HATTINGER
NOFAL
NIKOLIC

Aufgabe 19

IK: Anz. Inn. Knoten
B: Anz. Blätter

a)

*) Betrachte Insert und seine Auswirkungen auf die Anzahl IK und B:

Insert: 1)* Wenn (Parentknoten hat kein Kind):
wird Parent von Blatt zu
Innerem Knoten und ein Blatt
wird hinzugefügt
⇒ B konstant, IK+1

2)* Wenn (Parentknoten hat ein Kind)
wird ein neues Blatt hinzugefügt
⇒ B+1, IK konstant

) Da 2) nur für jedes ausgeführte 1)* anwendbar ist, muss die Anzahl Durchführungen von von 2)* ≤ 1)* sein.

~~Daraus folgt IK+1 ≤ B IK+1 ≤ B~~

*) Daraus folgt: $IK + 1 \geq B$

↑
+1 wegen Wurzel (Initialisierung)

HATTINGEN
NOTAC
NIKOLIC

19b)

Trivialerweise wird in diesem Fall für jedes 1)* auch ein 2)* durchgeführt, was zu einem gleichmäßigem Anstieg beider Werte, Ik und B, führt.

Wenn jeder Ik zwei Kinder Besitzt, so ~~oder~~ müssen 1)* und 2)* gleich oft durchgeführt worden sein $\Rightarrow$ $\underline{Ik+1=B}$

Aufgabe 20
Nofal, Nikolic, Hattinger

```
 static int sumOfLeafDepths( TreeNode node, int depth ) {
            // When called as sumOfLeafDepths(root,0), this will compute the
            // sum of the depths of all the leaves in the tree to which root
            // points.  When called recursively, the depth parameter gives
            // the depth of the node, and the routine returns the sum of the
            // depths of the leaves in the subtree to which node points.
            // In each recursive call to this routine, depth goes up by one.
        if ( node == null ) {
               // Since the tree is empty and there are no leaves,
               // the sum is zero.
           return 0;
        }
        else if ( node.left == null && node.right == null) {
               // The node is a leaf, and there are no subtrees of node, so
               // the sum of the leaf depths is just the depth of this node.
           return depth;
        }
        else {
               // The node is not a leaf.  Return the sum of the
               // the depths of the leaves in the subtrees.
           return sumOfLeafDepths(node.left, depth + 1)
                        + sumOfLeafDepths(node.right, depth + 1);
        }
    } // end sumOfLeafDepth()
```

HATTINGER
NOFAL
NIKOLIC

Aufgabe 21)

-) Gewichteter Tree: Anz Nachfolger im linken bzw rechten Subtree bekannt.
-) kt kleinstes Element = k-tes Element im BST
-) Können feststellen ob sich Wert links oder rechts von root befindet.

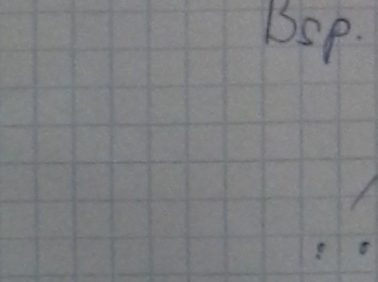
Dabei gilt:

Wenn k im linken Subtree ($k < \text{Weight}[\text{left}]$)
- starte Suche rekursiv im linken Subtree

~~sogar~~ falls root = k tes Element

Wenn k im rechten Subtree ($k > \text{Weight}[\text{left}] + 1$)
- starte Suche rekursiv im rechten Subtree.
- Setze k auf $k - \text{Weight}[\text{left}] - 1$.

Bsp.



Weil es nur noch das 5t kleinste im rechten Subtree ist.

-) Wählen ~~wir~~ nun die geeignete Abbruch bedingung:

Wenn genau k-1 Elemente im linken Subtree sind, so ~~ist~~ enthält ~~trivialerweise~~ wegen BST-Eigenschaften unser aktueller Node die gesuchte Zahl.