




Informatik II

Hashing

G. Zachmann
 Clausthal University, Germany
zach@in.tu-clausthal.de



Das Wörterbuch-Problem

- Das **Wörterbuch-Problem (WBP)** kann wie folgt beschrieben werden:
 - Gegeben: Menge von Objekten (Daten) die über einen eindeutigen Schlüssel (ganze Zahl, String, . . .) identifizierbar sind
 - Gesucht: Struktur zu Speicherung der Objektmenge, so daß mindestens die folgenden Operationen (Methoden) effizient ausführbar sind:
 - Suchen (Wiederfinden, Zugreifen)
 - Einfügen, Entfernen
- Andere Namen: *Dictionary, Symbol-Table, assoziatives Array*
- In Python: Datenstruktur **Dictionary**
 - so gesehen sind Hash-Tabellen / Dictionaries eine Verallgemeinerung von Arrays

```
d = {}
d["hallo"] = 12
d["Paul"] = 18
```

G. Zachmann Informatik 2 - SS 06 Hashing 2



Hash-Tabellen - Beispiele


- Memory Management: Tabelle im Betriebssystem
- Comiler: Variablen/Identifier in einem Programm


```
i int 0x87C50FA4
j int 0x87C50FA8
x double 0x87C50FAC
name String 0x87C50FB2
...
```
- Environment-Variablen in Unix (Variablenname, Attribut)


```
EDITOR=emacs
GROUP=mitarbeiter
HOST=vulcano
HOSTTYPE=sun4
LPDEST=hp5
MACHTYPE=sparc
...
```
- ausführbare Programme


```
PATH=~:/bin:/usr/local/gnu/bin:/usr/local/bin:/usr/bin:/bin:
```

G. Zachmann Informatik 2 - SS 06 Hashing 3



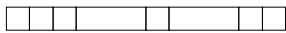
- Folgende Bedingungen können die Wahl einer Lösung des WBP beeinflussen:
 - Ort an dem die Daten gespeichert werden: Hauptspeicher, Platte, CDROM
 - Häufigkeit der Operationen:
 - überwiegend Einfügen & Löschen (dynamisches Verhalten)
 - überwiegend Suchen (statisches Verhalten)
 - annähernd Gleichverteilung
 - nichts bekannt
 - Weitere zu implementierende Operationen:
 - Durchlaufen der Menge in bestimmter Reihenfolge (etwa nach Schlüsselwert aufsteigend)
 - Mengen-Operationen: Vereinigung, Durchschnitt, Differenz, . . .
 - Aufspalten
 - Konstruieren
 - Kostenmaße zur Beurteilung der Lösung: Average-, Worst-, amortisierter Worst-Case
 - Ausführungsreihenfolge der Operationen:
 - sequentiell
 - nebenläufig

G. Zachmann Informatik 2 - SS 06 Hashing 4

Verschiedene Ansätze zur Lösung des WBP

- Strukturierung der aktuellen Schlüsselmenge: Listen, Bäume, Graphen, ...
- Aufteilung des gesamten Schlüssel-Universums: *Hashing*
- Hashing* (engl.: *to hash* = zerhacken) beschreibt eine spezielle Art der Speicherung einer Menge von Schlüsseln durch Zerlegung des Schlüssel-Universums
- die Position des Daten-Elements im Speicher ergibt sich (zunächst) durch **Berechnung direkt aus dem Schlüssel**
- Ort des Datensatzes d : Berechnung aus dem Schlüssel s von d
→ keine Vergleiche
→ konstante Zeit
- Datenstruktur: lineares Array der Größe m → **Hash-Tabelle**

Schlüssel s 0 1 2 i $m-2$ $m-1$



Der Speicher wird zerlegt in m gleich große Behälter (Buckets).

G. Zachmann Informatik 2 - SS 06 Hashing 5

Implementierung in Python

```
class TableEntry( object ):
    def __init__( key, value ):
        self.key = key
        self.value = value

class HashTable( object ):
    def __init__( capacity ):
        self.capacity = capacity
        self.table = capacity * [ None ]
        # wir nehmen hier an, daß entry ein statisches Array sei,
        # mit fester Größe, wie das auch in C++/Java der Fall wäre

    # Hash-Funktion
    def h( key ): ...

    # Füge value mit Schlüssel key ein, falls noch nicht vorhanden
    def insert( key, value ): ...

    # Lösche Element mit key aus Tabelle, falls vorhanden
    def delete( key ): ...

    # Suche Element mit key und liefere dessen Wert
    def search( key ): ...
```

G. Zachmann Informatik 2 - SS 06 Hashing 6

Idee des Hashings

- Notation:
 - U = Universum aller möglichen Keys
 - K = Menge von Keys gerade im Wörterbuch
 - $|K| = n$
- wenn U sehr groß, ist Array für ganz U nicht praktikabel
 - Außerdem: $|K| \ll |U|$
- Idee der Hash-Tabelle: benutze eine Tabelle, deren Größe proportional zu $|K|$ ist
 - definiere Funktion, die die Keys auf die Slots der Hash-Tabelle abbildet

G. Zachmann Informatik 2 - SS 06 Hashing 7

- Hash-Funktion:** Abbildung von U in alle Slots einer Hash-Tabelle $T[0..m-1]$

$$h : U \rightarrow \{0, 1, \dots, m-1\}$$
- bei Arrays: Key k wird abgebildet in den Slot $A[k]$
- bei Hash-Tabellen: Key k wird abgebildet in den Slot $T[h(k)]$ ab ("k hashes to ...")
- $h(k)$ ist der **Hash-Wert** oder **Hash-Adresse** des Keys k

G. Zachmann Informatik 2 - SS 06 Hashing 8

- Normalerweise gilt $|U| \gg m \Rightarrow h$ kann nicht injektiv sein \Rightarrow Kollisionen sind unvermeidlich
- Belegungsfaktor (load factor):

$$\alpha = \frac{\# \text{ gespeicherter Keys}}{\text{Größe der Hash-Tabelle}} = \frac{|K|}{m}$$

G. Zachmann Informatik 2 - SS 06 Hashing 9

- Hash-Verfahren :=
 - möglichst "gute" Hash-Funktion +
 - Strategie zur Auflösung von Adresskollisionen
- Annahme im Folgenden: Tabellengröße m ist fest

G. Zachmann Informatik 2 - SS 06 Hashing 11

Anforderungen an gute Hash-Funktionen

- Eine Kollision tritt dann auf, wenn bei Einfügen eines Elementes mit Schlüssel k der Slot $T[h(k)]$ schon belegt ist
- Eine Hash-Funktion h heißt **perfekt für eine Menge von Schlüsseln** S , falls keine Kollisionen für S auftreten
- Ist h perfekt und $|S| = n$, dann gilt: $n \leq m$, der **Belegungsfaktor** (BF) der Hash-Tabelle ist $\frac{n}{m} \leq 1$
- Eine Hash-Funktion ist **gut**, wenn:
 - für viele Schlüssel-Mengen auch bei hohem Belegungsfaktor die Anzahl der Kollisionen möglichst klein ist
 - sie effizient zu berechnen ist

G. Zachmann Informatik 2 - SS 06 Hashing 12

Beispiel einer Hash-Funktion

- $U = \{\text{alle Namen von Variablen/Identifiern in einer Programmiersprache}\}$
- Beispiel: Hash-Funktion für Strings


```
# m = size of table
def h( s, m ):
    k = 0
    for i in range( 0, len(s) ):
        k += int( s[i] )
    return k % m
```
- Folgende Hash-Adressen werden generiert für $m = 13$:

Schlüssel s	h(s)
Test	0
Hallo	2
SE	9
Algo	10
- h wird besser, je größer m gewählt wird

G. Zachmann Informatik 2 - SS 06 Hashing 13

Zur Wahrscheinlichkeit einer Kollision

- Die Anforderungen "hoher Belegungsfaktor" und "Kollisionsfreiheit" stehen in Konflikt zueinander
- Für eine Menge K , mit $|K|=n$, und Tabelle T , mit $|T|=m$, gilt:
 - für $n > m$ sind Konflikte unausweichlich
 - für $n < m$ gibt es eine (Rest-) Wahrscheinlichkeit $P(n, m)$ für das Auftreten mindestens einer Kollision
- Wie findet man eine Abschätzung für $P(n, m)$?
 - für beliebigen Schlüssel s ist die W'keit dafür, daß $h(s) = j$ mit $j \in \{0, \dots, m-1\}$: $P[h(s) = j] = \frac{1}{m}$ falls Gleichverteilung gilt
 - es ist $P(n, m) = 1 - \bar{P}(n, m)$, wenn $\bar{P}(n, m)$ die W'keit dafür ist, daß es beim Speichern von n Elementen in m Slots zu **keinen** Kollisionen kommt

G. Zachmann Informatik 2 - SS 06

Hashing 14

- werden n Schlüssel nacheinander auf die Slots T_0, \dots, T_{m-1} verteilt (bei Gleichverteilung), gilt jedes mal $P[h(s) = j] = \frac{1}{m}$
- die W'keit für **keine** Kollision im Schritt i ist $p_i = \frac{m-(i-1)}{m}$
- damit ist

$$P(n, m) = 1 - p_1 \cdot p_2 \cdot \dots \cdot p_n = 1 - \frac{m(m-1) \cdot \dots \cdot (m-n+1)}{m^n}$$

- Beispiel: Es ist etwa $P(23,365) > 50\%$ und $P(50,365) \approx 97\%$ (Geburtstagsparadoxon)

G. Zachmann Informatik 2 - SS 06

Hashing 15

Gebräuchliche Hash-Funktionen

- Zunächst Divisions-Methode
- für $U = \text{Integer}$ wird die Divisions-Rest-Methode verwandt:

$$h(s) = (a \times s) \bmod m \quad (a \neq 0, a \neq m, m \text{ Primzahl})$$
- Für Zeichenreihen der Form $s = s_1 s_2 \dots s_k$ nimmt man etwa:

$$h(s) = \left(\left(\sum_{i=1}^k B^i s_i \right) \bmod 2^w \right) \bmod m$$
 - etwa mit $B = 131$ und $w = \text{Wortbreite des Rechners}$ ($w = 32$ oder $w = 64$ ist üblich).

G. Zachmann Informatik 2 - SS 06

Hashing 16

- (Einfache) Divisions-Rest-Methode:

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

- Wahl von m ?
- Beispiele:
 - m gerade $\rightarrow h(k)$ gerade $\Leftrightarrow k$ gerade
 - Problematisch, wenn letztes Bit Sachverhalt ausdrückt (z.B. 0 = weiblich, 1 = männlich)
 - $m = 2^p$ liefert p niedrigsten Dualziffern von k , d.h., höhere Ziffern gehen gar nicht in die Hash-Adresse ein
- Regel: Wähle m prim, wobei m keine Zahl $2^i \pm j$ teilt, wobei i und j kleine, nicht-negative Zahlen sind, d.h., wähle m prim, aber nicht "zu nahe" an einer 2-er-Potenz

G. Zachmann Informatik 2 - SS 06

Hashing 17

Multiplikative Methode

- Satz von Vera Turán Sós [1957]:
Sei Θ eine irrationale Zahl. Platziert man die Punkte $\Theta - \lfloor \Theta \rfloor, 2\Theta - \lfloor 2\Theta \rfloor, \dots, n\Theta - \lfloor n\Theta \rfloor$ in das Intervall $[0, 1]$, dann haben die $n + 1$ Intervallteile höchstens 3 verschiedene Längen. Außerdem fällt der nächste Punkt $(n + 1)\Theta - \lfloor (n + 1)\Theta \rfloor$ in eines der größten (schon existierenden) Intervallteile.
- Fazit: die so gebildeten "Punkte" liegen ziemlich gleichmäßig gestreut im Intervall $[0, 1]$
- Es gilt: von allen Zahlen $\theta, 0 \leq \theta \leq 1$, führt der goldene Schnitt
$$\theta^{-1} = \frac{\sqrt{5} - 1}{2} \approx 0.6180339$$
 zur gleichmäßigsten Verteilung.

G. Zachmann Informatik 2 - SS 06 Hashing 18

- Wähle Konstante $\theta, 0 < \theta < 1$
 1. Berechne $k\theta \bmod 1 := k\theta - \lfloor k\theta \rfloor$
 2. $h(k) = \lfloor m(k\theta \bmod 1) \rfloor$
- Beispiel:

$$\theta^{-1} = \frac{\sqrt{5} - 1}{2} \approx 0.6180339$$

$$k = 123456$$

$$m = 10000$$

$$h(k) = \lfloor 10000(123456 \cdot 0.61803 \dots \bmod 1) \rfloor$$

$$= \lfloor 10000(76300.0041151 \dots \bmod 1) \rfloor$$

$$= \lfloor 41.151 \dots \rfloor = 41$$

G. Zachmann Informatik 2 - SS 06 Hashing 19

Praktische Berechnung von h

- Wahl von m unkritisch \rightarrow wähle $m = 2^p$
- Ann.: k passe in ein einzelnes Wort, d.h., k hat w Bits
- Wähle $\theta = \frac{s}{2^w}$ mit $0 < s < 2^w$
- Dann ist $k \cdot \theta = \frac{k \cdot s}{2^w} = \frac{r_1 2^w + r_0}{2^w}$
- r_1 ist der ganzzahlige Teil von kA ($\lfloor kA \rfloor$) und r_0 ist der gebrochene Rest ($kA \bmod 1 = kA - \lfloor kA \rfloor$)

G. Zachmann Informatik 2 - SS 06 Hashing 20