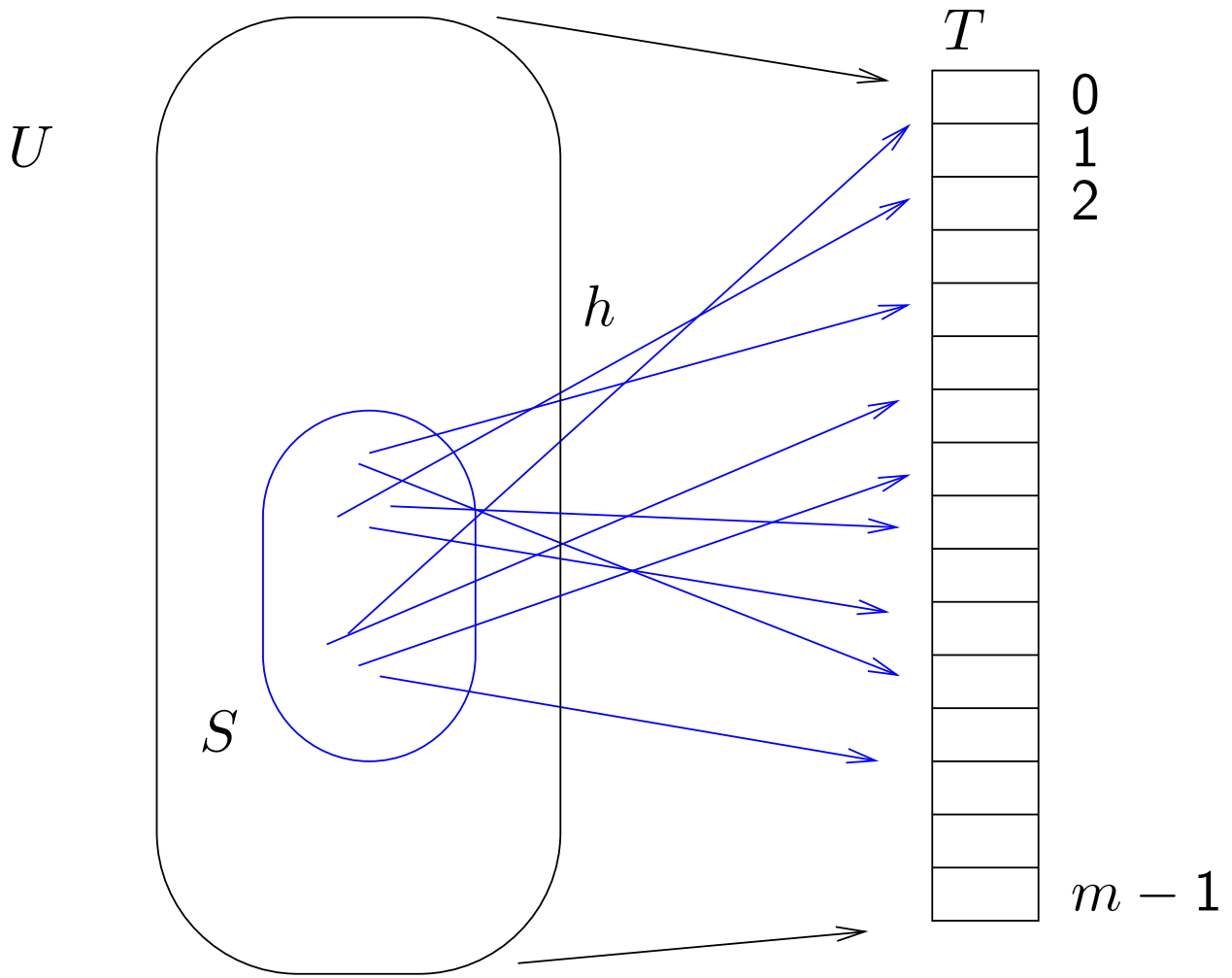
Menge potentieller Schlüssel (Universum) kann sehr groß sein!

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. $h : U \to \{0, 1, \ldots, m-1\}$.

Hashing



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 $O(1)$

Belegungsfaktor

Quotient $\beta := n/m$ heißt Belegungsfaktor oder Auslastungsfaktor einer Hashtabelle der Größe m.

Mittlerer Aufwand für dictionary operations als Funktion in β abschätzbar

$\rightarrow$ Anzahl aktueller Schlüssel geht nur indirekt in Aufwand ein

Hashing

Herausforderung:

Anzahl möglicher Schlüssel viel größer als Hashtabelle, also $|U| >> m$

- Hashfunktion muss verschiedene Schlüssel x_1 und x_2 auf gleiche Adresse abbilden.
- x_1 und x_2 beide in aktueller Schlüsselmenge
  $\longrightarrow$ Adresskollision

Hashverfahren gegeben durch:

- eine Hashtabelle,
- eine Hashfunktion, die Universum der möglichen Schlüssel auf Adressen einer Hashtabelle abbildet,
- eine Strategie zur Auflösung möglicher Adresskollisionen.

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.

Voraussetzung:

$U = \{0, 1, \ldots, N-1\}$ und

$h : U \to \{0, 1, \ldots, m-1\}$

Häufig: Divisions- oder Kongruenzmethode

$h(x) :\equiv x \bmod m \to |h^{-1}(i)| \in \{\lfloor N/m \rfloor, \lceil N/m \rceil\}$

Wahl einer Hashfunktion

$h(x) :\equiv x \bmod m$

Problem: Daten oft nicht gleichverteilt!

Beispiel: Texte in Zahlen übertragen, oft viele Leerzeichen, bestimmte Wörter häufiger etc.

Wichtig: geeignete Wahl von m

- m Zweierpotenz: $x \bmod m$ wählt nur die letzten $\log m$ Bits
- m Primzahl: $x \bmod m$ *beeinflusst* alle Bits

Beispiel: $m = 11$ und $S = \{49, 22, 6, 52, 76, 34, 13, 29\}$

Hashwerte: $h(49) = 49 \bmod 11 = 5$, $h(22) = 22 \bmod 11 = 0$, 6, 8, 10, 1, 2, 7

Zur Erinnerung:

Im Allgemeinen unvermeidbar, dass Kollisionen auftreten, denn aus $N >> m$ folgt Existenz eines Speicherplatzes i mit $|h^{-1}(i)| \geq N/m$.

Frage:

Sei $n := |S|$. Wie wahrscheinlich sind Kollisionen bei $n << m$?

Geburtstagsparadoxon

Bei wie vielen (zufällig gewählten) Personen ist es *wahrscheinlich*, dass hiervon zwei am selben Datum (Tag und Monat) Geburtstag haben?

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) = $\frac{m-1}{m} \cdot \frac{m-2}{m} \ldots \frac{m-n+1}{m}$

Beispiel: $m = 365$

Prob(23 Daten kollisionsfrei) ≈ 0.49

Prob(50 Daten kollisionsfrei) ≈ 0.03

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

$$\underbrace{\frac{m-1}{m}\cdots}_{1}\underbrace{\frac{m-m^{1/2}}{m}\cdots\frac{m-2m^{1/2}+1}{m}}_{\left(\frac{m-m^{1/2}}{m}\right)^{m^{1/2}}}$$

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

Hashing muss mit Kollisionen leben und benötigt Strategien zur Kollisionsbehandlung!

Kollisionsbehandlung

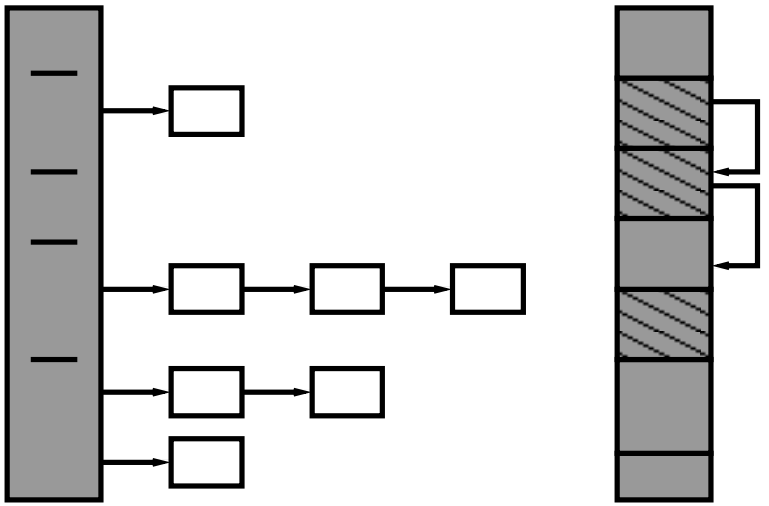
Hashverfahren unterscheiden sich in Strategien zur Kollisionsbehandlung:

- mittels verketteter Listen:
  Jede Komponente der Hashtabelle enthält Zeiger auf Überlaufliste.
- mittels offener Adressierung:
  Im Kollisionsfall nach fester Regel alternativen freien Platz in Hashtabelle suchen. Für jeden Schlüssel Reihenfolge, in der Speicherplätze betrachtet werden, vorgegeben:
  Sondierungsfolge

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 $h(x) = i$.

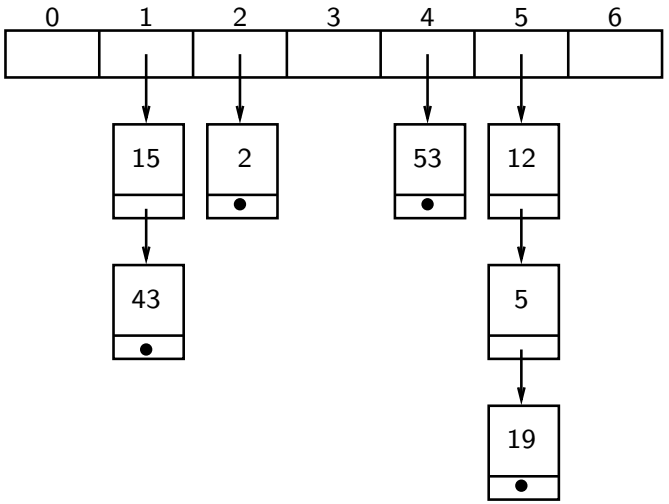
Vorteil: Alle Operationen werden unterstützt und $n > m$ ist möglich. (Für $n >> m$ jedoch Rehashing ratsam) Nachteil: Speicherplatzbedarf für Zeiger

- search(x): Berechne $h(x)$ und suche in Liste $L(h(x))$.
- insert(x) (nach erfolgloser Suche): Berechne $h(x)$ und füge x in Liste $L(h(x))$ ein.
- delete(x) (nach erfolgreicher Suche): Berechne $h(x)$, suche x in Liste $L(h(x))$ und entferne x.

Beispiel Verkettung der Überläufer

Beispiel: $m = 7$ und $h(x) = x \bmod m$

$$S = \{2, 5, 12, 15, 19, 43, 53\}$$



Analyse

Bei zufälligen Daten und ideal streuenden Hashfunktion gilt für

$$X_{ij} := \begin{cases} 1 & i\text{-tes Datum kommt in Liste } L(j) \\ 0 & \text{sonst} \end{cases}$$

$\text{Prob}(X_{ij} = 1) = \frac{1}{m}$

$\rightarrow \text{E}(X_{ij}) = 1 \cdot \frac{1}{m} + 0 \cdot \frac{m-1}{m} = \frac{1}{m}$

$X_j = X_{1j} + \cdots + X_{nj}$ zählt Anzahl Daten in Liste $L(j)$.

$\text{E}(X_j) = \text{E}(X_{1j} + \cdots + X_{nj}) = \text{E}(X_{1j}) + \cdots + \text{E}(X_{nj}) = \frac{n}{m}$

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