Contents

Section outline

BST critique

- BST operations are simple
- However, tree can get skewed or become degenerate in course of time
- Resulting tree is unbalanced
- All operations can take $O(n)$ on a degenerate tree
- Is it possible to ensure that the BST remains balanced?

BST critique

- BST operations are simple
- However, tree can get skewed or become degenerate in course of time
- Resulting tree is unbalanced
- All operations can take $O(n)$ on a degenerate tree
- Is it possible to ensure that the BST remains balanced?
- Solution was presented by G M Adelson-Velskii and E M Landis in 1962 – was called AVL tree
  - BST framework is still used
  - height is maintained in each node, dynamically
  - tree is restructured from time-to-time using rotations

Balanced binary trees

Definition (Balance factor)

The balance factor of a node is the height of its left subtree minus the height of its right subtree (sometimes opposite).

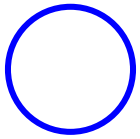
Definition (Balanced binary tree node)

A node of a binary tree with balance factor 1, 0, or -1 is considered balanced.

Definition (Height balanced binary tree)

An empty binary tree is balanced. A non-empty binary tree is balanced if its left and right subtrees are balanced and the root node is also balanced.

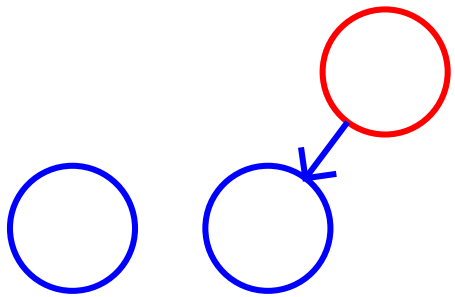
Some worst case HB trees



$h = 0$

$N(h) = 1$

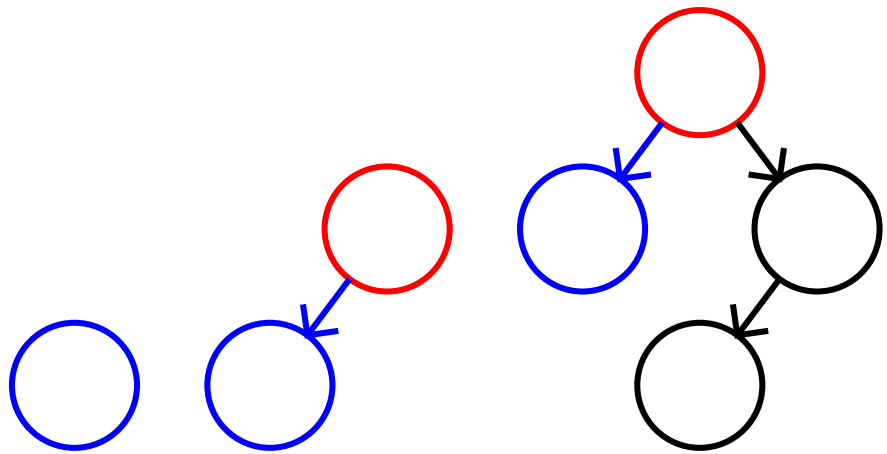
Some worst case HB trees



$h = 0$ $h = 1$

$N(h) = 1$ $N(h) = 2$

Some worst case HB trees

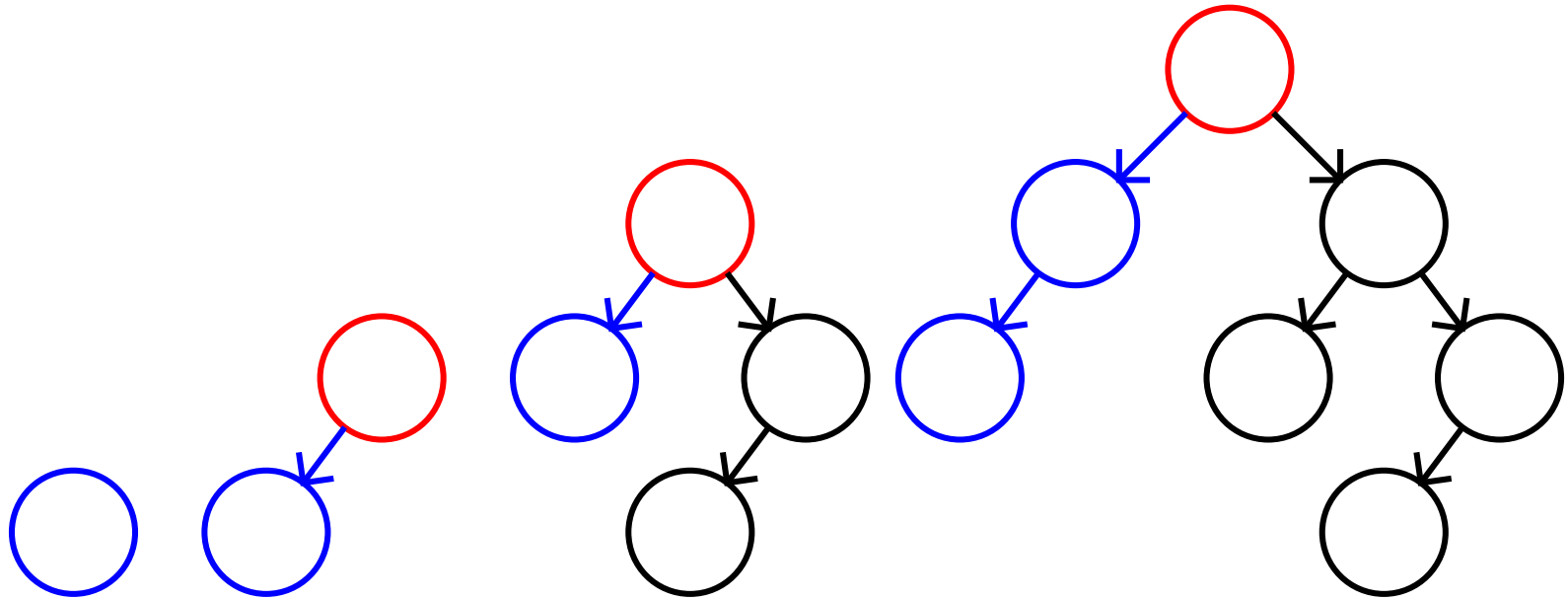


$h = 0$ $N(h) = 1$

$h = 1$ $N(h) = 2$

$h = 2$ $N(h) = 4$

Some worst case HB trees



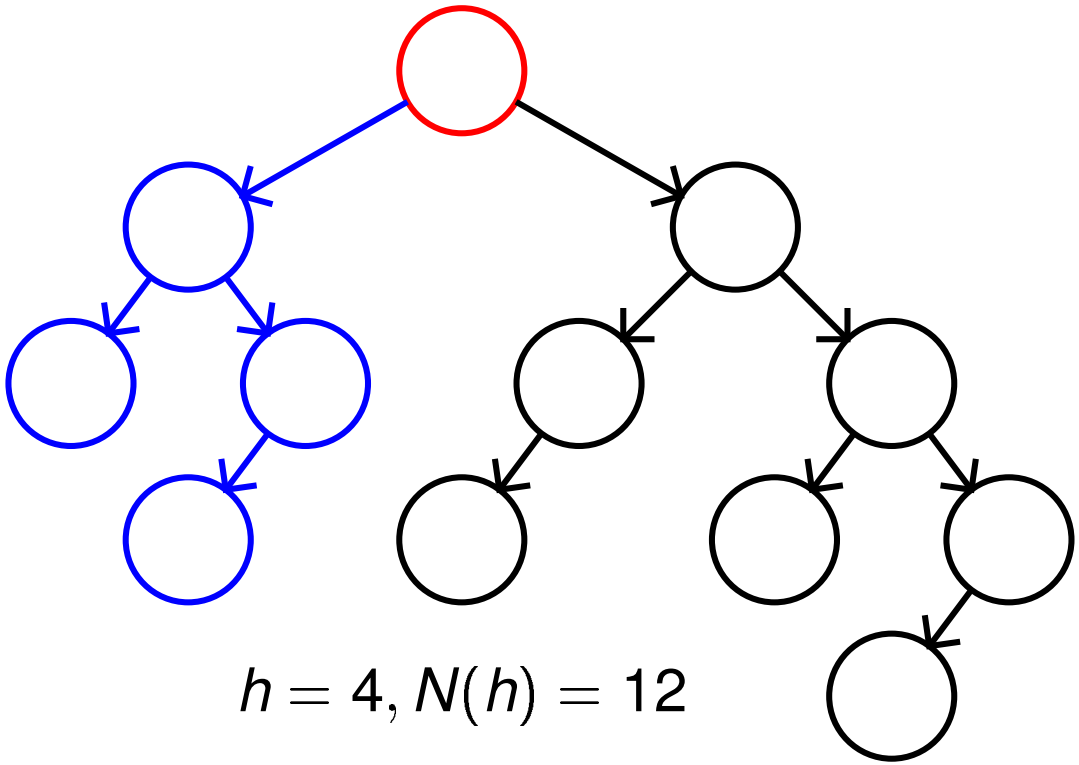
$h = 0$
$N(h) = 1$

$h = 1$
$N(h) = 2$

$h = 2$
$N(h) = 4$

$h = 3, N(h) = 7$

Some worst case HB trees (contd.)



$h = 4, N(h) = 12$

- Let $N(h)$ represent the minimum number of nodes in a balanced binary tree of height h
- $N(0) = 1, N(1) = 2, N(2) = 4$
- $N(k) = 1 + N(k-1) + N(k-2)$

Minimum number of nodes in a HB binary tree

- Adding 1 to both sides, $\overline{N(k)+1} = \overline{1+N(k-1)} + \overline{1+N(k-2)}$

Minimum number of nodes in a HB binary tree

- Adding 1 to both sides, $\overline{N(k)+1} = \overline{1+N(k-1)} + \overline{1+N(k-2)}$
- Let $f(k) = N(k)+1$

  Now, $f(h) = f(h-1) + f(h-2)$ and

  $f(0) = N(0)+1 = 2 = F(3)$, $f(1) = N(1)+1 = 3 = F(4)$,

  $f(2) = N(2)+1 = 5 = F(5)$

  Thus, $f(h) = F(h+3), h \geq 0$

$\therefore$ $N(h) = F(h+3) - 1, h \geq 0 = \frac{\phi^{h+3} - \psi^{h+3}}{\sqrt{5}} - 1$, $\phi = \frac{1+\sqrt{5}}{2}$, $\psi = \frac{1-\sqrt{5}}{2}$

$\therefore$ $N(h) \geq \phi^h \Leftrightarrow h \leq \log_\phi n$

- $N(h)$ grows exponentially in h,

Minimum number of nodes in a HB binary tree

- Adding 1 to both sides, $\overline{N(k)+1} = \overline{1+N(k-1)} + \overline{1+N(k-2)}$
- Let $f(k) = N(k)+1$

  Now, $f(h) = f(h-1) + f(h-2)$ and

  $f(0) = N(0)+1 = 2 = F(3)$, $f(1) = N(1)+1 = 3 = F(4)$,

  $f(2) = N(2)+1 = 5 = F(5)$

  Thus, $f(h) = F(h+3), h \geq 0$

$\therefore$ $N(h) = F(h+3) - 1, h \geq 0 = \frac{\phi^{h+3} - \psi^{h+3}}{\sqrt{5}} - 1$, $\phi = \frac{1+\sqrt{5}}{2}$, $\psi = \frac{1-\sqrt{5}}{2}$

$\therefore$ $N(h) \geq \phi^h \Leftrightarrow h \leq \log_\phi n$

- $N(h)$ grows exponentially in h, searching possible in $O(\lg n)$ time

Minimum number of nodes in a HB binary tree

- Adding 1 to both sides, $\overline{N(k)+1} = \overline{1+N(k-1)} + \overline{1+N(k-2)}$
- Let $f(k) = N(k)+1$
  Now, $f(h) = f(h-1) + f(h-2)$ and
  $f(0) = N(0)+1 = 2 = F(3)$, $f(1) = N(1)+1 = 3 = F(4)$,
  $f(2) = N(2)+1 = 5 = F(5)$
  Thus, $f(h) = F(h+3), h \geq 0$

$\therefore$ $N(h) = F(h+3) - 1, h \geq 0 = \frac{\phi^{h+3} - \psi^{h+3}}{\sqrt{5}} - 1$, $\phi = \frac{1+\sqrt{5}}{2}$, $\psi = \frac{1-\sqrt{5}}{2}$

$\therefore$ $N(h) \geq \phi^h \Leftrightarrow h \leq \log_\phi n$

- $N(h)$ grows exponentially in h, searching possible in $O(\lg n)$ time

Golden ratio

Lengths a and b are in golden ratio if $\frac{a+b}{a} = \frac{a}{b} \equiv 1 + \frac{b}{a} = \frac{a}{b} \equiv 1 + \frac{1}{x} = x$

Thus, $x^2 - x - 1 = 0$, $x = \frac{1 \pm \sqrt{5}}{2}$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618034$ (golden ratio),

$\psi = \frac{1-\sqrt{5}}{2} \approx -0.618034$

AVL trees – search, insert and delete

Definition (AVL trees)

AVL trees are height balanced binary search trees, with permissible balance factor in $[-1, 1]$

AVL trees – search, insert and delete

Definition (AVL trees)

AVL trees are height balanced binary search trees, with permissible balance factor in $[-1, 1]$

Search

- Same as BST search – $\Theta(\lg N)$ complexity

AVL trees – search, insert and delete

Definition (AVL trees)

AVL trees are height balanced binary search trees, with permissible balance factor in $[-1, 1]$

Search
- Same as BST search – $\Theta(\lg N)$ complexity

Insert
- First as BST insertion
- Restructure tree by rotation if some nodes become unbalanced

AVL trees – search, insert and delete

Definition (AVL trees)

AVL trees are height balanced binary search trees, with permissible balance factor in $[-1, 1]$

Search
- Same as BST search – $\Theta(\lg N)$ complexity

Insert
- First as BST insertion
- Restructure tree by rotation if some nodes become unbalanced

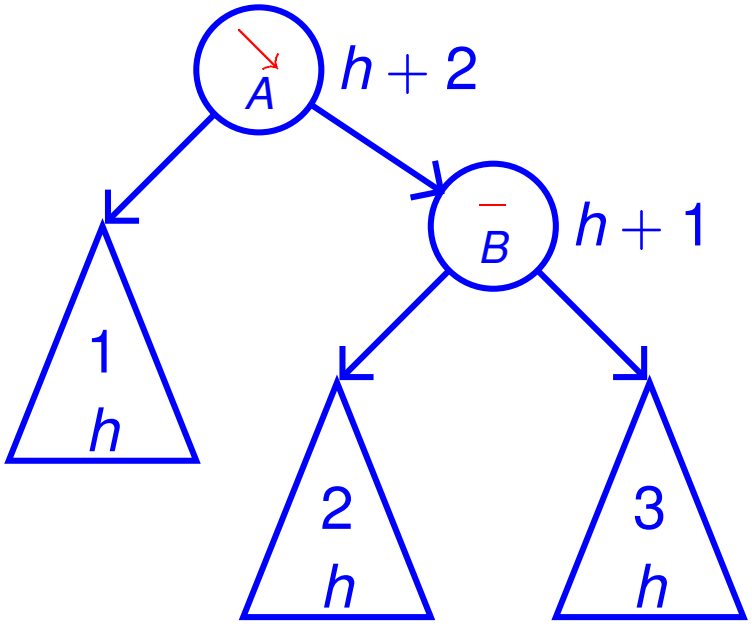
Delete
- First as BST deletion
- Restructure tree by rotation if some nodes become unbalanced

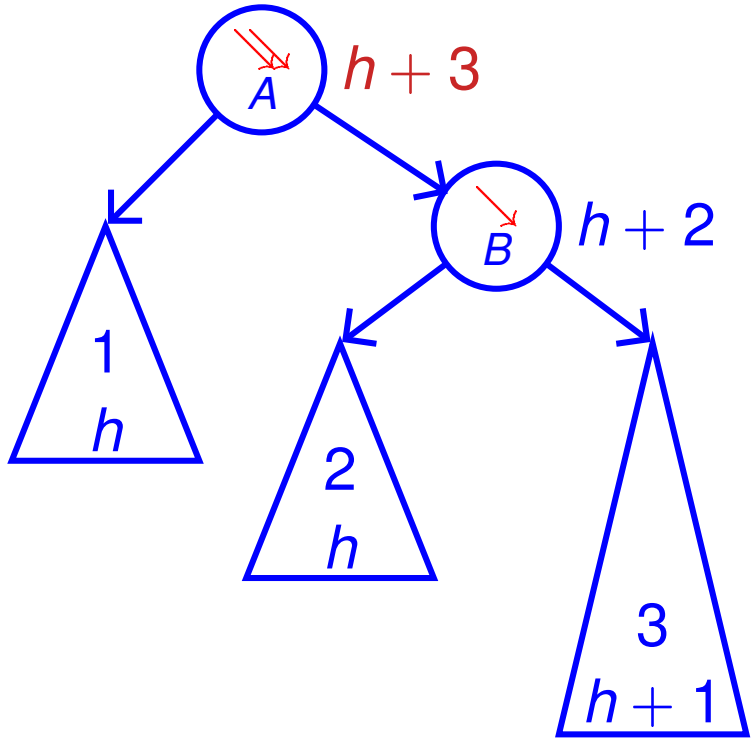
```
typedef int keyTyp;

typedef struct
avlTTag {
  keyTyp key;
  int height;
  struct avlTTag
    *lChild, *rChild;
} avlTTyp, *avlTPtr;
```

Insertion in AVL trees

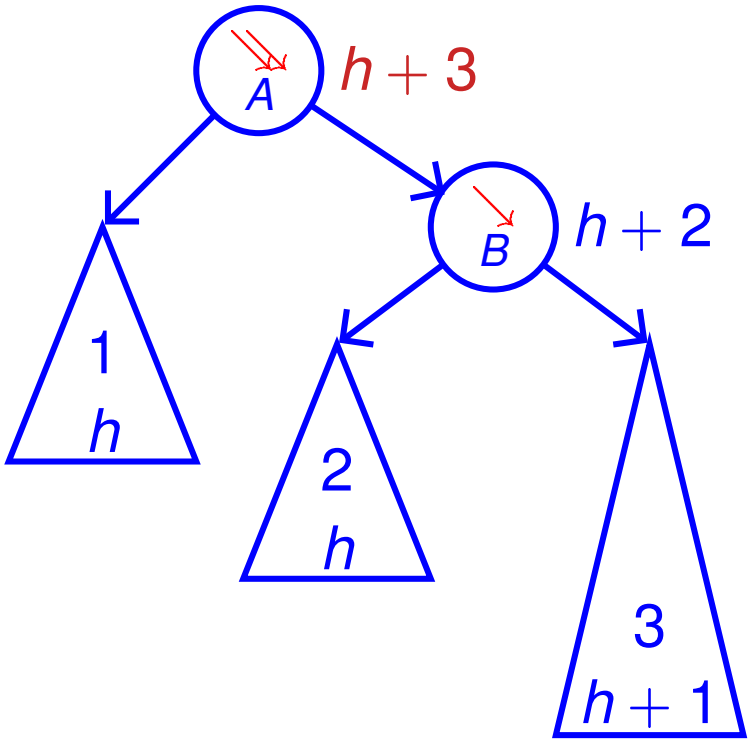
Tree becomes unbalanced after insertion into the right sub-tree of B





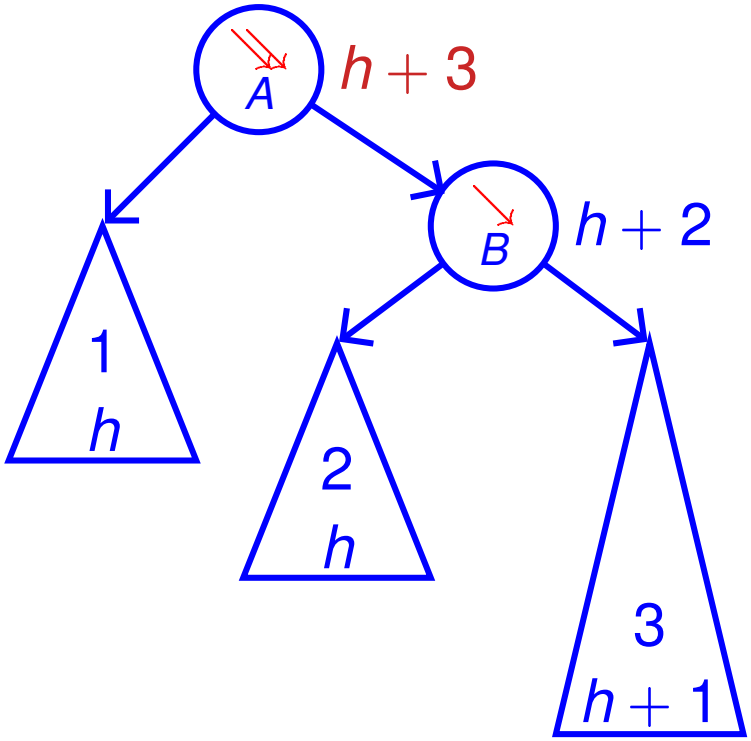
Correction by single rotation after insertion

- Right sub-tree of B gains height over its left sub-tree



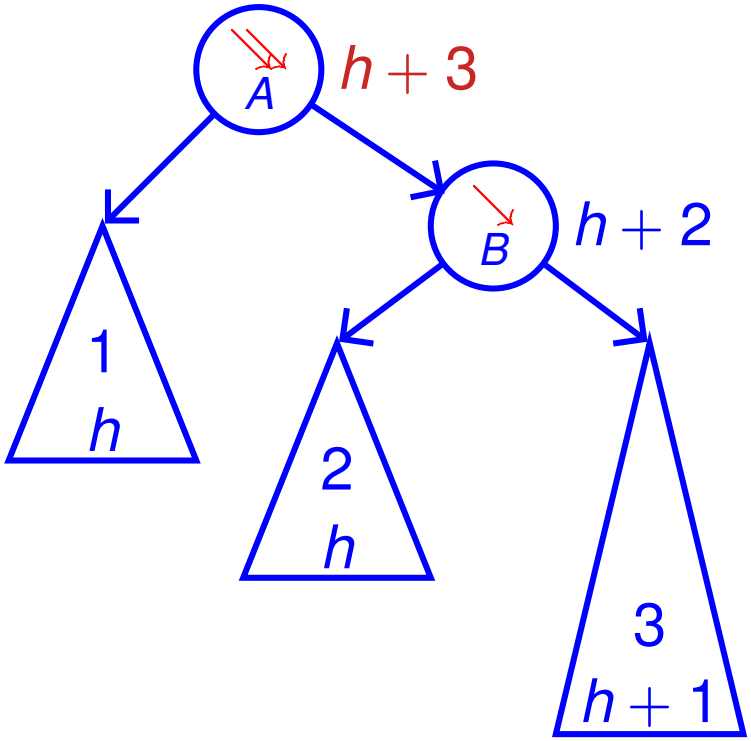
Correction by single rotation after insertion

- Right sub-tree of *B* gains height over its left sub-tree
- Balance disturbed at *A*, while retracing path back via *R-ST*



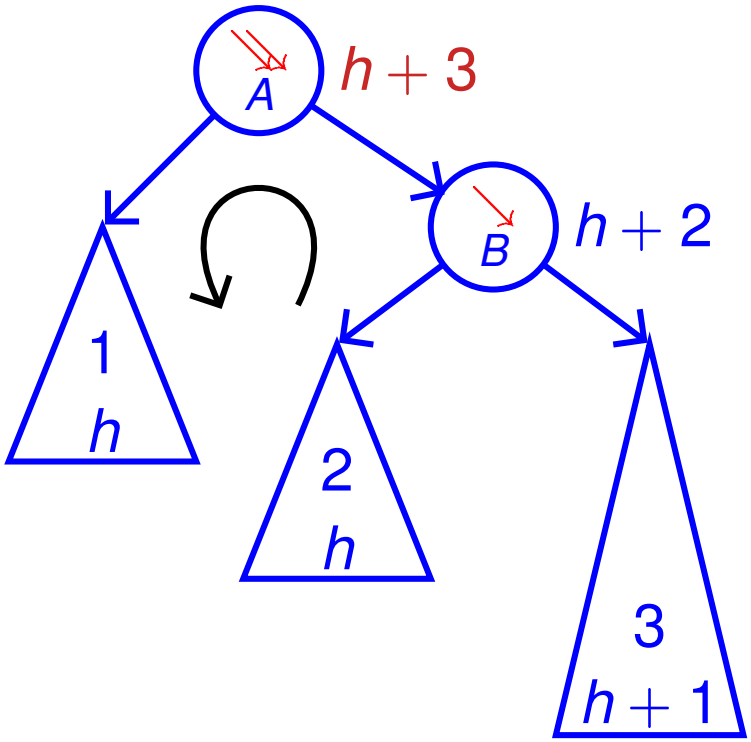
Correction by single rotation after insertion

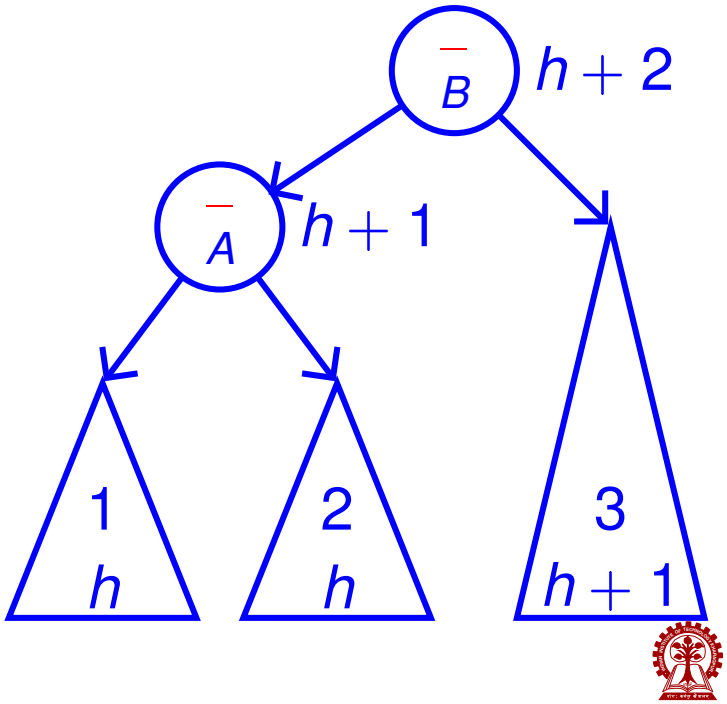
- Right sub-tree of B gains height over its left sub-tree
- Balance disturbed at A, while retracing path back via R-ST
- Visit B to find height(R-ST of B)>height(L-ST of B)



Correction by single rotation after insertion

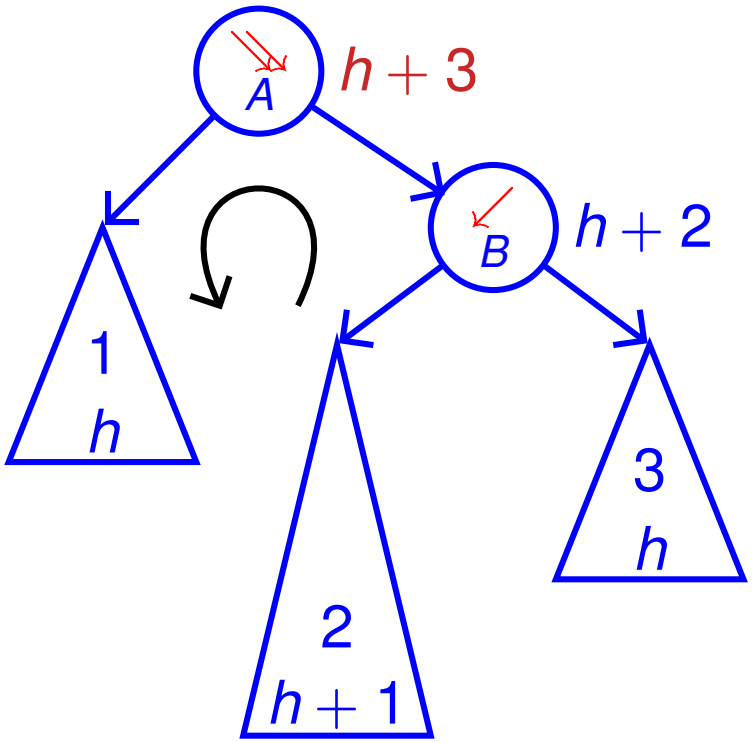
- Right sub-tree of B gains height over its left sub-tree
- Balance disturbed at A, while retracing path back via *R-ST*
- Visit B to find height(*R-ST* of B)>height(*L-ST* of B)
- Correct by applying single (left) rotation

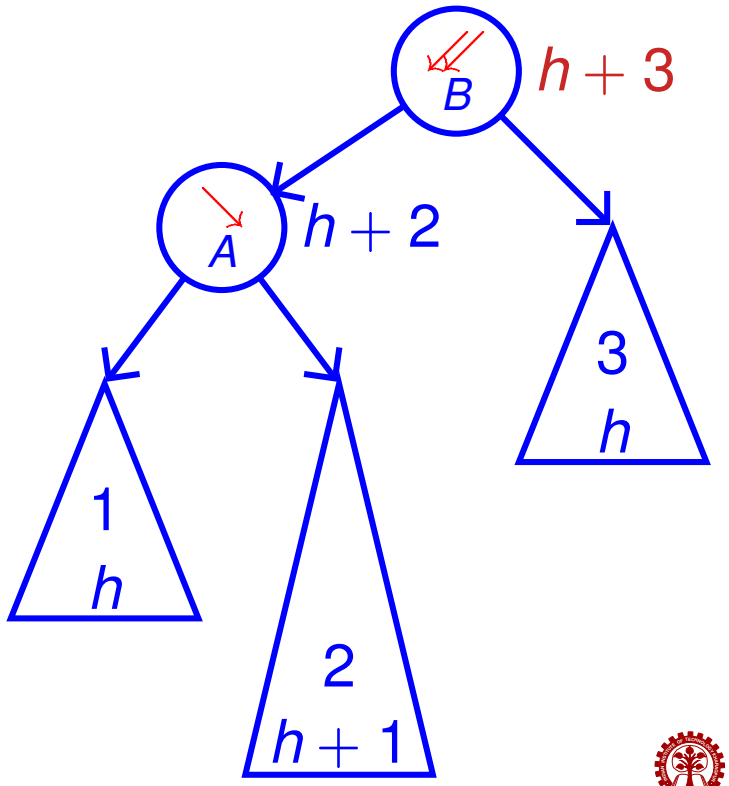




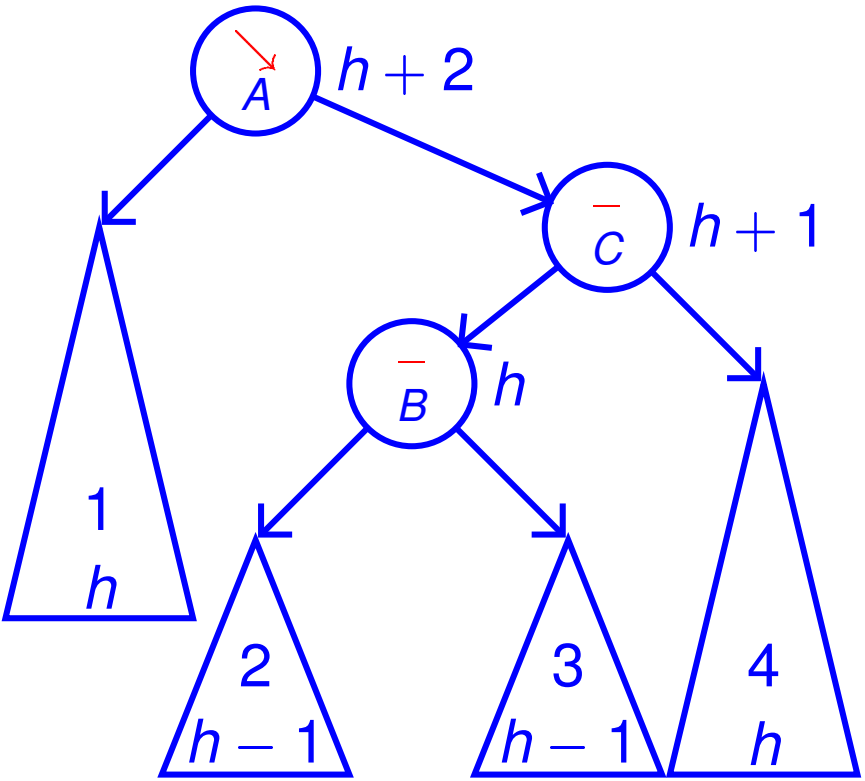
Failure to correct by single rotation after insertion

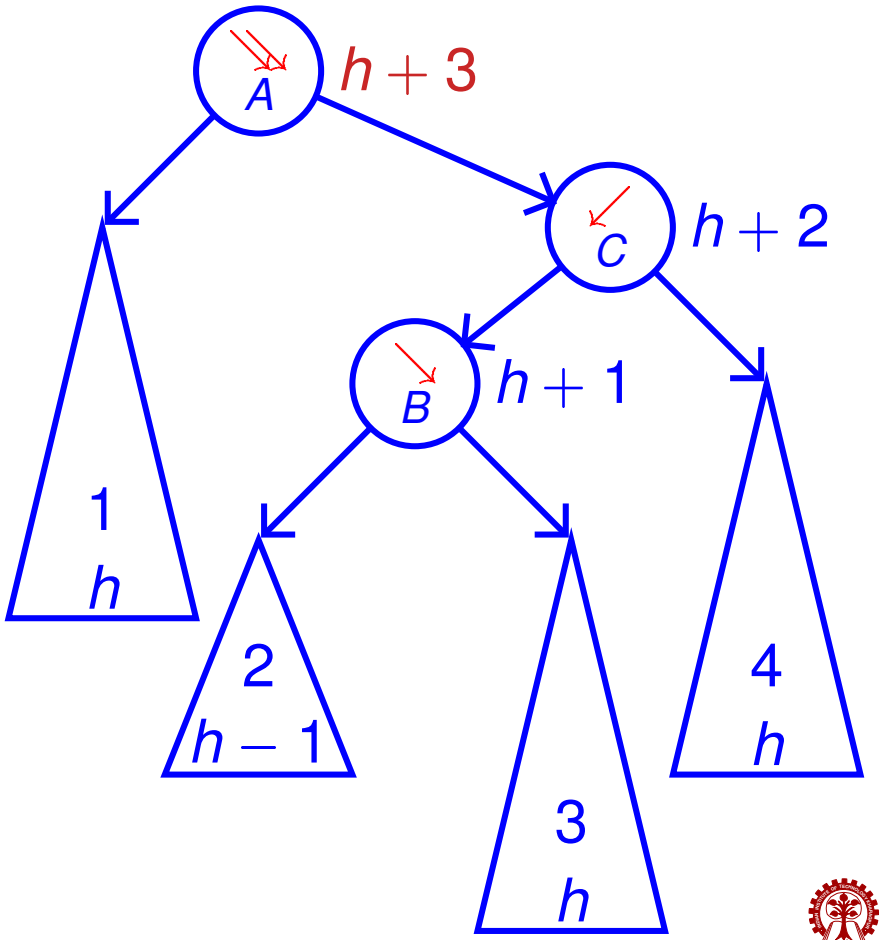
- Left sub-tree of B gains height over its left sub-tree
- Application of single (left) rotation fails to restore balance



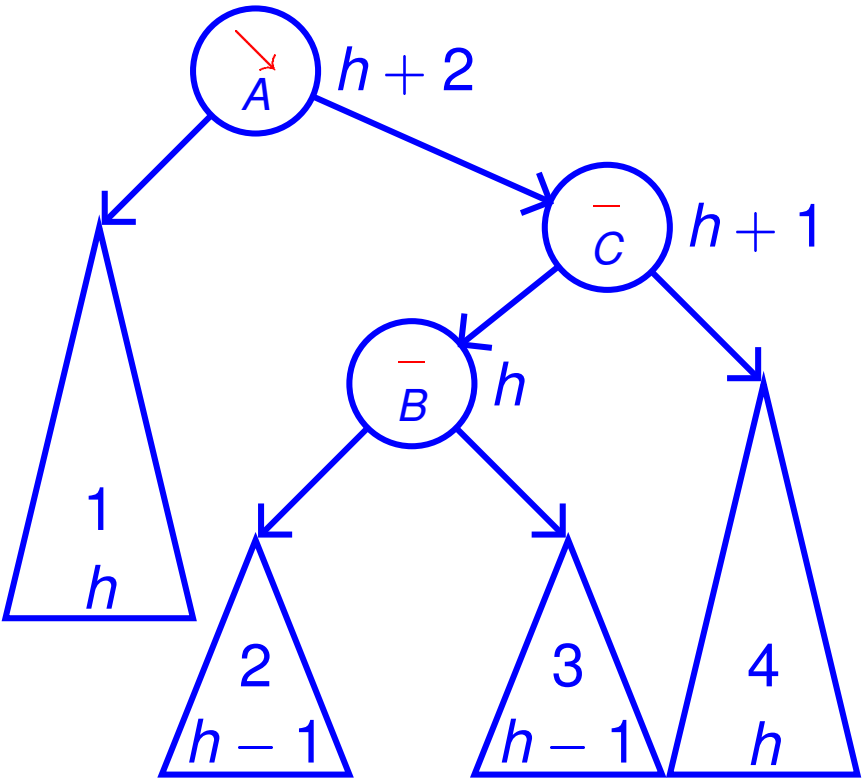


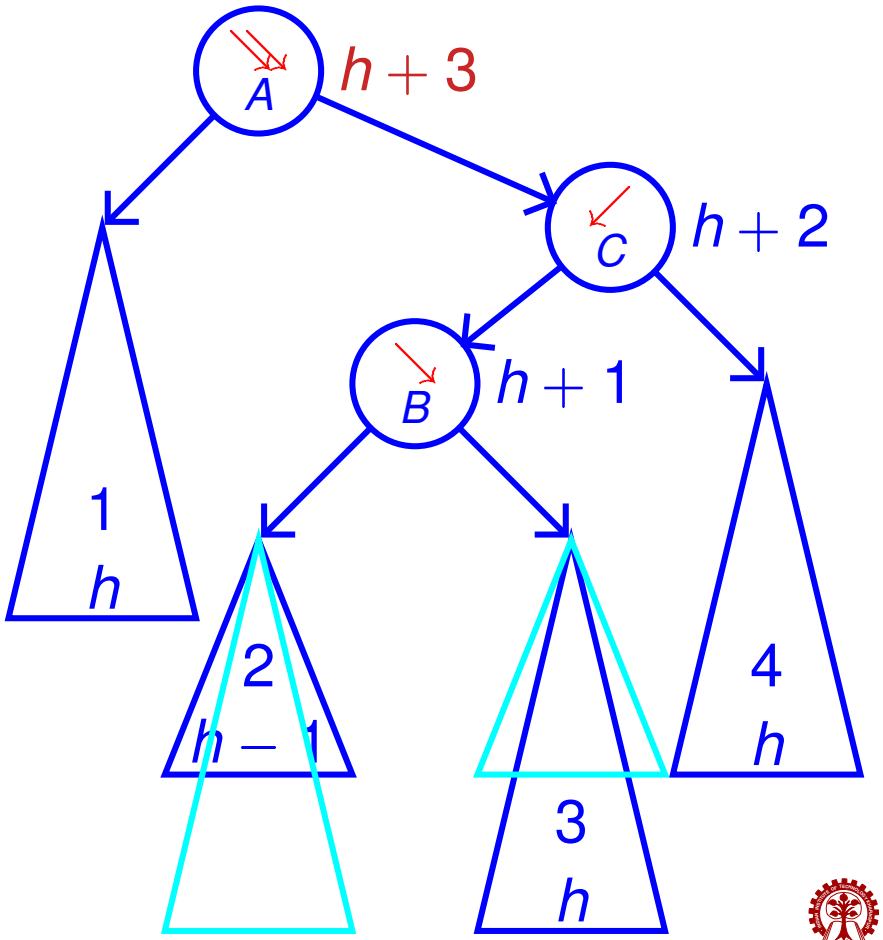
L-ST of R-child gaining height





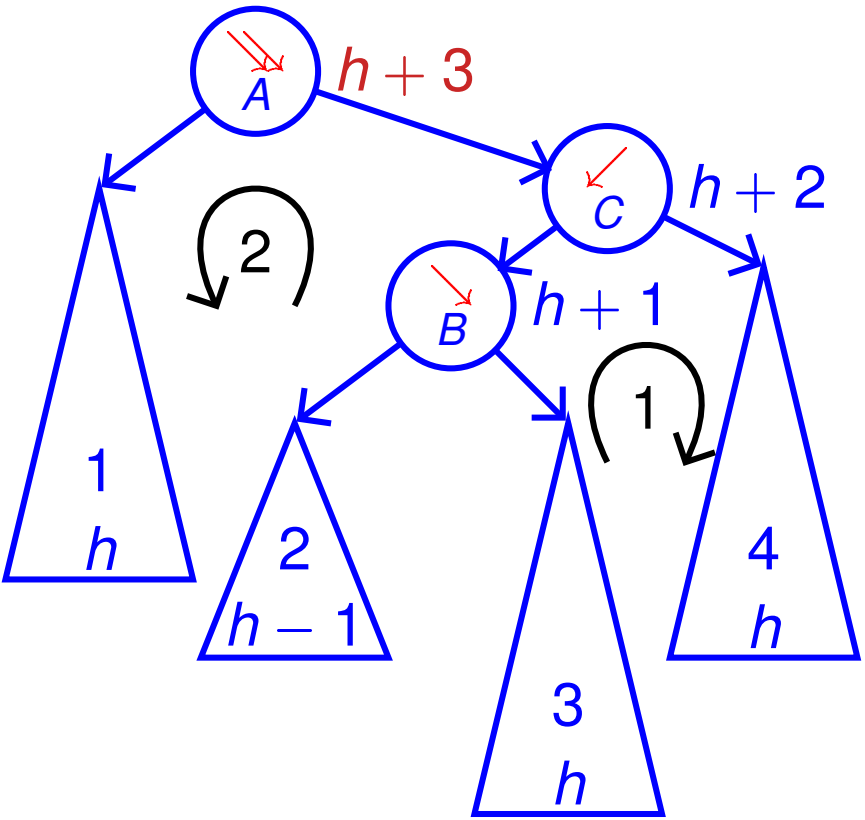
L-ST of R-child gaining height

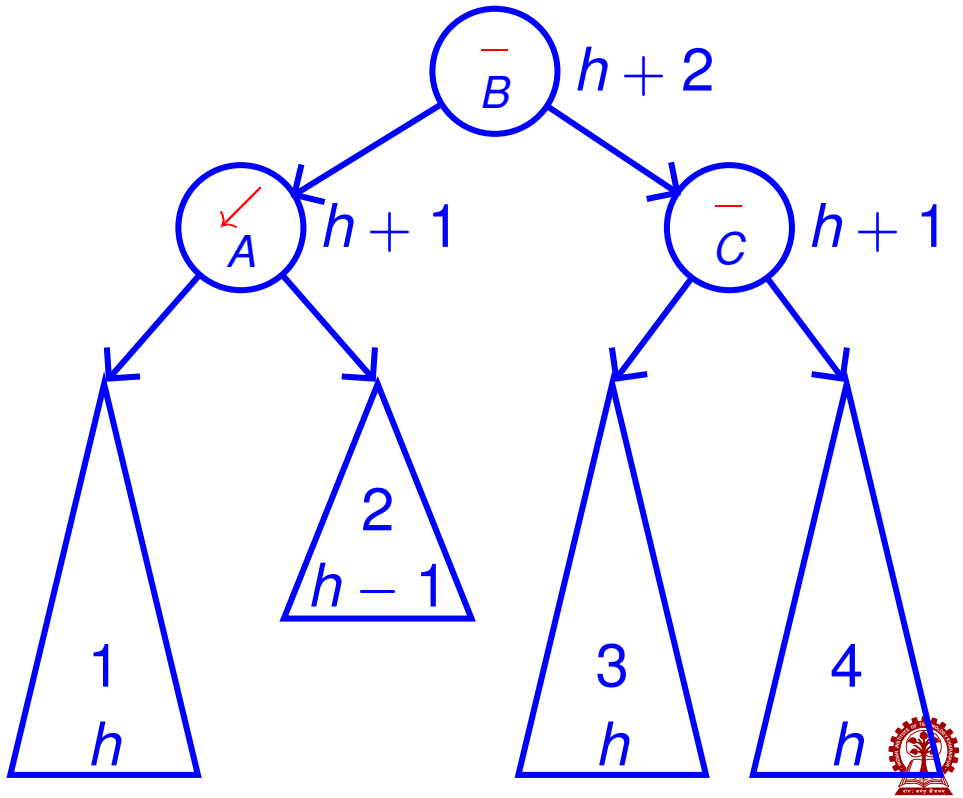




Correction by double rotation after insertion

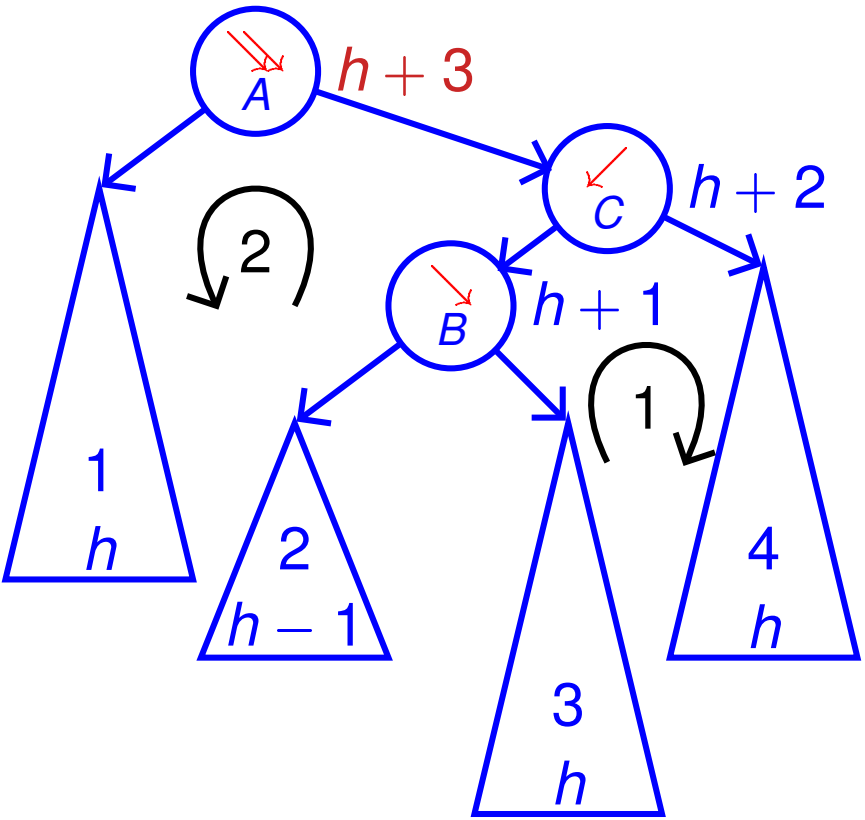
- Balance disturbed at A, while retracing path back to root

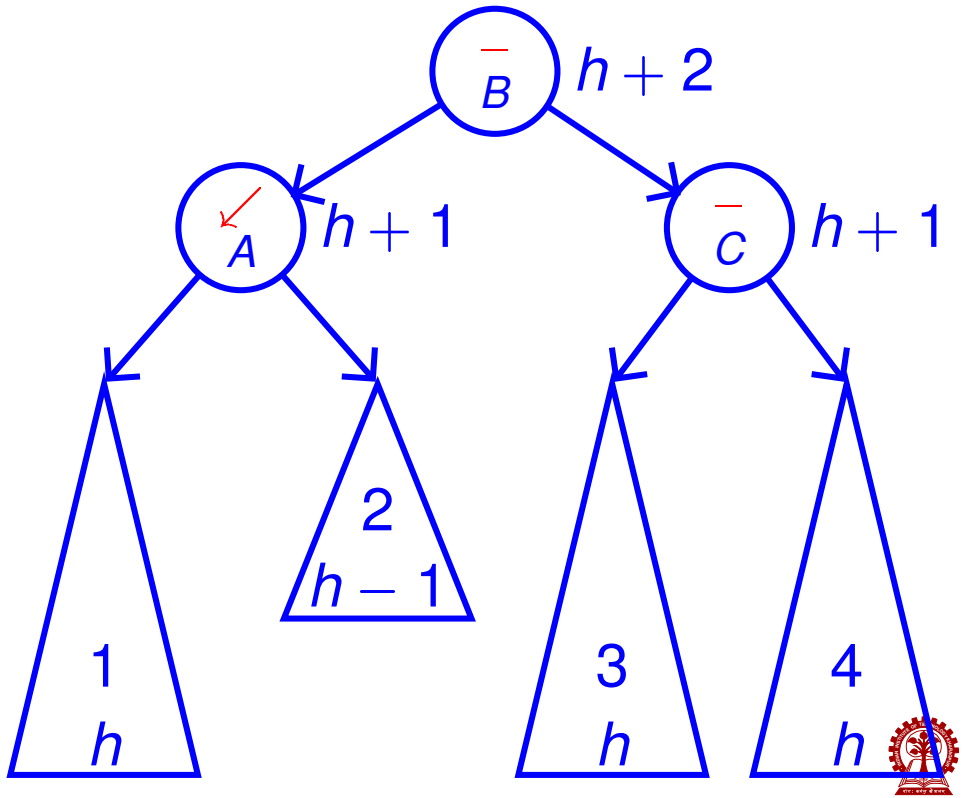




Correction by double rotation after insertion

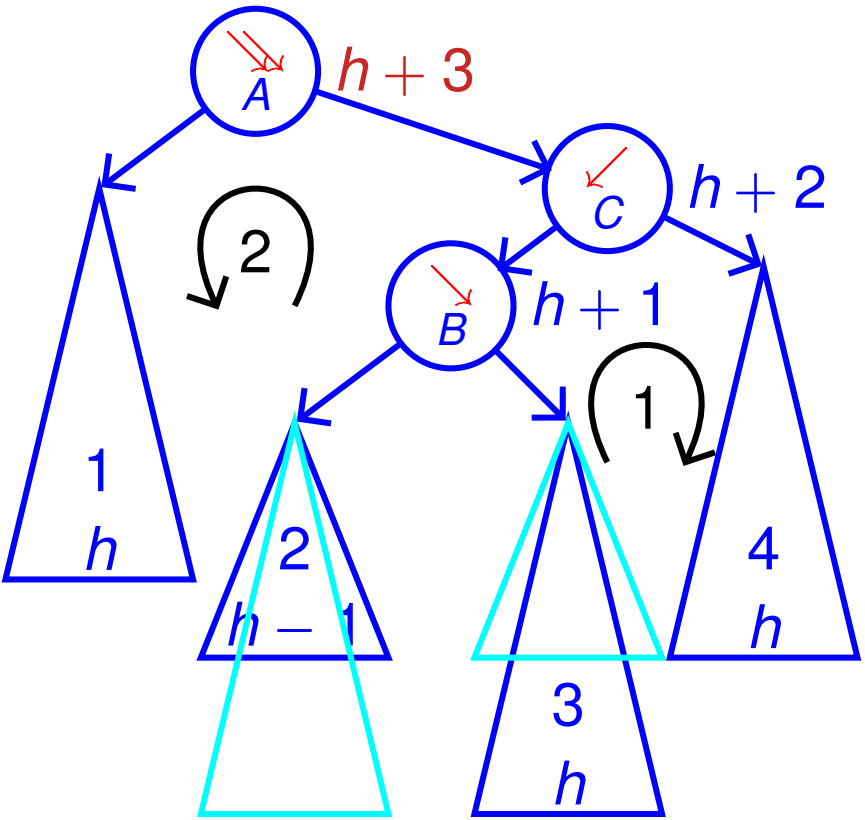
- Balance disturbed at A, while retracing path back to root
- Revisit C to find height(L-ST (B) of C)>height(R-ST of C)
- Correct by applying double (right-left) rotation

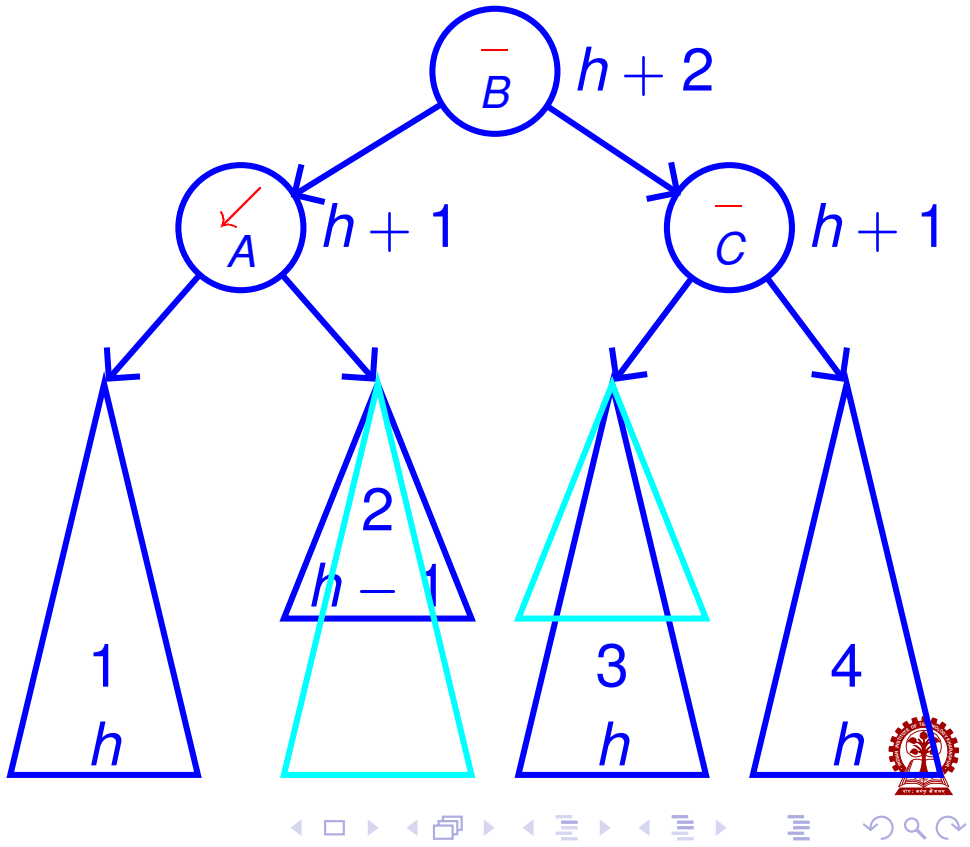




Correction by double rotation after insertion

- Balance disturbed at A, while retracing path back to root
- Revisit C to find height(L-ST (B) of C)>height(R-ST of C)
- Correct by applying double (right-left) rotation
- Same rotation works, irrespective of the balance of B





Other cases of balance restoration

- Balance disturbed at A, while retracing path back via L-ST rooted at B
- Revisit B to find height(L-ST of B)>height(R-ST of B)
- Correct by applying single (right) rotation

Other cases of balance restoration

- Balance disturbed at A, while retracing path back via L-ST rooted at B
- Revisit B to find height(L-ST of B)>height(R-ST of B)
- Correct by applying single (right) rotation

- Balance disturbed at A, while retracing path back via L-ST rooted at C
- Revisit C to find height(R-ST (B) of C)>height(L-ST of C)
- Correct by applying double (left) rotation
- Same rotation works, irrespective of the balance of B

Complexities of operations

Search Needs traversing to a leaf node – $\Theta(\lg N)$

Insert

- Needs traversing to a leaf node
- Correction by rotations up to two levels
- $\Theta(\lg N)$ worst case complexity

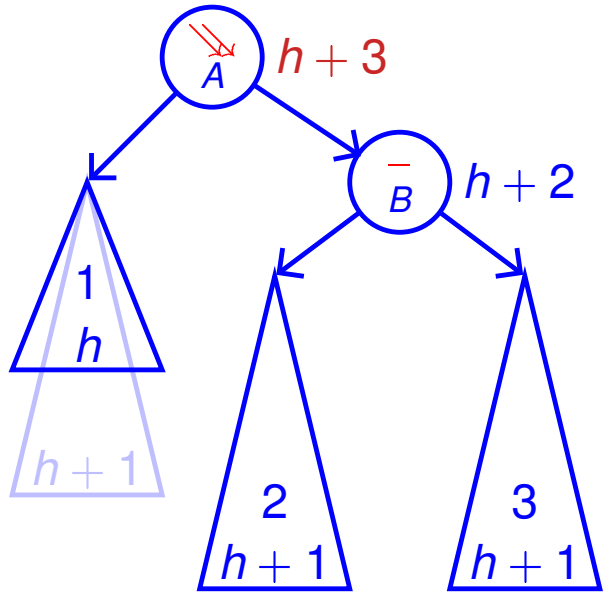
Deletion in AVL trees

- As normal binary search tree deletion
- Final node deleted will be either a leaf or have just one subtree (what about nodes with two (non-empty) subtrees?)
- If the deleted node has one subtree, then that subtree contains only one node (why?)
- Retrace back from the deleted node checking the balance of each node, balance the node by rotation if unbalanced

Deletion in AVL trees

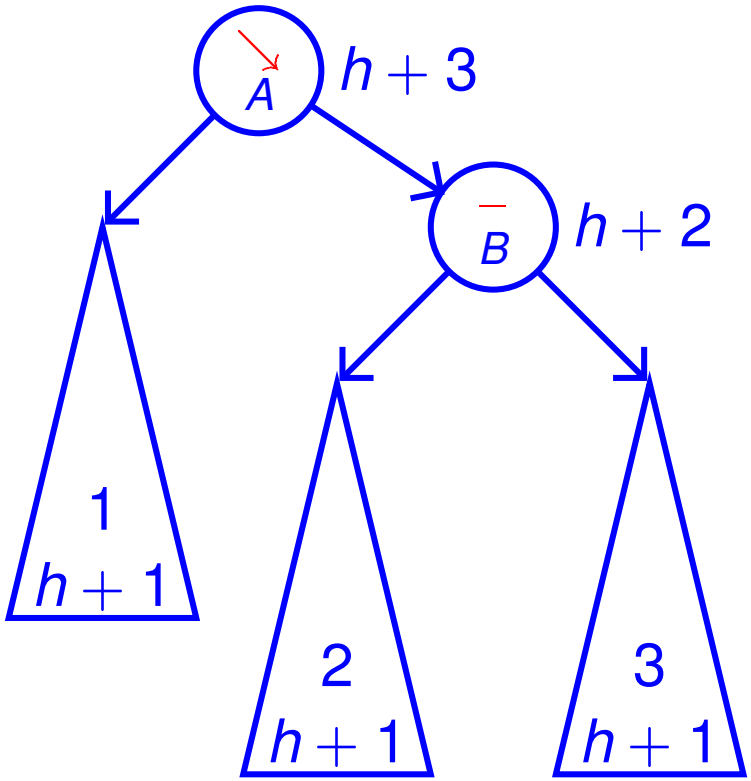
- As normal binary search tree deletion
- Final node deleted will be either a leaf or have just one subtree (what about nodes with two (non-empty) subtrees?)
- If the deleted node has one subtree, then that subtree contains only one node (why?)
- Retrace back from the deleted node checking the balance of each node, balance the node by rotation if unbalanced

- Imbalance may happen at a node such as A due to deletion in its L-ST
- Then examine r-child (B)
- For deletion in R-ST, examine l-child



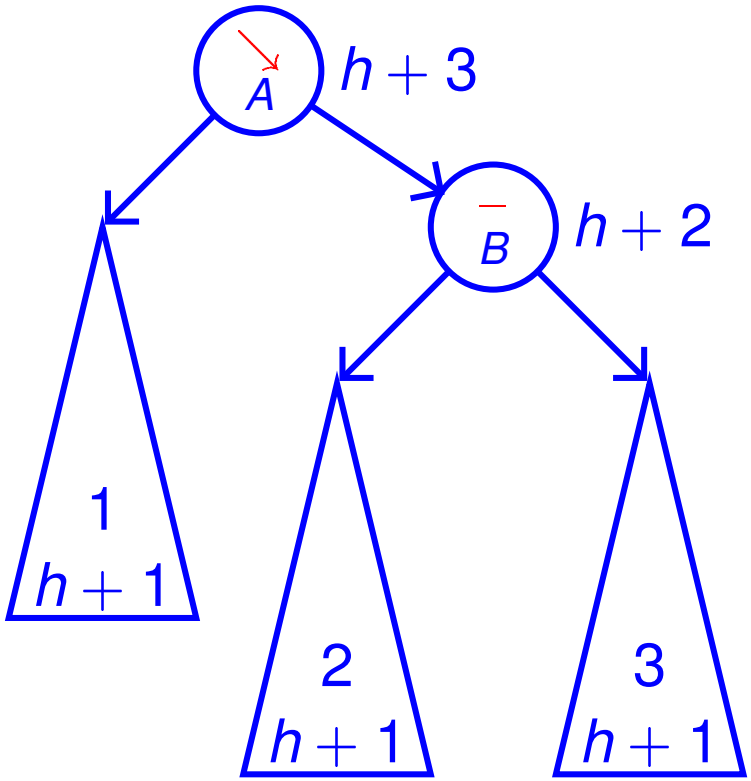
L-ST of root loosing height after deletion

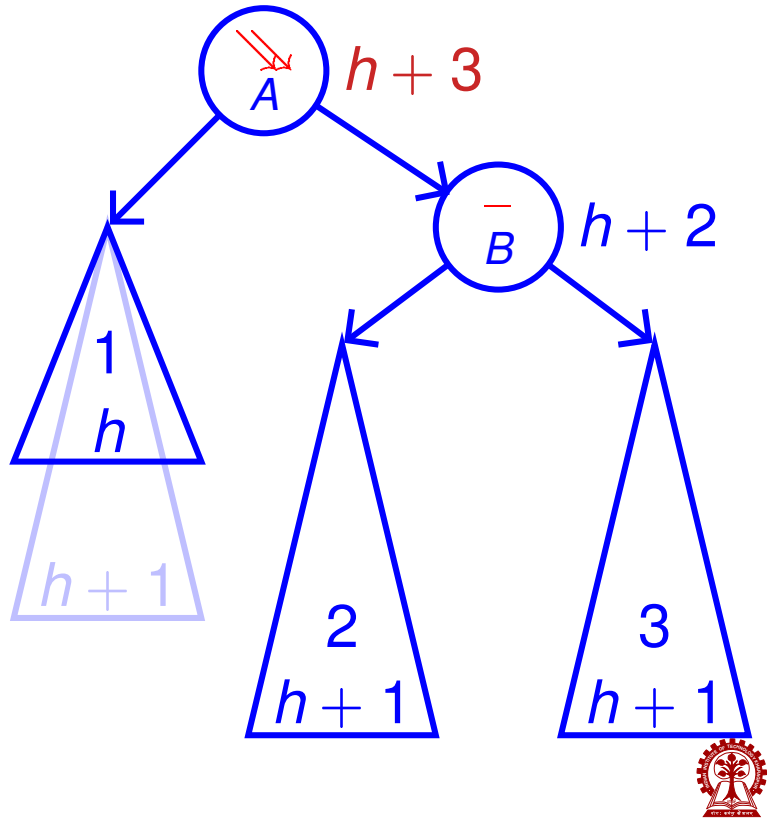
- A key is deleted from the L-ST of A
- Tree becomes unbalanced after deletion from the left sub-tree of A



L-ST of root loosing height after deletion

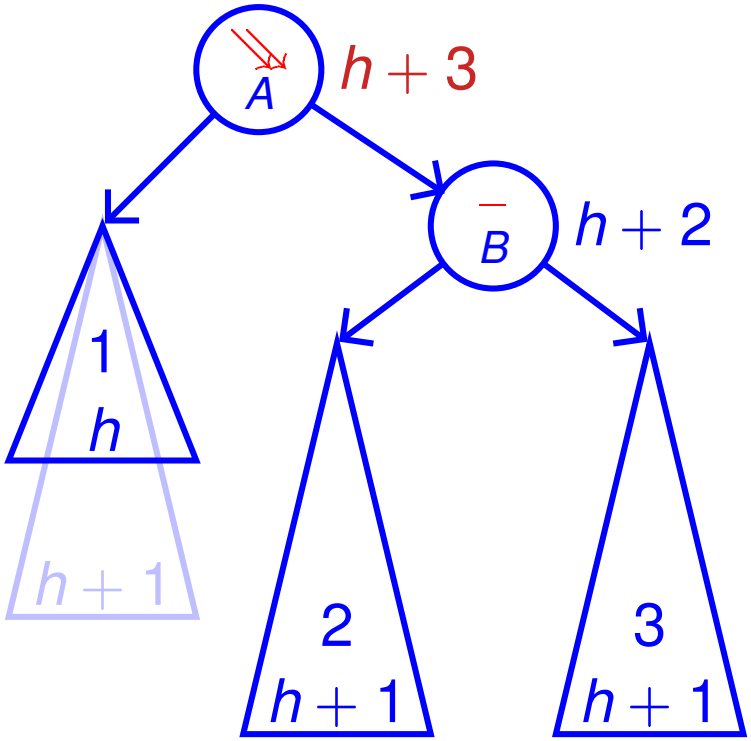
- A key is deleted from the L-ST of A
- Tree becomes unbalanced after deletion from the left sub-tree of A





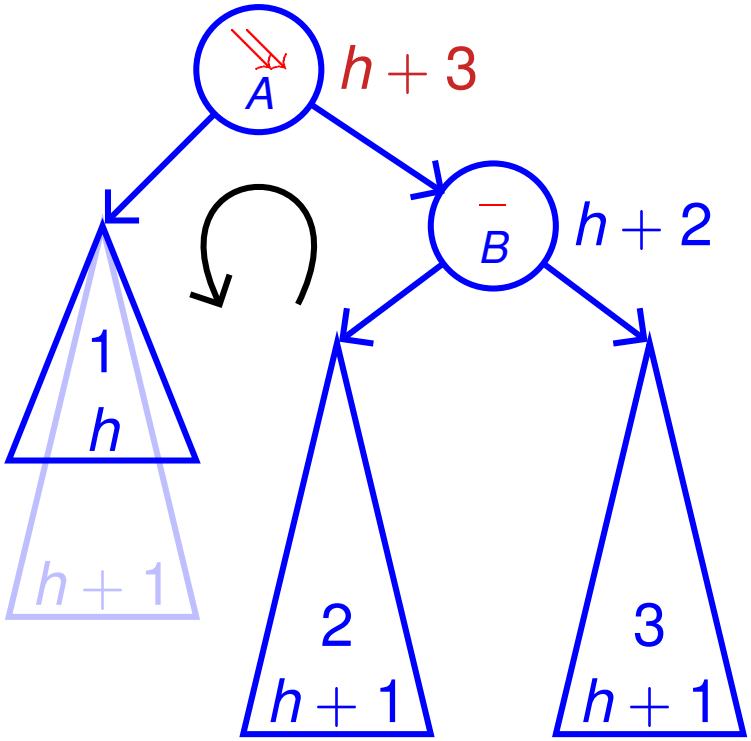
Correction by single rotation after deletion

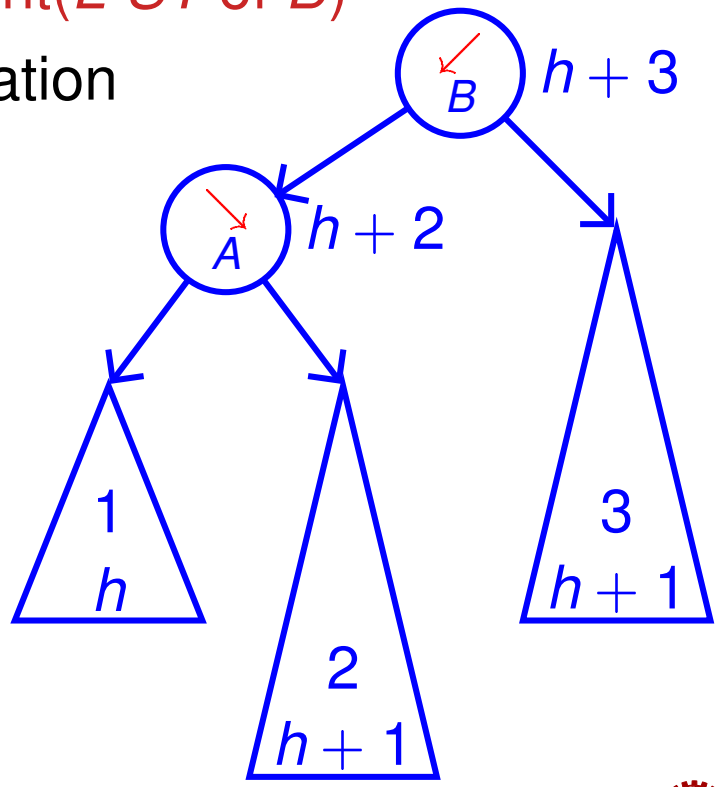
- Left subtree of root has lost height after deletion
- Balance disturbed at A, while retracing path back via *L-ST*



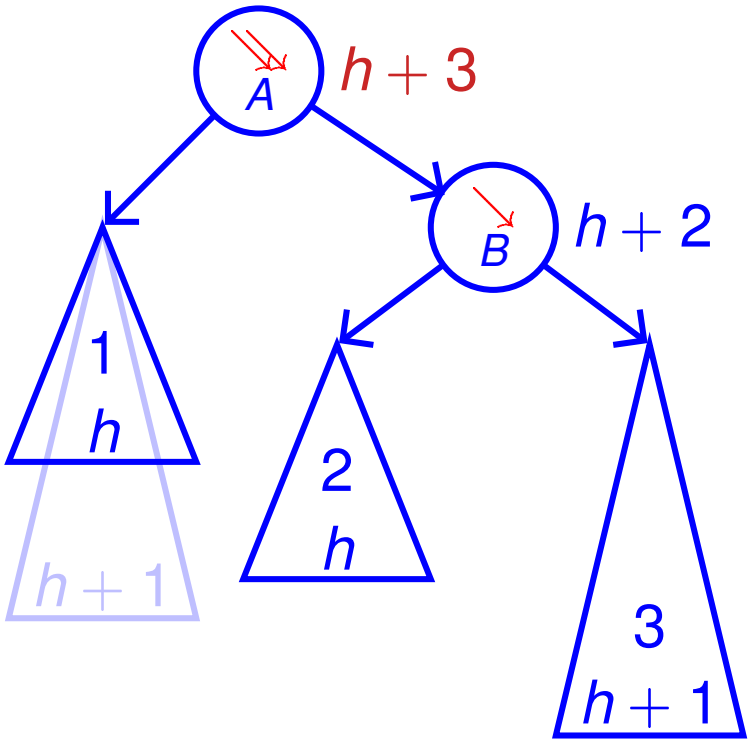
Correction by single rotation after deletion

- Left subtree of root has lost height after deletion
- Balance disturbed at *A*, while retracing path back via *L-ST*
- Revisit *B* to find height(*R-ST* of *B*)≥height(*L-ST* of *B*)
- Correct by applying single (left-right) rotation

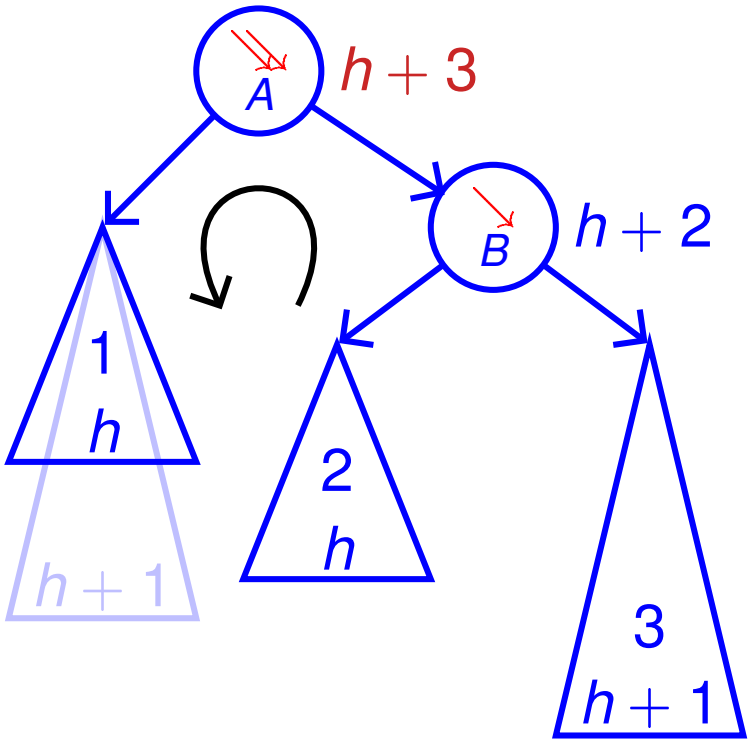


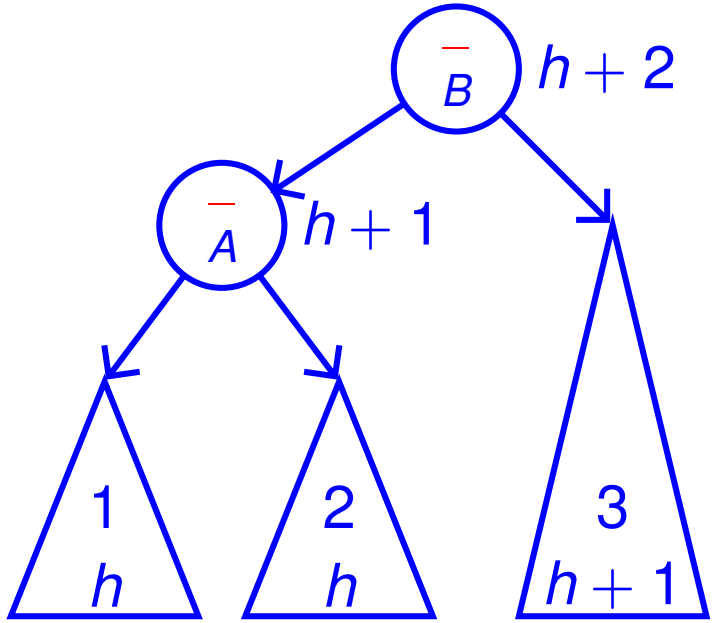


height(*R-ST* of *B*)>height(*L-ST* of *B*) after deletion



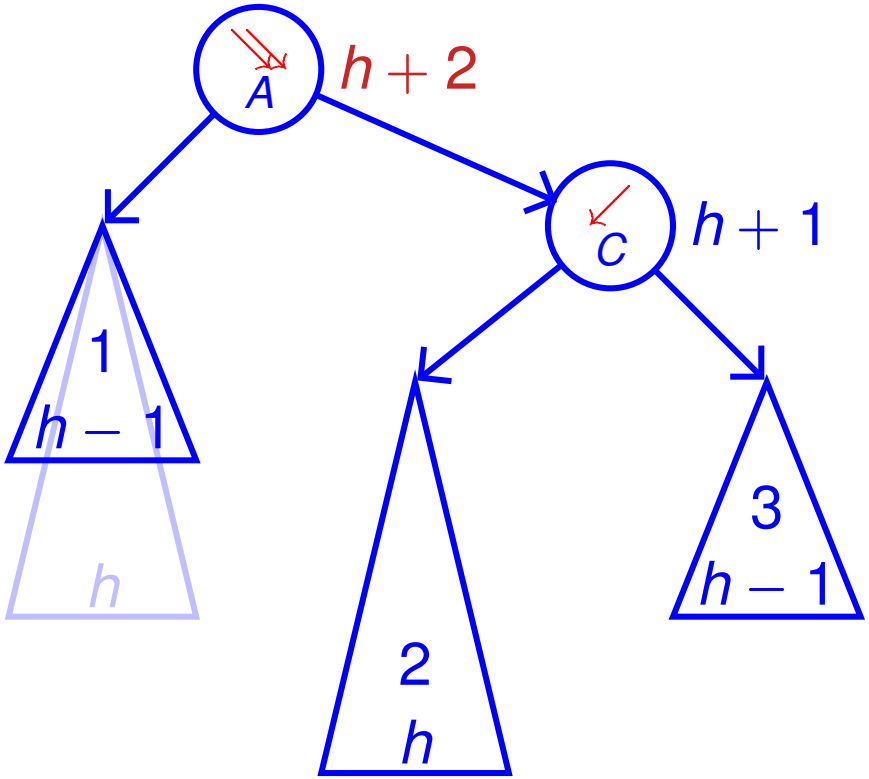
height(*R-ST* of *B*)>height(*L-ST* of *B*) after deletion





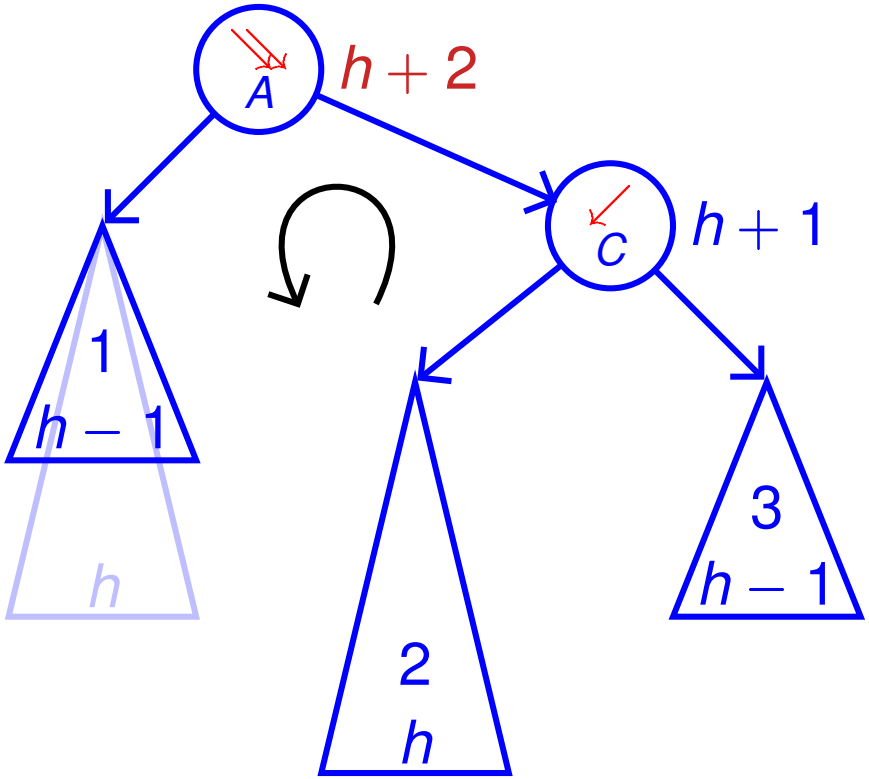
Another case of deletion

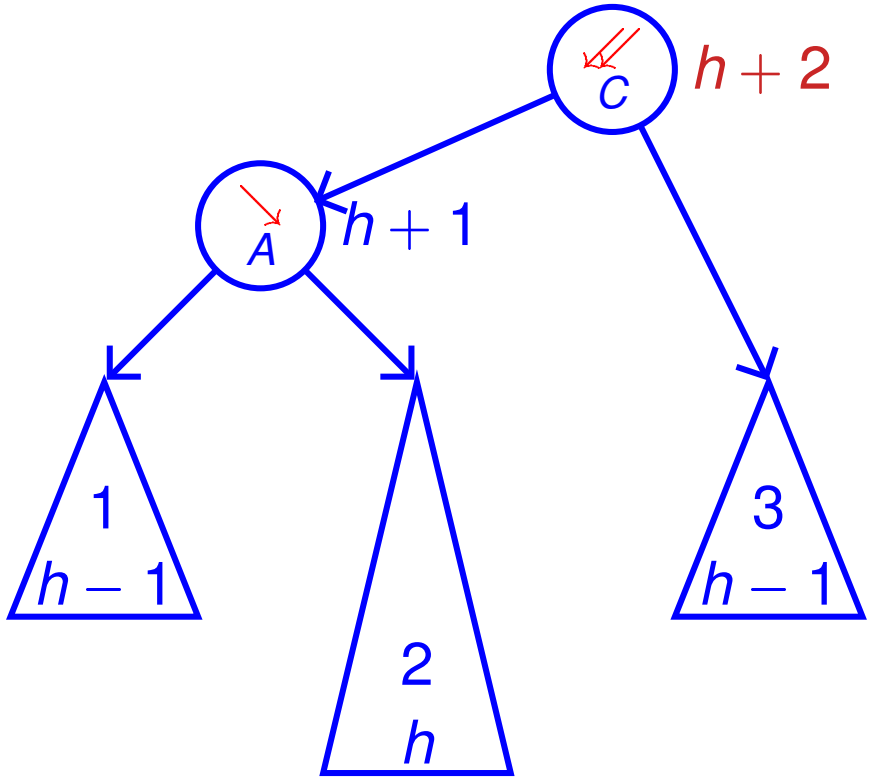
- Now, after deletion height(*R-ST* of *C*)<height(*L-ST* of *C*)
- Does applying single (left) rotation work?



Another case of deletion

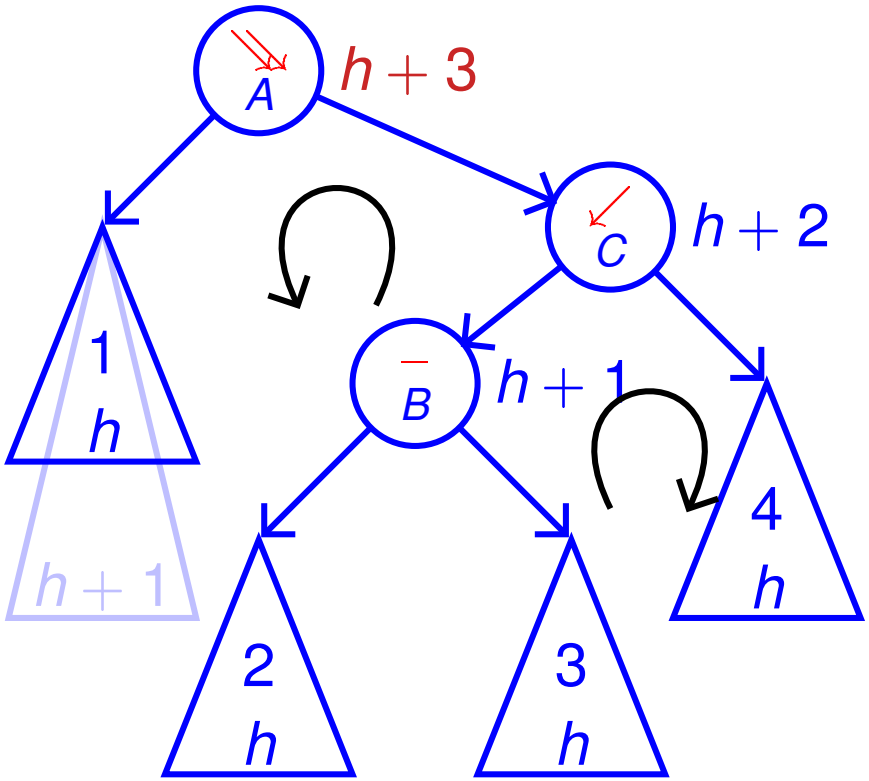
- Now, after deletion height(*R-ST* of *C*)<height(*L-ST* of *C*)
- Does applying single (left) rotation work?





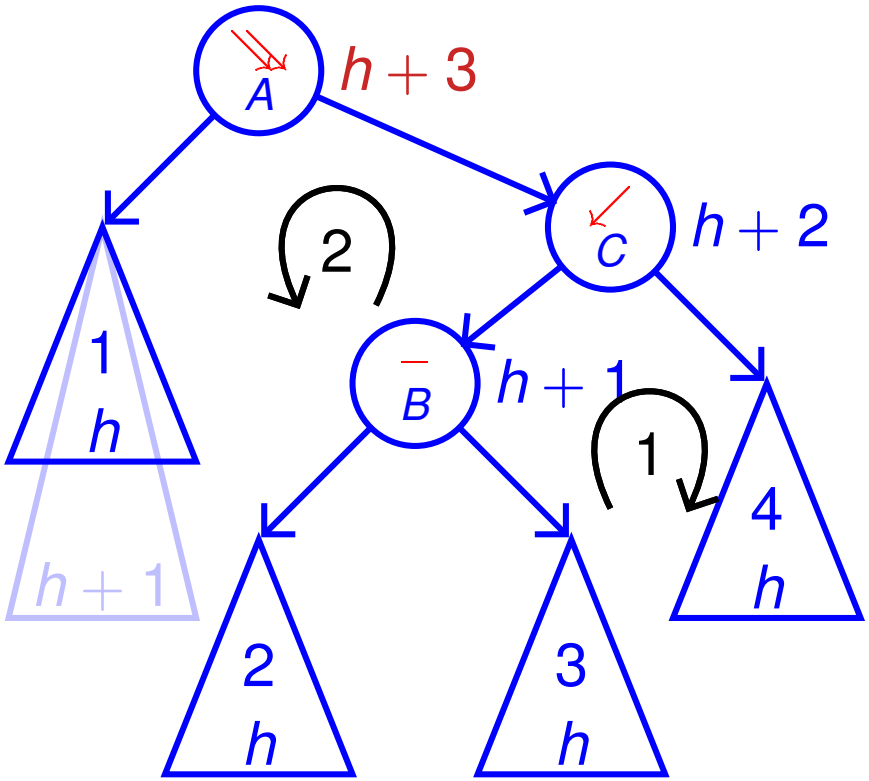
Correction by double rotation after deletion

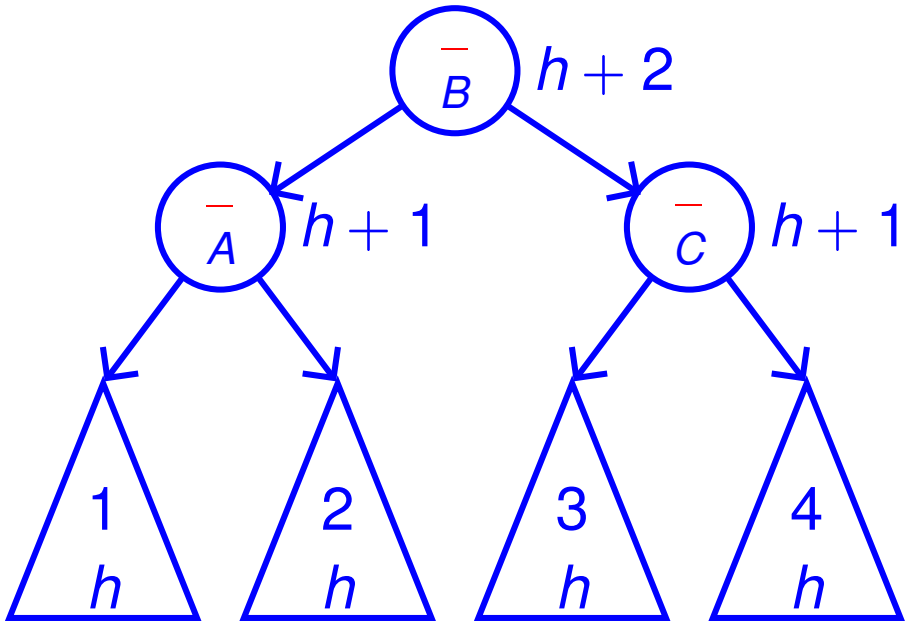
- After deletion height(*R-ST* of *C*)<height(*L-ST* of *C*)
- What happens if correction by double rotation is attempted?



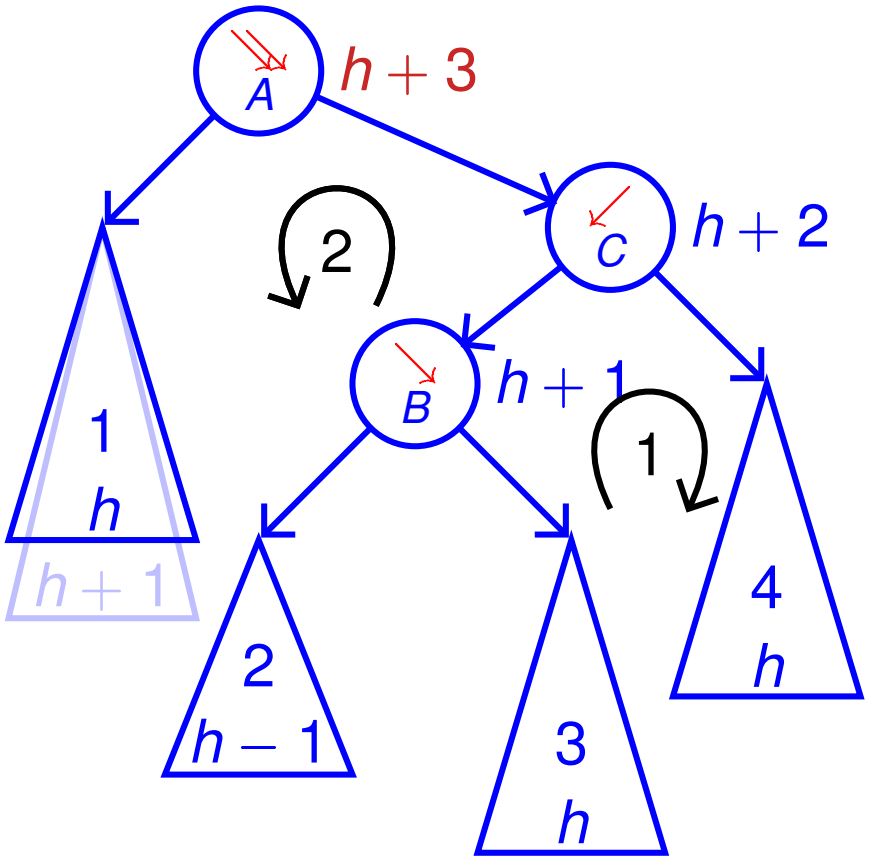
Correction by double rotation after deletion

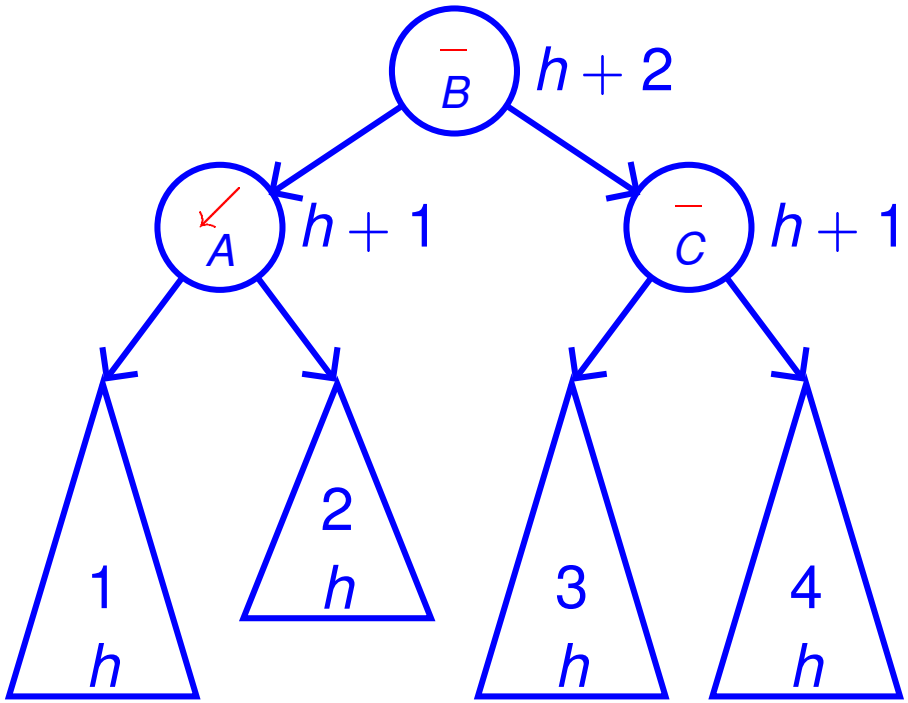
- After deletion height(*R-ST* of *C*)<height(*L-ST* of *C*)
- What happens if correction by double rotation is attempted?
- Now balance is restored
- There is a reduction in height, corrections must continue upwards



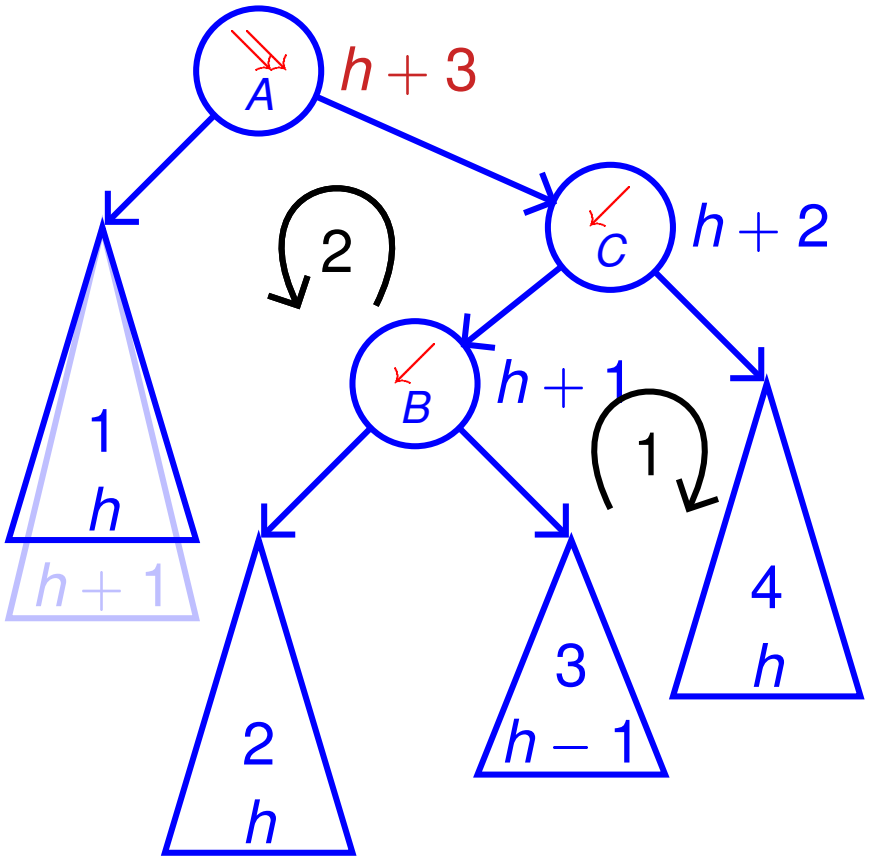


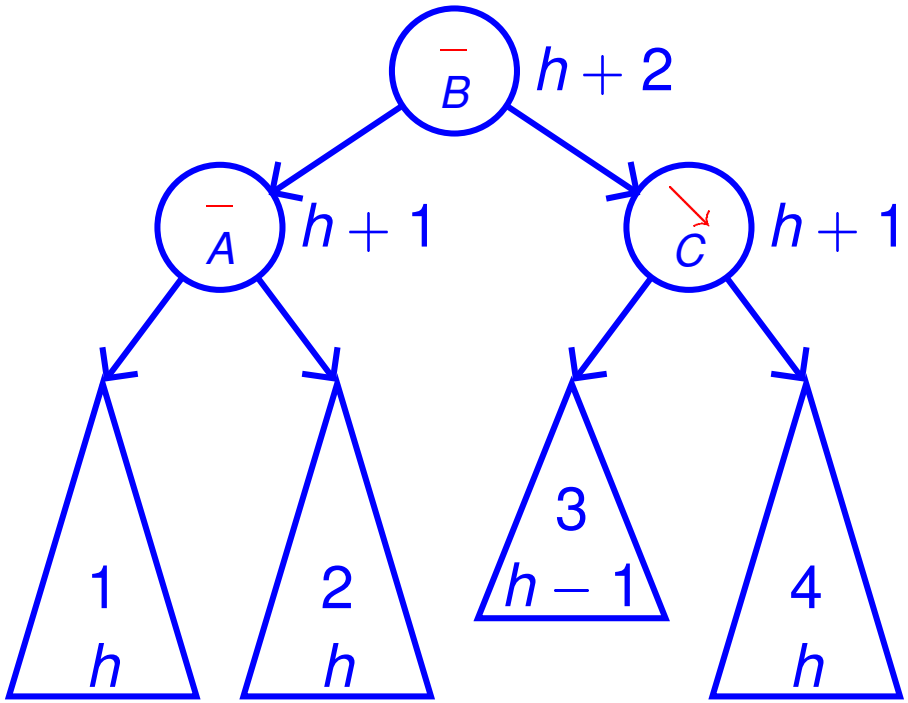
Other cases of double rotation





Other cases of double rotation (contd.)





Complexities of operations

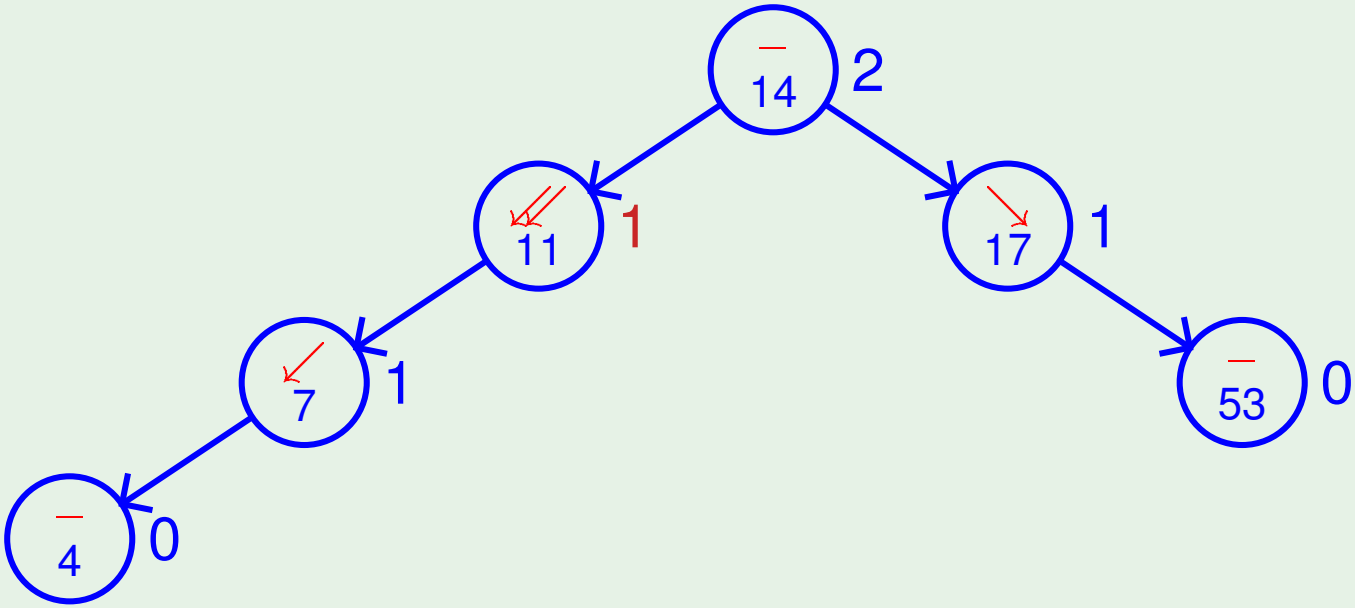
Search Needs traversing to a leaf node – $\Theta(\lg N)$

Insert
- Needs traversing to a leaf node
- Correction by rotations up to two levels
- $\Theta(\lg N)$ worst case complexity

Delete
- Needs traversing to a leaf node or a level above
- Correction by rotations, may propagate back to root node, *monotonically*
- $\Theta(\lg N)$ worst case complexity

Examples of AVL tree operations

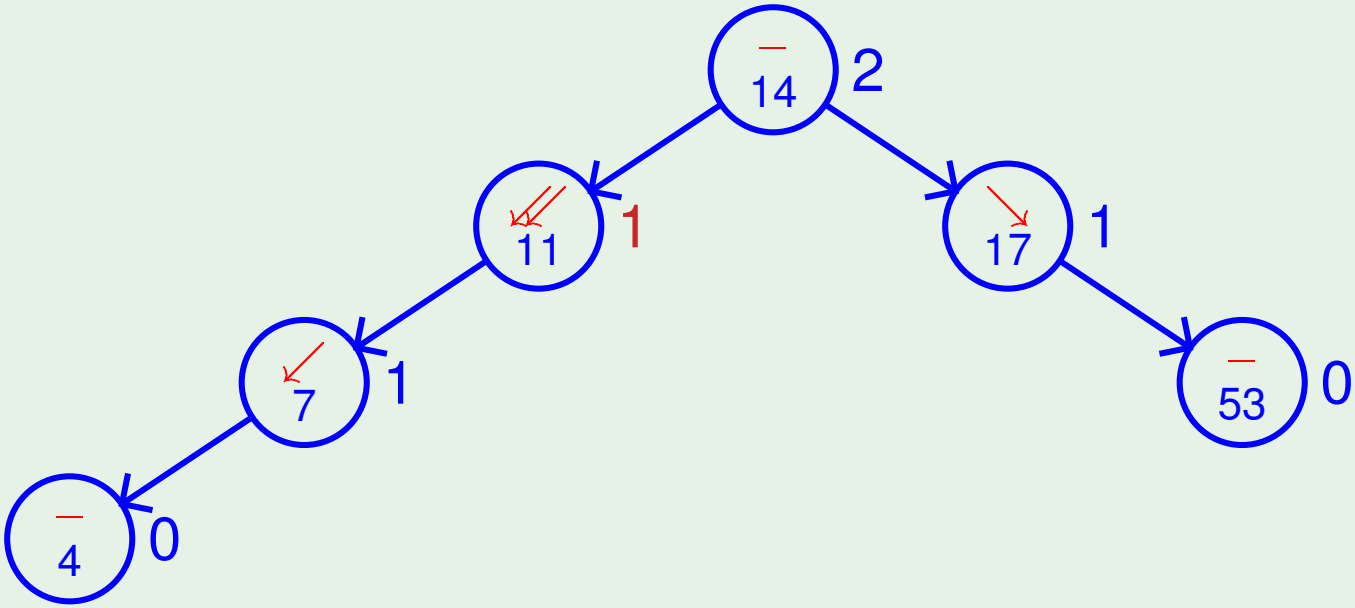
Example (Insert: 14, 17, 11, 7, 53, 4)



Revisit 7:

Examples of AVL tree operations

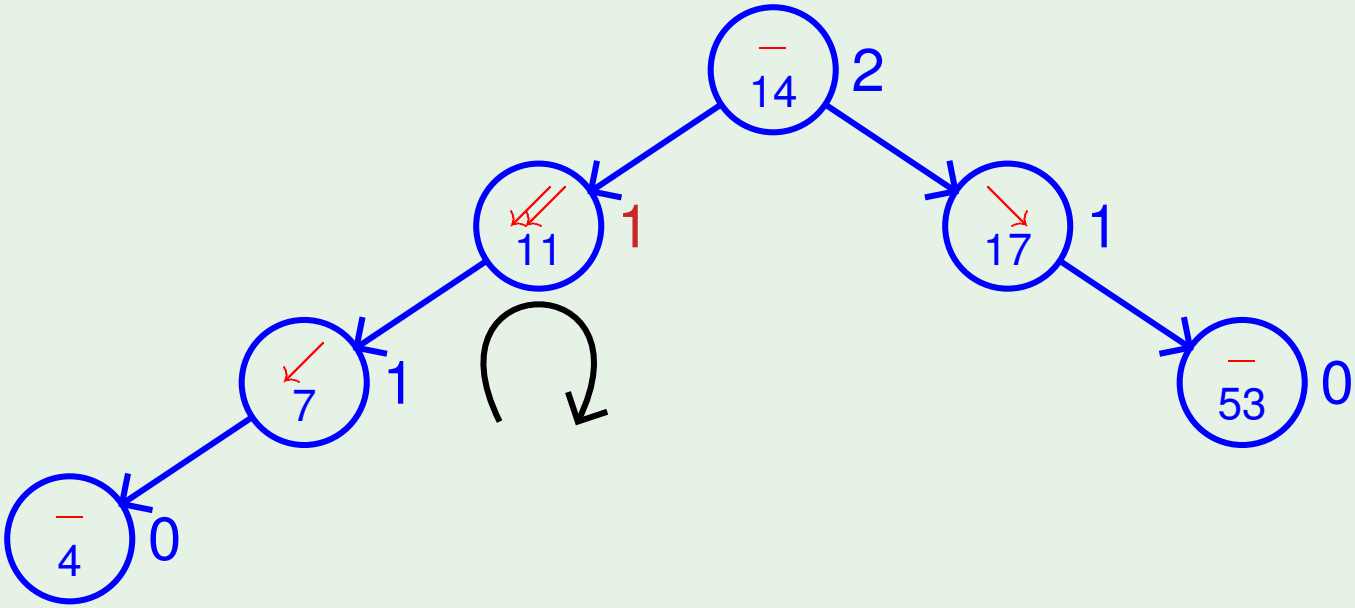
Example (Insert: 14, 17, 11, 7, 53, 4)



Revisit 7: height(*L-ST*) $>$ height(*R-ST*)

Examples of AVL tree operations

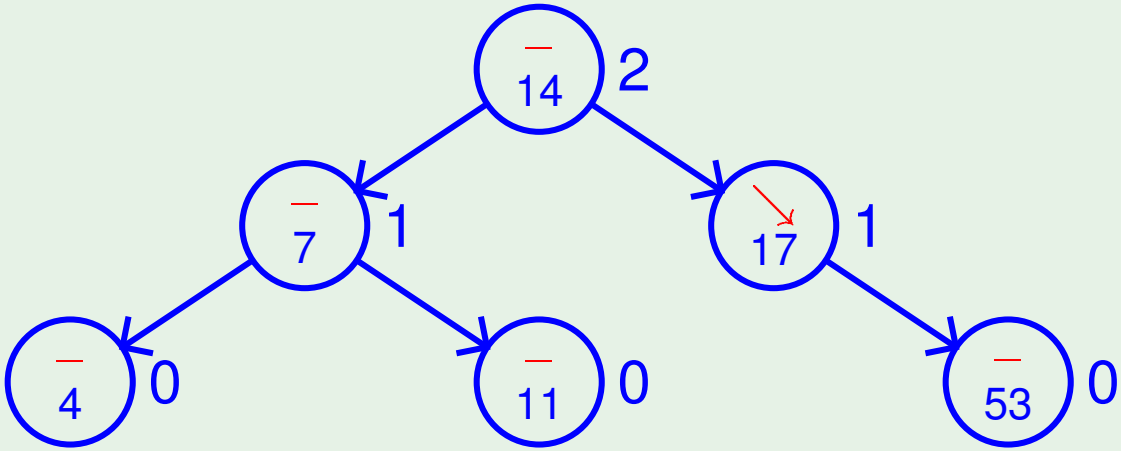
Example (Insert: 14, 17, 11, 7, 53, 4)



Revisit 7: height(*L-ST*) $>$ height(*R-ST*), apply right rotation

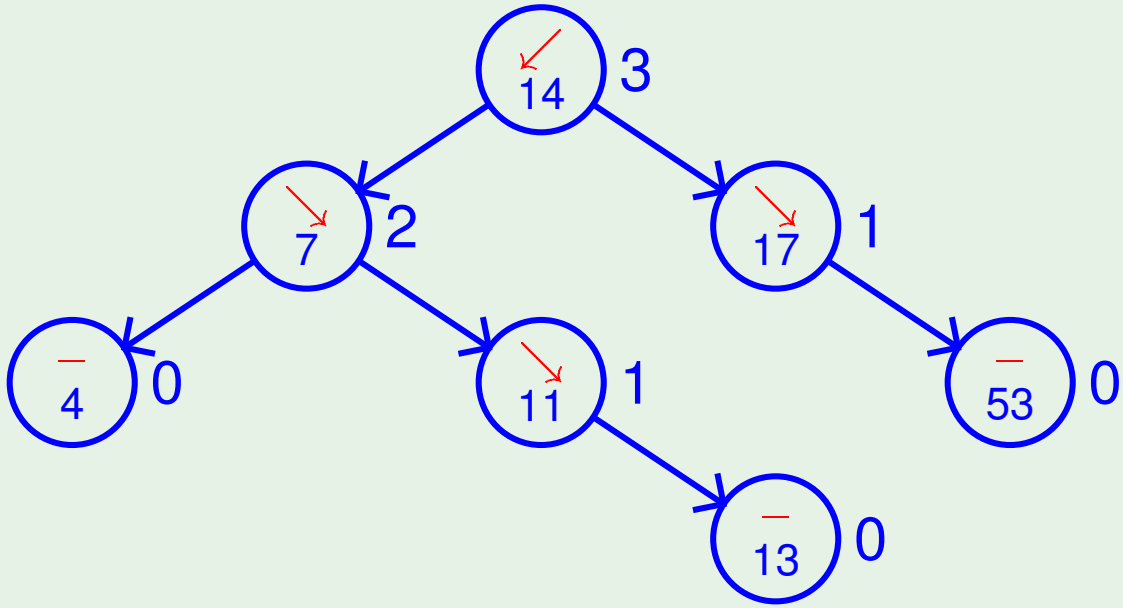
Examples of AVL tree operations (contd.)

Example



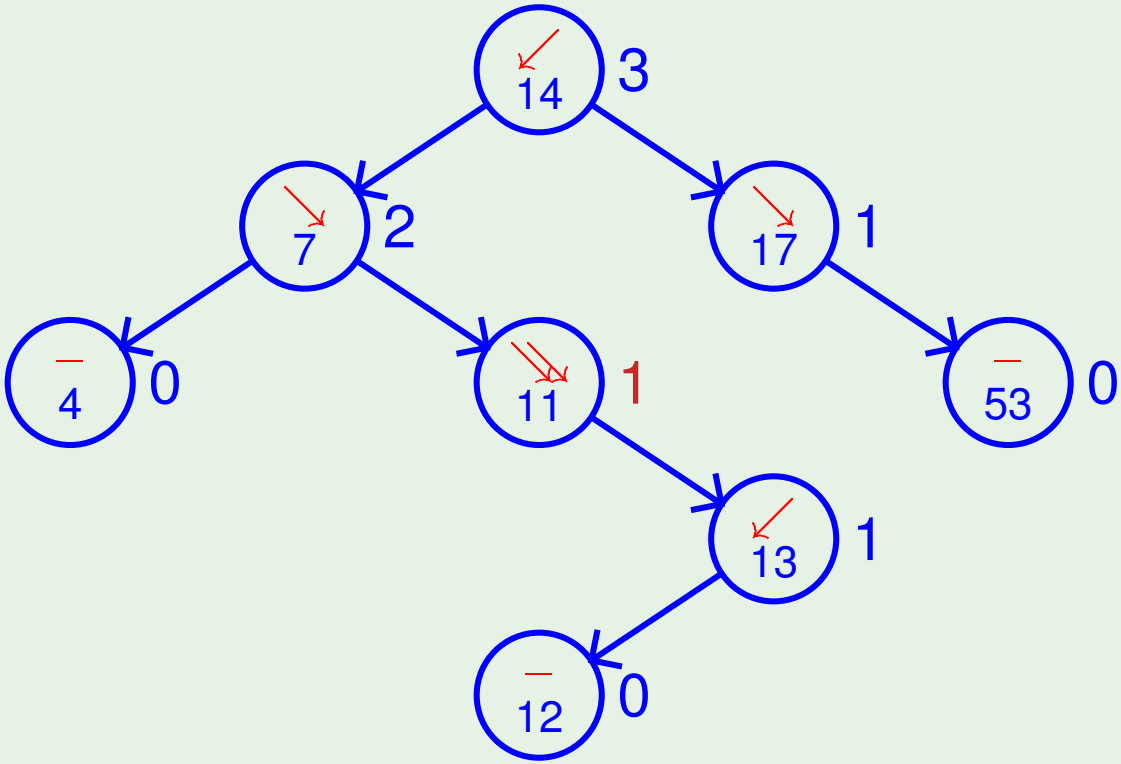
Examples of AVL tree operations (contd.)

Example (Insert: 13, 12)



Examples of AVL tree operations (contd.)

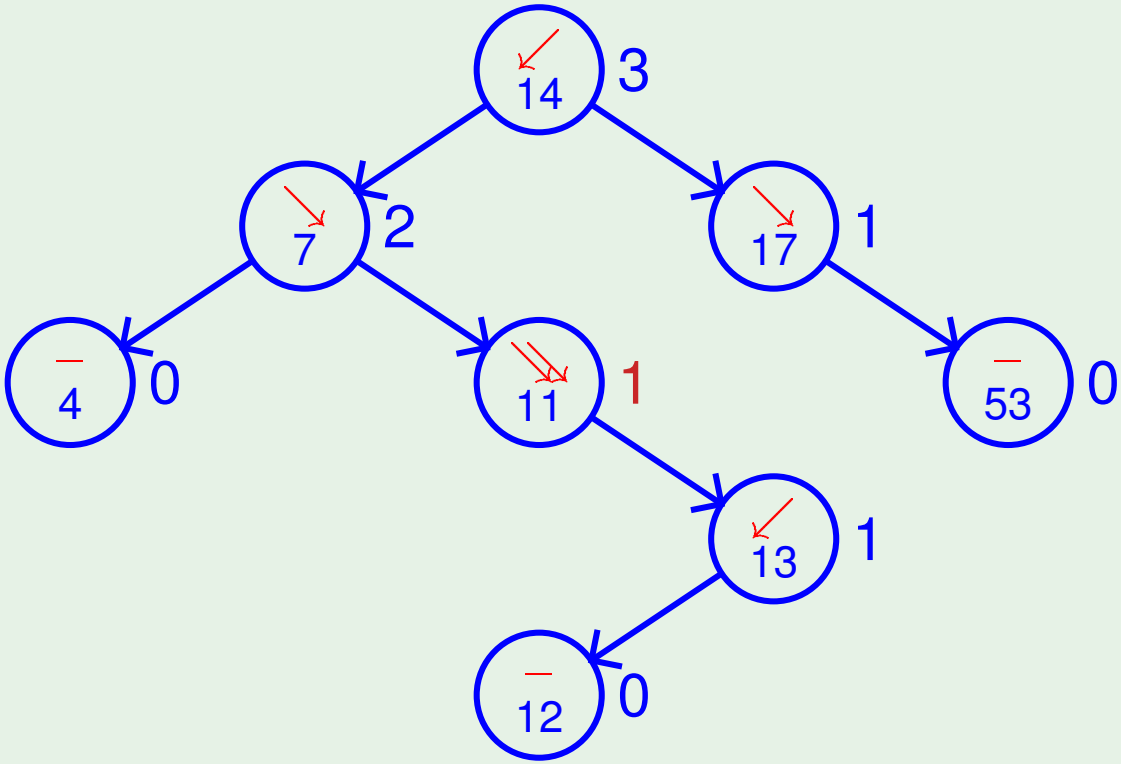
Example (Insert: 12)



Revisit 13:

Examples of AVL tree operations (contd.)

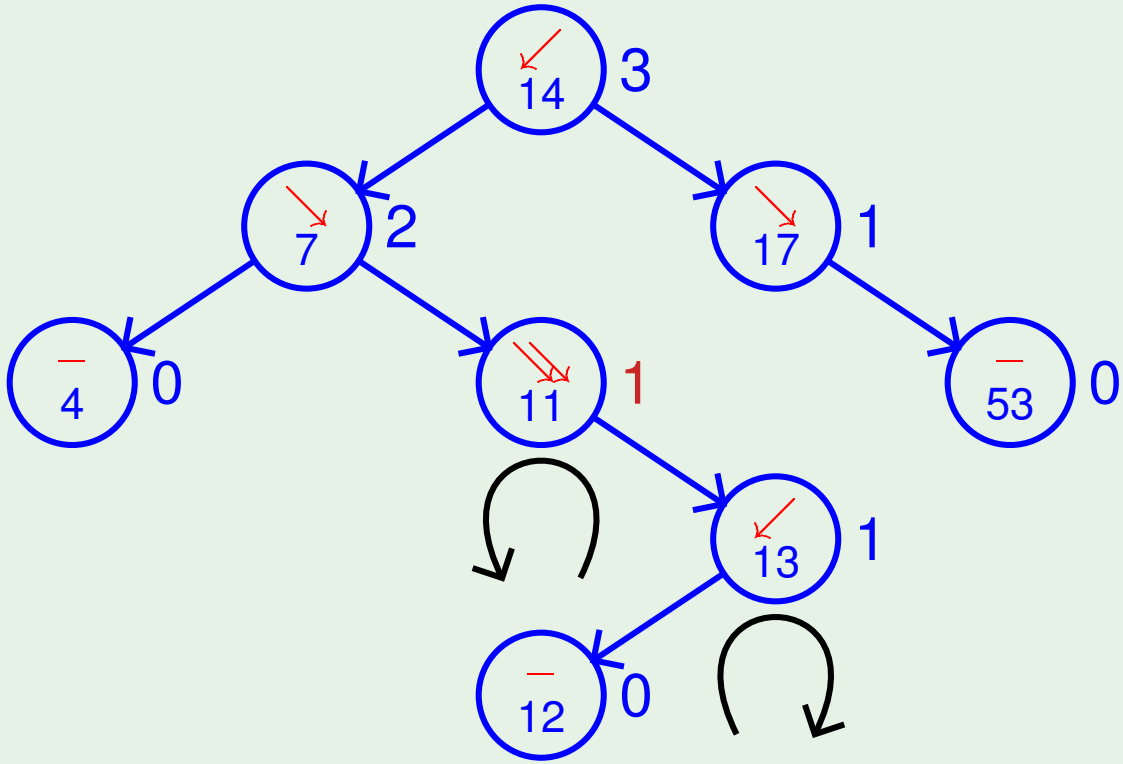
Example (Insert: 12)



Revisit 13: height(*R-ST*) $<$ height(*L-ST*)

Examples of AVL tree operations (contd.)

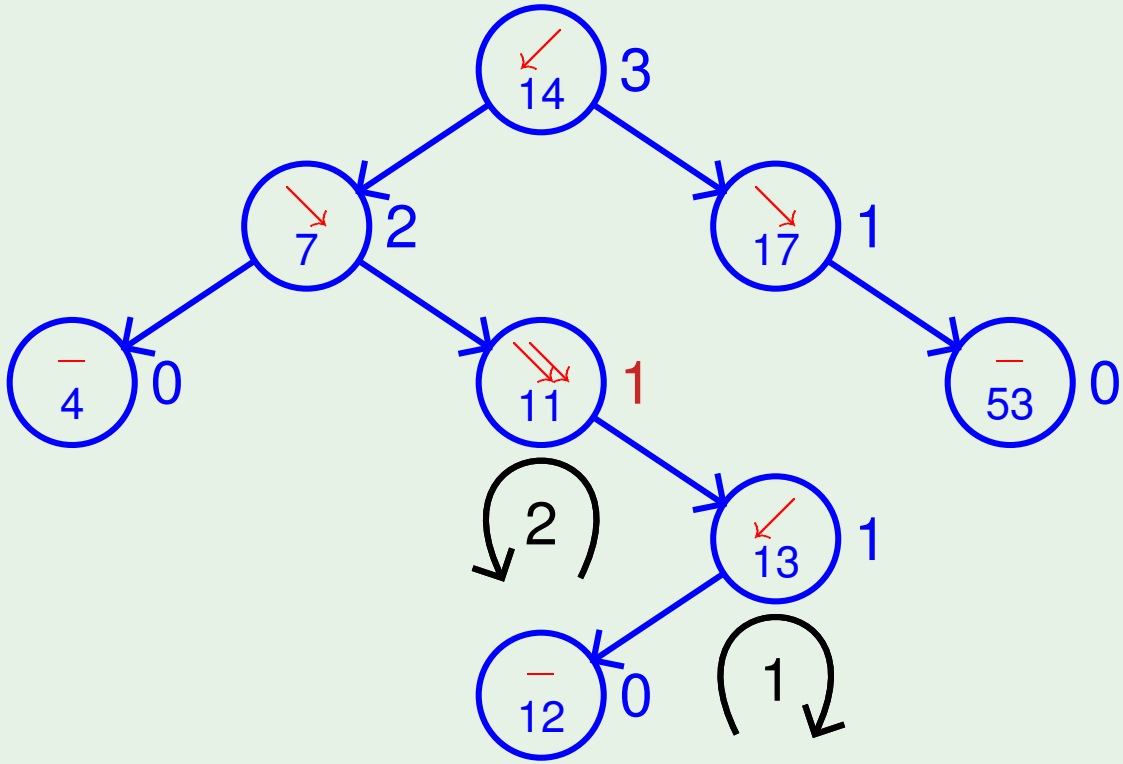
Example (Insert: 12)



Revisit 13: height(*R-ST*) $<$ height(*L-ST*), apply right-left rotation

Examples of AVL tree operations (contd.)

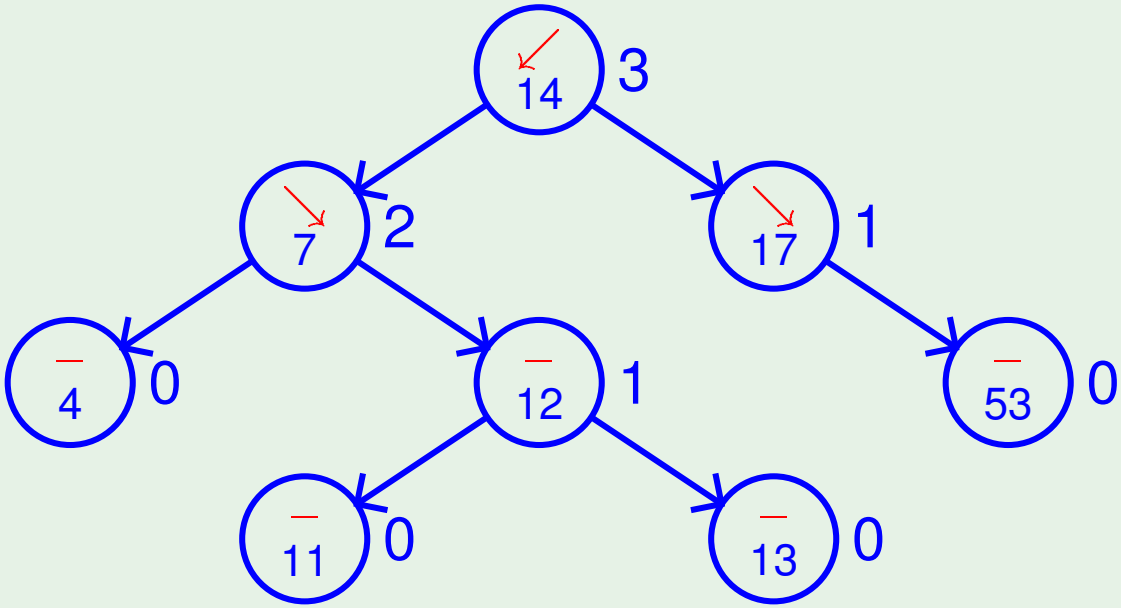
Example (Insert: 12)



Revisit 13: height(*R-ST*) $<$ height(*L-ST*), apply right-left rotation

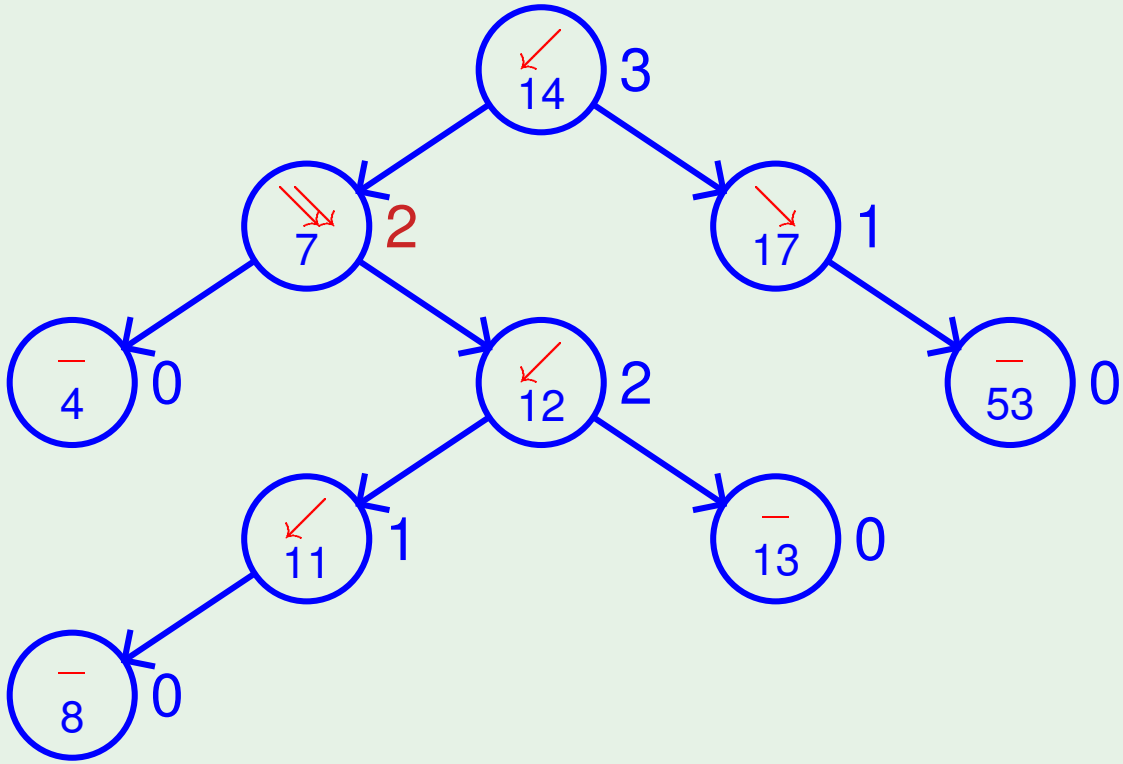
Examples of AVL tree operations (contd.)

Example (Insert: 12)



Examples of AVL tree operations (contd.)

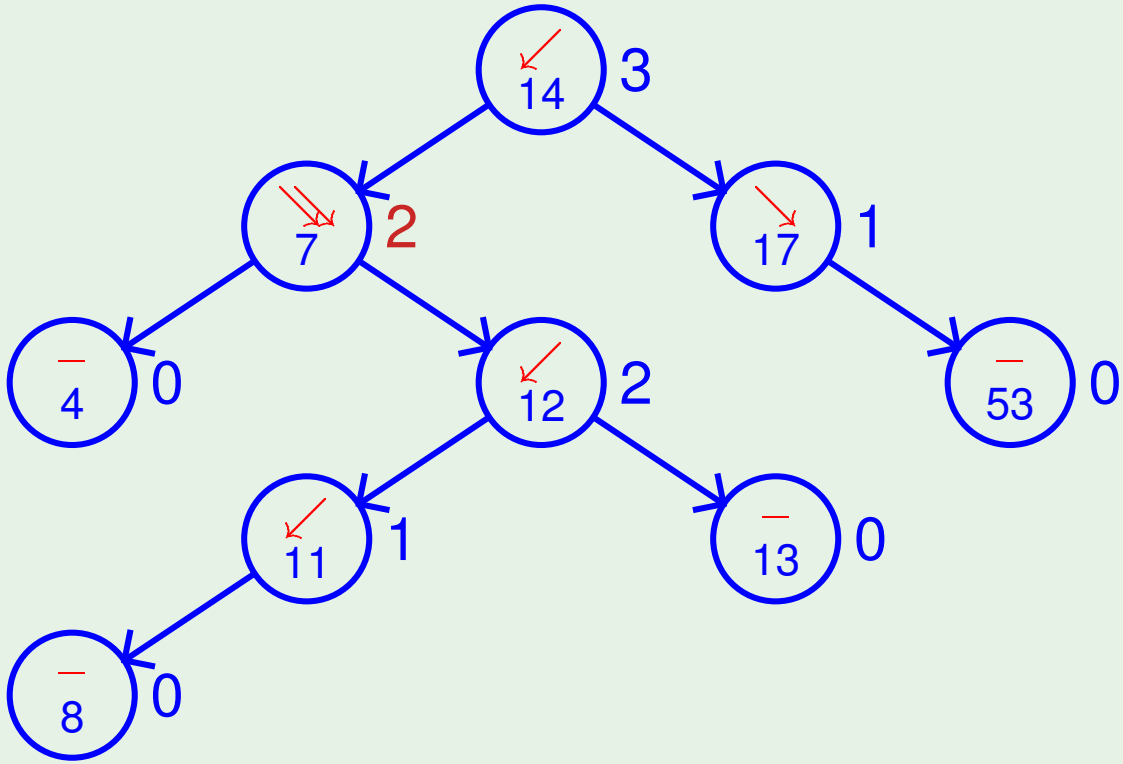
Example (Insert: 8)



Revisit 12:

Examples of AVL tree operations (contd.)

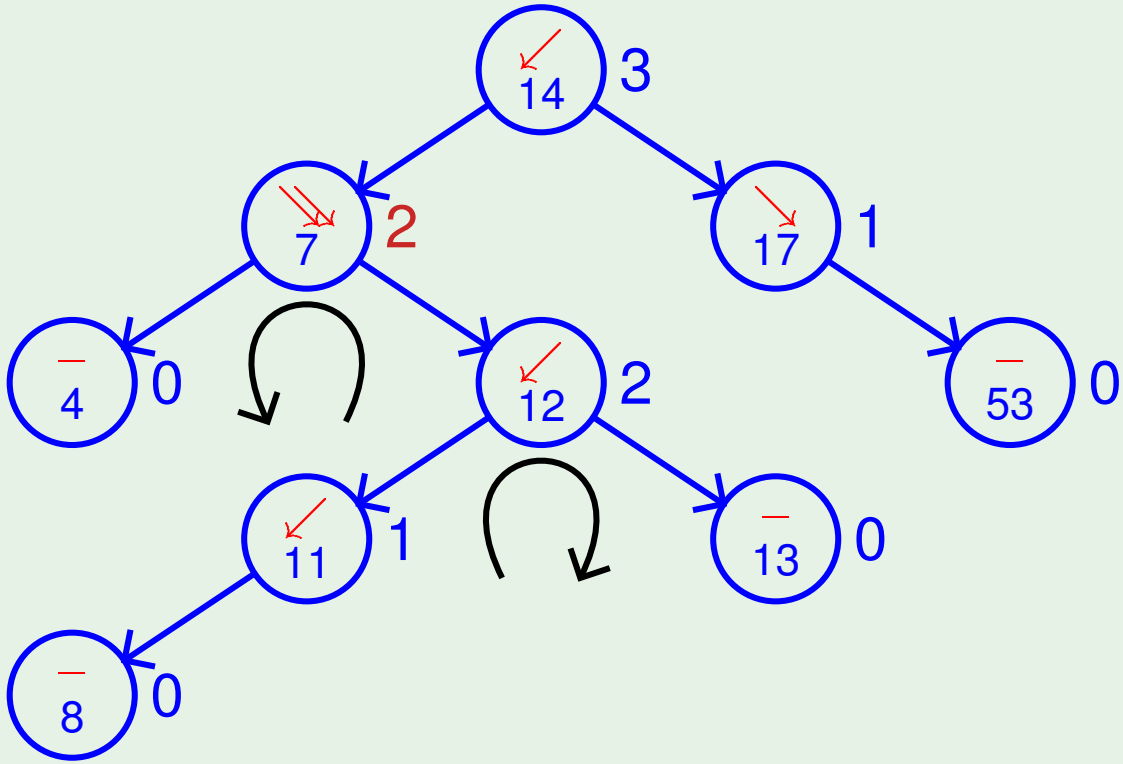
Example (Insert: 8)



Revisit 12: height(*L-ST*) $>$ height(*R-ST*)

Examples of AVL tree operations (contd.)

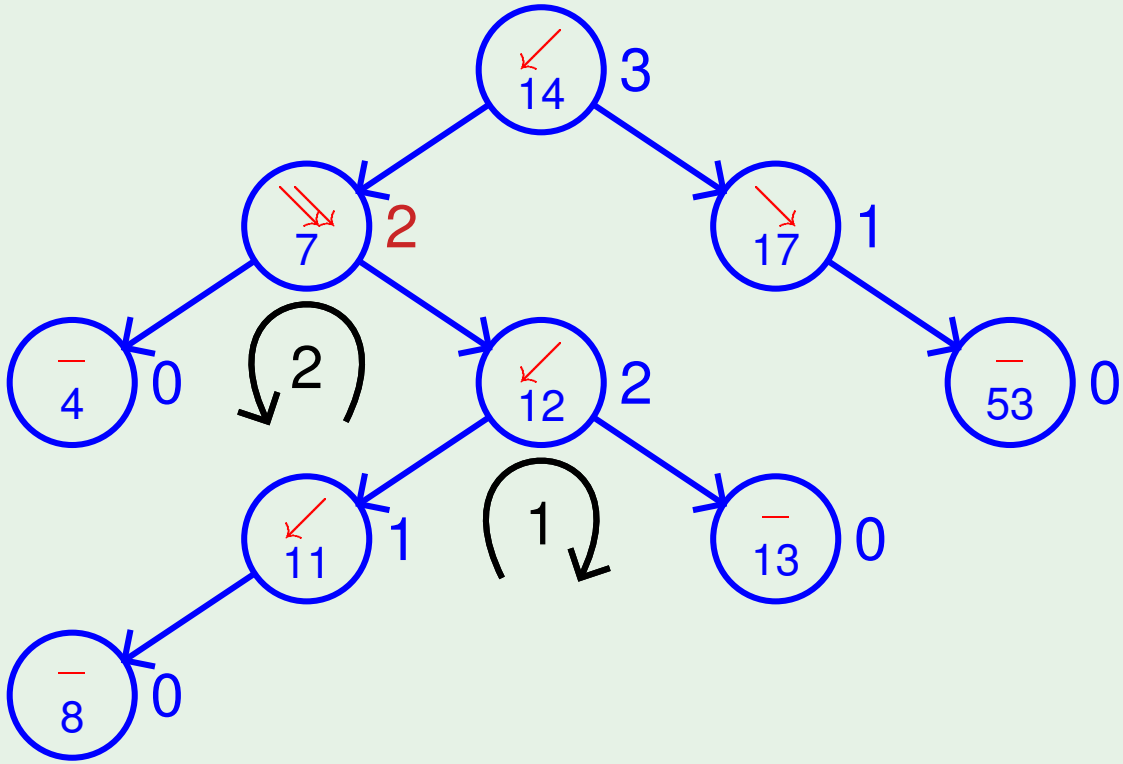
Example (Insert: 8)



Revisit 12: height(*L-ST*) $>$ height(*R-ST*), apply double rotation

Examples of AVL tree operations (contd.)

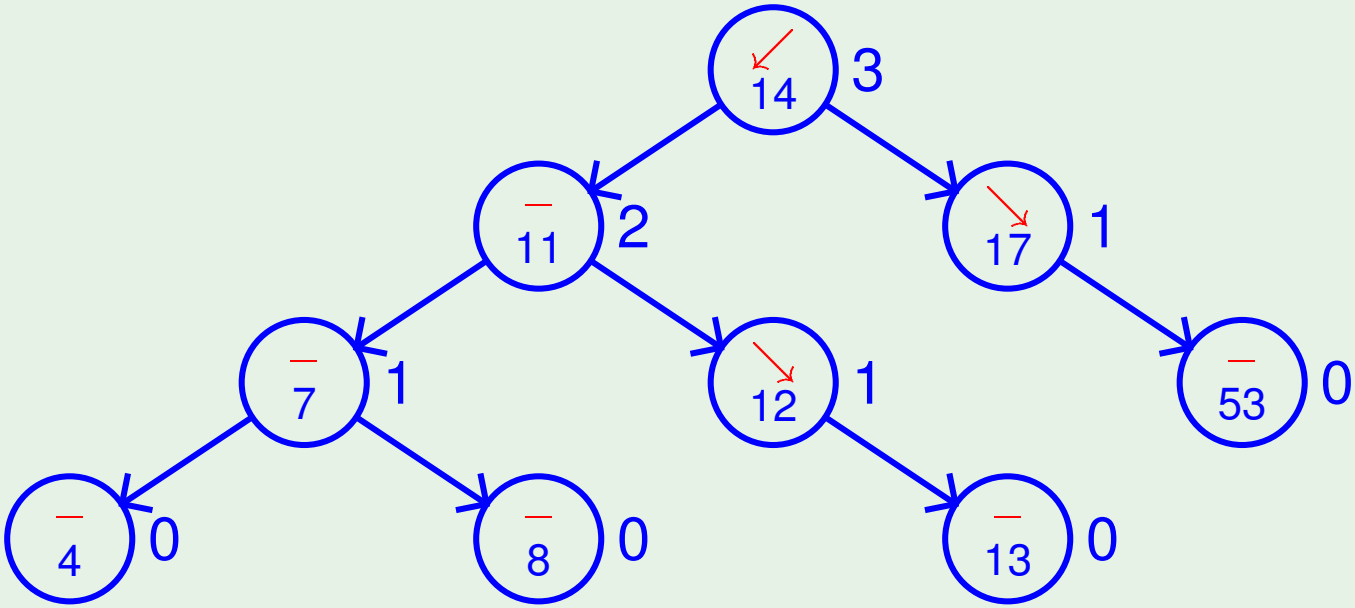
Example (Insert: 8)



Revisit 12: height(*L-ST*) $>$ height(*R-ST*), apply double rotation

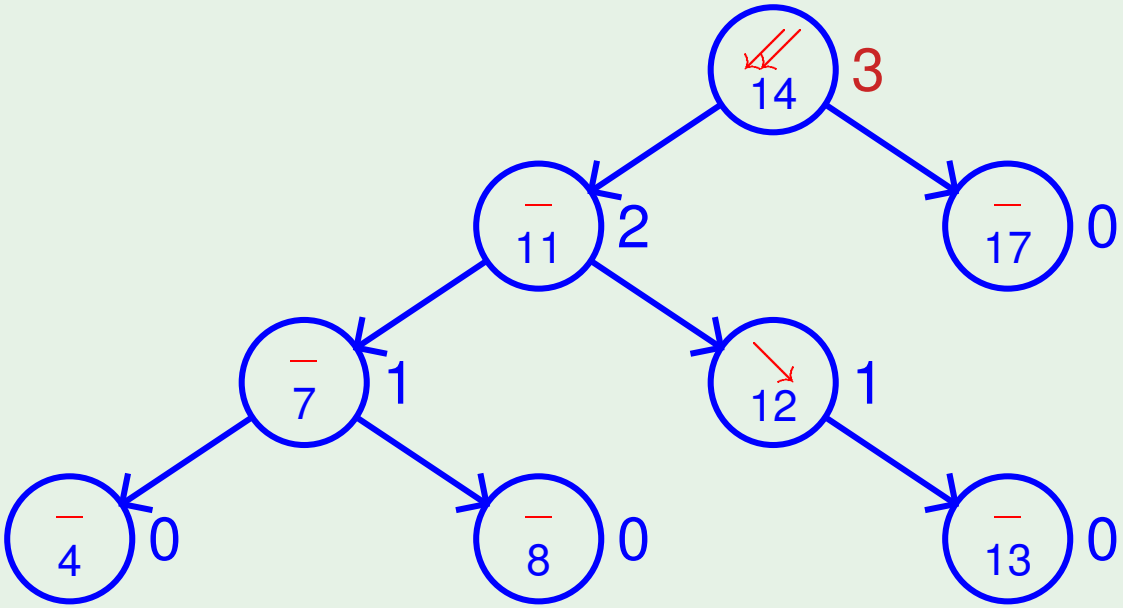
Examples of AVL tree operations (contd.)

Example (Insert: 8, Delete: 53)



Examples of AVL tree operations (contd.)

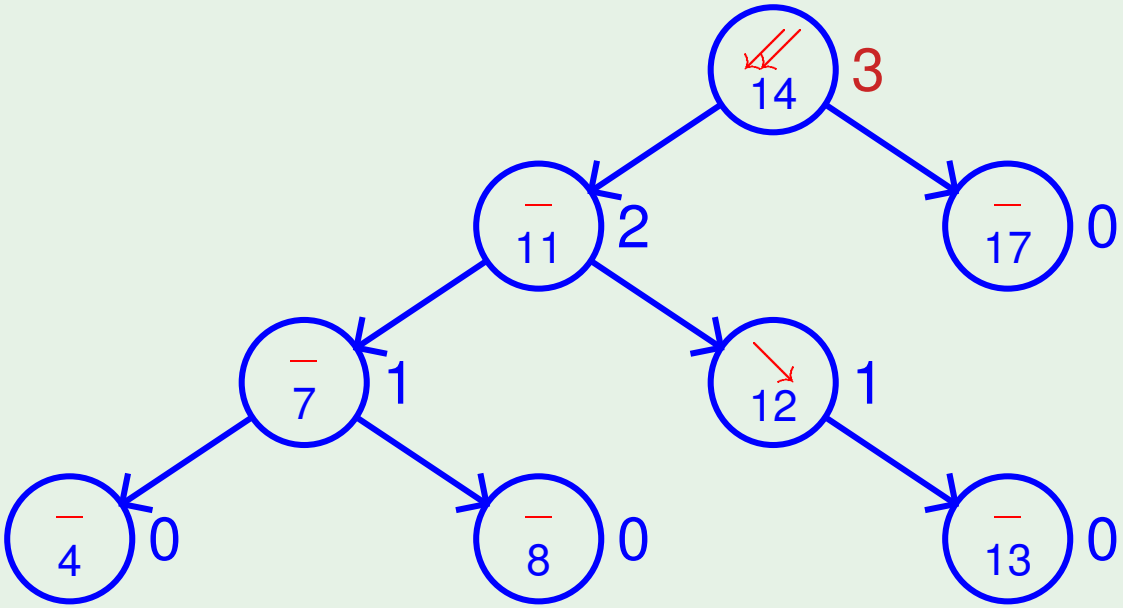
Example (Delete: 53)



Check 11:

Examples of AVL tree operations (contd.)

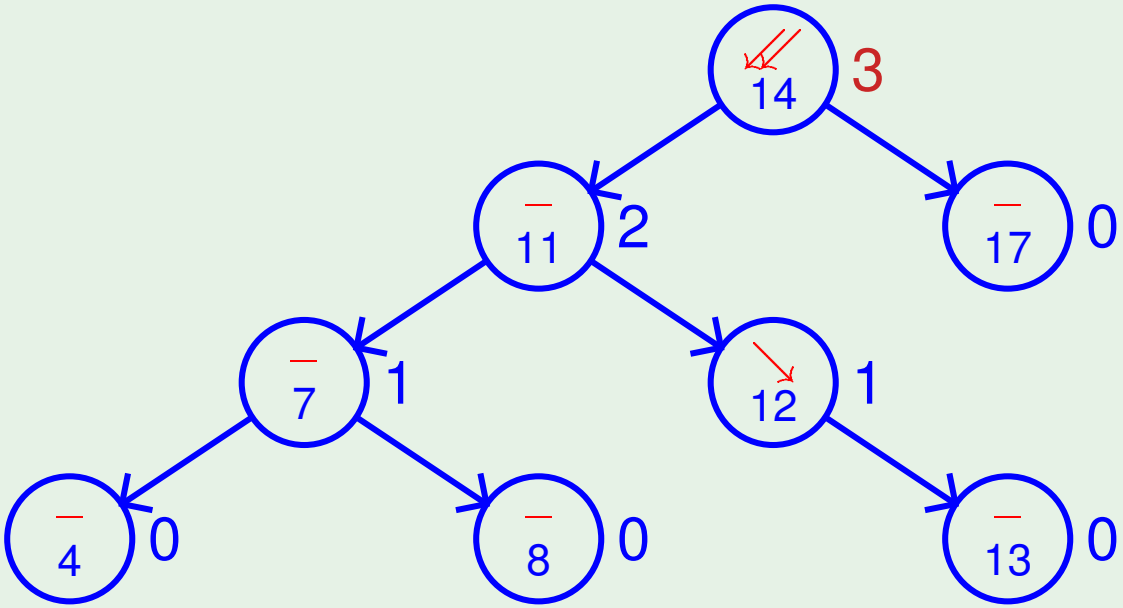
Example (Delete: 53)



Check 11: height(*L-ST*) $\geq$ height(*R-ST*)

Examples of AVL tree operations (contd.)

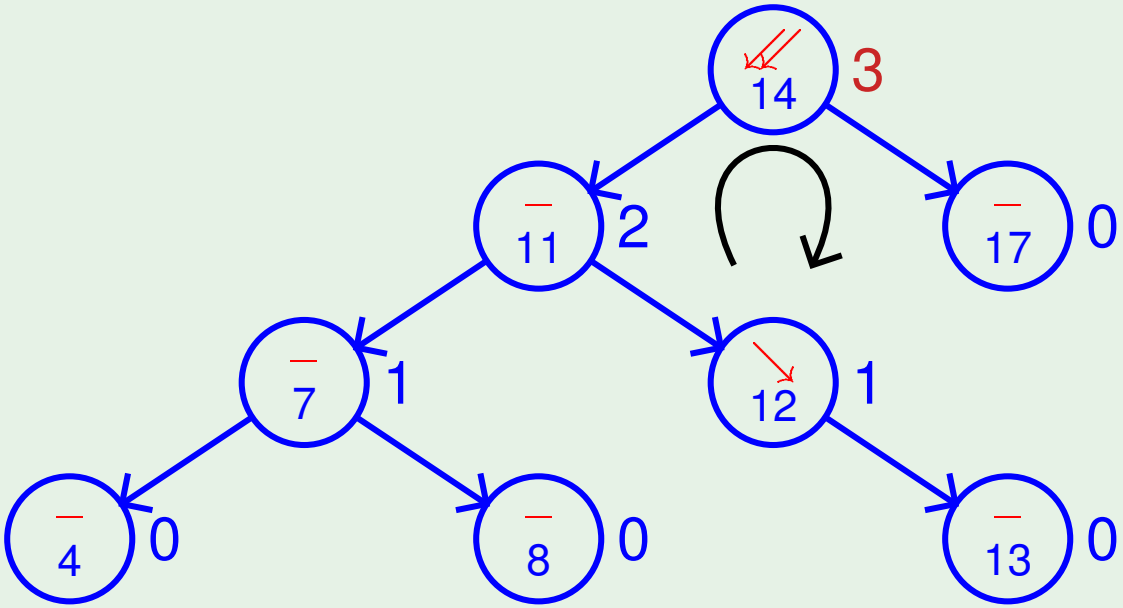
Example (Delete: 53)



Check 11: height(*L-ST*) $\geq$ height(*R-ST*), apply single rotation

Examples of AVL tree operations (contd.)

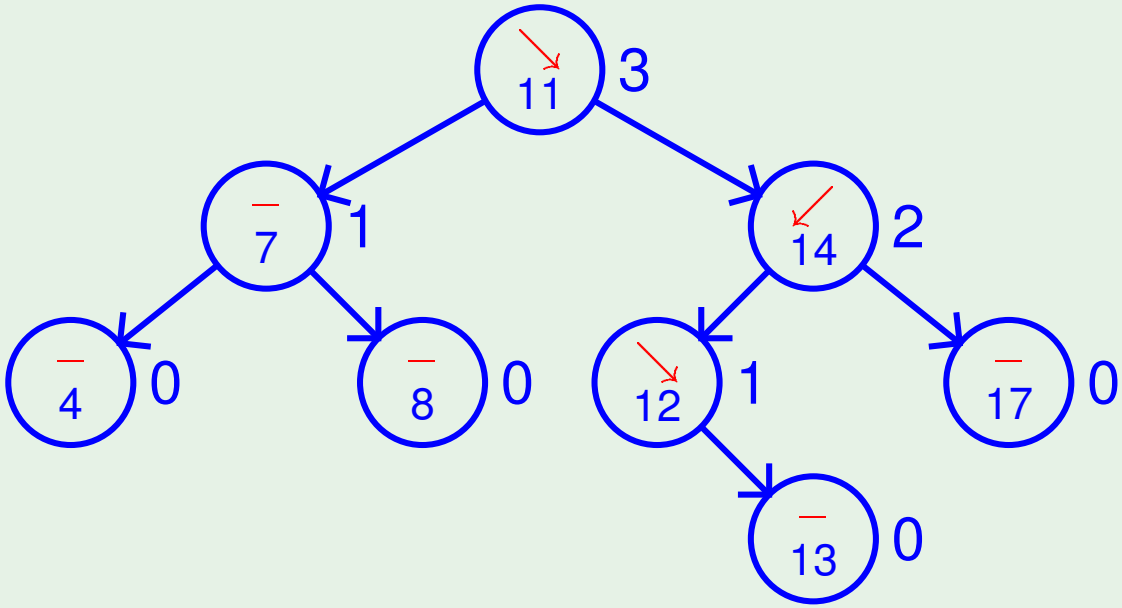
Example (Delete: 53)



Check 11: height(*L-ST*) $\geq$ height(*R-ST*), apply single rotation

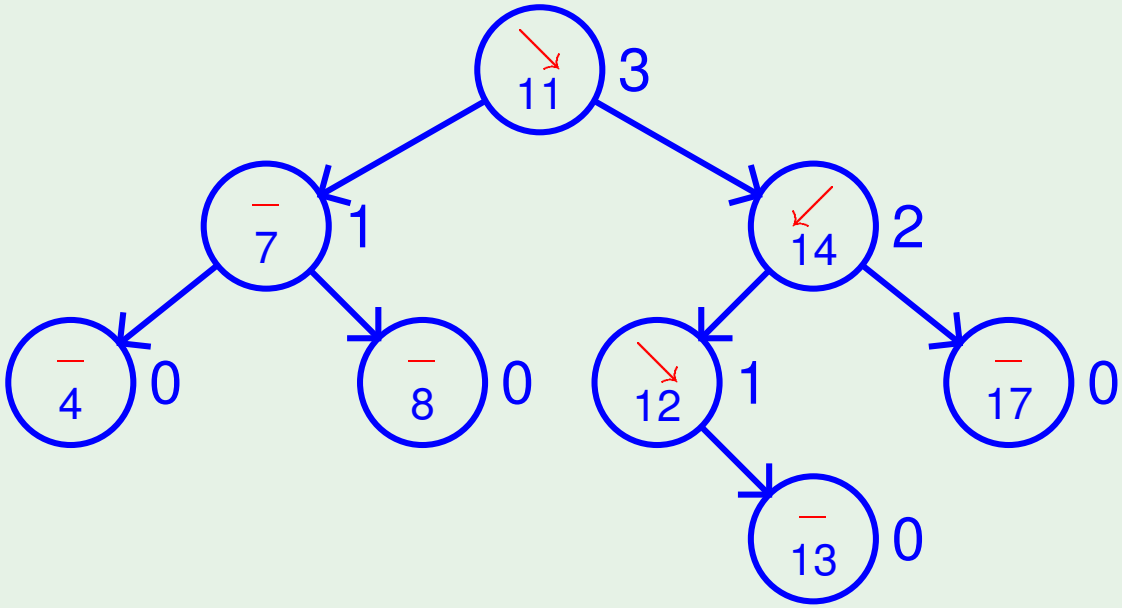
Examples of AVL tree operations (contd.)

Example (Delete: 53, 11)



Examples of AVL tree operations (contd.)

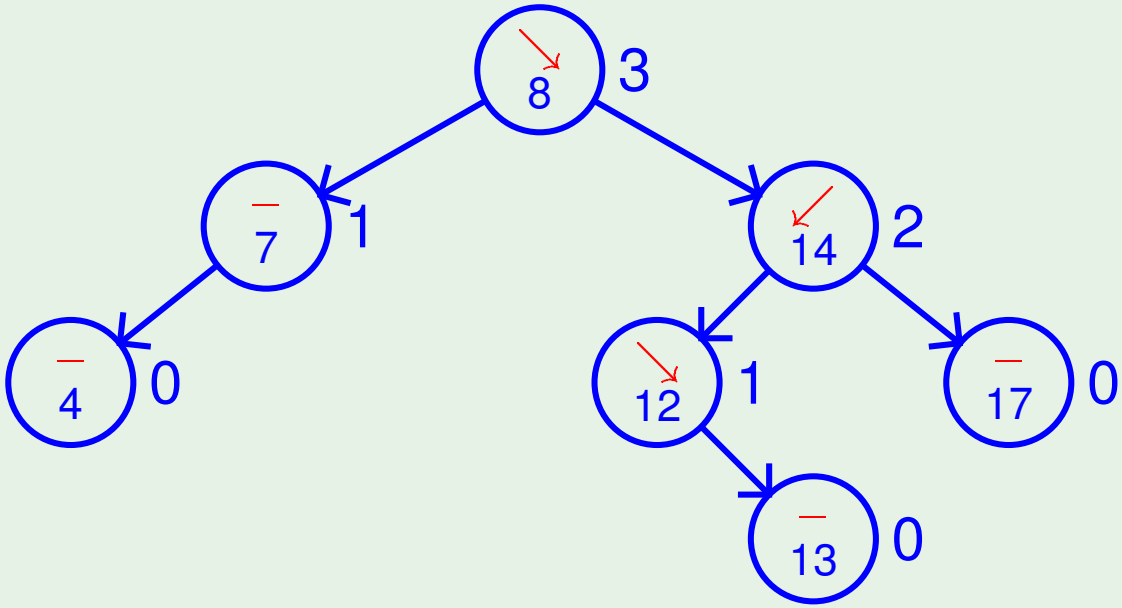
Example (Delete: 53, 11)



Double would also work as L-ST and R-ST were of equal height, but with more effort

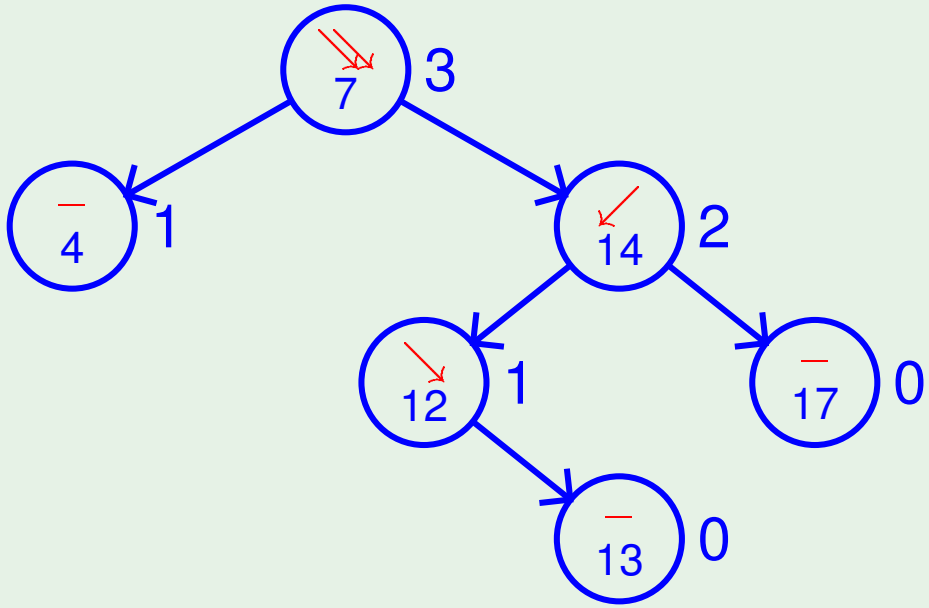
Examples of AVL tree operations (contd.)

Example (Delete: 11, 8)



Examples of AVL tree operations (contd.)

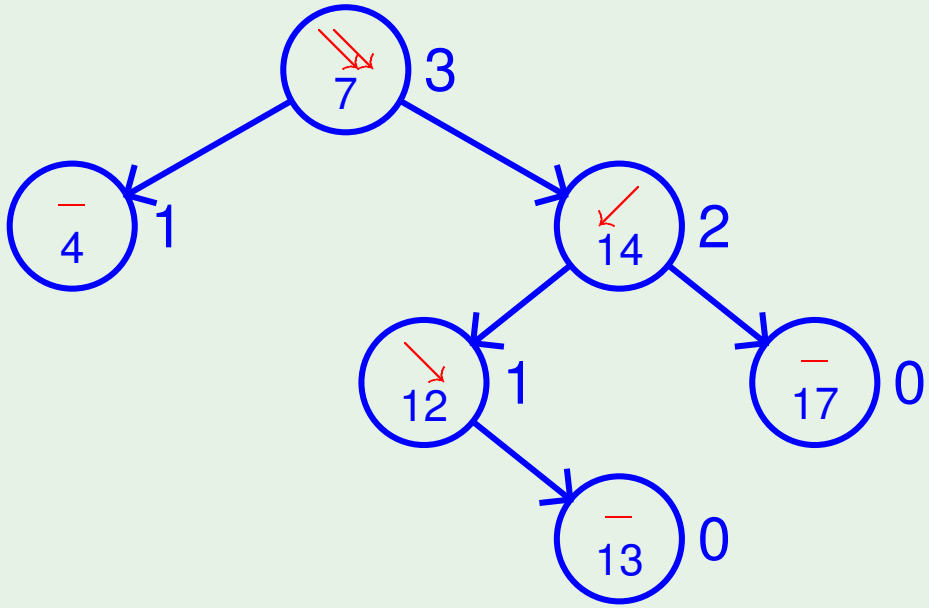
Example (Delete: 8)



Check 14:

Examples of AVL tree operations (contd.)

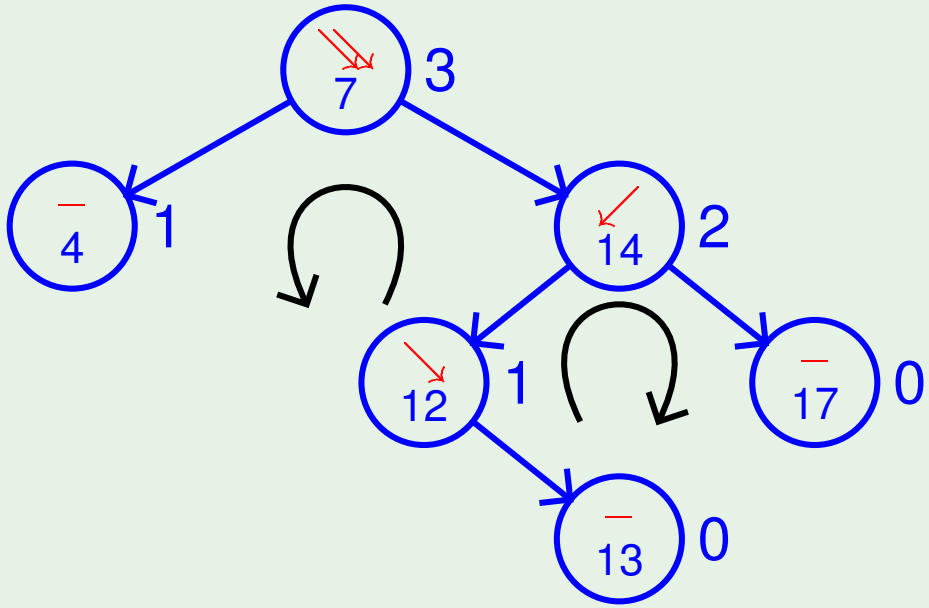
Example (Delete: 8)



Check 14: height(*L-ST*) $>$ height(*R-ST*)

Examples of AVL tree operations (contd.)

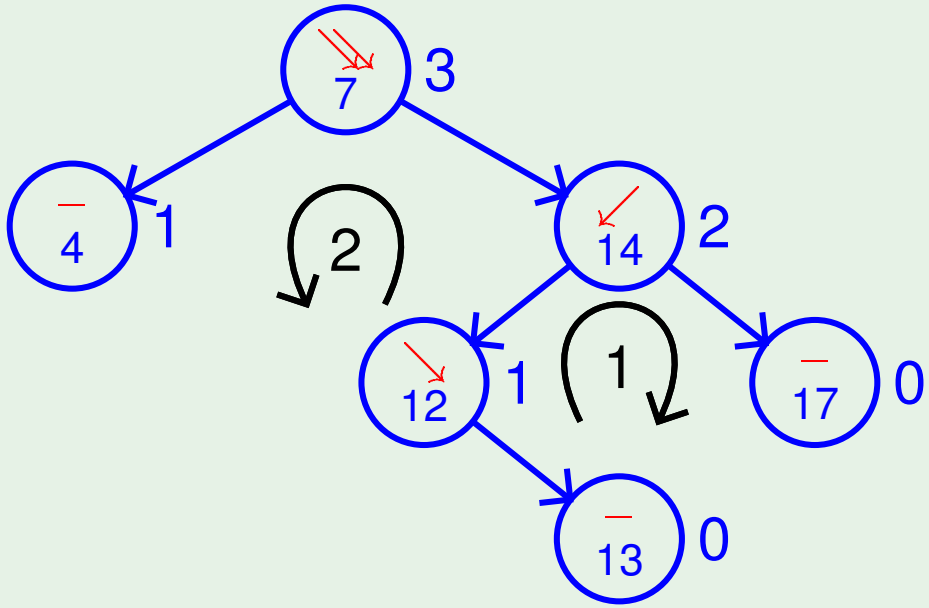
Example (Delete: 8)



Check 14: height(*L-ST*) $>$ height(*R-ST*), apply double rotation

Examples of AVL tree operations (contd.)

Example (Delete: 8)



Check 14: height(*L-ST*) $>$ height(*R-ST*), apply double rotation

Examples of AVL tree operations (contd.)

Example (Delete: 8)