Informatik II: Algorithmen und Datenstrukturen SS 2013

Vorlesung 6a, Dienstag, 28. Mai 2013

(Dynamische Felder: 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 Ü5 (Hashing)
  - Korrekturen zu den Hashklassen aus der Vorlesung 6b
  - Anzahl noch aktiver TeilnehmerInnen der Vorlesung
- Dynamische Felder
  - Benutzung in Java und C++
  - Eigene Implementierung
  - Ein schönes Beispiel wo man Mathematik braucht, um zu verstehen, warum man es so machen muss und nicht anders
  - Übungsblatt 6: Erweiterung des Codes aus der Vorlesung und Laufzeitanalyse dazu (nach dem Vorbild aus der Vorlesung)

Erfahrungen mit dem Ü6 (Hashing)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 28. Mai 15:30
  - Aufgabe 1 für einige schwer zu verstehen

    Das liegt in der Natur von universellem Hashing !
  - Lieber wieder reine Mathe-Übungsblätter

    Sie sollen aber in der Informatik beides lernen :-)
  - Wie testet man Zufall … gar nicht, ist aber auch nicht nötig

    Z.B. Hashfunktionen einfach für feste Werte testen
  - Das Wetter am Dienstag wird grausam WEIL
  - Natürlich wird das Wetter am Dienstag perfekt

Korrektur Hashklassen Vorlesung 6b

UNI FREIBURG

- Es gab da einige kleine Fehler

  Positivbeispiel

  – **PB 1:** $a \cdot x + b \bmod p \bmod m$ p Primzahl $\geq u$

    a und b müssen zufällig aus $\{0, \ldots, p-1\}$ gewählt werden

    nicht nur aus $\{0, \ldots, u-1\}$, sonst nicht universell !

  – **PB 2:** $a \bullet x \bmod m$ dabei war $\bullet$ eine Art "Skalarprodukt"

    m muss eine Primzahl sein, sonst nicht universell !

  – **PB 3:** $a \cdot x \bmod 2^k \operatorname{div} 2^{k-l}$ $u = 2^k$, $m = 2^l$

    a muss eine **ungerade** Zahl aus $\{0, \ldots, u-1\}$ sein !

Anzahl TeilnehmerInnen an der VL

UNI FREIBURG

- Hier ein paar Zahlen
  - Im WS 12/13 angefangen mit B.Sc. Informatik: 148
  - Zu Beginn für die Vorlesung angemeldet: 116
  - Es geben jetzt noch Übungsblätter ab: 47
  - Mehr werden die Klausur dann wohl auch nicht mitschreiben

    Man braucht ja (aus gutem Grund) 50% der Punkte
    aus den Übungsblättern für die Zulassung zur Klausur

Was ist los?

Statische Felder

UNI FREIBURG

- ... gibt es sowohl in Java:

```
int[] numbers = new int[100];  // Array of 100 ints, initializ. to 0.
System.out.println(numbers[12]);  // Prints 0.
String[] strings = new String[10];  // Array of 10 strings.
System.out.println(strings[7]);  // Prints empty string.
strings[8] = "doof";
```

- ... als auch in C++

```
int[] numbers = new int[100];  // Pointer to 100 ints, no initializ.
printf("%d\n", numbers[12]);  //  Prints random number.
string[] strings = new string[10];  // Pointer to 10 strings.
printf("%s\n", strings[7].c_str());  // Prints empty string.
strings[8] = "doof";
```

- Größe muss bei der Erzeugung festgelegt werden!
  - Die benötigte Größe ergibt sich aber oft erst im Laufe des Progrs

Statische Felder

UNI FREIBURG

- Der Name "statisch" ist etwas irreführend
  - Es hat nichts mit dem keyword static in Java oder in C++ zu tun
  - Die Felder sind auch nicht statisch in dem Sinne, dass der Speicherplatz schon vor der Ausführung des Programmes alloziert wird, im Gegenteil
  - Was statisch ist, ist die **Größe** des Feldes
    - die muss bei der Erzeugung des Feldes explizit angegeben werden
    - und kann danach nicht mehr geändert werden
  - Von daher wäre Feld fester Größe bzw. fixed-size array ein besserer Name

Dynamische Felder

UNI FREIBURG

- ... können beliebig vergrößert / verkleinert werden
  - In Java haben wir dafür bisher immer ArrayList benutzt

```
ArrayList<String> strings = new ArrayList<String>();
strings.add("doof");
strings.add("doofer");
strings.add("am doofsten");
System.out.println(strings.get(0));  // Will print doof.
strings.clear();  // Remove all elements.
```

  - In C++ nimmt man dafür std::vector

```
vector<string> strings;
strings.push_back("doof");
...
strings.resize(2);  // Keep only first 2 elements.
strings.clear();  // Remove all elements.
```

Dynamische Felder 1/5

UNI FREIBURG

- Grundprinzip
  - Man hat intern ein fixed-size array (FSA)

    … von einer ausreichenden Größe
  - Wenn Elemente dazu kommen oder entfernt werden

    … und die Größe des internen FSA nicht mehr passt

    … erzeugt man ein neues FSA passender Größe

    … und kopiert die Elemente vom alten in das neue FSA

    … diesen Vorgang nennt man **Reallokation**
  - Das implementieren wir jetzt zusammen
    - Erst mal append (= neues Element anhängen)
    - Dann auch removeLast (= letztes Element entfernen)

Dynamische Felder 2/5

UNI FREIBURG

- Vergrößerungsstrategie 1 ... die einfachste Variante
  - Wir vergrößern das Feld nach jedem append
  - Und machen es immer genauso groß, wie wir es brauchen
  - Beobachtung: akkumulierte Laufzeit sieht quadratisch aus
- Analyse
  - Sei $T(n)$ die Laufzeit für eine Folge von n mal append
  - Sei T_i die Laufzeit für das i-te append
  - Dann ist $T_i \geq A \cdot i$ für irgendeine Konstante A

    weil wir bei der Reallokation i Elemente umkopieren müssen
  - Das macht zusammen $\sum_{i=1}^{n} T_i \geq A \cdot \underbrace{\sum_{i=1}^{n} i}_{=\frac{1}{2}n(n+1)} \geq \frac{A}{2} \cdot n^2 = \Omega(n^2)$

Dynamische Felder 3/5

- Vergrößerungsstrategie 2 ... etwas vorrausschauender
  - Idee: beim Vergrößern zusätzlichen Platz lassen ... wie viel?
  - Lassen wir erst mal Platz für C zusätzliche Elemente, für ein beliebiges festes C, zum Beispiel C = 100 oder C = 1000
  - Beobachtung: akkumulierte Laufzeit immer noch quadratisch
- Analyse
  - Die meisten append Operationen kosten jetzt nur O(1)
  - Aber für i = C, 2C, 3C, ... ist nach wie vor $T_i \geq A \cdot i$
  - Das macht zusammen:

$$\sum_{i=1}^{n} T_i \geq \sum_{j=1}^{\lfloor n/C \rfloor} T_{j \cdot C} \geq \sum_{j=1}^{\lfloor n/C \rfloor} A \cdot j \cdot C = A \cdot C \cdot \sum_{j=1}^{\lfloor n/C \rfloor} j \geq A \cdot C \cdot \frac{1}{2} \cdot n^2/C^2$$

T_{100}, T_{200}

$$\frac{1}{2}\lfloor \tfrac{n}{C} \rfloor \cdot (\lfloor \tfrac{n}{C} \rfloor + 1) \geq \frac{1}{2} \cdot \left(\frac{n}{C}\right)^2 = \frac{A}{2C} \cdot n^2 = \Omega(n^2)$$

Dynamische Felder 4/5

UNI FREIBURG

- Vergrößerungsstrategie 3
  - Idee: immer doppelt so viel vergrößern wie nötig
  - Jetzt sieht die Laufzeitkurze linear aus (mit Sprüngen)
- Analyse
  - Jetzt Reallokationen nur noch bei $i = 1, 2, 4, 8, 16, \ldots$

    An den Stellen ist dann nach wie vor nur $T_i \leq A \cdot i$

    Für alle anderen i's ist $T_i \leq A$
  - Das macht zusammen:

$$\sum_{i=1}^{n} T_i \leq A \cdot n + \sum_{j=0}^{\lfloor \log_2 n \rfloor} 2^j \cdot A \leq 3A \cdot n = O(n)$$

$$1 + 2 + 4 + \cdots + 2^k = 2^{k+1} - 1 \leq 2 \cdot n - 1 \leq 2 \cdot n$$

$$k = \lfloor \log_2 n \rfloor \leq \log_2 n$$

$$1 + 2 + 4 + 8 + 16 = 32 - 1$$
$$2^4 \quad 2^5$$

Dynamische Felder 5/5

UNI FREIBURG

- Was machen wir wenn Element entfernt werden
  - Analog zum Vergrößern, könnten wir das Feld auf die Hälfte verkleinern wenn es nur noch halbvoll ist

    Aber Achtung: wenn man danach ein append macht muss man es gleich wieder vergrößern

    Außerdem: wenn wir nach einer Vergrößerung ein removeLast machen, muss man gleich wieder verkleinern
  - Deswegen machen wir es (erst mal) so:

    wenn ganz voll, Vergrößerung auf doppelte Größe

    wenn ein Viertel voll, Verkleinerung auf halbe Größe

    Für das Ü6 sollen Sie das verallgemeinern !

Literatur / Links

UNI FREIBURG

- Dynamische Felder: Implementierung
  - In C++ und in Java

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

    http://docs.oracle.com/javase/1.4.2/docs/api/java/util/ArrayList.html
- Doof

  http://de.wiktionary.org/wiki/doof