

7.3 AVL-Trees

Definition 1

AVL-trees are binary search trees that fulfill the following balance condition. For every node v

$$|\text{height}(\text{left sub-tree}(v)) - \text{height}(\text{right sub-tree}(v))| \leq 1 .$$

Lemma 2

An AVL-tree of height h contains at least $F_{h+2} - 1$ and at most $2^h - 1$ internal nodes, where F_n is the n -th Fibonacci number ($F_0 = 0, F_1 = 1$), and the height is the maximal number of edges from the root to an (empty) dummy leaf.

7.3 AVL-Trees

Definition 1

AVL-trees are binary search trees that fulfill the following balance condition. For every node v

$$|\text{height}(\text{left sub-tree}(v)) - \text{height}(\text{right sub-tree}(v))| \leq 1 .$$

Lemma 2

An AVL-tree of height h contains at least $F_{h+2} - 1$ and at most $2^h - 1$ internal nodes, where F_n is the n -th Fibonacci number ($F_0 = 0, F_1 = 1$), and the height is the maximal number of edges from the root to an (empty) dummy leaf.

Proof.

The upper bound is clear, as a binary tree of height h can only contain

$$\sum_{j=0}^{h-1} 2^j = 2^h - 1$$

internal nodes.

Proof (cont.)

Induction (base cases):

1. an AVL-tree of height $h = 1$ contains at least one internal node, $1 \geq F_3 - 1 = 2 - 1 = 1$.
2. an AVL tree of height $h = 2$ contains at least two internal nodes, $2 \geq F_4 - 1 = 3 - 1 = 2$



Proof (cont.)

Induction (base cases):

1. an AVL-tree of height $h = 1$ contains at least one internal node, $1 \geq F_3 - 1 = 2 - 1 = 1$.
2. an AVL tree of height $h = 2$ contains at least two internal nodes, $2 \geq F_4 - 1 = 3 - 1 = 2$



Proof (cont.)

Induction (base cases):

1. an AVL-tree of height $h = 1$ contains at least one internal node, $1 \geq F_3 - 1 = 2 - 1 = 1$.
2. an AVL tree of height $h = 2$ contains at least two internal nodes, $2 \geq F_4 - 1 = 3 - 1 = 2$

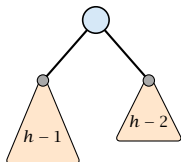


Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.

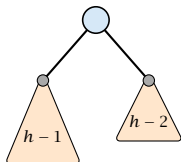
Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.



Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.

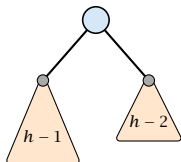


Let

$$g_h := 1 + \text{minimal size of AVL-tree of height } h .$$

Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.



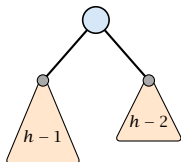
Let

$$g_h := 1 + \text{minimal size of AVL-tree of height } h .$$

Then

Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.



Let

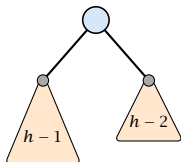
$$g_h := 1 + \text{minimal size of AVL-tree of height } h .$$

Then

$$g_1 = 2 \qquad \qquad \qquad = F_3$$

Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.



Let

$$g_h := 1 + \text{minimal size of AVL-tree of height } h .$$

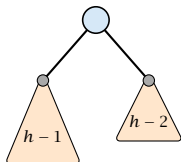
Then

$$g_1 = 2 \qquad \qquad \qquad = F_3$$

$$g_2 = 3 \qquad \qquad \qquad = F_4$$

Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.



Let

$$g_h := 1 + \text{minimal size of AVL-tree of height } h .$$

Then

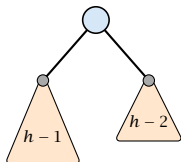
$$g_1 = 2 \qquad = F_3$$

$$g_2 = 3 \qquad = F_4$$

$$g_{h-1} = 1 + g_{h-1} - 1 + g_{h-2} - 1, \qquad \text{hence}$$

Induction step:

An AVL-tree of height $h \geq 2$ of minimal size has a root with sub-trees of height $h - 1$ and $h - 2$, respectively. Both, sub-trees have minimal node number.



Let

$$g_h := 1 + \text{minimal size of AVL-tree of height } h .$$

Then

$$g_1 = 2 \qquad = F_3$$

$$g_2 = 3 \qquad = F_4$$

$$g_{h-1} = 1 + g_{h-1} - 1 + g_{h-2} - 1, \qquad \text{hence}$$

$$g_h = g_{h-1} + g_{h-2} \qquad = F_{h+2}$$

7.3 AVL-Trees

An AVL-tree of height h contains at least $F_{h+2} - 1$ internal nodes.

Since

$$n + 1 \geq F_{h+2} = \Omega \left(\left(\frac{1 + \sqrt{5}}{2} \right)^h \right),$$

we get

$$n \geq \Omega \left(\left(\frac{1 + \sqrt{5}}{2} \right)^h \right),$$

and, hence, $h = \mathcal{O}(\log n)$.

7.3 AVL-Trees

We need to maintain the balance condition through rotations.

For this we store in every internal tree-node v the **balance** of the node. Let v denote a tree node with left child c_ℓ and right child c_r .

$$\text{balance}[v] := \text{height}(T_{c_\ell}) - \text{height}(T_{c_r}) ,$$

where T_{c_ℓ} and T_{c_r} , are the sub-trees rooted at c_ℓ and c_r , respectively.

7.3 AVL-Trees

We need to maintain the balance condition through rotations.

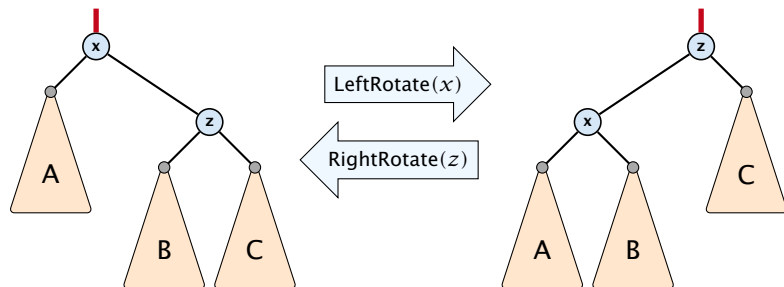
For this we store in every internal tree-node v the **balance** of the node. Let v denote a tree node with left child c_ℓ and right child c_r .

$$\text{balance}[v] := \text{height}(T_{c_\ell}) - \text{height}(T_{c_r}) ,$$

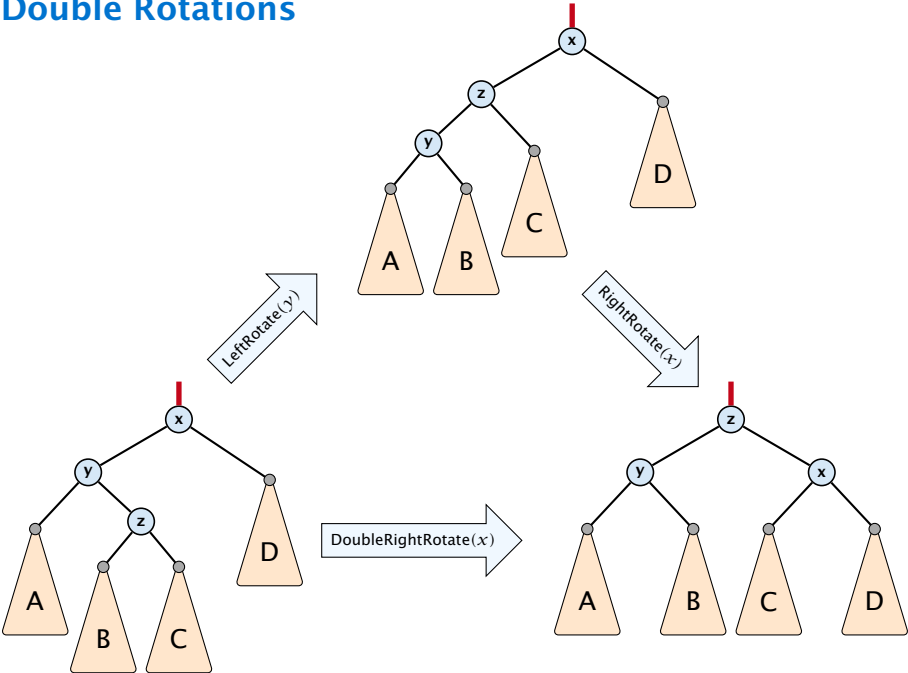
where T_{c_ℓ} and T_{c_r} , are the sub-trees rooted at c_ℓ and c_r , respectively.

Rotations

The properties will be maintained through rotations:



Double Rotations



AVL-trees: Insert

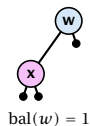
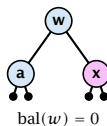
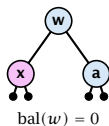
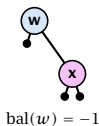
- ▶ Insert like in a binary search tree.

AVL-trees: Insert

- ▶ Insert like in a binary search tree.
- ▶ Let w denote the parent of the newly inserted node x .

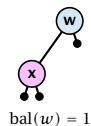
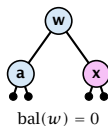
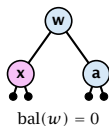
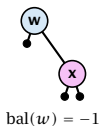
AVL-trees: Insert

- ▶ Insert like in a binary search tree.
- ▶ Let w denote the parent of the newly inserted node x .
- ▶ One of the following cases holds:



AVL-trees: Insert

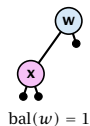
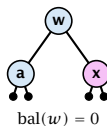
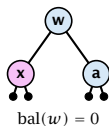
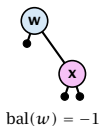
- ▶ Insert like in a binary search tree.
- ▶ Let w denote the parent of the newly inserted node x .
- ▶ One of the following cases holds:



- ▶ If $\text{bal}[w] \neq 0$, T_w has changed height; the balance-constraint may be violated at ancestors of w .

AVL-trees: Insert

- ▶ Insert like in a binary search tree.
- ▶ Let w denote the parent of the newly inserted node x .
- ▶ One of the following cases holds:



- ▶ If $\text{bal}[w] \neq 0$, T_w has changed height; the balance-constraint may be violated at ancestors of w .
- ▶ Call $\text{AVL-fix-up-insert}(\text{parent}[w])$ to restore the balance-condition.

Invariant at the beginning of AVL-fix-up-insert(ν):

1. The balance constraints hold at all descendants of ν .
2. A node has been inserted into T_c , where c is either the right or left child of ν .
3. T_c has increased its height by one (otw. we would already have aborted the fix-up procedure).
4. The balance at node c fulfills $\text{balance}[c] \in \{-1, 1\}$. This holds because if the balance of c is 0, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

Invariant at the beginning of AVL-fix-up-insert(ν):

1. The balance constraints hold at all descendants of ν .
2. A node has been inserted into T_c , where c is either the right or left child of ν .
3. T_c has increased its height by one (otw. we would already have aborted the fix-up procedure).
4. The balance at node c fulfills $\text{balance}[c] \in \{-1, 1\}$. This holds because if the balance of c is 0, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

Invariant at the beginning of AVL-fix-up-insert(v):

1. The balance constraints hold at all descendants of v .
2. A node has been inserted into T_c , where c is either the right or left child of v .
3. T_c has increased its height by one (otw. we would already have aborted the fix-up procedure).
4. The balance at node c fulfills $\text{balance}[c] \in \{-1, 1\}$. This holds because if the balance of c is 0, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

Invariant at the beginning of AVL-fix-up-insert(v):

1. The balance constraints hold at all descendants of v .
2. A node has been inserted into T_c , where c is either the right or left child of v .
3. T_c has increased its height by one (otw. we would already have aborted the fix-up procedure).
4. The balance at node c fulfills $\text{balance}[c] \in \{-1, 1\}$. This holds because if the balance of c is 0, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

Invariant at the beginning of AVL-fix-up-insert(v):

1. The balance constraints hold at all descendants of v .
2. A node has been inserted into T_c , where c is either the right or left child of v .
3. T_c has increased its height by one (otw. we would already have aborted the fix-up procedure).
4. The balance at node c fulfills $\text{balance}[c] \in \{-1, 1\}$. This holds because if the balance of c is 0, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

AVL-trees: Insert

Algorithm 11 AVL-fix-up-insert(v)

- 1: **if** $\text{balance}[v] \in \{-2, 2\}$ **then** DoRotationInsert(v);
- 2: **if** $\text{balance}[v] \in \{0\}$ **return**;
- 3: AVL-fix-up-insert(parent(v));

We will show that the above procedure is correct, and that it will do at most one rotation.

Algorithm 12 DoRotationInsert(v)

```
1: if balance[ $v$ ] = -2 then // insert in right sub-tree
2:     if balance[right[ $v$ ]] = -1 then
3:         LeftRotate( $v$ );
4:     else
5:         DoubleLeftRotate( $v$ );
6: else // insert in left sub-tree
7:     if balance[left[ $v$ ]] = 1 then
8:         RightRotate( $v$ );
9:     else
10:        DoubleRightRotate( $v$ );
```

AVL-trees: Insert

It is clear that the invariant for the fix-up routine holds as long as no rotations have been done.

We have to show that after doing one rotation all balance constraints are fulfilled.

We show that after doing a rotation at v :

- ▶ v fulfills balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ The height of T_v is the same as before the insert-operation took place.

We only look at the case where the insert happened into the right sub-tree of v . The other case is symmetric.

AVL-trees: Insert

It is clear that the invariant for the fix-up routine holds as long as no rotations have been done.

We have to show that after doing one rotation **all** balance constraints are fulfilled.

We show that after doing a rotation at v :

- ▶ v fulfills balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ The height of T_v is the same as before the insert-operation took place.

We only look at the case where the insert happened into the right sub-tree of v . The other case is symmetric.

AVL-trees: Insert

It is clear that the invariant for the fix-up routine holds as long as no rotations have been done.

We have to show that after doing one rotation **all** balance constraints are fulfilled.

We show that after doing a rotation at v :

- ▶ v fulfills balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ The height of T_v is the same as before the insert-operation took place.

We only look at the case where the insert happened into the right sub-tree of v . The other case is symmetric.

AVL-trees: Insert

It is clear that the invariant for the fix-up routine holds as long as no rotations have been done.

We have to show that after doing one rotation **all** balance constraints are fulfilled.

We show that after doing a rotation at v :

- ▶ v fulfills balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ The height of T_v is the same as before the insert-operation took place.

We only look at the case where the insert happened into the right sub-tree of v . The other case is symmetric.

AVL-trees: Insert

It is clear that the invariant for the fix-up routine holds as long as no rotations have been done.

We have to show that after doing one rotation **all** balance constraints are fulfilled.

We show that after doing a rotation at v :

- ▶ v fulfills balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ The height of T_v is the same as before the insert-operation took place.

We only look at the case where the insert happened into the right sub-tree of v . The other case is symmetric.

AVL-trees: Insert

It is clear that the invariant for the fix-up routine holds as long as no rotations have been done.

We have to show that after doing one rotation **all** balance constraints are fulfilled.

We show that after doing a rotation at v :

- ▶ v fulfills balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ The height of T_v is the same as before the insert-operation took place.

We only look at the case where the insert happened into the right sub-tree of v . The other case is symmetric.

AVL-trees: Insert

It is clear that the invariant for the fix-up routine holds as long as no rotations have been done.

We have to show that after doing one rotation **all** balance constraints are fulfilled.

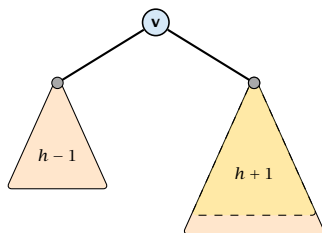
We show that after doing a rotation at v :

- ▶ v fulfills balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ The height of T_v is the same as before the insert-operation took place.

We only look at the case where the insert happened into the right sub-tree of v . The other case is symmetric.

AVL-trees: Insert

We have the following situation:

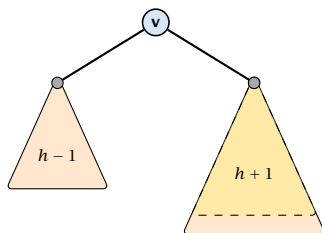


The right sub-tree of v has increased its height which results in a balance of -2 at v .

Before the insertion the height of T_v was $h + 1$.

AVL-trees: Insert

We have the following situation:



The right sub-tree of v has increased its height which results in a balance of -2 at v .

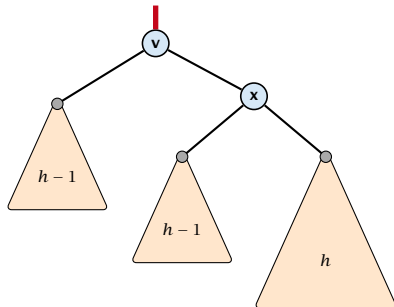
Before the insertion the height of T_v was $h + 1$.

Case 1: $\text{balance}[\text{right}[v]] = -1$

We do a left rotation at v

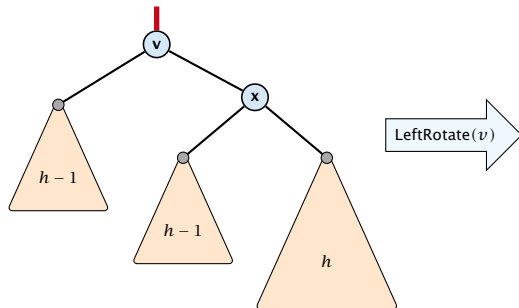
Case 1: $\text{balance}[\text{right}[v]] = -1$

We do a left rotation at v



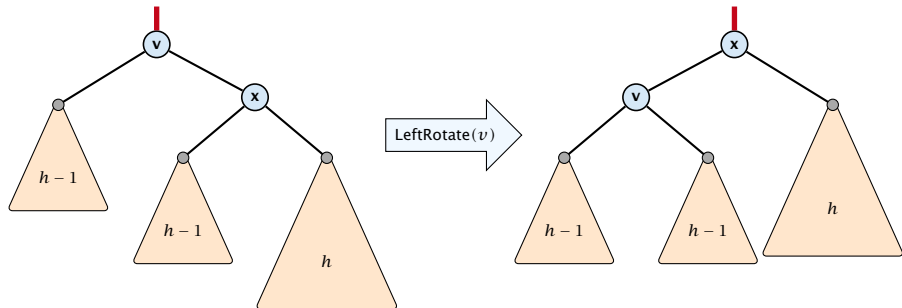
Case 1: $\text{balance}[\text{right}[v]] = -1$

We do a left rotation at v



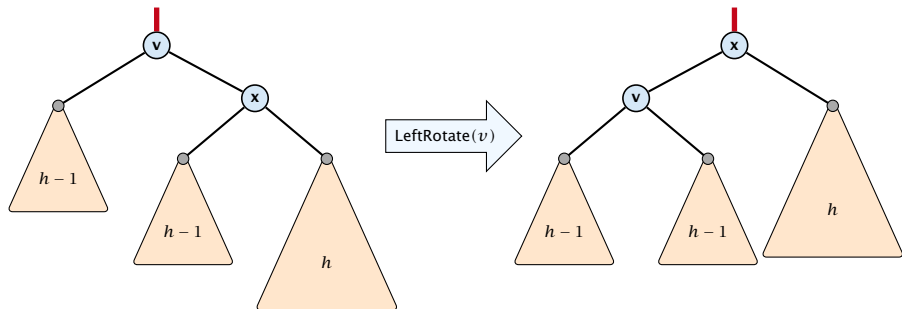
Case 1: $\text{balance}[\text{right}[v]] = -1$

We do a left rotation at v



Case 1: $\text{balance}[\text{right}[v]] = -1$

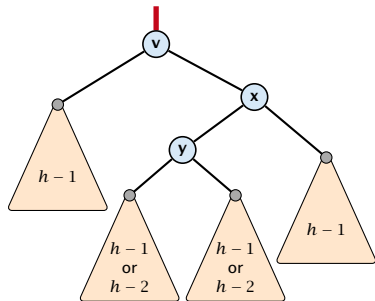
We do a left rotation at v



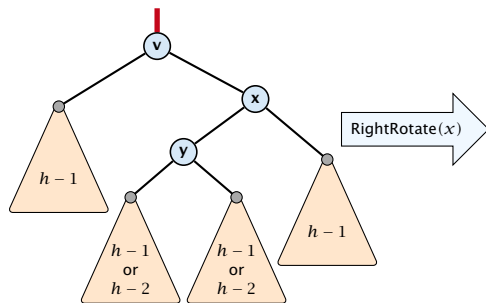
Now, the subtree has height $h + 1$ as before the insertion.
Hence, we do not need to continue.

Case 2: $\text{balance}[\text{right}[v]] = 1$

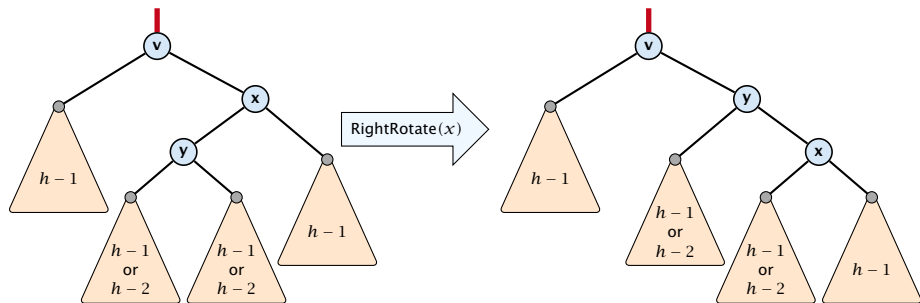
Case 2: $\text{balance}[\text{right}[\nu]] = 1$



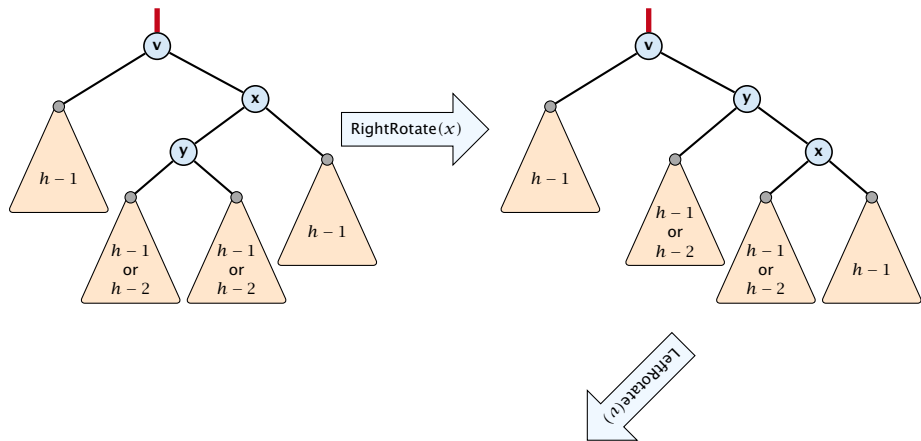
Case 2: $\text{balance}[\text{right}[v]] = 1$



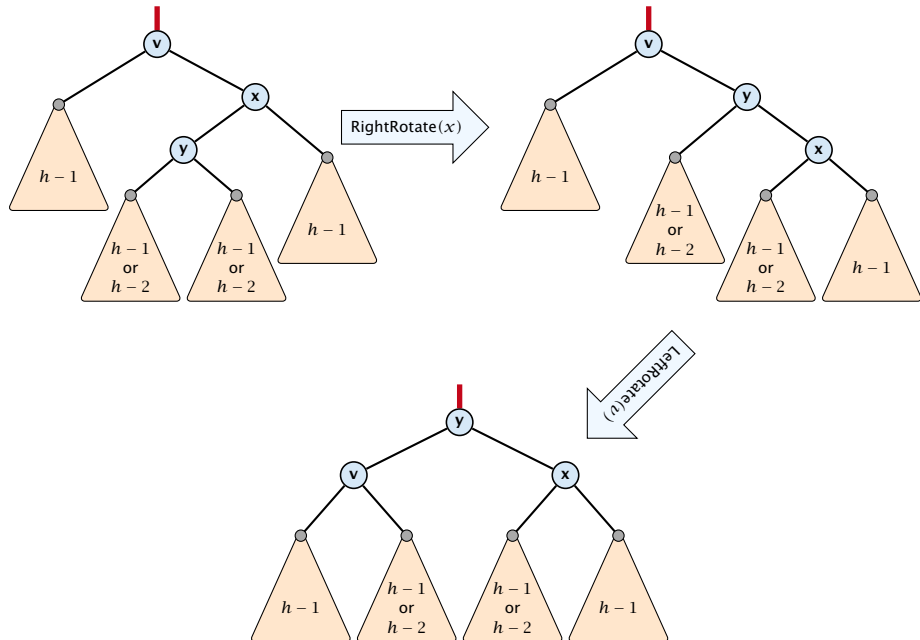
Case 2: $\text{balance}[\text{right}[v]] = 1$



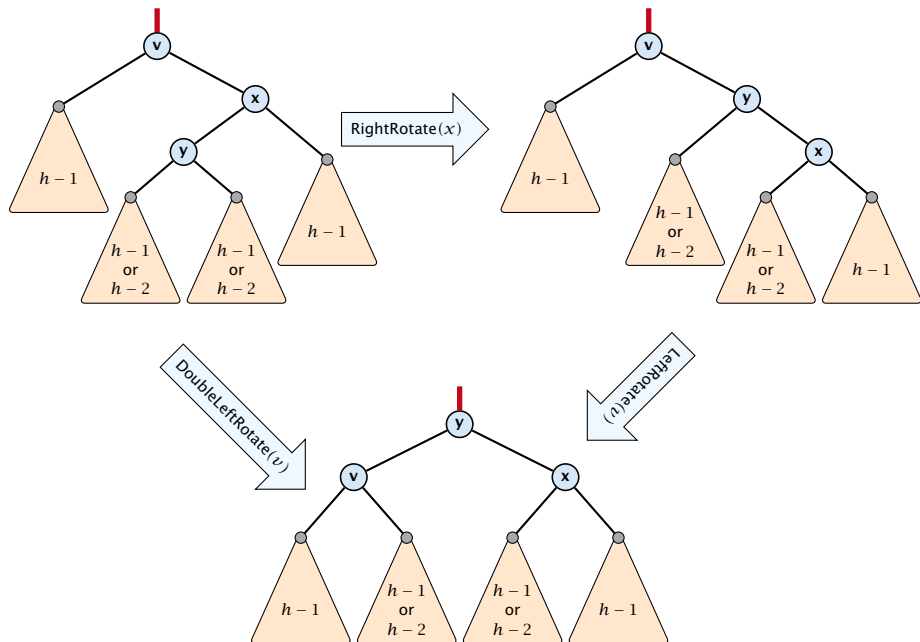
Case 2: $\text{balance}[\text{right}[v]] = 1$



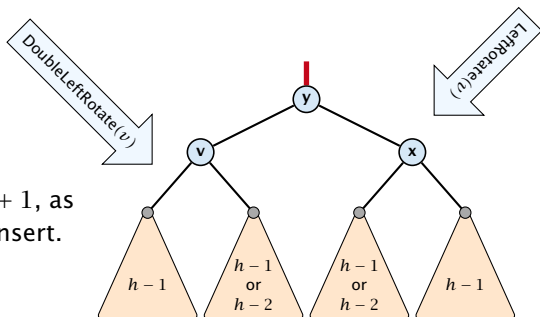
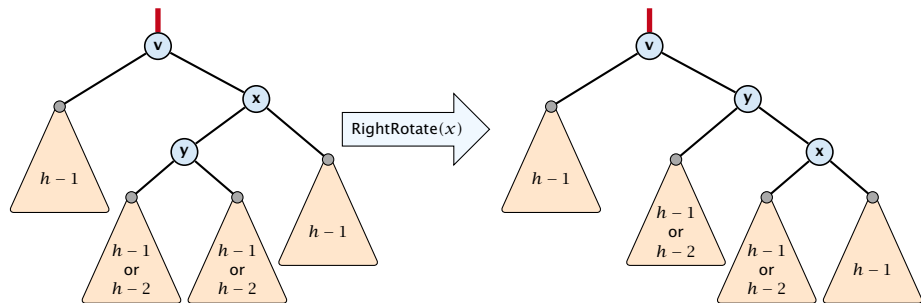
Case 2: $\text{balance}[\text{right}[v]] = 1$



Case 2: $\text{balance}[\text{right}[v]] = 1$



Case 2: $\text{balance}[\text{right}[v]] = 1$



Height is $h + 1$, as before the insert.

AVL-trees: Delete

- ▶ Delete like in a binary search tree.
- ▶ Let v denote the parent of the node that has been spliced out.
- ▶ The balance-constraint may be violated at v , or at ancestors of v , as a sub-tree of a child of v has reduced its height.
- ▶ Initially, the node c —the new root in the sub-tree that has changed—is either a dummy leaf or a node with two dummy leaves as children.



Case 1



Case 2

In both cases $\text{bal}[c] = 0$.

- ▶ Call $\text{AVL-fix-up-delete}(v)$ to restore the balance-condition.

AVL-trees: Delete

- ▶ Delete like in a binary search tree.
- ▶ Let v denote the parent of the node that has been **spliced out**.
- ▶ The balance-constraint may be violated at v , or at ancestors of v , as a sub-tree of a child of v has reduced its height.
- ▶ Initially, the node c —the new root in the sub-tree that has changed—is either a dummy leaf or a node with two dummy leaves as children.



Case 1



Case 2

In both cases $\text{bal}[c] = 0$.

- ▶ Call $\text{AVL-fix-up-delete}(v)$ to restore the balance-condition.

AVL-trees: Delete

- ▶ Delete like in a binary search tree.
- ▶ Let v denote the parent of the node that has been **spliced out**.
- ▶ The balance-constraint may be violated at v , or at ancestors of v , as a sub-tree of a child of v has reduced its height.
- ▶ Initially, the node c —the new root in the sub-tree that has changed—is either a dummy leaf or a node with two dummy leaves as children.



Case 1



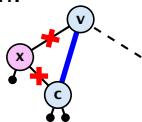
Case 2

In both cases $\text{bal}[c] = 0$.

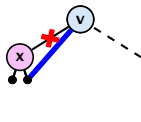
- ▶ Call $\text{AVL-fix-up-delete}(v)$ to restore the balance-condition.

AVL-trees: Delete

- ▶ Delete like in a binary search tree.
- ▶ Let v denote the parent of the node that has been **spliced out**.
- ▶ The balance-constraint may be violated at v , or at ancestors of v , as a sub-tree of a child of v has reduced its height.
- ▶ Initially, the node c —the new root in the sub-tree that has changed—is either a dummy leaf or a node with two dummy leaves as children.



Case 1



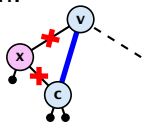
Case 2

In both cases $\text{bal}[c] = 0$.

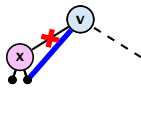
- ▶ Call $\text{AVL-fix-up-delete}(v)$ to restore the balance-condition.

AVL-trees: Delete

- ▶ Delete like in a binary search tree.
- ▶ Let v denote the parent of the node that has been **spliced out**.
- ▶ The balance-constraint may be violated at v , or at ancestors of v , as a sub-tree of a child of v has reduced its height.
- ▶ Initially, the node c —the new root in the sub-tree that has changed—is either a dummy leaf or a node with two dummy leaves as children.



Case 1



Case 2

In both cases $\text{bal}[c] = 0$.

- ▶ Call $\text{AVL-fix-up-delete}(v)$ to restore the balance-condition.

Invariant at the beginning AVL-fix-up-delete(v):

1. The balance constraints holds at all descendants of v .
2. A node has been deleted from T_c , where c is either the right or left child of v .
3. T_c has decreased its height by one.
4. The balance at the node c fulfills $\text{balance}[c] = 0$. This holds because if the balance of c is in $\{-1, 1\}$, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

Invariant at the beginning AVL-fix-up-delete(v):

1. The balance constraints holds at all descendants of v .
2. A node has been deleted from T_c , where c is either the right or left child of v .
3. T_c has decreased its height by one.
4. The balance at the node c fulfills $\text{balance}[c] = 0$. This holds because if the balance of c is in $\{-1, 1\}$, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

Invariant at the beginning AVL-fix-up-delete(v):

1. The balance constraints holds at all descendants of v .
2. A node has been deleted from T_c , where c is either the right or left child of v .
3. T_c has decreased its height by one.
4. The balance at the node c fulfills $\text{balance}[c] = 0$. This holds because if the balance of c is in $\{-1, 1\}$, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

Invariant at the beginning AVL-fix-up-delete(v):

1. The balance constraints holds at all descendants of v .
2. A node has been deleted from T_c , where c is either the right or left child of v .
3. T_c has decreased its height by one.
4. The balance at the node c fulfills $\text{balance}[c] = 0$. This holds because if the balance of c is in $\{-1, 1\}$, then T_c did not change its height, and the whole procedure would have been aborted in the previous step.

AVL-trees: Delete

Algorithm 13 AVL-fix-up-delete(v)

- 1: **if** $\text{balance}[v] \in \{-2, 2\}$ **then** DoRotationDelete(v);
- 2: **if** $\text{balance}[v] \in \{-1, 1\}$ **return**;
- 3: AVL-fix-up-delete(parent[v]);

We will show that the above procedure is correct. However, for the case of a delete there may be a logarithmic number of rotations.

AVL-trees: Delete

Algorithm 13 AVL-fix-up-delete(v)

- 1: **if** $\text{balance}[v] \in \{-2, 2\}$ **then** DoRotationDelete(v);
- 2: **if** $\text{balance}[v] \in \{-1, 1\}$ **return**;
- 3: AVL-fix-up-delete(parent(v));

We will show that the above procedure is correct. However, for the case of a delete there may be a logarithmic number of rotations.

Algorithm 14 DoRotationDelete(v)

```
1: if balance[ $v$ ] = -2 then // deletion in left sub-tree
2:     if balance[right[ $v$ ]]  $\in$  {0, -1} then
3:         LeftRotate( $v$ );
4:     else
5:         DoubleLeftRotate( $v$ );
6: else // deletion in right sub-tree
7:     if balance[left[ $v$ ]] = {0, 1} then
8:         RightRotate( $v$ );
9:     else
10:        DoubleRightRotate( $v$ );
```

AVL-trees: Delete

It is clear that the invariant for the fix-up routine hold as long as no rotations have been done.

We show that after doing a rotation at v :

- ▶ v fulfills the balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ If now $\text{balance}[v] \in \{-1, 1\}$ we can stop as the height of T_v is the same as before the deletion.

We only look at the case where the deleted node was in the right sub-tree of v . The other case is symmetric.

AVL-trees: Delete

It is clear that the invariant for the fix-up routine hold as long as no rotations have been done.

We show that after doing a rotation at v :

- ▶ v fulfills the balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ If now $\text{balance}[v] \in \{-1, 1\}$ we can stop as the height of T_v is the same as before the deletion.

We only look at the case where the deleted node was in the right sub-tree of v . The other case is symmetric.

AVL-trees: Delete

It is clear that the invariant for the fix-up routine hold as long as no rotations have been done.

We show that after doing a rotation at v :

- ▶ v fulfills the balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ If now $\text{balance}[v] \in \{-1, 1\}$ we can stop as the height of T_v is the same as before the deletion.

We only look at the case where the deleted node was in the right sub-tree of v . The other case is symmetric.

AVL-trees: Delete

It is clear that the invariant for the fix-up routine hold as long as no rotations have been done.

We show that after doing a rotation at v :

- ▶ v fulfills the balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ If now $\text{balance}[v] \in \{-1, 1\}$ we can stop as the height of T_v is the same as before the deletion.

We only look at the case where the deleted node was in the right sub-tree of v . The other case is symmetric.

AVL-trees: Delete

It is clear that the invariant for the fix-up routine hold as long as no rotations have been done.

We show that after doing a rotation at v :

- ▶ v fulfills the balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ If now $\text{balance}[v] \in \{-1, 1\}$ we can stop as the height of T_v is the same as before the deletion.

We only look at the case where the deleted node was in the right sub-tree of v . The other case is symmetric.

AVL-trees: Delete

It is clear that the invariant for the fix-up routine hold as long as no rotations have been done.

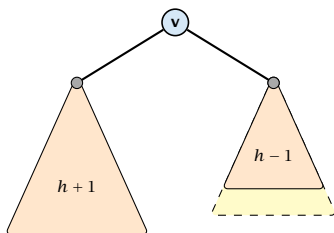
We show that after doing a rotation at v :

- ▶ v fulfills the balance condition.
- ▶ All children of v still fulfill the balance condition.
- ▶ If now $\text{balance}[v] \in \{-1, 1\}$ we can stop as the height of T_v is the same as before the deletion.

We only look at the case where the deleted node was in the right sub-tree of v . The other case is symmetric.

AVL-trees: Delete

We have the following situation:

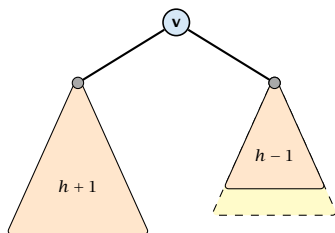


The right sub-tree of v has decreased its height which results in a balance of 2 at v .

Before the deletion the height of T_v was $h + 2$.

AVL-trees: Delete

We have the following situation:

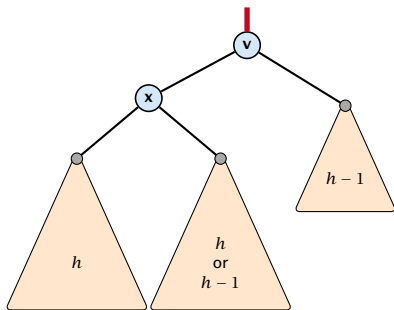


The right sub-tree of v has decreased its height which results in a balance of 2 at v .

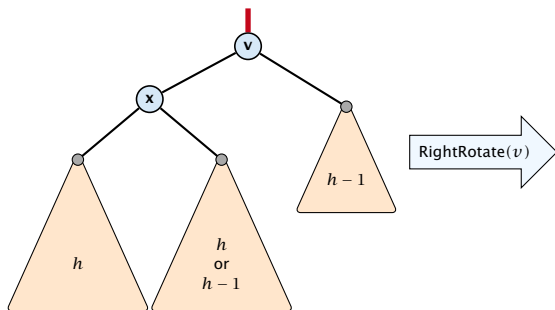
Before the deletion the height of T_v was $h + 2$.

Case 1: $\text{balance}[\text{left}[v]] \in \{0, 1\}$

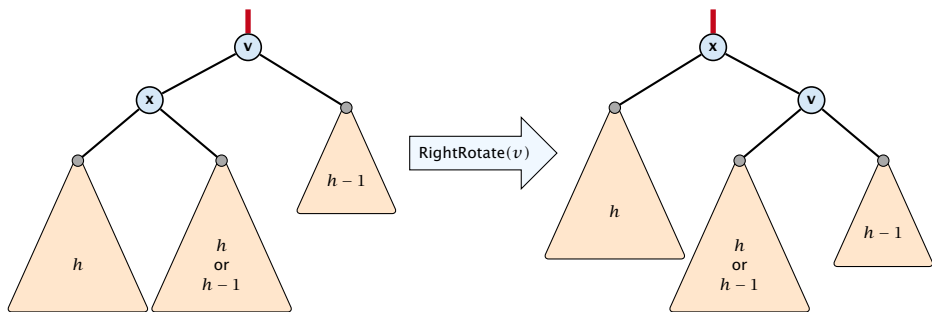
Case 1: $\text{balance}[\text{left}[v]] \in \{0, 1\}$



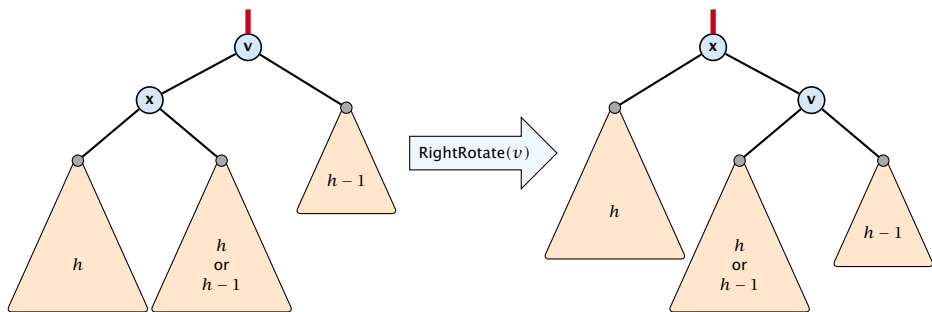
Case 1: $\text{balance}[\text{left}[v]] \in \{0, 1\}$



Case 1: $\text{balance}[\text{left}[v]] \in \{0, 1\}$

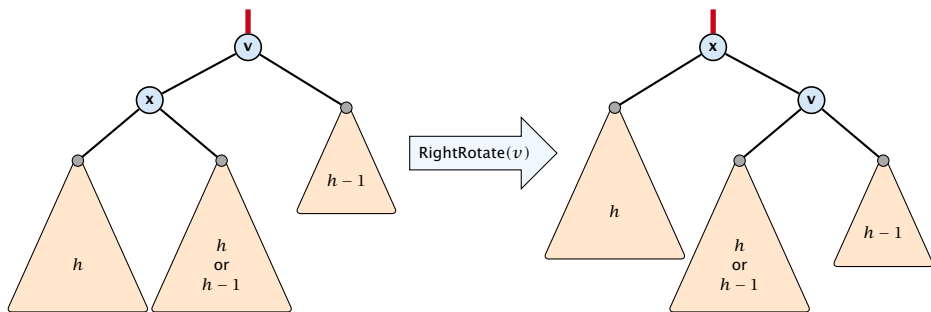


Case 1: $\text{balance}[\text{left}[v]] \in \{0, 1\}$



If the middle subtree has height h the whole tree has height $h + 2$ as before the deletion. The iteration stops as the balance at the root is non-zero.

Case 1: $\text{balance}[\text{left}[v]] \in \{0, 1\}$

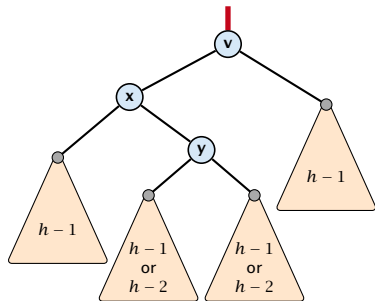


If the middle subtree has height h the whole tree has height $h + 2$ as before the deletion. The iteration stops as the balance at the root is non-zero.

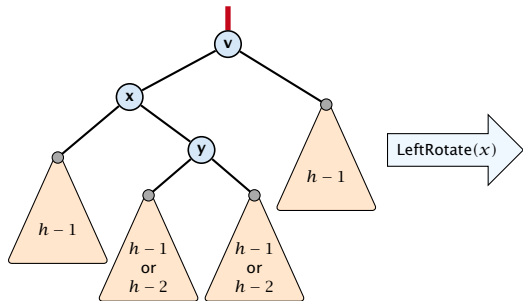
If the middle subtree has height $h - 1$ the whole tree has decreased its height from $h + 2$ to $h + 1$. We do continue the fix-up procedure as the balance at the root is zero.

Case 2: $\text{balance}[\text{left}[v]] = -1$

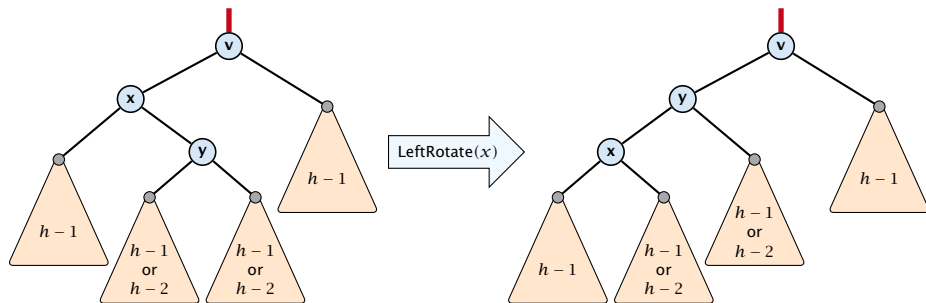
Case 2: $\text{balance}[\text{left}[v]] = -1$



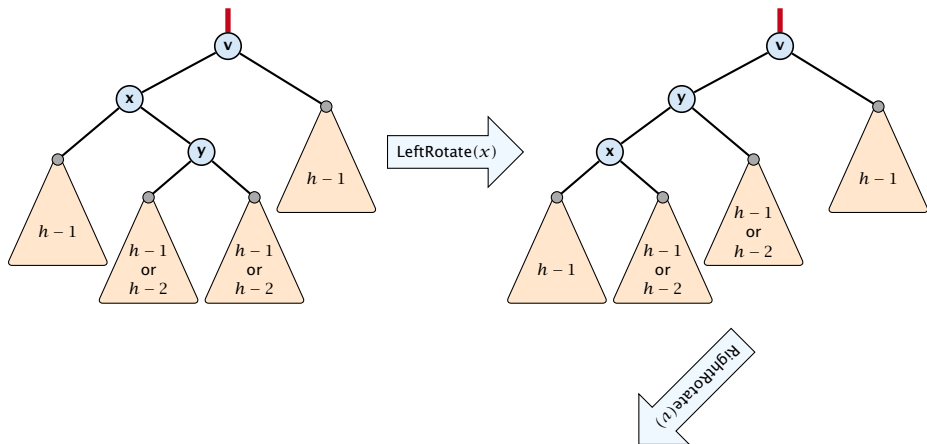
Case 2: $\text{balance}[\text{left}[v]] = -1$



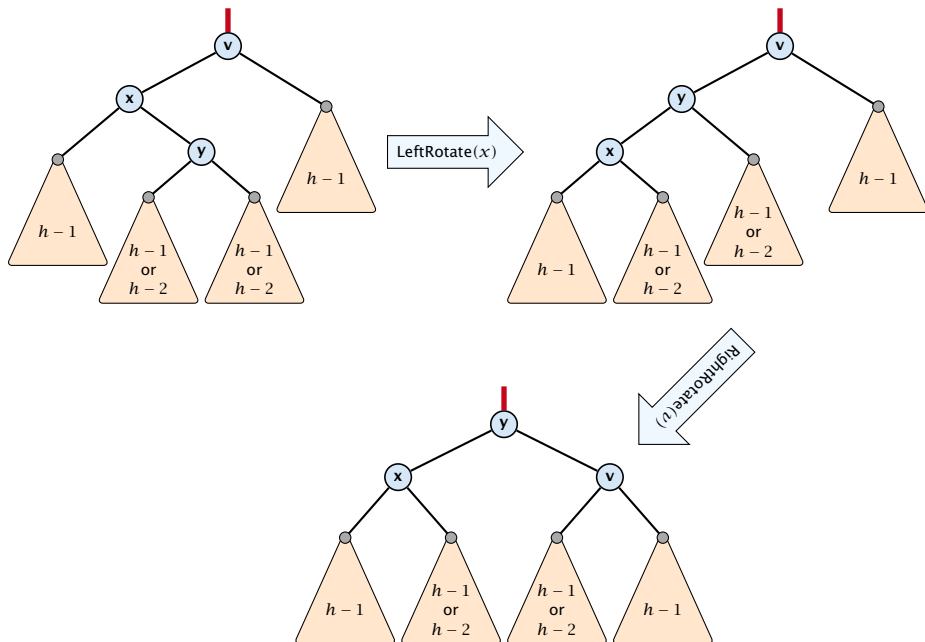
Case 2: $\text{balance}[\text{left}[v]] = -1$



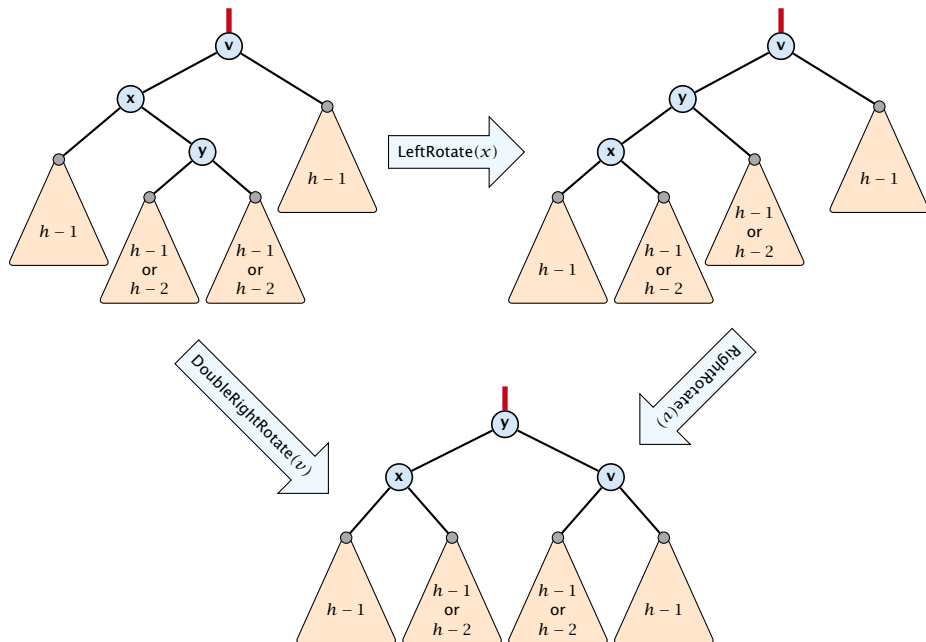
Case 2: $\text{balance}[\text{left}[v]] = -1$



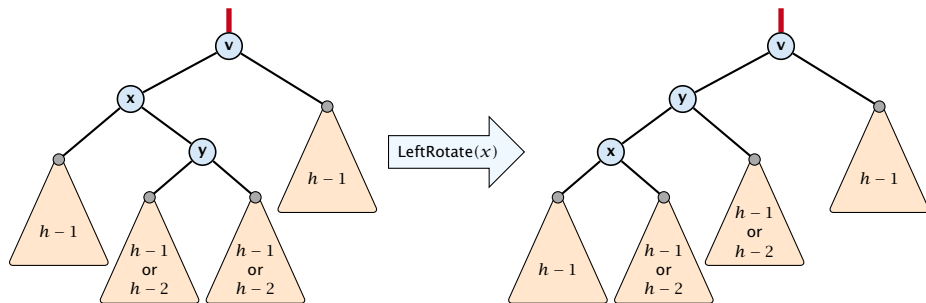
Case 2: $\text{balance}[\text{left}[v]] = -1$



Case 2: $\text{balance}[\text{left}[v]] = -1$



Case 2: $\text{balance}[\text{left}[v]] = -1$



Sub-tree has height $h + 1$, i.e., it has shrunk. The balance at y is zero. We continue the iteration.

