Algorithmen und Datenstrukturen

SoSe 2008 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Algorithmen und Datenstrukturen | Gesamtübersicht

- Organisatorisches / Einführung
- Grundlagen: RAM, $\mathcal{O}$-Notation, Rekursion, Datenstrukturen
- Sortieren
- Wörterbücher und Mengen
- Graphen und Graphalgorithmen

Abstrakter Datentyp Menge: **Algebraischer Standpunkt**

Eine *Menge* wird beschrieben durch ein Tupel $(\mathrm{D}, \chi, \epsilon, \in, \iota, \sqcup, \sqcap, \setminus, \eta)$ mit D Datentyp (Sorte); daraus abgeleitet: $set(\mathrm{D})$, $list(\mathrm{D})$ [letzteres wird vorausgesetzt] $\rightsquigarrow$ mehrsortige Signatur

$\chi :\rightarrow set(\mathrm{D})$	Erschaffen einer Menge
$\epsilon : set(\mathrm{D}) \rightarrow \mathbb{B}$	Testen, ob Menge leer ist
$\in: set(\mathrm{D}) \times \mathrm{D} \rightarrow bool$	Element-Testen
$\iota : set(\mathrm{D}) \times \mathrm{D} \rightarrow set(\mathrm{D})$	Einfügen (insert)
$\sqcup : set(\mathrm{D}) \times set(\mathrm{D}) \rightarrow set(\mathrm{D})$	Vereinigung
$\sqcap : set(\mathrm{D}) \times set(\mathrm{D}) \rightarrow set(\mathrm{D})$	Durchschnitt
$\setminus : set(\mathrm{D}) \times set(\mathrm{D}) \rightarrow set(\mathrm{D})$	Differenz
$\eta : set(\mathrm{D}) \rightarrow list(\mathrm{D})$	Aufzählen (enumeration)

Viele weiter Operationen sind denkbar / wünschenswert.

Implementierungen I

Sind die darzustellenden Mengen Teil eines kleinen, endlichen *Universums* (Wertebereichs) $\mathrm{U} = \{\mathrm{u}_1, \ldots, \mathrm{u}_N\}$, so wähle *Bitvektor-Darstellung*:
$s : \{0, 1\}[1..N]$, $s[i] = 1$ gdw. i ist in der durch s dargestellten Menge enthalten.
Während das Einfügen somit in $\mathcal{O}(1)$ Zeit geht,
kosten insbesondere Vereinigung, Durchschnitt und Differenz sowie der Elementtest Zeit $\mathcal{O}(N)$.
Das ist zu teuer für viele Anwendungen.

Hinweis: Ist das Universum “sehr klein”, so passen Bitvektoren evtl. bereits in ein einzelnes Speicherwort; dann gehen auch alle “teuren” Operationen in konstanter Zeit.

Implementierungen II

Die Feld-Implementierung (Bitvektordarstellung) ist speicheraufwändig.
$\rightsquigarrow$ ungeeignet bei größeren Universen.

Sind die betrachteten Teilmengen dennoch klein, bietet sich eine *Listenimplementierung* an.

Variante 1: *ungeordnete Listen*: größter Nachteil: Das Vereinigen (analog für andere Mengenoperationen) zweier Mengen der Größe n und m kostet $\mathcal{O}(n \cdot m)$ Zeit.

Variante 2: *geordnete Listen* (Vorauss.: Definition einer geeigneten Ordnung auf den Mengen)
Dann geht z.B. das Vereinigen zweier Mengen der Größe n und m in $\mathcal{O}(n + m)$ Zeit.
Idee: paralleler Durchlauf durch beide Listen (siehe Tafel)
Es genügt als Grunddatenstruktur eine einfach verkettete Liste. (Warum?)

Wörterbücher: Mengen mit eingeschränkten Operationen

Für viele Anwendungen werden nicht alle Operationen auf Mengen benötigt. Grundlegend wichtig scheinen oft die folgenden Operationen zu sein:
Nachschlagen (Lookup), also die Elementanfrage
Einfügen (Insert)
Löschen (Delete) eines einzelnen Elementes (musste gemäß bisheriger Spezifikation durch andere Mengenoperationen nachgebildet werden (wie genau?))
Initialisieren (init/clear)
Überlegen wir: Wie teuer sind die Operationen bei den drei vorgestellten Implementierungen?

Wörterbücher: Bessere Datenstrukturen

1. Hashing

2. binäre Suchbäume

3. AVL-Bäume

Hashing

Eine Abbildung $h : K \to S$ heißt *Hash-Funktion* oder *Schlüsseltransformation*, wenn $|K| \geq |S|$ gilt. Insbesondere liefert h eine *Hashtabelle* der Größe $|S|$.
S heißt auch die Menge der *Ziel-Schlüssel* oder *Behälter*.
Die Menge K repräsentiert die Daten, die "gehasht" werden sollen.
Diese Repräsentation selbst erfolgt durch *Urschlüssel*.
Typischerweise wird die Menge der Ziel-Schlüssel als Anfangsstück der natürlichen Zahlen gewählt: $S \subseteq \{0, \ldots, m-1\}$.
Diese Menge heißt dann auch *Adressraum*. In dieser Lesart geht es also um das geeignete Abbilden von Urschlüsseln in den Speicher.

Hinweis: Typischerweise wird in der Praxis immer nur eine kleine Teilmenge $K' \subseteq K$ mit h abgebildet. Die Menge $S' := \{h(k) | k \in K'\}$ sind dann die tatsächlich genutzten Schlüssel.

Das Verhältnis $\beta = \frac{|K'|}{|S|}$ liefert uns den *Belegungsfaktor*. (Manchmal auch $\frac{|S'|}{|S|}$.)

Der Fall $k \neq k' \wedge h(k) = h(k')$ wird als *Kollision* bezeichnet. Eine injektive Hash-Funktion heißt *perfekt*, u.a. weil sie keine Kollisionen erzeugt.

Hashing

Kriterien für eine gute Hash-Funktion:

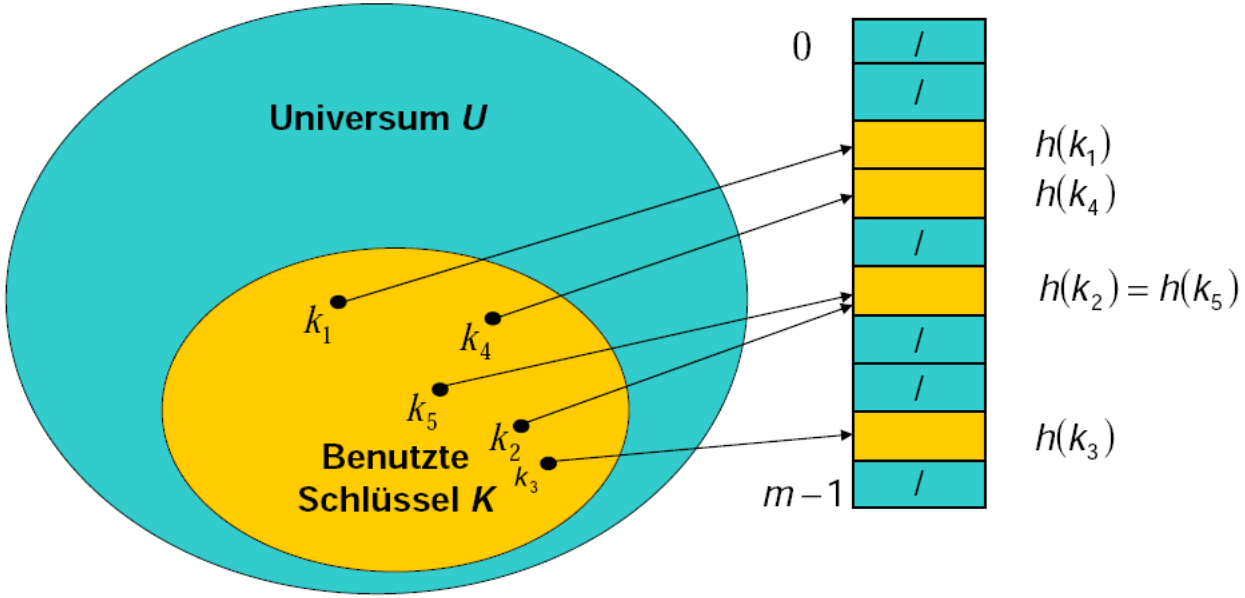
1. Theoretisch: Surjektivität;
   praktisch: gute Speicherausnutzung, ausgedrückt durch Belegungsfaktor

2. Kollisionsarmut (im Widerstreit zu 1.)

3. Schnelle Berechenbarkeit

4. Die zu speichernden Urschlüssel sollten mit der Hash-Funktion möglichst gleichmäßig über alle Zielschlüssel verteilt werden.

Hashing - Illustration



Zum Kollisionsproblem

Angenommen, unsere Hashfunktion verteilt die n tatsächlich zu verteilenden Urschlüssel völlig gleichmäßig auf die m Behälter (jedesmal, wenn so eine Zuweisung ansteht), mit $\mathrm{n} < \mathrm{m}$.

Sei $\mathrm{P}(\mathrm{i})$ die Wahrscheinlichkeit dafür, dass der i-te Urschlüssel auf einen freien Behälter abgebildet wird, wenn alle "vorigen" Urschlüssel ebenfalls kollisionsfrei abgebildet wurden.
$\rightsquigarrow$ Wahrscheinlichkeit, dass keine Kollision nach i Zuweisungen erfolgte, beträgt $\mathrm{P}(1) \cdot \mathrm{P}(2) \cdots \mathrm{P}(\mathrm{i})$.
Für $\mathrm{n} = 23$ und $\mathrm{m} = 365$ entspricht das dem *Geburtstagsparadoxon*:
Lediglich 23 zufällig ausgewählte Personen genügen, um mit Wahrscheinlichkeit $> 0,5$ zu gewährleisten, dass zwei von ihnen am selben Tag Geburtstag haben.
Für uns bedeutet es: Kollisionsfreiheit ist nicht völlig zu umgehen.

Zum Kollisionsproblem

Schauen wir uns die Wahrscheinlichkeitsrechnerei genauer an:

Die Wahrscheinlichkeit, dass k zufällig gewählte Schlüssel paarweise verschiedene Hashwerte haben, ist:

$$1 \cdot \frac{m-1}{m} \cdot \frac{m-2}{m} \cdots \frac{m-k+1}{m} = \prod_{i=0}^{k-1} \left(1 - \frac{i}{m}\right) \leq \prod_{i=0}^{k-1} e^{-i/m} = e^{-k(k-1)/2m}.$$

Diese Wahrscheinlichkeit wird kleiner als 50%, wenn $k \geq 1 + 1/2\sqrt{1 + 8m \ln 2}$.

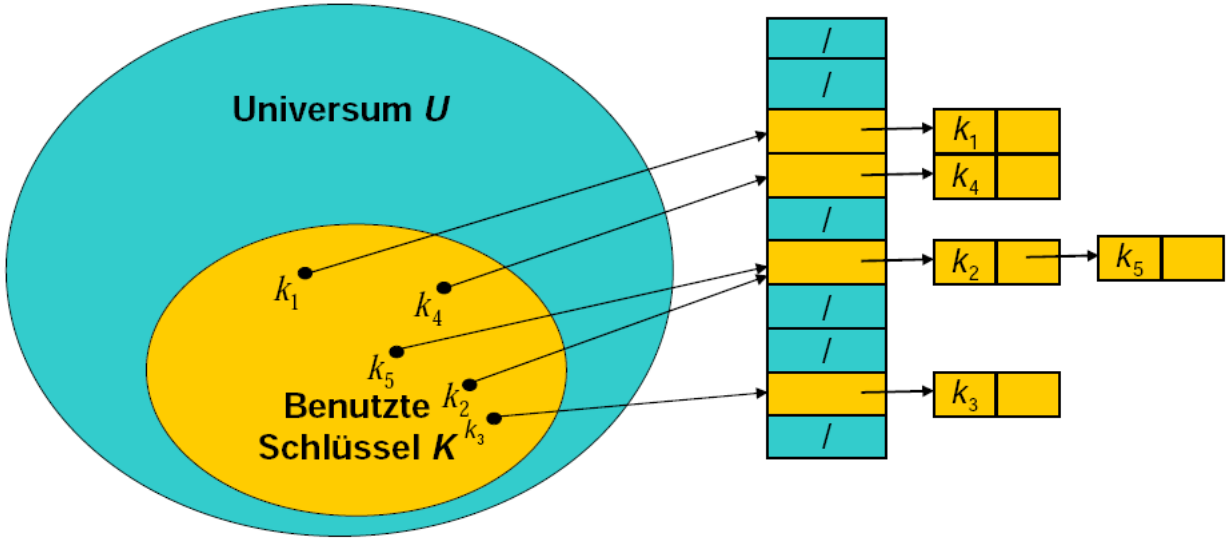
Man überprüfe den Fall $m = 365$ und $k = 23$.

Zum Umgang mit dem Kollisionsproblem

(1) *Hashing mit Verkettung* (auch *offenes Hashing* genannt):
Jeder Behälter wird durch eine beliebig erweiterbare Liste dargestellt.
Die Hashtabelle enthält daher Listenköpfe.

(2) *Hashing mit offener Adressierung*: Neben der Hashfunktion $\mathrm{h} = \mathrm{h}_0$ werden noch weitere Hashfunktionen h_i definiert, die eine Kollisionsbehandlung ermöglichen.

Hashing mit Listen - Illustration



Hashing mit Verkettung Betrachte $h : K \to S$.

$$\delta_h(x, y) := \begin{cases} 1 & \text{falls } x \neq y \text{ und } h(x) = h(y) \\ 0 & \text{sonst} \end{cases}$$

Wir erweitern auf Urschlüsselmengen $U \subseteq K$ mit:
$\delta_h(x, U) = \sum_{y \in U} \delta_h(x, y)$.
Ist U die Menge der abgespeicherten Urschlüssel, so ist $\delta_h(x, U)$ die Länge der Liste an der Stelle $h(x)$.
Wie groß ist $\delta_h(x, U)$? Wir betrachten zwei Fälle:
Im schlechtesten Fall: $|U|$; dann generiert Hashing zu linearer Suche.
Im mittleren Fall: Analyse folgt...

Hashing mit Verkettung (mittlerer Fall) Betrachte $h : K \to S = \{0, \ldots, m-1\}$.

Annahmen: (1) h verteilt K gleichmäßig.
(2) Die Urschlüsselmenge $U = \{x_1, \ldots, x_n\}$ wird gemäß Gleichverteilung und unabhängig voneinander "gezogen"
Frage: Wie groß ist $\delta_h(x, U)$ im Mittel ?
Sei $h_i = h(x_i)$. Die ZV h_i nimmt nach Annahmen die Zahlen in $[0..m-1]$ gleichwahrscheinlich an.
Wie groß ist die Wahrscheinlichkeit, dass ℓ der Zahlen $h_1, \ldots, h_n$ gleich einer bestimmten Zahl z sind ?
Lösung ist Produkt dreier Terme (siehe DSL VL 17/18):
$\binom{n}{\ell}$: # Möglichkeiten, ℓ Elemente aus n vielen auszuwählen
$m^{-\ell}$: Wahrscheinlichkeit für jedes der ausgewählten Elemente, mit z übereinzustimmen.
$\left(\frac{m-1}{m}\right)^{n-\ell}$: Wahrscheinlichkeit für jedes der nicht-ausgewählten Elemente, mit z nicht übereinzustimmen.
Dieses Produkt ist also die Wahrscheinlichkeit dafür, dass eine feste aber beliebige Liste die Länge ℓ besitzt.

Hashing mit Verkettung (mittlerer Fall) (Forts.)

$$
\begin{aligned}
E[\delta_h(x,\{x_1,\ldots,x_n\})] &= \sum_{\ell\geq 0} P(\delta_h(x,\{x_1,\ldots,x_n\})=\ell)\cdot\ell \\
&= \sum_{\ell\geq 1} \binom{n}{\ell} m^{-\ell}\left(\frac{m-1}{m}\right)^{n-\ell}\cdot\ell \\
&= \frac{n}{m}\cdot\sum_{\ell\geq 1}\frac{(n-1)!}{(\ell-1)!(n-\ell)!}\left(\frac{1}{m}\right)^{\ell-1}\left(\frac{m-1}{m}\right)^{n-\ell} \\
&= \frac{n}{m}\cdot\underbrace{\sum_{\ell\geq 1}\binom{n-1}{\ell-1}\left(\frac{1}{m}\right)^{\ell-1}\left(\frac{m-1}{m}\right)^{(n-1)-(\ell-1)}}_{\left(\frac{1}{m}+\frac{m-1}{m}\right)^{n-1}=1} \\
&= \frac{n}{m}
\end{aligned}
$$

Hashing mit Verkettung (mittlerer Fall) Betrachte $h : K \to S = \{0, \ldots, m-1\}$.

Vereinfachte Rechnung

Sei $L_p(n)$ Länge der p-ten Liste (nach n Einfügungen).

$L_p(n)$ ist eine ZV.

Aus Symmetriegründen gilt für den mittleren Fall (Erwartungswert): $E[L_p(n)] = E[L_q(n)]$ für beliebige $0 \leq p, q < m$.

Da h Hash-Funktion, gilt: $\sum_{i=0}^{m-1} L_i(n) = n$, und damit auch für den Erwartungswert:

$$n = E[\sum_{i=0}^{m-1} L_i(n)] = \sum_{i=0}^{m-1} E[L_i(n)] = m \cdot E[L_p(n)].$$

Die mittlere Listenlänge beträgt also n/m.

Insbesondere für $n \approx m$ ist das ein erfreuliches Ergebnis.

Hashing mit Verkettung (mittlerer Fall)

Unter den gemachten Annahmen können wir also schlussfolgern, wenn wir den Belegungsfaktor $\beta = \frac{n}{m}$ setzen:
1. Die mittleren Kosten der n-ten Einfügung sind $\mathcal{O}(\beta)$.
2. Die mittleren Kosten von n Einfügungen sind daher $\mathcal{O}(\beta n)$.
Der Belegungsfaktor entspricht dabei der mittleren Listenlänge.

Das bedeutet: Mittleres Verhalten von Hashing mit Verkettung ist sehr gut beim Einfügen, nämlich $\mathcal{O}(1)$, wenn β klein ist.
Platzausnutzung ist gut, wenn β nicht zu klein ist,
z.B. bei $\beta \approx \frac{1}{4}$ ist beides gewährleistet.

Ungünstige Belegungsfaktoren kann man durch entsprechendes Verdoppeln bzw. Halbieren der Hashtabelle (*Rehash*) vermeiden.
Dazu: *Familie von Hashfunktionen* $h_i : K \to [0..2^i - 1]$, $i = 1, 2, 3, \ldots$.
Benutze h_i, solange $2^{i-2} \leq n \leq 2^i$, also $\frac{1}{4} \leq \beta \leq 1$.
Falls $n = 2^i + 1$, verdoppele Tabelle: Speichere alle Elemente mit Hilfe von h_{i+1} in neue Tabelle.
Falls $n = 2^{i-2} - 1$, halbiere Tabelle: Umspeichern mit h_{i-1}.

Eigenschaften:
(1) Zugriff und Einfügen kosten weiterhin $\mathcal{O}(1)$ im Mittel.
(2) Übergang von h_i auf h_{i+1} bzw. auf h_{i-1} kostet $\mathcal{O}(2^i)$.
Lohnt sich der Aufwand ?

Verhältnis von Rehash-Kosten zu den übrigen Kosten

Beobachtungen:

(1) Nach einem Rehash ist $\beta = \frac{1}{2}$.

(2) Beginnen wir, mit h_i zu arbeiten, können wir (wegen (1)) wenigstens

$$\min(2^i - 2^{i-1}, 2^{i-1} - 2^{i-2}) \geq 2^{i-2}$$

Operationen mit h_i durchführen, bevor wieder ein Rehash nötig wird.

(3) Wenn wir die Kosten für das Rehash auf diese Operationen umlegen (*amortisierte Analyse*), so benötigt jede dieser Operationen immer noch $\mathcal{O}(1)$ Zeit.

Also: Rehash lohnt sich !

Hinweis: Der benutzte Verdoppelungs- bzw. Halbierungstrick ist oft einsetzbar.

Diskussion der gemachten Annahmen

Annahmen: (1) h verteilt K gleichmäßig,
Diese Annahme ist leicht zu erfüllen. Ist m ein Teiler von $N = |K|$, so genügt $h(x) = x \bmod m$ (*Divisionsmethode*).
Ist $N >> m$, so ist die Annahme (1) "fast" erfüllt, selbst wenn m kein Teiler von N ist.
(2) Die Urschlüsselmenge $U = \{x_1, \ldots, x_n\}$ wird gemäß Gleichverteilung und unabhängig voneinander "gezogen".
Dies ist im Grunde vom "Benutzer" der Hashfunktion abhängig, der aber wiederum die genaue Darstellung von K oft nicht kennt und daher (selbst wenn er wollte) dieses Kriterium nicht erfüllen kann.
Untersuchungen an praktischen Beispielen haben jedoch gezeigt, dass die theoretischen Ergebnisse "meist" mit denen aus der Praxis übereinstimmen.
Genaueres: Mehlhorn, Abschnitt II.2.1 über Symboltafeln bei Compilern

Folgerung aus den gemachten Annahmen

Satz: Unter den genannten Annahmen gilt:
Hashing mit Verkettung gewährleistet, dass die mittleren Kosten des Nachschlagens, Einfügens und Löschens eines Elementes eines Wörterbuches $\mathcal{O}(1)$ betragen.
Die Kosten im schlechtesten Fall sind jedoch linear, genauer $\mathcal{O}(n)$.
Der Platzbedarf ist ebenfalls linear, genauer $\mathcal{O}(m)$.

Die mittlere Länge der längsten Liste MLK

Unter den selben Annahmen gilt weiter:
Satz: Die mittleren Kosten für den schlechtesten Fall bei Hashing (mit Verkettung) von n Elementen betragen $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$, wenn $\beta \leq 1$.

Beweis: Sei ℓ_i die Länge der Liste i. $P[\ell_i \geq j] \leq \binom{n}{j} m^{-j}$.

$$P[\max_i \ell_i \geq j] \leq \sum_{i=0}^{m-1} P[\ell_i \geq j] \leq m \cdot \binom{n}{j} m^{-j} \leq n \cdot \left(\frac{n}{m}\right)^{j-1} \cdot \frac{1}{j!}$$

$$\mathrm{MLK} = E[\max_i \ell_i] = \sum_{j\geq 1} P[\max_i \ell_i \geq j] \leq \sum_{j\geq 1} \min\left(1, n \cdot \left(\frac{n}{m}\right)^{j-1} \cdot \frac{1}{j!}\right)$$

Sei $j_0 = \min\left\{j \mid n \cdot \left(\frac{n}{m}\right)^{j-1} \cdot \frac{1}{j!} \leq 1\right\} \leq \min\{j \mid n \leq j!\}$, da $n \leq m$ ($\beta \leq 1$).
Wegen $j! \geq (j/2)^{j/2}$ folgt $j_0 \in \mathcal{O}((\log n)/(\log\log n))$ und so die Beh.

Ein hilfreicher Rechentrick

Für ZV X gilt: $P[X = k] = P[X \geq k] - P[X \geq k+1]$.
Das liefert für den Erwartungswert:

$$E[X] = \sum_k P[X = k]k = \sum_k k \cdot (P[X \geq k] - P[X \geq k+1]) = \sum_k P[X \geq k]$$

Das sieht man leicht ein durch die Struktur der Summanden.

Die mittlere Länge der längsten Liste MLK: Wozu ?
Lastverteilung bei paralleler Abarbeitung
Simulation eines globalen Speichers durch massiv parallele Architektur

Das nächste Mal

Hashing mit offener Adressierung

(Perfektes Hashing)

Sollten Sie Schwierigkeiten beim Nachvollziehen der Analysen im mittleren Fall haben, so mag das an mangelnden Kenntnissen (im Umgang mit oder in überhaupt) Wahrscheinlichkeitsrechnung liegen.
Bitte bemühen Sie sich, entsprechende Kenntnisse zu erwerben.
Die Formeln in den folgenden Vorlesungen werden nicht angenehmer...