Informatik II (D-ITET)

Tutorial 3

TA: Marian George
marian.george@inf.ethz.ch

Distributed Systems Group, ETH Zürich
Solution U2.A1

Bracket and indented form
\[ S(R(H(K)), P(A(N,O), Q,T), V(J,F(G))) \]

Can you reconstruct the tree given by the comma representation shown above without any ambiguity?

Yes, if the position of the nodes is irrelevant (left or right successor?)
Solution U2.A1

Remarks:

1. Beware of the number of brackets

2. Those two are called leaves just not single nodes.

3. Longest path depends on whether the tree is defined as a directed graph.
   Maths: Tree = Connected and acyclic graph
   CS: Tree = Connected, acyclic and directed graph
   Forest: Acyclic graph (directed/undirected)
Solution U2.A2

recursiveSort()
Informatik II - Übung 03

recursiveSort(4)

recursiveSort(3)

recursiveSort(2)

recursiveSort(1)

recursiveSort(0)

Sorted!

9 <- findLargest(0,3)

Swap

5 <- findLargest(1,3)

Swap

2 <- findLargest(2,3)

Swap

No need for further swap!

→ List is now in decreasing order!

Animation von Simon Mayer
Solution U2.A2

SWAP function

```java
/**
 * swaps two fields of {@link RandomArray#numbers}
 *
 * @param i a valid index into {@link RandomArray#numbers}
 * @param j a valid index into {@link RandomArray#numbers}
 */
private void swap(int i, int j)
{
    int tmp = numbers[j];
    numbers[j] = numbers[i];
    numbers[i] = tmp;
}
```
## Solution U2.A2

**Remark:** SWAP can also be implemented without a temporary variable

\[
\begin{array}{c|c}
  x & y \\
  1010 & 0011 \\
  - & - \\
  1001 & 0011 \\
  1001 & 1010 \\
  0011 & 1010 \\
\end{array}
\]

X := X XOR Y
Y := X
X := X XOR Y
Solution U2.Ae

SWAP inside a loop: good idea?

```c
void recursiveSort( int until ) {
    // 0 elements are considered to be sorted
    if( until == 0 )
        return;

    // sort first until-1 elements in the array
    recursiveSort( until - 1 );

    // bring the greatest element from the rest to position until-1
    for( int i = until; i < a.length; i++ )
    {
        if( a[i] > a[until-1] ){
            swap(until-1, i);
        }
    }
}
```
Solution U2.A2

Better: search first, then exchange

```java
void recursiveSort( int until ) {
    // 0 elements are considered to be sorted
    if( until == 0 )
        return;

    // sort first until-1 elements in the array
    recursiveSort( until - 1 );

    // find index of greatest element after until-1
    int maxIndex = until - 1;
    for( int i = until; i < a.length; i++ ) {
        if( a[i] > a[maxIndex] ) {
            maxIndex = i;
        }
    }

    // swap elements at maxIndex and until-1
    swap( until-1, maxIndex );
}
```

For 15 values, on average ~57 swaps in first case and ~11 swaps in second case.
Solution U2.A2

Coding Style - Formatting

Eclipse: $\text{Ctrl}+\text{Shift}+\text{F}$ and the code is nicely formatted (indented)

```java
while ((e+i)<=14) {
    if (a[e] > a[e+i]) {
        e++;
        i=1;
    }
    else
        i++;
}
```
Solution U2.A2

Coding Style – No hardcoding!

```java
if (x < 10) {
    x < a.length
}
```

```java
if (myString.compareTo("hello world") == 0);
```

```java
private static final String REF = "hello world";
... 
if (myString.compareTo(REF) == 0);
```
Solution U2.A2

Coding Style – Loops

```java
for (int i=0; i < MAX_I; ++i) {
    nextIterationStep();
}
```

```java
int timeout = 0;
while(!userInteraction()) {
    Thread.yield();
    timeout++;
}
```
Solution U2.A2

- Coding Style – Difference?

```cpp
if (index >= boundary)
    return;
else if (array[index] == 'x')
    return;
```

```cpp
if (index >= boundary ||
    array[index] == 'x')
    return;
```

**Y in expression (X || Y) is only evaluated if X == false (border effect)**

```cpp
if (index < boundary)
    if (array[index] == 'x')
        array[index] = '\0';
```

```cpp
if (index < boundary &&
    array[index] == 'x')
    array[index] = '\0';
```

**Y in expression (X && Y) is only evaluated if X == true**

```cpp
int counter = 0;
while (counter < n) {
    ...
    counter++;
}
```

```cpp
for (int counter = 0;
    counter < n;
    counter++) {
    ...
}
```

**Warning: counter is still defined outside the loop!**

**Clean counting: counter can be reused out of the for loop.**
Solution U2.A2

Coding Style – Efficiency

Object initialization is expensive!

```java
void initialize() {
    for (int i=0; i<a.length; i++) {
        Random r = new Random();
        a[i] = r.nextInt(1000);
    }
}
```

```java
void initialize() {
    Random r = new Random();
    for (int i=0; i<a.length; i++) {
        a[i] = r.nextInt(1000);
    }
}
```
Solution U2.A3a

- Root at index 0
- Successor of i is at $2i + 1$ and $2i + 2$
- $2\text{height} - 1 = 2\text{depth} \leq \text{array.length} \leq 2\text{depth} + 1 = 2\text{height}$

```c
int leftChild( node ){
    return 2 * node + 1;
}

int rightChild( node ){
    return 2 * node + 2;
}

int father( node ){
    return (node - 1) / 2;
}

(father(0) = -1/2 = 0)
```
Solution U2.A3c

- **checkTree()**

- **Test the assignment of the array**
  - **Test:** Each node has a father
    - The root is its own father.
    - Solution is trivial.
  - **What happens with empty nodes?**
    - No need for a father.

```java
//
/* Check if the given array represents a valid binary tree.
 * @param array a binary tree encoded as char array
 * @throws IllegalArgumentException if check fails
 */
private static void checkTree(char[] array)
{
    if (array.length == 0) throw new IllegalArgumentException("At least one, probably empty node is required.");
    for (int i=0; i<array.length; i++) {
        if (array[i] != ' ') {
            int f = father(i);
            if (array[f] == ' ') {
                throw new IllegalArgumentException("node number " + i + " has no father");
            }
        }
    }
}
```
Solution U2.A3b

- `toString()`

```java
/**
 * recursive toString
 *
 * @param node index of the root of the subtree which has to be converted.
 * @param indentation the current indentation
 * @return indented form of the selected subtree
 */
private String toString(int node, String indentation) {
    assert (tree[node] != ' ');

    String s = indentation;
    s = s + tree[node] + '\n';

    int c1 = leftChild(node);
    if (isNode(c1)) {
        s = s + toString(c1, indentation + " ");
    }

    int c2 = rightChild(node);
    if (isNode(c2)) {
        s = s + toString(c2, indentation + " ");
    }

    return s;
}
```
HINTS FOR U3
Hints U3

1. Objects and references (String vs. StringBuffer)
   - Caesar cipher algorithm, how to decrypt a message? Runtime when using StringBuffer instead of String?
   - Describe the behavior of the Main method.
   - Why are Strings ever used at all?

2. Syntax diagrams
   - Given the diagrams, which expressions can be produced?

3. Syntax checker for trees
   - Complete the syntax diagrams from the class
   - Implement the syntax checker
Hints U3.A1

- **String – Immutable**
  - Optimization possible because static
  - Modification only through copy

- **StringBuffer – Mutable**
  - Easily modifyable (without copy)
  - Some operations are more expensive (e.g. search)
Hints U3.A1

Difference between `String` and `StringBuffer`

```
String myString = "hello";
myString = myString + " world";
```

JAVA String concatenation

```
StringBuffer myStringBuffer = "hello";
myStringBuffer.append(" world");
```

StringBuffer Method

Animation by Beat Saurenmann
Hints U3.A1

String myString = "hello";
myString = myString+" world";
myString = myString+" how";
myString = myString+" are";
myString = myString+" you";
myString = myString+" today";

Speicher
"hello"
" hello world"
"hello world how"
"hello world how are"
"hello world how are you"
"hello world how are you today"

Garbage Collector

Animation von Beat Saurenmann
Hints U3.A1

- Caesar Cipher
  - Replace each plaintext letter with the letter at three places further down the alphabet.

Hints U3.A1

- Caesar Cipher
  - Given: encrypt(String s) method
    - The method uses a String object to represent the encrypted text
    - The String is appended with a new character in each iteration

```java
public static String encrypt(String s) {
    String ret = "";
    for (int i = 0; i != s.length(); ++i) {
        ret = ret + (char) (s.charAt(i) + 3);
    }
    return ret;
}
```
Hints U3.A1

- Caesar cipher, how to decrypt a message?
- Runtime analysis when using StringBuffer instead of String?
- Analyze the behavior of the Main method.
- Why are Strings ever used at all?
Hints U3.A2 – Syntax diagram

Syntax diagram, as presented in class…

Var:

Clause:

Expr:

\[ \neg (X_1 \lor X_2 \ldots \lor X_n) \land \neg (X_1 \lor X_2 \ldots \lor X_n) \]

e.g. \( (\neg X_1 \lor X_2) \land \neg (X_n) \)
Hints U3.A3

Implementing a syntax checker for trees

- First:
  - Extension of tree syntax to empty subtrees
- Syntax checker
  - Own methods for «Tree», «Successor» and «Nodes»

Approach: `public static int f(String str, int offset){...}

Give the new offset to the above mentioned method. Think about how large the offset should be in the end and what happens if you don’t get the right value

Hint: Solution of 3a (String, StringBuffer) should be integrated

Possible problems and their resolutions/explanations:

`StringIndexOutOfBoundsException` — you are trying to access character at position \( n \) in the string, but the array is shorter than \( n \).
Have fun!