

# Theoretische Grundlagen der Informatik II

## Blatt 5

Markus Vieth, David Klopp, Christian Stricker

26. November 2015



# Aufgabe 1

a-d)

Node.java

```

1  class Node<T extends Comparable<T>> extends AbstractNode<T>{
2
3      public Node(T key){
4          super(key);
5      }
6
7      public boolean insert (T key)
8      {
9          int direction = key.compareTo(this.key);
10
11         // use direction to choose if left or right, zero means equality
12         if (direction < 0) {
13             if (this.left != null)
14                 this.left.insert(key);
15             else {
16                 this.left = new Node<T>(key);
17                 return true;
18             }
19         } else if (direction > 0) {
20             if (this.right != null)
21                 this.right.insert(key);
22             else {
23                 this.right = new Node<T>(key);
24                 return true;
25             }
26         }
27         // should be more sophisticated than xkcd random dice roll https://xkcd.com/221/
28         return false;
29     }
30
31     public boolean isIn (T key){
32         int direction = key.compareTo(this.key);
33         // use direction to choose if left or right, zero means equality
34
35         if (direction == 0) {
36             return true;
37         } else if (direction < 0 && this.left != null) {
38             return this.left.isIn(key);
39         } else if (this.right != null){
40             return this.right.isIn(key);
41         }
42
43         // should be more sophisticated than xkcd random dice roll https://xkcd.com/221/
44         return false;
45     }
46
47     public void preorder() {
48         System.out.println(this.key);
49
50         // linker Teilbaum
51         if (this.left != null )
52             this.left.preorder();
53
54         // rechter Teilbaum
55         if (this.right != null )
56             this.right.preorder();
57     }
58
59     public void inorder() {
60         // linker Teilbaum
61         if (this.left != null )
62             this.left.inorder();

```

```

64     System.out.println(this.key);
65
66     // rechter Teilbaum
67     if (this.right != null )
68         this.right.postorder();
69 }
70
71 public void postorder()
72 {
73     // linker Teilbaum
74     if (this.left != null )
75         this.left.postorder();
76
77     // rechter Teilbaum
78     if (this.right != null )
79         this.right.postorder();
80
81     System.out.println(this.key);
82 }
83
84
85 public int maxDepth(){
86
87     if (left == null && right == null)
88         return 1;
89     if (left == null)
90         return right.maxDepth()+1;
91
92     int leftD = left.maxDepth();
93
94     if (right == null)
95         return leftD+1;
96
97     int rightD = right.maxDepth();
98
99     return leftD>rightD ? leftD+1 : rightD+1 ;
100 }
101 }

```

## Tree.java

```

1  import java.util.Random;
2
3  // Wrapper class, there is no need for customization
4  class Tree<T extends Comparable<T>> extends AbstractTree<T> {
5
6      public Tree(T key) {
7          root = new Node<T>(key);
8      }
9
10     public boolean insert(T key) {
11         return root.insert(key);
12     }
13
14     public boolean isIn(T key) {
15         return root.isIn(key);
16     }
17
18     // all the traversal methods! :D |--> :(
19     public void preorder() {
20         root.preorder();
21     }
22
23     public void inorder() {
24         root.inorder();
25     }
26
27     public void postorder() {
28         root.postorder();
29     }
30 }

```

```

31 // recursive depth method
32 public int maxDepth() {
33     return root.maxDepth();
34 }

36 public static void main(String args[]) {

38     // new Integer tree (put in 2 as root key)
39     Tree<Integer> T = new Tree<Integer>(2);
40     // put in 3 (true)
41     System.out.println(T.insert(3));
42     // put in 3 (false - keys are unique!)
43     System.out.println(T.insert(3));
44     // 2 in tree (true)
45     System.out.println(T.isIn(2));

47     System.out.println("-----Preorder-----");
48     T.preorder();
49     System.out.println("-----Inorder-----");
50     T.inorder();
51     System.out.println("-----Postorder-----");
52     T.postorder();

55     for (int k = 2; k <= 6; k++) { //Cases 10^2,10^3,10^4,10^5,10^6

57         final int[] sum = new int[10];
58         int nMax = (int)Math.pow(10.0, (double)k);

60         final int[] test = new int[nMax];
61         for (int i=0; i < test.length; i++) //Fills array with numbers from 1-10^k
62             test[i] = i+1;

64         Thread[] threads = new Thread[10]; //10 tests

66         for (int i = 0; i < 10; i++) {
67             final int j = i;
68             threads[i] = new Thread(new Runnable() {
69                 @Override
70                 public void run() {

72                     int[] numbers = test.clone(); //Working copy

74                     Random random = new Random();

77                     for (int i = 0; i < numbers.length; i++) { //swapping numbers
78                         int toSwap = random.nextInt(numbers.length);
79                         int temp = numbers[i];
80                         numbers[i] = numbers[toSwap];
81                         numbers[toSwap] = temp;
82                     }

84                     Tree<Integer> tree = new Tree<Integer>(numbers[0]); //put numbers in Tree
85                     for(int n = 1; n < nMax; n++) {
86                         tree.insert(numbers[n]);
87                     }

89                     System.out.print(".");
90                     sum[j] =tree.maxDepth();//save maxDepth

92                 }

94             });
95             threads[i].start(); //start thread
96         }
97         for (Thread thread: threads) //wait till all threads finished
98             try {
99                 thread.join();

```

```

100         } catch (InterruptedException e) {
101             // TODO Auto-generated catch block
102             e.printStackTrace();
103         }

105         int result = 0;
106         for (int i: sum)
107             result += i;

109         System.out.println();
110         System.out.println("k = "+k+" durchschnittliche Tiefe = "+result/10);
111     }

113 }
114 }

```

zu d)

$k = 2$  : durchschnittliche Tiefe = 13

$k = 3$  : durchschnittliche Tiefe = 21

$k = 4$  : durchschnittliche Tiefe = 33

$k = 5$  : durchschnittliche Tiefe = 40

$k = 6$  : durchschnittliche Tiefe = 50

Da beim verzehnfachen der Elemente sich die Tiefe konstant um ca. 9 vergrößert, wächst die Baumtiefe asymptotisch in  $\log n$

e)

### Szenario 1

Es sollen Daten mit nicht fortlaufendem Wert/Schlüssel gefunden werden.

**Beispiel** Das finden eines Wortes in einem Wörterbuch. Ein Array bräuchte hier im Schnitt  $\frac{n}{2}$  Schritte, ein Binärer Suchbaum  $\log_2 n$  und ist somit schneller.

### Szenario 2

Es sollen Duplikate gefunden werden.

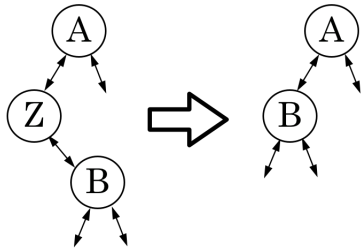
**Beispiel** 2 Datenbanken sollen zusammen gelegt werden und Duplikate sollen gelöscht werden. Ein Array bräuchte hier im Worst-Case  $\sum_{i=1}^n i \in O(n^2)$  Schritte, ein Binärer Suchbaum  $O(n \log n) \ni \sum_{i=1}^n \log_2 i \leq \sum_{i=1}^n \log_2 n = n \log_2 n$  und ist somit schneller.

## Aufgabe 2

Anmerkung: Pfeile ohne Endknoten deuten einen beliebig großen Teilbaum an. Teilbäume bei Knoten D wurden vergessen einzuzichnen, können aber natürlich auch existieren.

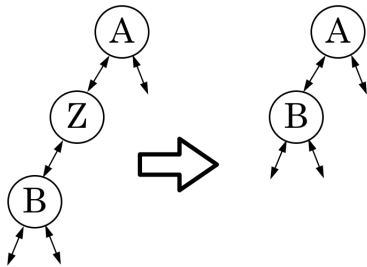
a)

Ersetze z mit seinem rechten Kind. Da z kein linkes Kind besitzt, ist das Löschen von z fertig.



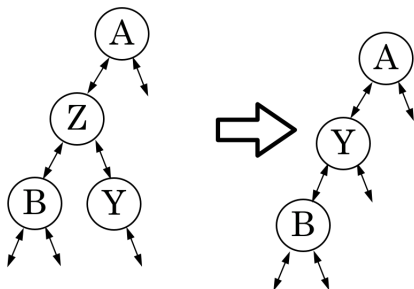
b)

Ersetze z mit dem linken Kind. Da z kein rechtes Kind besitzt, ist das Löschen von z fertig.



c)

Hänge den linken Teilbaum von z an y und tausche y mit z.



d)

Suche das kleinste Element y vom rechten Teilbaum von z. Setze y als Parent vom rechten Kind von z und das rechte Kind von z als rechtes Kind von y. Danach ersetze z mit y und setze das linke Kind von z an die Stelle vom linken Kind von y. Das Löschen ist fertig.

