Informatik II
Übung 02

Marian George
marian.george@inf.ethz.ch
Outlook

- Exercise 1 solution discussion
- Exercise 2 tackles (trees, recursion, sorting)
Outlook

- Exercise 1 solution discussion
- Exercise 2 tackles (trees, recursion, sorting)
Solution U1.A1

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

a) Is it possible to prove the correctness of the algorithm by induction over \( a \)?
   It is **not** possible to prove the correctness by induction over \( a \).
   The induction base already fails for \( b > 1 \!\).
   The size of \( a \) is ever-growing (in each step, \( a \to 2 \times a \) and induction never stops).
   \( \rightarrow \) No conclusion is possible on already proven cases and no induction hypothesis is formulated.

b) Does the algorithm terminate?
   Yes, if we can make the value of \( b \) reach 1
   Is that the case?
   Yes! Because \( b \) is always halved in each call of the function
   \( \rightarrow \) After \( \lfloor \log_2(b) \rfloor \) step, the value of \( b=1 \) is reached!
c) How do we prove the correctness of the algorithm when the smallest case is $b=0$?

The definition of the function becomes:

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

The induction hypothesis becomes:

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

The induction step is similar to the original, as

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

In 1b) we have shown that the case of $b=1$ is always reached. Since the integer division of 1 by 2 gives 0, then the case of $b = 0$ is also always reached. No change in the proof of this step is required.
Solution U1.A2a – Recursive method calls

gerade(int x)

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

⇒ x (or x+1)

derdopple(int x)

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

⇒ x (or x+1)

halbiere(int x)

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

⇒ ⌊x/2⌋ (or ⌊x/2⌋+1)
Solution U1.A2b

The total number of calls to the three methods in terms of \( a \) and \( b \) by a single call to \( f \)

```java
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 each case \( \text{gerade}(b), \text{verdopple}(a) \) and \( \text{halbiere}(b) \) are called. The number of calls (with results from part A2a) is therefore at most

\[
b+1 + a+1 + \lfloor b/2 \rfloor +1 \approx a + 3b/2 + 3
\]
Solution U1.A2c

Total number of method calls:

It is not \((\# \text{ calls of } f) \times (\# \text{ Total number of calls for a single instance of } f)\)

With the results of 2b), we get:

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

The recursion terminates when \(b = 0\). This is reached after \(k = \lfloor \log_2 b \rfloor + 1\) calls, because \(b\) is halved after each step.

In the end, you are going to get \(\approx 2ab - a + 3b\)
Solution U1.A3

```java
/**
 * 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);
}
```
public static int mult(int a, int b) 
{
    try {
    if (a <= 0)
            throw new IllegalArgumentException("A negative!");
    }
catch(IllegalArgumentException e)
    {
        ...
    }
}
Outlook

- Exercise 1 solution discussion
- Exercise 2 tackles (trees, recursion, sorting)
Exercise 2

Overview

1. Rooted Trees (theory)
   a. Representation of tree using (i) brackets and (ii) indented
   b. Given brackets representation, (i) draw tree and (ii) give indented format
   c. Can the tree in 1b be reconstructed nonambiguously? Why/Why not?
   d. For the trees in 1a and 1b: Give (i) height of tree [1 node has height 1], (ii) longest paths [trees are directed!], and (iii) set of leaves

2. Recursive Sorting
   a. Constructor: Create array of given size and fill with random numbers
   b. Build method toString
   c. Create recursiveSort(int until) to sort numbers in descending order

3. Binary Trees
   - Check Trees
Overview on some different types of trees

- **General Tree**: Every node has X child nodes

- **Binary Tree**: Each node has at most two child nodes

- **Binary/Ternary Search Tree (BST)**: Nodes are saved in an ordered form

- **Trie** (from «Retrieval»): Not the content but the position of the node that matters, i.e. edges carry information!
  (e.g. Suffix tree → text autocomplete)

**Task**: Deal with different representations of trees!
Q2: Recursive Sorting

- **Constructor**
  - Produce array of randomly generated numbers
  - Import Random Class (package: `import java.util.Random`)

```java
//RandomGenerator
Random r = new Random();

//Array...

//1 random number generieren:
r.nextInt(1000);
```

- **Method toString()**

```java
String s = "";
for ( int i=0; i < array.length, i++; )
    ...
return s;
```
Q2: Recursive Sorting

- **recursiveSort**(int until)

  - Core idea of recursion is to reduce the problem to smaller instances of the same problem ...

  - Given a list of (N) element

<table>
<thead>
<tr>
<th>To order list of i elements in descending order, I need the following...</th>
</tr>
</thead>
<tbody>
<tr>
<td>... Sorting the first (i - 1) elements with descending order</td>
</tr>
<tr>
<td>... Search for the largest element in the list</td>
</tr>
<tr>
<td>... place it in the first place of the list</td>
</tr>
</tbody>
</table>

- The empty list is a sorted list
recursiveSort(4)

recursiveSort(3)

recursiveSort(2)

recursiveSort(1)

Ist sortiert!

swap(0,2)

2 <- findLargest(0,3)

recursiveSort(0)

swap(1,2)

3 <- findLargest(2,3)

swap(2,3)

Swap is not necessary anymore...

→ List sorted in descending order!
Q3: Binary Tree as an Array

- Binary trees can be saved as arrays given proper interpretation

- The idea is as follow:
  - Set the root of the tree to have an index 0
  - The next two array position from (i) are saved, namely the positions \((2i+1)\) and \((2i+2)\)
  - What is the size of the array that stores the binary tree?

\[
2^{\text{height}-1} \leq \text{array.length} < 2^{\text{height}}
\]
Q3: Binary Tree as an Array

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';

Is this also possible with general (= non-binary) trees?
Q3: Binary Tree as an Array

- **toString()**
  Idea:
  - `toString()` call ➔ `toString(int node, String indentation)`
    E.g. `toString(0, " ");`

- **checkTree()**
  Idea:
  - Root at index 0
  - Direct successor $i$ to $2i + 1$ and $2i + 2$
    - $2^{high-1} \leq \text{array.length} < 2^{high}$

- **Check if this applies for the passed array**
  - Test: Every element has a parent node
    - "The root is its own father."
  - What about the empty nodes?
Trees in Computer Science...

Images: http://kitabundsunnah.wordpress.com/, http://www.cs.lmu.edu/courses
Have Fun!