Informatik II: Algorithmen und Datenstrukturen SS 2015

Vorlesung 6a, Dienstag, 2. Juni 2015

(Dynamische Felder: Implementierung)

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

UNI FREIBURG

Blick über die Vorlesung heute

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü5 (Universelles Hashing)
  - Treffen mit Ihrem Tutor
- Inhalt
  - Felder fester Größe: in Python, Java, C++
  - Dynamische Felder: Implementierung + Laufzeitanalyse

  Wir werden sehen: mit Rumprobieren kommt man irgendwann nicht mehr weiter

  Man braucht mal wieder Mathematik, um zu verstehen, was wirklich passiert (das machen wir dann morgen)

Erfahrungen mit dem Ü5 1/2

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 2. Juni 12:00
  - Für viele das aufwändigste Übungsblatt bisher

    Aber nicht viel Code, siehe Lösungsbeispiel gleich
  - Viele Verständnisprobleme zu universellem Hashing

    Das ist normal, aber ich denke/hoffe, dass das Übungsblatt geholfen hat, es besser zu verstehen

    Nur vom Zuhören versteht man das einfach nicht
  - "Bösen Pfingstferien haben sich in den Weg gestellt"
  - Punkte nicht proportional zum Arbeitsaufwand
  - Fragen zum Universum werden vermisst

Erfahrungen mit dem Ü5 2/2

UNI FREIBURG

- Auszüge Musterlösung
  - Aufgabe 1 war für jede Klasse von Hashfunktionen nur sehr wenig (und teilweise sehr einfacher) Code
  - Dazu eine(!) weitere Funktion für die Berechnung des Histogramms für eine gegeben Klasse

    Ich mache das kurz für zwei der Klassen vor …

Treffen mit Ihrem Tutor

- Grund
  - Wir wollen Sie auch alle mal persönlich kennen lernen
  - Weitere Gelegenheit für Fragen und Feedback aller Art
  - Wir wollen auch schauen, ob es Sie wirklich gibt und ob es wirklich Sie sind, die die Übungsblätter machen

    Es ist aber keine Prüfung … außer Sie haben abgeschrieben
- Vorgehen
  - Sie werden von Ihrem Tutor angeschrieben
  - So ein Treffen dauert ca. 30 Minuten
  - Das Treffen ist Zulassungsvoraussetzung

Felder fester Größe 1/4

- Eigenschaften
  - Die Größe des Feldes wird bei der Deklaration festgelegt

    Vorteil: effizient umzusetzen, weil nur einmal eine feste Menge Speicher alloziert werden muss.

    Nachteil: bei manchen Anwendungen weiß man vorher nicht, ein wie großes Feld man benötigt

  - In **Python** gibt es deswegen gar keine solchen Felder

    Der Schwerpunkt bei Python liegt auf möglichst einfacher und schneller Entwicklung, nicht auf effizientem Code

    Da das Ü6 mit Feldern fester Größe realisiert werden muss, deswegen diesmal nur Java oder C++ zur Auswahl

Felder fester Größe 2/4

- Felder fester Größe in Java

```
int[] numbers = new int[100];          Feld von 100 ints
System.out.println(numbers[12]);       Druckt "0"

String[] strings = new String[10];     Feld von 10 strings
System.out.println(strings[7]);        Druckt "null"
strings[8] = "doof";                   Setzt das 9-te Element
```

In Java werden die Elemente des Feldes bei der Deklaration grundsätzlich **immer** initialisiert

Bei nativen Typen wie int mit 0, bei Objekten mit null

Felder fester Größe 3/4

UNI FREIBURG

- Felder fester Größe in C++

```
int[] numbers = new int[100];          Feld von 100 ints
cout << numbers[12] << endl;           Druckt irgendwas

string[] strings = new string[10];     Feld von 10 strings
cout << strings[7] << endl;            Druckt leeren string
strings[8] = "doof";                   Setzt das 9-te Element
```

In C++ werden die Elemente bei nativen Datentypen wie int grundsätzlich **nicht** initialisiert (aus Effizienzgründen)

Bei Objekten wird dagegen grundsätzlich der Default-Konstruktor der Klasse aufgerufen

Felder fester Größe 4/4

- Begriffsverwirrung
  - Felder fester Größe werden auch manchmal **statische Felder** genannt, weil sich ihre Größe nicht ändert
  - Das hat aber nichts mit statischer (vs. dynamischer) Allokation bzw. dem keyword **static** zu tun

    Dabei geht es darum, ob Speicherplatz schon vom Compiler (statisch) oder erst zur Laufzeit (dynamisch) zugewiesen wird

    Statisch zugewiesener Platz hat aber immer auch eine feste Größe

    Aber ein Feld fester Größe kann auch dynamisch zugewiesen werden

Dynamische Felder 1/9

- Eigenschaften
  - Können im Laufe ihres Lebens beliebig vergrößert und verkleinert werden
  - Das (nicht-triviale) Speichermanagement ist dabei vor der Benutzerin versteckt

    Für das Ü6 sollen Sie es gerade implementieren
  - Zu Beginn kann das Feld entweder leer sein oder schon eine gewisse Größe haben

    Wenn man weiß, dass man eine gewisse Größe sowieso braucht, spart man sich dann am Anfang das Management

Dynamische Felder 2/9

- Dynamische Felder in Java
  - In Java nimmt man dafür ArrayList

```
ArrayList<String> strings = new ArrayList<String>();
strings.add("doof");
strings.add("doofer");
strings.add("am doofsten");
System.out.println(strings.get(0));       Druckt "doof"
strings.remove(string.length() - 1);      Lösche letztes Element
```

ArrayList funktioniert **nicht** mit nativen Typen, z.B. int, man muss dann Integer nehmen

Das ist für große Datenmengen sehr ineffizient … wenn die Laufzeit zählt, sollte man dann lieber ein eigenes ArrayInt implementieren, das intern mit nativen Feldern realisiert ist

Dynamische Felder 3/9

- Dynamische Felder in C++
  - In C++ nimmt man dafür vector aus der STL

```
vector<string> strings;
strings.push_back("doof");
strings.push_back("doofer");
strings.push_back("am doofsten");
cout << strings[0] << endl;          Druckt "doof"
strings.pop_back();                  Lösche letztes Element
```

vector funktioniert mit **allen** Typen (nativ und Klassen)

vector ist außerdem genauso effizient wie ein Feld fester Größe, das wird durch "template" Programmierung erreicht ... siehe Programmieren in C++, Vorlesung 8

Dynamische Felder 4/9

- Dynamische Felder in Python
  - In Python hat man "Listen" und die sind immer dynamisch

```
strings = [];
strings.append("doof");
strings.append("doofer");
strings.append("am doofsten");
print(strings[0]);                    Druckt "doof"
strings.pop();                        Lösche letztes Element
```

Für native Typen gibt es in Python auch die Klasse **array**

Die ist effizienter als eine Liste, aber auch dynamisch in dem Sinne, dass sie Ihre Größe beliebig ändern kann ... wie gesagt, ein Feld fester Größe gibt es in Python nicht

Dynamische Felder 5/9

UNI FREIBURG

- Realisierung (intuitiv)
  - Intern ein fixed-size array (FSA) von ausreichender Größe
  - Wenn Elemente dazu kommen oder entfernt werden

    Schauen: passt die Größe des internen FSA noch?

    Falls nein: erzeuge ein neues FSA passender Größe und kopiere die Elemente vom alten in das neue FSA

    Diesen Vorgang nennt man **Reallokation**
  - Das implementieren wir jetzt zusammen

    Erst mal push_back neues Element anhängen

    Dann auch pop_back letztes Element entfernen

Dynamische Felder 6/9

- Vergrößerungsstrategie 1
  - Wir vergrößern das Feld nach jedem push_back … und machen es dabei aber immer nur genau **um eins** größer

    Beobachtung: akkumulierte Laufzeit sieht quadratisch aus
- Analyse
  - Sei T_i die Laufzeit für das i-te push_back
  - Dann ist $T_i \geq A \cdot i$ für irgendeine Konstante A

    Bei einer Reallokation müssen alle Elemente kopiert werden

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

Dynamische Felder 7/9

- Vergrößerungsstrategie 2
  - Wie vorher, aber jetzt bei jedem Vergrößern **um C größer**, für ein festes C, zum Beispiel C = 100 oder C = 1000

    Beobachtung: akkumulierte Laufzeit immer noch quadratisch
- Analyse
  - Sei wieder T_i die Laufzeit für das i-te push_back
  - Dann sind die meisten T_i jetzt O(1)
  - Aber für i = C, 2C, 3C, ... ist nach wie vor $T_i \geq A \cdot i$

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

$C = 100: \quad T_{100} + T_{200} + T_{300} + \ldots$

$$\frac{1}{2}\lfloor n/C \rfloor \left(\lfloor \tfrac{n}{C} \rfloor + 1\right)$$

$$\approx \frac{1}{2} \underbrace{A \cdot C / C^2}_{A/C} \cdot n^2 = \Omega(n^2)$$

Literatur / Links

- Mehlhorn / Sanders
  - Kapitel 3: Representing Sequences by Arrays …
- Plotly
  - https://plot.ly
- Doof
  - http://de.wiktionary.org/wiki/doof