



```

AVL-Insert(key, val, root):
    if root = NULL
        x = new node(key, val)
***    x.height = 0
        return x

    if key < root.key
        root.left = Insert(key, val, root.left)
        root.left.parent = root
    else
        root.right = Insert(key, val, root.right)
        root.right.parent = root

*** root.height = 1+max(root.left.height, root.right.height)
*** return Rebalance(root)

```

Rebalance(x):

```
if x.left.height = x.right.height - 2  
    return LeftRotate(x)
```

```
else if x.right.height = x.left.height - 2  
    return RightRotate(x)
```

```
else return x
```

Je toto dobre?

Rebalance(x):

```
if x.left.height = x.right.height - 2
```

```
  y = x.right
```

```
  if y.right.height = y.height - 2
```

```
    y = RightRotate(y); y.parent = x; x.right = y
```

```
  return LeftRotate(x)
```

```
else if x.right.height = x.left.height - 2
```

```
  y = x.left
```

```
  if y.left.height = y.height - 2
```

```
    y = LeftRotate(y); y.parent = x; x.left = y
```

```
  return RightRotate(x)
```

```
else return x
```

Rebuild(x):

  A=empty array

  inorder(x,A)

  return buildBalanced(A,1,size\_of(A))

inorder(x,A):

  if x = NULL return

  inorder(x.left,A)

  push(A,(x.key,x.val))

  inorder(x.right,A)

```
buildBalanced(A,from,to):
    mid = (from+to)/2
    root = new node(A[mid].key,A[mid].val)
    root.size = 1
    if (from<mid)
        // build left subtree
        left = buildBalanced(A,from,mid-1)
        root.left = left
        left.parent = root
        root.size += left.size
    if (to>mid)
        // build right subtree
        right = buildBalanced(A,mid+1,to)
        root.right = right
        right.parent = root
        root.size += right.size
    return root
```

```

Insert(key, val, root):
    if root = NULL
        x = new node(key, val)
**  x.size = 1
**  return (x, x, 0)

    if key < root.key
**  (root.left, x, depth) = Insert(key, val, root.left)
        root.left.parent = root
**  root.size += 1
    else
**  (root.right, x, depth) = Insert(key, val, root.right)
        root.right.parent = root
**  root.size += 1

**return (root, x, depth+1)

```

```

Scapegoat_Insert(key, val, root):
    (root, x, depth) = Insert(key, val, root)
    if depth > maxDepth(root.size)
        scapegoat = findScapegoat(x)
        parent = scapegoat.parent
        y = Rebuild(scapegoat)
        if (parent = NULL)
            return y
        else
            if (parent.left = scapegoat)
                parent.left = y
            else
                parent.right = y
            y.parent = parent
    return root

```



```
findScapegoat(x):  
    if (x.left and x.left.size > (2/3)*x.size)  
        return x  
    if (x.right and x.right.size > (2/3)*x.size)  
        return x  
    return findScapegoat(x.parent)
```