Informatik II: Algorithmen und Datenstrukturen SS 2013

Vorlesung 5b, Mittwoch, 15. Mai 2013

(Univ. Hashing Teil 2, Rehash, Cuckoo Hashing)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Weiter geht's mit (universellem) Hashing
  - HashSet vs. HashMap
  - Beispiele für universelle Klassen von Hashfunktionen
  - Dynamische Schlüsselmengen, Rehash
  - Cuckoo Hashing
  - Nochmal Histogramme

Hash Set vs. Hash Map

UNI FREIBURG

- Im Prinzip dasselbe, bis auf:
  - Bei der Hash Map werden key-value Paare gespeichert, wie bisher immer erklärt und gemalt
  - Bei einem Hash Set werden **nur** die keys gespeichert, es gibt darüber hinaus keine Values

    In vielen Anwendungen reicht das
  - In den folgenden Erklärungen / Beispielen werden wir der Einfachheit halber die values oft weglassen

Klassen von Hashfunktionen 1/5

- Negativbeispiel 1 H
  - Die Menge aller h mit $h(x) = a \cdot x + b \bmod m$

    für $a, b \in U$

    Das heißt: um eine zufällige solche Hashfunktion zu wählen, wählt man einfach zufällig a und b aus U
  - Das sind $|U|^2$ mögliche Hashfunktionen, also viele
  - Aber trotzdem **nicht** universell

für festes c, z.B. $c = 2$

Falls universell: $\forall x, y \in U \; x \neq y \quad |\{h \in H : h(x) = h(y)\}| \leq c \cdot \frac{|H|}{m}$

Um zu zeigen dass nicht universell reicht ein Paar $x, y \in U \; x \neq y \quad |\{h \in H : h(x) = h(y)\}| > c \cdot \frac{|H|}{m}$

Nehme z.B. $x = m$, $y = 2m$

$\Rightarrow a \cdot x + b \bmod m = b \bmod m$ (durch m teilbar)

$a \cdot y + b \bmod m = b \bmod m$

$\Rightarrow \forall h \in H : h(x) = h(y)$

$\Rightarrow |\{h \in H : \text{---} \; n \; \text{---}\}| = |H|$

z.B. $m = 10$, $x = 10$, $y = 20$, $a =$ irgendwas, $b = 7$

Klassen von Hashfunktionen 2/5

- Negativbeispiel 2
  - Die Menge aller Funktionen von $U \to \{0, \ldots, m-1\}$
  - Ist 1-universell
  - Aber als Klasse von Hashfunktionen ungeeignet

= wie „zufälliges Werfen"

weil: für ein zufälliges h_z aus dieser Menge ist $h_z(x)$ zufällig aus $\{0, \ldots, m-1\}$

und wir hatten ja schon gesehen, dass bei zufälligem Werfen $\Pr(\text{Kollision von } x \neq y) = m \cdot \frac{1}{m} \cdot \frac{1}{m} = \frac{1}{m}$

weil für ein solche „zufälliges Werfen" Funktion bräuchte man $\Theta(|S|)$ Platz zum Speichern und $\Theta(|S|)$ Zeit zum Auswerten! $|S| = n$

Klassen von Hashfunktionen 3/5

UNI FREIBURG

- Positivbeispiel 1
  - Sei p eine Primzahl mit $p > m$ und $U \subseteq \{0, \ldots, p-1\}$
  - Sei H die Menge aller h mit $h(x) = (a \cdot x + b) \bmod p \bmod m$

    wobei $a, b \in \{0, \ldots, p-1\}$
  - Die ist $\approx\mathbf{1}$-universell

    Siehe Exercise 4.11 in Mehlhorn/Sanders

Klassen von Hashfunktionen 4/5

UNI FREIBURG

- Positivbeispiel 2
  - Die Menge aller h mit $h(x) = a \bullet x \bmod m$, für ein $a \in U$
    - Schreibe $a = \Sigma_{i=0..k-1}\, a_i \cdot m^i$, wobei $k = \text{ceil}(\log_m |U|)$
    - Entsprechend $x = \Sigma_{i=0..k-1}\, x_i \cdot m^i$
    - Dann $a \bullet x := \Sigma_{i=0..k-1}\, a_i \cdot x_i$
    - Intuitiv: das "Skalarprodukt" der Darstellung zur Basis m
  - Die ist **1**-universell, siehe Theorem 4.4 in Mehlhorn/Sanders

z.B. $m = 10$, $U = \{0, \ldots, u-1\}$ $u = 10^3 = 1000$

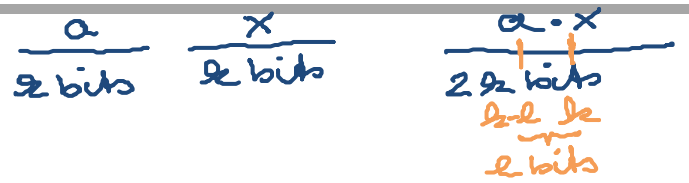
z.B. $a = 987 = 9 \cdot 10^2 + 8 \cdot 10 + 7 \cdot 1$

$x = 123 = 1 \cdot 10^2 + 2 \cdot 10 + 3 \cdot 1$

dann $a \bullet x = 9 \cdot 1 + 8 \cdot 2 + 7 \cdot 3 = 46$

$a \bullet x \bmod 10 = 6$

Klassen von Hashfunktionen 5/5



- Positivbeispiel 3
  - Die Menge aller h mit $h(x) = a \cdot x \bmod 2^k \operatorname{div} 2^{k-\ell}$ für $a \in U$

    ... wobei $|U| = 2^k$, $m = 2^\ell$ ... in der Regel $k >> \ell$

    Das $\cdot$ ist hier wieder das normale Produkt

    Das heißt $a \cdot x$ gibt eine Zahl aus $0..|U|^2$

    Die lässt sich also in Binärdarstellung mit $2k$ Bits darstellen

    Eine Position in der Hashtabelle lässt sich mit ℓ Bits darstellen

    $h(x)$ ist dann einfach der Wert der Bits $k-l..k-1$ von $a \cdot x$
  - Diese Menge von Hashfunktionen ist **2**-universell

    Siehe Exercise 4.14 in Mehlhorn / Sanders

Dynamische Schlüsselmengen 1/5

UNI FREIBURG

- Bisherige Annahme
  - Die Schlüsselmenge S ist vorher bekannt, insbes. $n = |S|$
  - Dann kann man leicht die Größe der Hashtabelle als $m = \Theta(n)$ wählen, so dass die **erwartete** Zeit pro Operation $\Theta(1)$ ist
  - Das gilt aber eben nur im Erwartungsfall = "im Durchschnitt"
  - Was, wenn die zufällig gewählte Hashfunktion schlecht ist ?

    Das ist unwahrscheinlich aber möglich

    Genau so, wie es auch für zufällige Schlüssel unwahrscheinlich aber möglich ist, dass alle bzw. übermäßig viele von der Funktion $h(x) = x \bmod m$ auf dieselbe Zahl abgebildet werden

Dynamische Schlüsselmengen 2/5

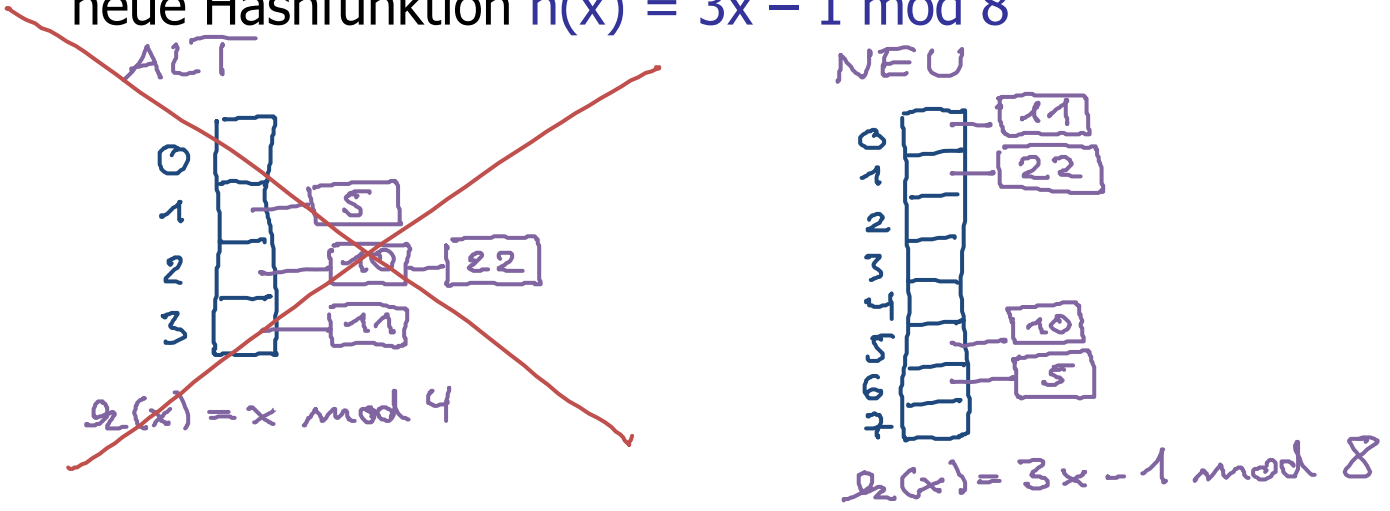
UNI FREIBURG

- Wachsende Schlüsselmengen
  - Wenn nach und nach Schlüssel dazu kommen, gibt es noch ein weiteres Problem:

    Die Größe der Hashtabelle reicht irgendwann nicht mehr aus, um Zeit $\Theta(1)$ pro Operation zu gewährleisten

Dynamische Schlüsselmengen 3/5

- Lösung für beide Probleme: Rehash
  - Bei einem Rehash wird einfach:
    1. Eine neue Hashfunktion ausgewählt
    2. Die Elemente von der alten in die neue Tabelle kopiert
    3. Die alte Tabelle gelöscht
  - Beispiel: S = {5, 10, 11, 22}, alte Hashf. h(x) = x mod 4, neue Hashfunktion h(x) = 3x – 1 mod 8



Dynamische Schlüsselmengen 4/5

UNI FREIBURG

- Kosten für einen Rehash
  - Ein Rehash ist teuer: er kostet Zeit $\Theta(n)$, wobei n die Anzahl Elemente zum Zeitpunkt des Rehash ist
  - Wenn man es richtig macht, ist er allerdings selten nötig:

    Bei einer universellen Klasse von Hashfunktionen, kommt es selten vor, dass eine Hashfunktion nicht gut ist

    Wenn die Hashtabelle zu klein geworden ist, und man die neue Hashtabelle doppelt so groß wählt ($m \rightarrow 2m$), dauert es lange, bis man wieder vergrößern muss

    Diese "Verdoppelungsstrategie" analysieren wir in einer späteren Vorlesung genauer (amortisierte Analyse)

Dynamische Schlüsselmengen 5/5

UNI FREIBURG

- Verkleinerung der Schlüsselmenge
  - Die Schlüsselmenge kann auch wieder kleiner werden, indem Schlüssel gelöscht werden

    in Java: remove(key) … in C++: erase(key)
  - Wenn |S| << m wird, kann man die Hashtabelle auch wieder verkleinern … siehe ebenfalls spätere Vorlesung
  - Macht man aber in der Praxis oft nicht, weil:

    1. In sehr vielen Anwendungen braucht man nur insert und lookup, kein remove bzw. erase

    2. Zu irgendeinem Zeitpunkt braucht man sowie den Platz für die maximale Anzahl Schlüssel der Anwendung

Perfektes Hashing

UNI FREIBURG

- Definition

  – Eine Hashfunktion $h : U \to \{0, \ldots, m-1\}$ heißt **perfekt** für eine Menge S, wenn Sie injektiv auf S ist

  Das heißt, für alle $x, y \in S$ mit $x \neq y$ gilt $h(x) \neq h(y)$

  – Eine perfekte Hashfunktion heißt **minimal**, wenn $m = |S|$

  – Beispiel dafür: $S = \{3, 17, 84, 22\}$ $h(x) = x \bmod 4$

  – **Hinweis:** es gibt Verfahren, um für ein gegebenes S eine Hashfunktion $h : U \to \{0, \ldots, m-1\}$ zu konstruieren, so dass:

    1. h ist eine perfekte Hashfunktion für S mit $m = \Theta(|S|)$
    2. h kann in $\Theta(m)$ Platz gespeichert werden
    3. Für alle $x \in U$ kann $h(x)$ in Zeit $\Theta(1)$ berechnet werden

Cuckoo Hashing 1/5

UNI FREIBURG

- Informale Beschreibung
  - Es gibt eine Hashtabelle der Größe m wie gehabt
  - Es gibt **zwei** Hashfunktionen $h_1, h_2 : U \to \{0, \ldots, m-1\}$
  - Jede Position der Hashtabelle hat nur Platz für **ein** Element
  - Versuche ein neues Element x bei $h_1(x)$ zu speichern
  - Falls schon belegt von einem Element y, dann speichere y bei $h_i(y)$ falls vorher bei $h_j(y)$ gespeichert, $\{i, j\} = \{1, 2\}$
  - Falls neuer Platz für y belegt von einem Element z, dann verfahre genau so mit z … und so weiter

Cuckoo Hashing 2/5

UNI FREIBURG

- Informale Beschreibung, Fortsetzung
  - Es kann so zu einem **Zyklus** kommen:

    x schmeißt y_1 raus, y_1 schmeißt y_2 raus, y_2 schmeißt y_3 raus, ..., $y_{\ell-1}$ schmeißt y_ℓ raus, y_ℓ schmeißt wieder x raus, und der alternative Platz für x ist auch besetzt

    Das passiert insbesondere dann, wenn ein y_i rausgeschmissen wird und $h_1(y_i) = h_2(y_i)$

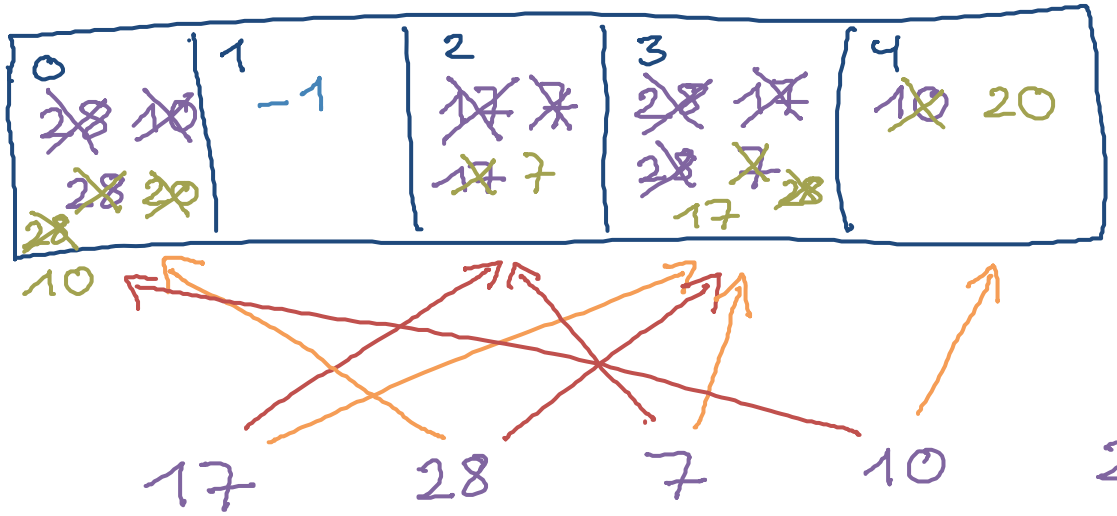
  - Wenn das passiert, wählt man neue Hashfunktionen h_1 und h_2 und macht einen Rehash wie erklärt

Cuckoo Hashing 3/5

- Beispiel (ohne Rehash falls |S| > m/2)

  m = 5, $h_1(x) = x \bmod 5$, $h_2(x) = 2x - 1 \bmod 5$

  insert(17), insert(28), insert(7), insert(10), lookup(17) → YES, lookup(8) → NO, insert(20)



ZYKLUS!

$h_1(x) = x \bmod 5$

$h_2(x) = (2x - 1) \bmod 5$

HINWEIS: man braucht ein Element das heißt „KEINE ZAHL HIER" NO KEY z.B. −1

Cuckoo Hashing 4/5

UNI FREIBURG

- Wahl der Hashfunktionen
  - Die Hashfunktionen wählt man einfach wieder (beide, unabhängig voneinander) aus einer universellen Klasse
  - Dann kann man zeigen, dass es hinreichend selten zu einem Zyklus (und dem dann nötigen teuren Rehash) kommt, solange $|S| \leq m/2$
  - Bei wachsender Schlüsselmenge wie gehabt ein Rehash mit $m \rightarrow 2m$, sobald $|S| > m/2$

Cuckoo Hashing 5/5

UNI FREIBURG

- Laufzeit
  - Die Laufzeit von lookup(x) ist **immer** $\Theta(1)$

    Man muss ja immer nur an zwei Positionen nachschauen, nämlich $h_1(x)$ und $h_2(x)$ ... das ist gerade der Clou an CH !
  - Dasselbe gilt dann auch für remove(x)

    An beiden Positionen nachschauen, und wo gefunden einfach löschen, die Position ist dann für zukünftige insert Operationen wieder frei
  - Man kann zeigen, dass ein insert(x) **im Durchschnitt** in Zeit $\Theta(1)$ geht

    Beweis siehe Referenzen ... aber nicht Klausur-Elefant

Hashfunktionen für beliebige Objekte

- Bisher haben wir nur Zahlen gehasht
  - Es lässt sich aber leicht aus jedem beliebigem Objekt eine Zahl herleiten, im einfachsten Fall:

    Das Objekt ist in k Bytes $B_0 \ldots B_{k-1}$ im Rechner repräsentiert

    Fasse diese Bytes als Zahl auf: $\Sigma_{j=0,\ldots,k-1} B_j \cdot 256^j$
  - Alternativ kann man das Objekt auch direkt "hashen":

    Sei zum Beispiel s ein String, dann

    h(s) = Summe der Ascii-Codes der Zeichen mod m
  - Für "zufällige" Objekte klappt das wieder gut

    Wobei man da aufpassen muss, dass h(...) auch zufällig ist
  - Sonst braucht man wieder eine universelle Klasse

Histogramme 1/2

UNI FREIBURG

- Brauchen Sie für das Ü5, Aufgabe 1
  - Für jede der drei Klassen dort, berechnen Sie eine Liste von geschätzten Kollisions-Wahrscheinlichkeiten

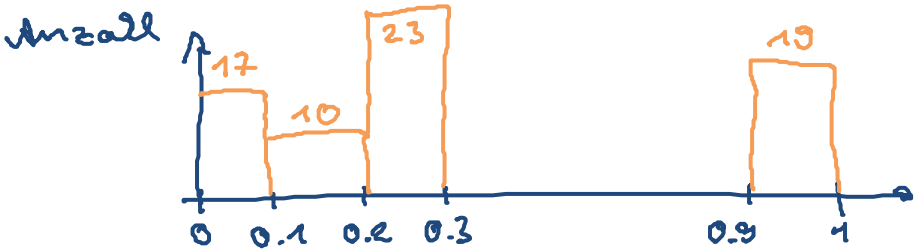
    Und zwar $u \cdot (u - 1) = 65\,280$ viele
  - Die visualisiert man am besten mit einem Histogramm:

    Werte $x_1, x_2, x_3, x_4, \ldots$ Wertebereich hier $[0,1]$

    Unterteile Wertebereich in n disjunkte Teil-Intervalle

    In unserem Fall hier kann man die gleich groß wählen

    Zähle für jedes Teil-Intervall I die Anzahl aller $x_i \in I$



Histogramme 2/2

UNI FREIBURG

- Wie malt man so ein Histogramm?
  - Anzahlen pro Bereich zeilenbasiert in eine Datei ausgeben

```
0.0     345
0.1     47
0.2     1234
...
```

  - Dann z.B. einfach mit gnuplot

```
set term png
set output "histogram.png"
plot "data.txt" using 1:2 with boxes
```

  - Geht aber auch mit R, S, Mathematica, Excel, ...

Literatur / Links

- Cuckoo Hashing
  - In Wikipedia

    http://en.wikipedia.org/wiki/Cuckoo_hashing
  - Die Originalarbeit dazu

    Rasmus Pagh und Flemming Rodler

    Cuckoo Hashing, ESA 2001 und Journal of Algorithms 2004
  - Antiprokrastinationsservice

    http://www.superkontrolle.ch (150 CHF)