Algorithmen und Datenstrukturen (für ESE) WS 2011 / 2012

Vorlesung 4, Dienstag, 22. November 2011

(Hash Map Implementierung)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Ihre Erfahrungen mit dem 3. Übungsblatt
  - Diverse Erinnerungen
  - Tipp für Windows Benutzer: cygwin
- Hash Maps / Hash Tables
  - Eine mögliche Realisierung von einem assoziativen Array
  - Und dabei lernen wir insbesondere was Hashing ist
  - **Übungsblatt:** Eine eigene HashMap Klasse schreiben, und zeigen dass die Hash Funktion dazu was taugt

Ihre Erfahrungen mit dem 3. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback Stand 22.11 16:00
  - Blatt wieder schwieriger und zeitaufwändiger als das Ü2
  - Viele haben mehr als 5 Stunden gebraucht, einige 20 Stunden
  - Manche aber auch < 1 Stunde
  - Defizite / Nachholbedarf beim Programmieren
  - Programmieren angenehmer als das Beweisen
  - Doofe Fehler kosten viel Zeit
  - Von Fehler geträumt ... am nächsten Morgen die Lösung gewusst
  - Aufgabenstellung etwas undeutlich
  - Collections.sort() in der Vorlesung erklären
  - Tipps (zum Beispiel cut Befehl) auf Folie festhalten wäre nett
  - Vim Tricks erklären, wenn ich sie benutze wäre nett
  - Einführung in das Arbeiten mit der Konsole wäre nett
  - Schöne Grüße an mich
  - Entspannter Vorlesungsstil ... kann man auch nachts um 2 hören

Diverse Erinnerungen

UNI FREIBURG

- Unit Tests
  - Für jede nicht-triviale Methode einen Unit Test
  - Mindestens einen normalen Fall testen
  - Mindestens zwei "Grenzfälle" testen, falls es solche gibt
- Abgaben
  - Bitte an die Namenskonventionen halten, z.B.
    - erfahrungen.txt und nicht Erfahrung.txt
    - non-code und nicht noncode
    - uebungsblatt_4 und nicht Übungsblatt4 ... usw.
    - Das gibt sonst, nach einmaliger freundlicher Erinnerung, exponentiell steigenden Punktabzug:

      1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, ...

Tipp für Windows-Benutzer

- Cygwin
  - Download unter www.cygwin.com
  - Dann haben Sie in Ihrer normalen DOS shell auch alle bekannten Unix/Linux Befehle
  - Insbesondere: cut, head, tail, less, more, sort, uniq, ...

Wie baut man eine Map?

UNI FREIBURG

- Zur Erinnerung
  - Ein assoziatives Array ist wie ein normales Array, nur dass die Indizes nicht 0, 1, 2, ... sind, sondern irgendwas
- Problem
  - Schnell ein Element mit einem bestimmten Schlüssel finden
  - Naive Lösung: Paare von Schlüsseln und Werten in einem normalen Feld (Java: ArrayList, C++: vector) speichern

    Array<KeyValuePair>
  - Bei n Schlüsseln kostet die Suche dann bis zu $\Theta(n)$ Zeit
  - Mit einer Hash Map geht es im günstigsten Fall in Zeit $\Theta(1)$ ... und zwar egal wieviele Elemente schon in der Map sind!

HashMap — Grundidee

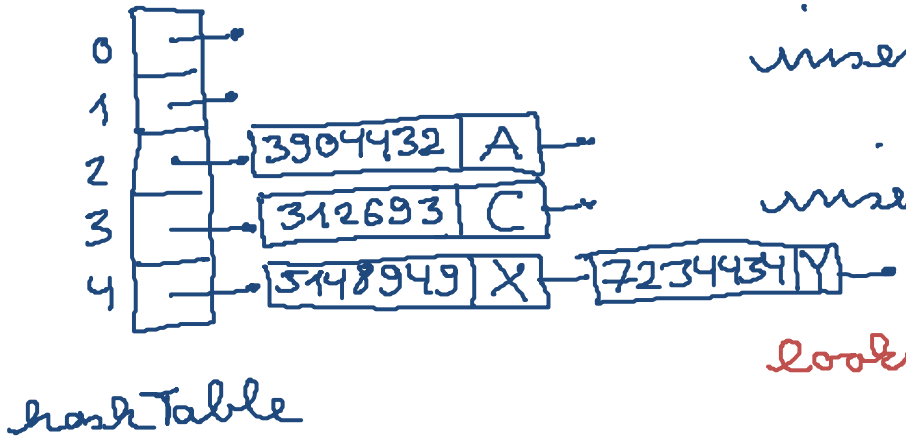
UNI FREIBURG

- Grundidee
  - Abbildung der Schlüssel auf die Indizes von einem normalen Feld, mit Hilfe einer sogenannten Hashfunktion
- Ein einfaches Beispiel
  - Schlüsselmenge { 312692, 3904433, 5148949 }
  - Hashfunktion $h(x) = x \bmod 5$, also Wertebereich [0..4]
    - $h(312692) = 2$, $h(3904433) = 3$, $h(5148949) = 4$
  - Ein gewöhnliches Feld T der Größe 5 (die Hashtabelle)
  - Wir speichern das Element mit Schlüssel x in $T[h(x)]$
  - In unseren Beispiel jetzt Zugriff in $\Theta(1)$ Zeit
  - Problem: zwei Schlüssel mit $x \neq y$ aber $h(x) = h(y)$
  - Das nennt man **Kollision**

HashMap — Kollisionen

- Einfache Lösung
  - Jeder Eintrag der Hashtabelle kann nicht nur ein key-value Paar speichern, sondern eine Menge davon

    Array<Array<KeyValuePair>> hashTable;
  - Beispiel



insert (312693, C)
h(·) = 3
insert (3904432, A)
h(·) = 2
insert (5148949, X)
h(·) = 4
lookup (3904432) ?
h(·) = 2 ⇒ A
lookup (7128163) ?
h(·) = 3 ⇒ null
insert (7234434, Y)

HashMap — Kollisionen

UNI FREIBURG

- Laufzeit für die Schlüsselsuche
  - Im besten Fall werden die Schlüssel gleichmäßig auf das Feld verteilt
    - Bei n Schlüsseln und einer Hashtabelle der Größe m sind das dann $\approx n/m$ Schlüssel pro Eintrag
  - Im schlechtesten Fall werden alle Schlüssel auf denselben Eintrag der Hashtabelle abgebildet
    - Dann sind wir wieder bei Zeit $\Theta(n)$
- Lösung
  - Wir wählen die Hashfunktion zufällig aus einer geeigneten Menge von Hashfunktionen, so dass die Schlüssel im Erwartungsfall gleichmäßig verteilt sind
  - Das nennt man **universelles Hashing**

Universelles Hashing

- Definition
  - Sei U die Menge der möglichen Schlüssel (Universum) und sei m die Größe der Hashtabelle
  - Sei H eine Menge von Hashfunktionen $U \rightarrow \{1, \ldots, m\}$
  - H ist c-universell wenn für alle $x, y \in U$ mit $x \neq y$ gilt dass $|\{h \in H : h(x) = h(y)\}| \leq c \cdot |H| / m$
- Satz
  - Sei H eine c-universelle Klasse von Hashfunktionen, $c \geq 1$
  - Sei S eine Menge von Schlüsseln und $h \in H$ zufällig gewählt
  - Sei S_i die Menge der Schlüssel x mit $h(x) = i$
  - Dann ist $E(|S_i|) \leq c \cdot |S| / m$ für alle i
  - Insbesondere: Falls $m = \Omega(|S|)$ gilt $E(|S_i|) = O(1)$

$\Rightarrow$ wenn $h \in H$ zufällig gewählt dann $Pr[h(x) = h(y)] \leq \frac{c \cdot |H|/m}{|H|} = \frac{c}{m}$

Crash-Kurs Wahrscheinlichkeitsrechnung 1/2

UNI FREIBURG

- Grundlegende Definitionen (diskreter Fall)
  - Wahrscheinlichkeitsraum Ω von sog. Elementarereignissen
  - Die haben Wahrscheinlichkeiten ... Bedingung $\Sigma_{e \in \Omega} \Pr(e) = 1$
  - Ereignis E = Teilmenge von Ω, Wahrsch. $\Pr(E) = \Sigma_{e \in E} \Pr(e)$
  - Zum Beispiel: zweimal würfeln, dann $\Omega = \{1,...,6\}^2$
    - Jedes e aus Ω hat dann Wahrscheinlichkeit $\Pr(e) = 1/36$
    - E = beide Augenzahlen sind gerade, dann $\Pr(E) =$ 1/4
- Zufallsvariable
  - ... weist einem Ausgang des Zufallsexperiments eine Zahl zu
    - zum Beispiel: X = Summe der Augenzahlen
  - Sowas wie $X = 12$ oder $X \geq 7$ sind dann einfach Ereignisse
  - Zum Beispiel $\Pr(X=2) =$ 1/36 und $\Pr(X=4) =$ 1/12

(1,3)
(2,2)
(3,1)

Crash-Kurs Wahrscheinlichkeitsrechnung 2/2

- Erwartungswert einer Zufallsvariablen

  $2 \cdot \frac{1}{36} + 3 \cdot \frac{2}{36} + 4 \cdot \frac{3}{36} + 5 \cdot \frac{4}{36} + \dots$

  - **Definition:** $\mathbf{E}(X) = \sum k \cdot \Pr(X = k)$

    (Intuitiv: gewichtetes Mittel der möglichen Werte von X, wobei die Gewichte die Wahrscheinlichkeiten der entspr. Werte sind)

  - **Satz:** Für beliebige (diskrete) Zufallsvariablen $X_1, \dots, X_n$ gilt

    $\mathbf{E}(X_1 + \dots + X_n) = \mathbf{E}(X_1) + \dots + \mathbf{E}(X_n)$

  - **Korollar:** Bei einem Zufallsexperiment tritt das Ereignis E mit Wahrscheinlichkeit p auf. Sei X die Anzahl der Auftreten von E bei n Ausführungen dieses Experimentes, dann ist $\mathbf{E}(X) = n \cdot p$

    Beweis des Korollars:

    $X_i := \begin{cases} 1, & \text{wenn } E \text{ bei } i\text{-ter Ausführung auftritt} \\ 0, & \text{sonst.} \end{cases}$

    Indikatorvariable

    dann $X = \sum_{i=1}^{n} X_i$

    $E(X_i) \overset{\text{Def.}}{=} 0 \cdot \Pr(X_i = 0) + 1 \cdot \Pr(X_i = 1) = p$

    $E(X) \overset{\text{Satz}}{=} \sum_{i=1}^{n} E(X_i) = n \cdot p$ Induktion

Universelles Hashing

S = Schlüsselmenge
m = Größe Hashtabelle
zum Verständnis: $c = 1$

- Beweis des Satzes

$$E(|S_i|) \leq c \cdot |S|/m$$

$S_i = |\{x \in S \mid h(x) = i\}|$
$h \in H$ zufällig gewählt
H war c-universell
(sonst $|S_i| = 0$ und die Behauptung stimmt auch)

Fall: $\exists x \in S : h(x) = i$

$$I_y := \begin{cases} 1, & \text{wenn } h(x) = h(y) \\ 0, & \text{sonst} \end{cases} \qquad x \neq y$$

dann $|S_i| = 1 + \sum_{\substack{y \in S \\ y \neq x}} I_y$

dann $E(|S_i|) = 1 + \sum_{\substack{y \in S \\ y \neq x}} \underbrace{E(I_y)}_{= Pr(h(x) = h(y))}$

$\leq 1 + \frac{c}{m} \cdot (|S| - 1)$ $\qquad \leq \frac{c}{m}$ wegen c-Universalität von H

$= c \cdot |S|/m + 1 - \frac{c}{m} \leq 1 + c \cdot |S|/m$

∎ (FAST)

Universelles Hashing

- Wesentlich besser als 1-universell geht nicht
  - ... zumindest nicht für |U| >> m, hier ist ein Beweis:

Universelles Hashing — Negativbeispiele

- Negativbeispiel 1
  - Die Menge aller h mit $h(x) = a \cdot x \bmod m$, für ein $a \in U$
  - Ist nicht universell, warum?

$x, y \in U \quad x \neq y \quad |\{a \in U : a \cdot x \bmod m = a \cdot y \bmod m\}|$ soll klein sein, $< c \cdot |U|/m$

kann nicht gelten

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

$\Rightarrow a \cdot x \bmod m = 0, \quad a \cdot y \bmod m = 0$

- Negativbeispiel 2
  - Die Menge aller Funktionen von $U \rightarrow \{1,\ldots,m\}$
  - Ist 1-universell, warum?
  - Aber als Hashfunktionen ungeeignet, warum?

brauchen Platz $\Theta(|U|)$ um sie abzuspeichern.

Universelles Hashing — Positivbeispiele

- Positivbeispiel 1
  - Sei p eine große Primzahl, und zwar $p > m$ und $p \geq |U|$
  - Sei H die Menge aller h mit $h(x) = (a \cdot x + b) \bmod p \bmod m$ wobei $a, b \in U$
  - Die ist $\approx \mathbf{1}$-universell, siehe Exercise 4.11 in Mehlhorn/Sanders
- 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 $\mathbf{1}$-universell, siehe Theorem 4.4 in Mehlhorn/Sanders

Universelles Hashing — Positivbeispiele

UNI FREIBURG

- 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

Literatur / Links

- Hash Maps
  - In Mehlhorn/Sanders:

    4 Hash Tables and Associative Arrays
  - In Cormen/Leiserson/Rivest

    12 Hash Tables
  - In Wikipedia

    http://de.wikipedia.org/wiki/Hashtabelle

    http://en.wikipedia.org/wiki/Hash_table
- Hash Map in Java und in C++

  http://download.oracle.com/javase/1.4.2/docs/api/java/util/HashMap.html

  http://www.sgi.com/tech/stl/hash_map.html