Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 5, Montag, 22. November 2010
(Hashtabellen, Hashfunktionen)

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 4. Übungsblatt
- 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 4. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Den allermeisten hat es gefallen
  - "Endlich wieder Programmieren"
  - Es gibt aber auch einige, denen die Theorie / Mathe lieber ist
  - Einige hat das Programmieren viel Zeit gekostet

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 Elementen in einem normalen Feld speichern

    vector<pair<KeyType, ValueType> >
  - Bei k Schlüsseln kostet die Suche dann bis zu $\Theta(k)$ Zeit
  - Mit einer Hash Map geht es im günstigsten Fall in Zeit $\Theta(1)$, egal wieviele Elemente schon in der Map sind

HashMap — Grundidee

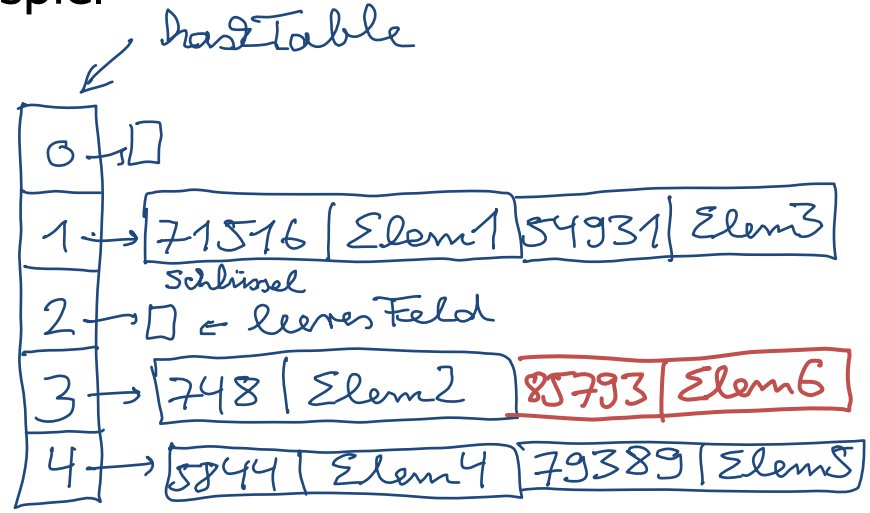
UNI FREIBURG

- 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 x und y 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 Element speichern, sondern ein Feld von Elementen

    vector<vector<pair<KeyType, ValueType> > > hashTable;
  - Beispiel



85793 2

h(...) = 3

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 Θ(Anzahl der Schlüssel)
- 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 x und y aus U mit $x \neq y$ gilt dass $|\{h \in H : h(x) = h(y)\}| \leq c \cdot |H| / m$
- Satz

Das heißt, wenn man $h \in H$ zufällig wählt $\Pr(h(x) = h(y)) \leq c/m$

  - Sei H eine c-universelle Klasse von Hashfunktionen
  - 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 $E(|S_i|) \leq 1 + c \cdot |S| / m$ für alle i
  - Insbesondere: Falls $m = \Omega(|S|)$ gilt $E(|S_i|) = O(1)$

Universelles Hashing

X = Augenzahl bei Wurf mit einem Würfel

$E(X) = \frac{1}{6}\cdot 1 + \frac{1}{6}\cdot 2 + \frac{1}{6}\cdot 3 + \frac{1}{6}\cdot 4 + \frac{1}{6}\cdot 5 + \frac{1}{6}\cdot 6$

$= \frac{1}{6}\cdot 21 = 3.5$

- Beweis Satz

$\{1,5,2\} = \{1,5\} \cup \{2,1\}$

$= \{1,5\} \dot\cup \{2\}$

S = Schlüsselmenge

$S_i = \{x \in S \mid h(x) = i\}$; $S = S_1 \dot\cup \ldots \dot\cup S_m$

$E(|S_i|) = ?$

Sei x ein Schlüssel mit $h(x) = i$ für ein festes i

(wenn es kein solches x gibt, dann ist $S_i = \emptyset$)

Indikatorvariable $I_y = \begin{cases} 1 & h(y) = i \\ 0 & h(y) \neq i \end{cases}$

Dann ist $|S_i| = 1 + \sum_{y \in S, x \neq y} I_y$

wegen Linearität des Erwartungswertes

$E(|S_i|) = E\left(1 + \sum_{y \in S, x \neq y} I_y\right) = 1 + \sum_{y \in S, x \neq y} E(I_y)$

$E(I_y) = Pr(h(y) = i) \cdot 1 + Pr(h(y) \neq i) \cdot 0$

weil $h \in H$ zufällig und H ist c-universell

$= Pr(h(y) = i) = Pr(h(y) = h(x)) \leq \frac{c}{m}$

Also $E(|S_i|) \leq 1 + |S| \cdot \frac{c}{m}$

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?

Sei die Schlüsselmenge $S = \{m, 2m, 3m, 4m, \ldots\}$
Dann ist für alle $x \in S$ $h(x) = 0$ egal für welches a.

- Negativbeispiel 2
  - Die Menge aller Funktionen von $U \to \{1,\ldots,m\}$
  - Ist 1-universell, aber nicht sehr nützlich, warum?

Es braucht sehr (zu) viel Platz, um so eine Funktion zu repräsentieren, nämlich $|U| \cdot \log m$ Bits

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 universell → Übungsaufgabe 1
- Positivbeispiel 2
  - Die Menge aller h mit $h(x) = a \bullet x \bmod m$, für ein $a \in U$
  - Dabei ist $\bullet$ ein spezielles (Skalar)produkt → nächste Folie
- Positivbeispiel 3
  - Die Menge aller h mit $h(x) = [a \cdot x]_{\text{Bits } k..\log m}$, für ein $a \in U$
  - Siehe Exercise 4.14 in Mehlhorn / Sanders

Beweis Positivbeispiel 2

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 C++ und Java

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

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