Algorithmen und Datenstrukturen (für ESE) WS 2011 / 2012

Vorlesung 7, Dienstag, 13. Dezember 2011

(Dynamische Felder, amortisierte Analyse)

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 5. Übungsblatt
  - Übungsgruppen / Treffen mit Ihrem Tutor bzw. Ihrer Tutorin
- Dynamische Felder
  - Was ist das (im Gegensatz zu statischen Feldern)
  - Standardbibliotheken dazu in Java und in C++
  - Ein paar eigene Implementierungsvarianten
  - Laufzeitanalyse dieser Varianten
  - Ein schönes Beispiel wo man Mathematik braucht, um zu verstehen, warum man es so machen muss und nicht anders
  - Übungsblatt 7: Erweiterung des Codes aus der Vorlesung und Laufzeitanalyse dazu (nach dem Vorbild aus der Vorlesung)

Ihre Erfahrungen mit dem Ü5 (PWs)

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback Stand 6.12 16:00
  - Übungsblatt war nicht einfach aber gut machbar
  - deleteMin war am schwierigsten (Pfad Wurzel → Blatt)
  - Vorlesung dazu sehr verständlich und hilfreich
  - Die meisten haben 4-6 Stunden gebraucht
  - Einige auch mehr, meistens wegen Bugs oder Irrwegen
  - Einige aber auch nur 1 Stunde
  - Feedback wird nicht nur eingefordert, sondern auch umgesetzt
  - Muss man wirklich auch die Hilfsmethoden testen?
  - Komisch wenn man keine Main schreiben muss
  - Jenkins hat diesmal was gebracht
  - Ordnerstruktur C++?
  - Das System zum Hochladen ist immer noch gewöhn.bedürftig

Übungsgruppen / Treffen mit TutorIn

UNI FREIBURG

- Übungsgruppen
  - ... gibt es ab jetzt keine mehr
  - Grund: die letzten Male ist kaum noch einer gekommen
- Jeder Teilnehmer / jede Teilnehmerin
  - ... soll sich einmal im Semester mit seinem Tutor / seiner Tutorin / ihrem Tutor / ihrer Tutorin treffen
  - Sie bekommen dazu eine Mail von Ihrem/r Tutor/in
  - Treffen dauert 15 – 30 Minuten
  - Grund 1: wir wollen auch nochmal persönlich schauen, wie es Ihnen geht und ob es Probleme gibt
  - Grund 2: wir wollen nicht, dass jemand schummelt

Statische Felder

- ... 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

- ... 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 — Implementierung

UNI FREIBURG

- Das Prinzip ist ganz einfach
  - Man hat intern ein fixed-size array
    - von der Größe, die man gerade braucht
  - Wenn Elemente dazu kommen:
    - erzeugt man ein neues fixed-size array der benötigten Größe
    - und kopiert die Elemente vom alten in das neue Feld
  - Wenn Elemente entfernt werden
    - erzeugt man ein neues fixed-size array der benötigten Größe
    - und kopiert die Elemente von alten in das neue Feld
  - Das implementieren wir jetzt erstmal zusammen
    - Vorlesung: nur append (= neues Element anhängen)
    - Übungsblatt: append **und** remove (= letztes El. entfernen)

Version 1

- Einfachste Vergrößerungsstrategie
  - Wir vergrößern das Feld nach jedem append
  - Und machen es immer genauso groß, wie wir es brauchen
  - Die Laufzeitkurze zeigt quadratisches Verhalten, warum?
- Analyse
  - Sei $T(n)$ die Laufzeit für eine Folge von n append Operationen
  - Sei T_i die Laufzeit für die i-te append Operation
  - Dann is $T_i \geq A \cdot i$ für irgendeine Konstante A
    - weil wir i Elemente reallozieren (umkopieren) müssen
  - Das macht zusammen:

$$T(n) = \sum_{i=1}^{n} T_i \geq \sum_{i=1}^{n} A \cdot i = A \cdot \underbrace{\sum_{i=1}^{n} i}_{= \frac{1}{2} \cdot n \cdot (n+1) \geq n} \geq A/2 \cdot n^2 = \Omega(n^2)$$

Version 2

- Eine etwas vorausschauendere Vergrößer.strategie
  - Idee: beim Vergrößern zusätzlichen Platz lassen
  - Aber wieviel?
  - Lassen wir erstmal Platz für C zusätzliche Elemente, für ein beliebiges festes C, zum Beispiel C = 100 oder C = 1000
  - Die Laufzeitkurve ist immer noch quadratisch, warum?
- 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:

$$T(m) = \sum_{i=1}^{m} T_i \geq \sum_{j=1}^{m/C} T_{j \cdot C} \geq A \cdot C \cdot \sum_{j=1}^{m/C} j \geq A \cdot C/2 \cdot (m/C)^2 = \frac{A}{2C} \cdot m^2 = \Omega(m^2)$$

$T_{j \cdot C} = T_{1000} + T_{2000} + T_{3000} + \ldots$ für $C = 1000$

Version 3

- Die "richtige" Vergrößerungsstrategie
  - Idee: immer doppelt so viel vergrößern wie nötig
  - Jetzt sieht die Laufzeitkurze linear aus (mit Sprüngen)
- Analyse
  - Jetzt kosten noch mehr append Operationen nur O(1) $\leq A'$
  - Für i = 1, 2, 4, 8, 16, ... ist $T_i \leq A'' \cdot i$ für irgendein A''
  - Das macht zusammen:

$$T(n) = \sum_{i=1}^{n} T_i \leq A' \cdot n + \underbrace{\sum_{j=0}^{k} T_{2^j}}_{T_1 + T_2 + T_4 + T_8 + T_{16} + \cdots}$$

$k = \lfloor \log_2 n \rfloor$ weil $2^{\log_2 n} = n$

$$\leq A' \cdot n + A'' \cdot \underbrace{\sum_{j=0}^{k} 2^j}_{1+2+4+8+16+\cdots}$$

$$= 2^{k+1} - 1$$

$$\leq 2 \cdot n$$

$$= O(n)$$

Dynamische Felder — Verkleinern

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
  - Deswegen lassen wir etwas Luft beim Verkleinern
    - z.B. auf 75% verkleinern wenn nur noch halbvoll
- Analyse
  - Jetzt wird's schwierig, warum?
  - Wenn wir eine beliebige Folge von append und remove Operationen haben, können wir nicht mehr so leicht vorhersagen, wann realloziert werden muss

Amortisierte Analyse 1/4

- Notation
  - Gegeben n Operationen $O_1, \ldots, O_n$
  - Sei s_i die Größe des Feldes nach Operation O_i $(s_0 := 0)$
  - Sei c_i die Kapazität des Feldes nach Operation O_i $(c_0 := 0)$
  - Sei $cost(O_i)$ die Zeit für Operation O_i (unser T_i von vorher)
    - wir nehmen an die ist $\leq A \cdot s_i$ falls Reallokation nötig
    - und ansonsten $\leq B$
    - für irgendwelche Zahlen A und B unabhängig von n
- Wir analysieren folgende Implementierungsversion
  - Falls O_i append: vergrößern genau dann wenn $s_{i-1} = c_{i-1}$
  - Falls O_i remove: verkleinern genau dann wenn $3 \cdot s_{i-1} \leq c_{i-1}$
  - In beiden Fällen sorgen wir dafür, dass danach $c_i = 3/2 \cdot s_i$

Amortisierte Analyse 2/4

- Beweisidee
  - Teuer sind nur die Operationen, wo realloziert werden muss
  - Wenn gerade realloziert wurde, dauert es eine Weile, bis wieder realloziert werden muss
  - Anders gesagt: nach einer teuren Operation kommt eine ganze Reihe billiger Operationen
  - **Idee für den Beweis:** wenn nach einer Operation die X gekostet hat X Operationen kommen die alle nur 1 kosten, sind die Gesamtkosten bei n Operationen höchstens $2 \cdot n$
- Formal beweisen wir Folgendes
  - Lemma: Wenn bei O_i eine Reallokation stattfindet und dann erst wieder bei O_j, dann ist $j - i > s_i / 2$
  - Korollar: $\text{cost}(O_1) + \ldots + \text{cost}(O_n) \leq (3A + B) \cdot n$

Amortisierte Analyse 3/4

- Beweis des Lemmas

  [Wenn bei O_i eine Reallokation stattfindet und dann erst wieder bei O_j, dann ist $j - i > s_i/2$]

  – Nach O_i ist die Kapazität genau $\text{floor}(3/2 \cdot s_i) = c_i$

  – Betrachte eine Operation O_j nach O_i mit $j - i \leq s_i/2$

1. Bis dahin können höchstens $j - i \leq s_i/2$ Elemente dazu gekommen sein
   $\Rightarrow s_j \leq s_i + s_i/2 = \frac{3}{2} \cdot s_i \Rightarrow s_j \leq \lfloor \frac{3}{2} \cdot s_i \rfloor = c_i$
   $\Rightarrow$ keine neue Vergrößerung nötig.
2. Bis dahin können höchstens $j - i \leq s_i/2$ Elemente weggenommen worden sein
   $\Rightarrow s_j \geq s_i - s_i/2 = s_i/2 \Rightarrow 3 \cdot s_j \geq \lfloor \frac{3}{2} \cdot s_i \rfloor = c_i$
   $\Rightarrow$ keine neue Verkleinerung nötig ∎

Amortisierte Analyse 4/4

- Beweis des Korollars

  [$cost(O_1) + \ldots + cost(O_n) \leq (3A + B) \cdot n$]

  – Seien die Reallokationen bei $O_{i1}, \ldots, O_{il}$

  – Die Kosten dafür sind A mal $s_{i1} + \ldots + s_{il}$

  – Nach Lemma ist $i2 > i1 + s_{i1}/2$, $i3 > i2 + s_{i2}/2$ usw.

$s_{i_1} < 2(i_2 - i_1)$; $s_{i_2} < 2(i_3 - i_2)$; ...

Also $s_{i_1} + s_{i_2} + \cdots < \underbrace{2(i_2 - i_1) + 2(i_3 - i_2) + \cdots}_{\text{sog. Teleskopsumme}}$

$= 2(i_e - i_1)$

$\leq 2 \cdot i_e \leq 2 \cdot n$ ∎

Literatur / Links

- Dynamische Felder
  - In Mehlhorn/Sanders:

    3.2 Unbounded Arrays
  - In Cormen/Leiserson/Rivest

    18.4 Dynamic Tables
  - In Wikipedia

    http://en.wikipedia.org/wiki/Dynamic_array
  - 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