

Ausagen:

Morgen: Projekt-Kickoff ab 14:15 Uhr

Hashing

→ gestreute Speicherung

Alternative zu Suchbäumen

unterstützt: - Suchen

- Einfügen
- Löschen (partiell)

Vorteil von Hashing gegenüber Suchb.:
 Aufwand pro Operation $O(1)$ im Mittel!

aber: schlechter als Suchbäume im Worst Case, nämlich $O(n)$.

Grundidee: Reserviere „Hash-Tabelle“
 im Speicher, in der die abgespeicherten
 Daten stehen (Vorstellung: array).

gegeben: Datensätze mit Schlüssel



$U = \text{Universum}$

z.B. alle Studierende
 der TU Berlin

Hashfkt.

$$h: U \rightarrow \{0, 1, \dots, m-1\}$$

einfachste Lösung („direct addressing“):
 h bijektive Abbildung

Nachteil: Hashtabelle (array) belegt viel mehr Speicher als nötig, falls U deutlich größer als die interessante Teilmenge ist.

$$u := \# \text{ Datensätze in Teilmenge} \ll |U|$$

Wichtig: Man kennt a priori nicht die Schlüssel der u Datensätze. Ganz U kommt in Frage.

Ziel: Größe der Hashtabelle soll ungefähr Anzahl der Datensätze entsprechen.



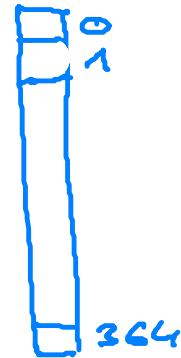
\Rightarrow i.A. gibt es $k_1 \neq k_2$ mit $h(k_1) = h(k_2)$.
Dies nennt man Kollision.

Kollisionen sind sehr wahrscheinlich,
lassen sich kaum vermeiden.

Experiment:



TU - Studis



Jan. 22

Feb. 8, 7

März 1

April 1

Mai 31

Juni 7, 3

Juli

Aug 22, 15, 14, 13, 23

Sept. 27, 30, 14

Okt. 6, 17, 5, 27

Nov. 3

Dez. 16, 24, 11

„Geburtsdays-
paradoxon“

$$n = 26$$

$$m = 365$$

Wahrscheinlichkeit dafür, dass keine
Kollision auftritt, unter der Annahme, dass
die $h(k)$ zufällig und gleichverteilt.

$$q(m, u) = \frac{m}{m} \cdot \frac{m-1}{m} \cdot \frac{m-2}{m} \cdot \dots \cdot \frac{m-u+1}{m} = \frac{m!}{(m-u)! \cdot m^u}$$

↑ ↑
Größe # Datensätze
des
Hash-Tab.

Wahrscheinlichkeit für eine Kollision:

| u | 20 | 23 | 30 | 50 | ... |
|---------------|-------|-------|-------|------|-----|
| $1 - q(m, u)$ | 0,411 | 0,507 | 0,706 | 0,97 | - - |

$m = 365$

Folgerung: Kollisionen treten fast sicher auf \rightarrow Hashverfahren muss mit Kollisionen umgehen können.

Vorher: Was sind gute Hash-Fkt.?

Man möchte, dass h gleichmäßig über den Adressraum streut, d.h. jede Adresse wird mit der gleichen Wahrscheinlichkeit $\frac{1}{m}$ gewählt („uniform hashing“)

In der Praxis ist dies nur angenähert realisierbar. Nutze diese Annahme jedoch für Analyse:

$$\forall j \quad \sum_{k \in U} p(k) = \frac{1}{m} \quad \text{mit } p(k) = \text{W'keit für Auftreten}$$

$h(k) = j$
W'keit dafür,
dass Speicherstelle j
getroffen wird.

von Schlüssel
 k in einer
Iteration.

Im Folgenden: Schlüssel k sind immer
ganze Zahlen.

Divisionsmethode: $h(k) = k \bmod m$

Dadurch wird Projektion von \mathbb{N} auf
 $\{0, \dots, m-1\}$ erreicht.

z.B.: $m = 2^q$
 k Binärzahl

h schneidet in diesem
Fall die q vordevsten Bits
ab und behält nur die
 q kleinsten Bits.

→ könnte u.U. zur Häufung von
Kollisionen führen.

Empirisch gute Wahl für m :

Primzahl weit entfernt von 2er-Potenzen.

Multiplikationsmethode:

Wähles festes $0 < A < 1$ und def.:

$$h(k) = \lfloor m \cdot (k \cdot A - \lfloor k \cdot A \rfloor) \rfloor$$

$$\underbrace{\underbrace{E[0,1]}_{E[0,m]}}_{E\{0,1,\dots,m-1\}}$$

z.B. $m=2^9$

empirisch gute Wahl für A : $\frac{\sqrt{5}-1}{2} \approx 0,618\dots$

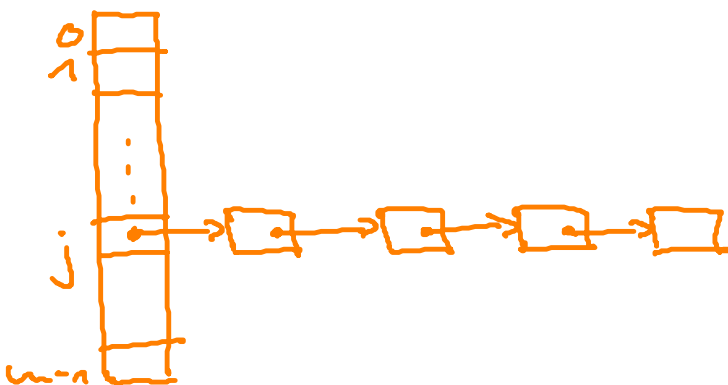
Bsp: $k=123456$, $m=10.000$, $A=\frac{\sqrt{5}-1}{2}$

$$h(k) = \lfloor 10.000 \cdot \underbrace{(123456 \cdot A - \lfloor 123456 \cdot A \rfloor)}_{\approx 0,0041151\dots} \rfloor = 41$$

jetzt: Umgang mit Kollisionen

- 1) Hashing mit Überlauf ("chaining")
- 2) Hashing mit Ersatzadresse ("open addressing")

ad 1) Lege pro Platz in der Hash-Tabelle eine Liste an; speichere alle Datensätze k mit $h(k)=j$ in der Liste zum Speicherplatz j ab.



Aufwand für das Einfügen (am Beginn der Liste): $O(1)$

Aufwand für Suchen: (im Mittel)

Ausgaben:

- Hashfkt. streut gut („uniform hashing“)
- Es sind n Datensätze in der Hash-Tabelle

⇒ im Mittel sind $\frac{n}{m}$ Datensätze pro Liste vorhanden.

⇒ Erfolgreiche Suche erfordert im Mittel Aufwand $O\left(1 + \frac{n}{m}\right)$

↑
Berechnung
von $h(k)$

↑
Durch-
gehen der
Liste

$\alpha = \frac{n}{m}$ „Auslastung der Hash-Tabelle“

Wähle typischerweise $m \in \Theta(n)$

Aufwand $O(1 + \alpha) = O(1)$

↑
Konstante

Betrachte noch erfolgreiche Suche:

Beachte: Die Liste $h(k)$ enthält mindestens ein Element (nämlich Element mit Schlüssel k). Daher ist die mittlere Länge der Liste an der Stelle $h(k)$ durch $1 + \frac{n-1}{m}$ beschw.

\Rightarrow Aufwand: $O(1 + 1 + \alpha)$
 \uparrow \uparrow \uparrow
 Berechne Länge der
 $u(k)$ Liste inkl. k

Aufwand für die 3 Operationen:

Einfügen: $O(1)$

Suchen: $O(1+\alpha)$ im Mittel

Löschen: $O(1+\alpha)$

\Rightarrow alle Operationen erfordern im Mittel konstanten Aufwand.

Aber: Worst-Case-Aufwand für Suchen ist $\Omega(n)$ (alle n Schlüssel in derselben Liste)

Bsp: Divisionsmethode

$n = 2000$ 3 Vergleiche im Mittel ok

$\Rightarrow \frac{n}{m} \approx 3 \Rightarrow m \approx \frac{n}{3}$

Wähle Primzahl $m = 701$, weit entfernt von Zweer-Potenzen 512, 1024

$\rightarrow h(k) = k \bmod 701$

hier: # Vergleiche bestimmt Größe der Hash-Tabelle

generell: Tradeoff zwischen Größe der Hash-Tabelle (Speicher) und # Vergl. beim Suchen (Zeit).