ETH zürich

Informatik II - Übung 02

Katja Wolff

katja.wolff@inf.ethz.ch

Themen

1. Ein Algorithmus und seine Implementierung in Java
2. Java: Elementare Aspekte
3. Klassen und Referenzen
4. Syntaxanalyse und Compiler
5. Pakete in Java
6. Objektorientierung
7. Exceptions
8. Binärbäume als Zeigergeflechte
9. Binärsuche
10. Backtracking
11. Spielbäume
12. Rekursives Problemlösen
13. Komplexität von Algorithmen
14. Simulation
15. Heaps
16. Parallele Prozesse und Threads

Übungsblatt 1

U1.A1

$$f(a,b) = a \times b = \begin{cases} a & , b = 1 \\ f(2a, b/2) & , b \text{ gerade} \\ a + f(2a, (b-1)/2) & , \text{sonst} \end{cases}$$

a) Induktionsbeweis über **a** möglich?

NEIN

Der Induktionsanfang schlägt bereits für **b** > 1 fehl!
a ist eine stetig wachsende Grösse
→ kein Rückschluss auf bereits bewiesene Fälle möglich und keine Induktionsannahme formulierbar

b) Terminiert der Algorithmus?

JA

b lässt sich immer auf **1** zurückführen.
Weil **b** immer halbiert wird gilt:
→ Nach $\lfloor \log_2(\mathbf{b}) \rfloor$ Schritten wird **b**=1 sein!

U2.A2

c) Wie ändert sich der **Korrektheits-** und **Terminierungsbeweis**, wenn:

$$f(a,b) = a \times b = \begin{cases} 0 & , b = 0 \\ f(2a, b/2) & , b \text{ gerade} \\ a + f(2a, (b-1)/2) & , \text{sonst} \end{cases}$$

Terminierungsbeweis:
Die Ganzzahldivision von 1 durch 2 ergibt 0. In 1b) haben wir gezeigt, dass es immer zu **b**=1 kommt, also es kommt auch immer zu **b**=0.

Korrektheitsbeweis:
Die Induktionsannahme lautet dann:

$$\forall a \in \mathbb{N}, \forall b \in \{0, \ldots, n\} : f(a,b) = a \cdot b$$

Der Induktionsschritt ist ähnlich wie im Original, da

$$(b-1)/2 \in \{0, \ldots, n\} \qquad b/2 \in \{0, \ldots, n\}$$

U1.A2a: Selbstaufrufe der Methoden

gerade(int x)

```
public static boolean gerade( int x ){
        if( x == 0 ) return true;
        return !gerade( x-1 );
}
```

→ x

verdopple(int x)

```
public static int verdopple( int x ){
        if( x == 0 ) return 0;
        return 2 + verdopple( x-1 );
}
```

→ x

halbiere(int x)

```
public static int halbiere( int x ){
        if( x == 0 ) return 0;
        if( x == 1 ) return 0;
        return halbiere( x-2 ) + 1;
}
```

→ $\lfloor x/2 \rfloor$

U1.A2b: Methodenaufrufe in einem Aufruf von f

Aufrufe der drei Methoden insgesamt in Abhängigkeit von **a** und **b** bei **einem** Aufruf von f ?

```
private static int f(int a, int b)
{
   if (b == 0) return 0;
   if (gerade(b)) return f(verdopple(a), halbiere(b));
   else return a + f(verdopple(a), halbiere(b));
}
```

In jedem Fall wird `gerade(b)`, `verdopple(a)` und `halbiere(b)` gerufen.
Der Anzahl der Aufrufe (mit Ergebnissen aus Teil A2a) ist also höchstens

$$b+1 \;+\; a+1 \;+\; \lfloor b/2 \rfloor +1 \;\approx\; a + 3b/2 + 3$$

U1.A2c: Gesamtzahl der Methodenaufrufe

Gesamtanzahl der Methodenaufrufe:

(Beachte: Abhängigkeit der Aufrufe von den Zahlen a und b!)

Die Rekursion endet, wenn ***b****=0* ist. Das ist der Fall nach $\mathbf{k = \lfloor \log_2 b \rfloor + 1}$ Aufrufen, da ***b*** in jedem Schritt halbiert wird.

Mit dem Ergebnis aus 2b) ergibt sich:

$$N(a,b) = \left(a + \frac{3b}{2} + 3\right) + N\left(2a, \frac{b}{2}\right) = \ldots = \sum_{i=0}^{k-1} 2^i a + \sum_{i=1}^{k} \frac{3b}{2^i} + k \cdot 3$$

Am Ende erhält man **≈ 2ab - a + 3b**

U1.A3

```
/**
 * This function implements the ancient Egyptian multiplication.
 *
 * @param a must be a positive integer
 * @param b must be a positive integer
 * @return the product of a and b
 * @throws IllegalArgumentException
 */
public static int mult(int a, int b) throws IllegalArgumentException
{
        if (a < 1) throw new IllegalArgumentException("Parameter a
                must be a positive integer but is " + a);
        if (b < 1) throw new IllegalArgumentException("Parameter b
                must be a positive integer but is " + b);
        return f(a, b);
}
```

U1.A3

```
public static int mult(int a, int b) throws IllegalArgumentException
{
  if (a <= 0)
        throw new IllegalArgumentException("Parameter a must be positive");
}
```

Ausblick: Übungsblatt 2

Generelle Hinweise

- „Helper Functions“ können benutzt werden
- Javadoc enthält manchmal nützliche Informationen (zusätzlich zum Aufgabenblatt)
- Bäume...
- Rekursion...

- Beispiel zur Rekursion: Suche in einem Binärbaum Baum

Blatt 2

- 1. Wurzelbäume
  - Trennung von Struktur und Darstellung
  - Klammerdarstellung
  - Darstellung in eingerückter Form

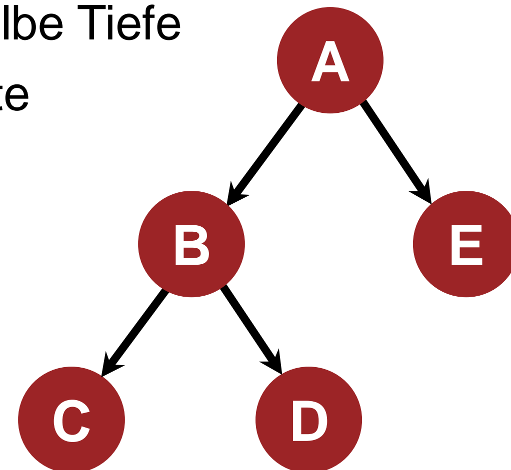
- 2. Rekursives Sortieren
  - Ausgabe von Objekten mittels toString()
  - Rekursiver Sortieralgorithmus

- 3. Binärbäume als Arrays
  - Wichtigste Sache: checkTree()

U2.A1: Darstellung eines Baumes

- **(Wurzel)baum:** zusammenhängender, gerichteter Graph aus Knoten und Kanten, ohne Zyklen
- **Binärbaum**: Wurzelbaum - jeder Knoten besitzt höchstens zwei Nachfolger
- **Geordneter Binärbaum**
- **Voller Baum**: kein Halbblatt existiert
- **Vollständiger Baum**: alle Blätter haben die selbe Tiefe
- **Entarteter Baum**: jeweils 0 oder 1 Kind → Liste
- **Höhe**

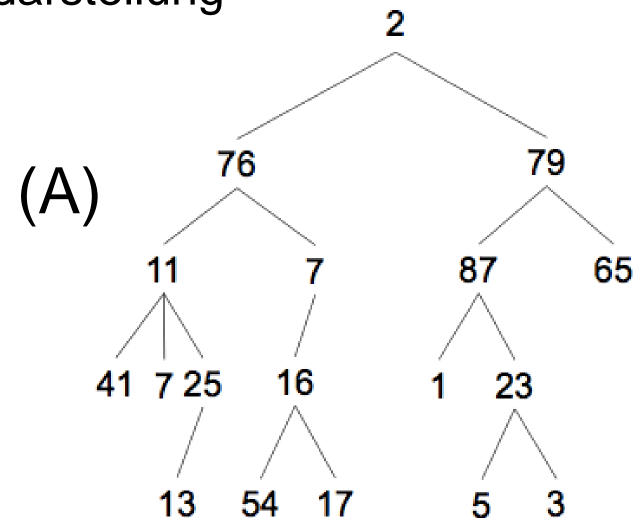
- **Baum geht von der Wurzel zu den Blättern (nicht zurück) → gerichteter Graph**



U2.A1: Darstellung eines Baumes

a) Graph gegeben: Klammerdarstellung und eingerückte Form von (A) gesucht
b) Klammerdarstellung gegeben: Graph und eingerückte Form von (B) gesucht
c) Sind Bäume (A) und (B) aus der Klammerdarstellung eindeutig rekonstruierbar?
d) Höhe, Längste Pfade, Blätter von (A) und (B) gesucht

(A)



(B)

```
S(R(H(K)),P(A(N,O),Q,T),V(J,F(G)))
```

U2.A2: Sortieren

- Gerüst auf der Webseite/Codeboard: u2a2.RandomArray
- 2a) Konstruktor
  - Array erzeugen und mit Zufallszahlen füllen
  - Klasse `Random` verwenden (package java.util)

```
//RandomGenerator erzeugen:
Random r = new Random();

//Array erzeugen

// eine random number generieren:
r.nextInt(1000);
```

- 2b) `toString()`
  - Stringrepräsentation des Arrays (Format in Javadoc vorgegeben)

```
String s = "";
for ( int i=0; i<array.length, i++ )
    …
return s;
```

U2.A2: Sortieren

- 2c) `recursiveSort(int until)`
  - Aufruf aus `sort()` mit array.length

- **Kernidee der Rekursion**: Reduzieren einer Probleminstanz auf eine *kleinere* Probleminstanz.
- **Gegeben**: Liste mit n Elementen

```
Um eine Liste absteigend zu sortieren, brauche ich nur...
        ... die ersten (i – 1) Elemente absteigend sortieren
        ... das grösste Element im Rest der Liste suchen
        ... und an Stelle i setzen
```

- Eine leere Liste ist bereits sortiert... ;-)

Wichtige Aufgabe für den Rest des Semesters: Viel Rekursion!

```
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 9 1 5 2 ]
[ 9 1 5 2 ]
[ 9 5 1 2 ]
[ 9 5 1 2 ]
[ 9 5 2 1 ]
[ 9 5 2 1 ]
```

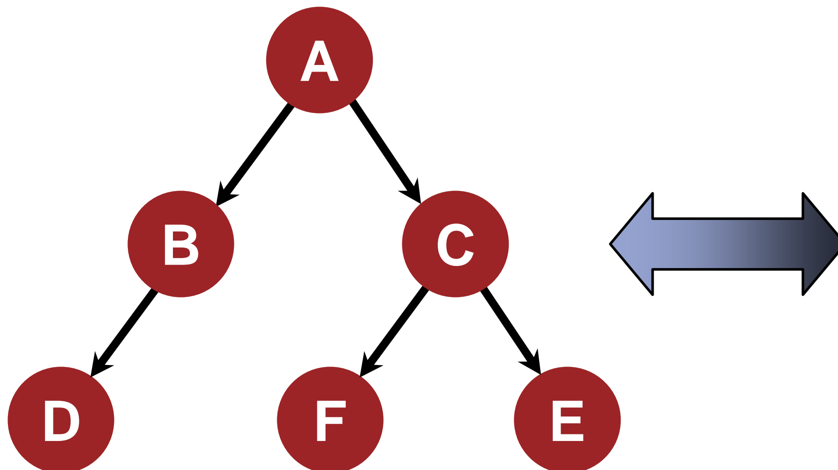
```
recursiveSort(4)
    recursiveSort(3)
        recursiveSort(2)
            recursiveSort(1)
                recursiveSort(0)
                Ist sortiert!
            9 <- findLargest(0,3)
            Swap
        5 <- findLargest(1,3)
        Swap
    2 <- findLargest(2,3)
    Swap
Kein swap mehr noetig...
```

→ Liste absteigend sortiert!

Animation von Simon Mayer

U2.A3: Binärbaum als Array

- Gerüst auf der Webseite/Codeboard: u2a3.BinaryTree
- Binärbäume kann man leicht in einem Array speichern, wenn dieses geeignet interpretiert wird.




```
char[] tree = new char[7];

tree[0] = 'A';
tree[1] = 'B';
tree[2] = 'C';
tree[3] = 'D';
tree[4] = ' ';
tree[5] = 'F';
tree[6] = 'E';
```

U2.A3: Binärbaum als Array

- Gerüst auf der Webseite/Codeboard: u2a3.BinaryTree
- Binärbäume kann man leicht in einem Array speichern.

- Idee:
  - Die Wurzel an Index 0 des Arrays zu setzen
  - Die beiden direkten Nachfolger von i an den Positionen $2i + 1$ und $2i + 2$ zu speichern
  - Wie gross ist das Array, welches den Binärbaum speichert?

  $2^{\text{Höhe}-1} \leq \text{array.length} < 2^{\text{Höhe}}$

1) Formeln nutzen – Implementierung ist dann einfach
2) Eltern-Knoten von Knoten 0 ist Knoten 0

U2.A3: toString()-Methode

3b) `toString()` – Binärbaum in eingerückter Form

Schwierigkeit: Einrückung – viele Möglichkeiten denkbar.

1) Rekursiv: Knoten „printen”, einmal einrücken, rekursiver Aufruf, und wieder ausrücken.
2) Nichtrekursiv: Zahl der Einrückungen merken und erhöhen/vermindern
3) Nichtrekursiv: Aktuelle Einrückung als String speichern

Idee für Rekursion: Helper-Methode:

- `toString()` ruft `toString(int node, String identation)` auf
- z.B. `toString(0,” ”);`

U2.A3: checkTree()-Methode

3c) `checkTree()` – Array gültige Repräsentation eines Baumes?

Zu prüfen:

- Jedes Element braucht einen Vater (Wurzel ist eigener Vater)
- Nichtleeres Array

Beispiele:

['A','B',' '] → gültig

['A','B'] → gültig

['A','B] → `throw IllegalArgumentException()`

[ ] → `throw IllegalArgumentException()`

→ Schaut auf die Unit Tests!

Random Numbers

Random Numbers in Java

Class Random

- An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. (See Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1.)
- If two instances of Random are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers.

From: http://docs.oracle.com/javase/7/docs/api/java/util/Random.html

…viel Spass!