Data Structures

Lecture 3*

AVL Trees

Yair Carmon

Spring semester 2022-3

* Based on slides by Uri Zwick, Haim Kaplan, Hanoch Levy , Amir Rubinstein, and TAUOnline

Definition

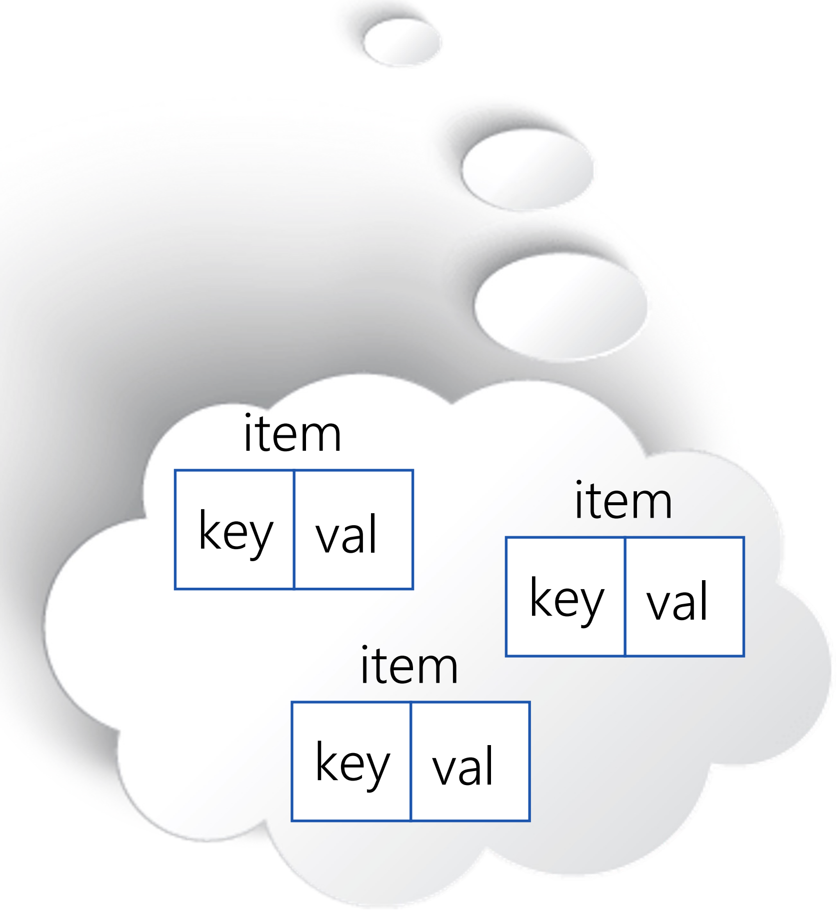
What Is an AVL tree?

ADT Dictionary - Reminder

- Maintain a set of items, with keys and associated values
  - Assume keys are distinct

- Support insert, delete, search

- If keys are totally ordered
  support min/max and successor/predecessor



ADT Dictionary - Reminder

Suppose a dictionary item x contains key and value, in the fields $x.key, x.val$

• Dictionary()	Create an empty dictionary
• Insert(D, x)	Insert x to D
• Delete(D, x)	Delete a given item x from D (assuming it exists)
• Search(D, k)	Return item with a given key k (if exists)
• Min(D)	Return item with the minimal key
• Max(D)	Return item with the maximal key
• Successor(D, x)	Return the successor of a given item x
• Predecessor(D, x)	Return the predecessor of a given item x

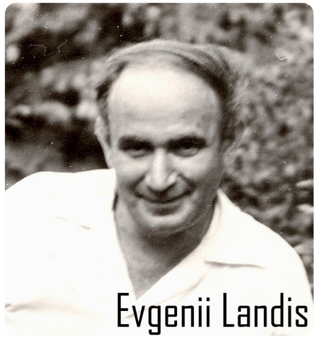
Assume that Items have distinct keys.

Motivation

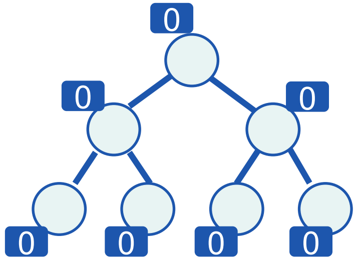
- We saw a dictionary implementation using binary search trees. The complexity of the above operations is $O(h+1)$ worst case, when h is the tree height.
  - h can be between $O(\log n)$ and $O(n)$
  - When $h = O(\log n)$ the tree is called **balanced**.

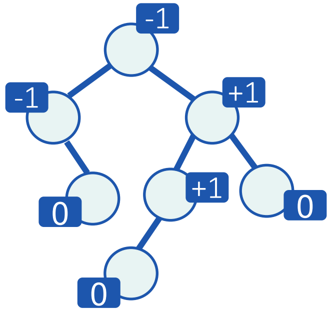
- Important examples:
  - **AVL trees** (*this lesson*) and their improved version, **WAVL trees**
  - **B trees**, **B^+ trees**
  - **Red-Black trees**

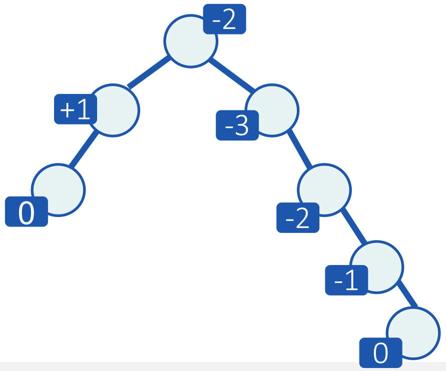
AVL Trees



AVL trees were invented by Adelson-Velsky, Landis at 1962.







<u>Balance factor</u>

$$\mathrm{BF}(v) = h(v.left) - h(v.right)$$

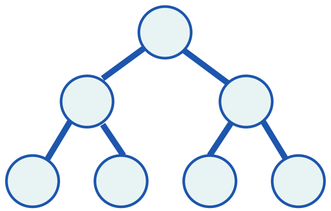
Reminder: The height of an empty tree is set to be -1.

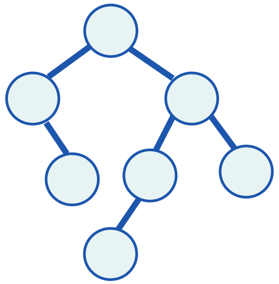
AVL Trees

Definition:
An **AVL** tree is a binary search tree where each node v maintains the following attribute:

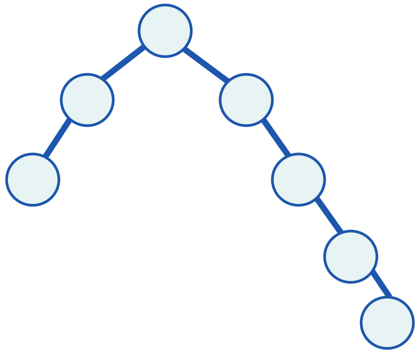
$$|BF(v)| \leq 1$$

Examples:





Non-example:



Upper Bound of an AVL Tree Height

Claim: An AVL tree with n nodes satisfies $h = O(\log n)$.

Proof: next

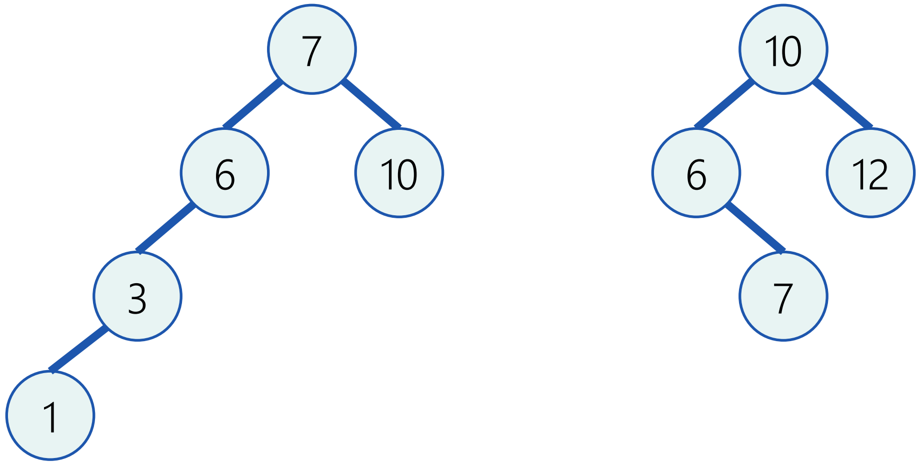
Conclusion

All queries (search, minimum/maximum, predecessor/successor) are executed in $O(\log n)$ time **worst case** in AVL trees.

AVL Tree Is a Balanced BST

And what about inserting and deleting an item?

Also in logarithmic time, but these operations may violate the tree balance and create nodes that are “AVL criminals”. For example:



Later on we will understand how to deal with this issue.

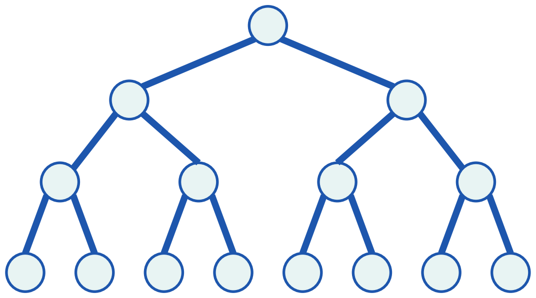
Upper Bound of an AVL Tree Height

Intuition

In a balanced tree $h = O(\log n)$. Meaning $n = \Omega(\alpha^h)$, for some constant $\alpha > 1$.

This is: the number of nodes in a balanced tree is exponential in the height of the tree.

For example, in a perfectly balanced binary search tree (complete binary tree), $h = O(\log_2 n)$ and $n = \Omega(2^h)$.



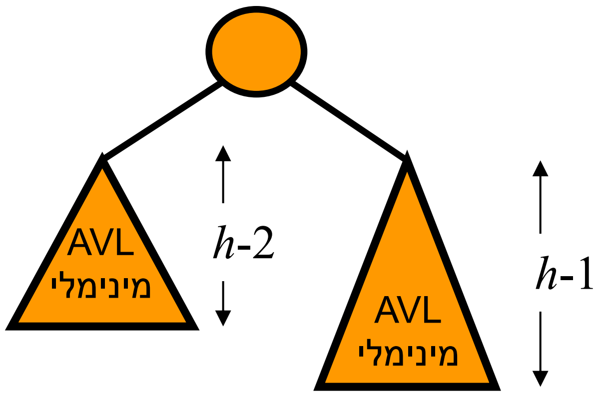
We will prove that the above holds for an **AVL tree** with the constant $\Phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ (**golden ratio**), hence $h = O(\log_\Phi n)$.

Proof That AVL is Balanced

$h = O(logn)$ for an AVL Tree

חסם לגובה עץ AVL - הוכחה

כיצד ניראה עץ AVL בגובה h בעל מספר הצמתים מינימאלי ?



(או הפוך)

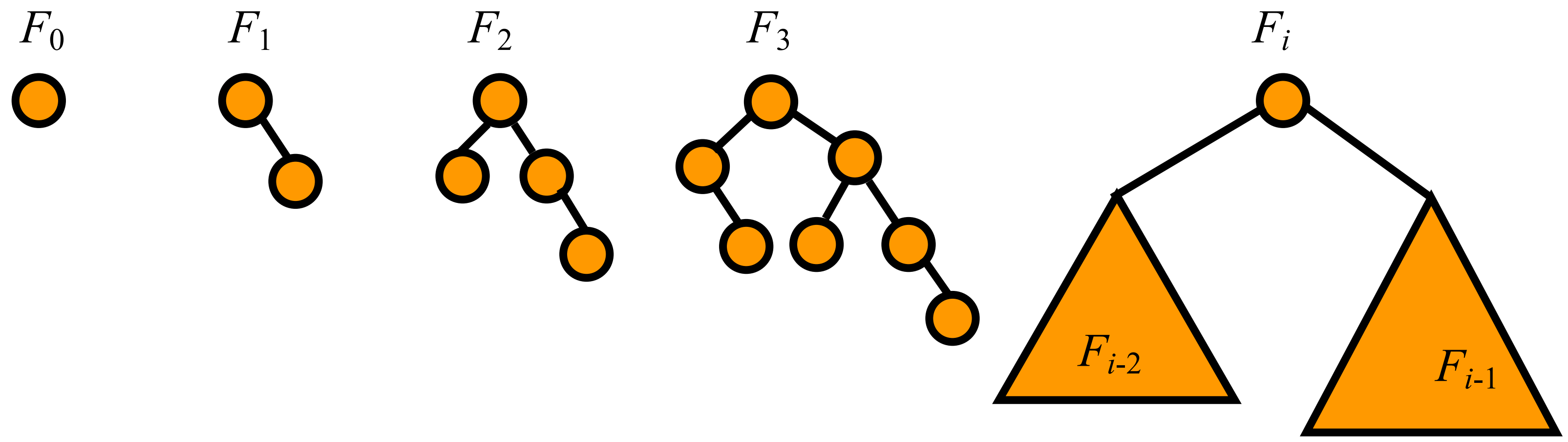
עץ כזה נקרא **<u>עץ פיבונאצ'י</u>**.

<u>תזכורת</u>: סדרת פיבונאצ'י: $f_n = f_{n-1} + f_{n-2}$ $\quad f_1 = f_2 = 1$

$$f_n = \frac{\Phi^n - \overline{\Phi}^n}{\sqrt{5}}$$ כאשר $$\Phi = \frac{1+\sqrt{5}}{2} \approx 1.618 \qquad \overline{\Phi} = 1 - \Phi = \frac{1-\sqrt{5}}{2} \approx -0.618$$

חסם לגובה עץ AVL – הוכחה (2)

הגדרת עצי פיבונאצ'י ברקורסיה:



<u>תכונות (תרגיל: הוכיחו כל אחת מהתכונות)</u>

1. גובהו של F_h הוא h.
2. $|F_h| = |F_{h\text{-}1}| + |F_{h\text{-}2}| + 1$ (כאשר $|F_i|$ הוא מספר הצמתים ב- F_i).
3. F_h הוא עץ AVL בעל מספר צמתים מינימלי מבין כל עצי ה- AVL בגובה h.

חסם לגובה עץ AVL – הוכחה (3)

<u>טענה</u>: מספר הצמתים ב- F_h הוא $\mathbf{|F_h| = f_{h+3} - 1}$ צמתים כאשר f_i הוא מספר פיבונצ'י ה- i.

<u>הוכחה</u>: באינדוקציה על h.

<u>בסיס</u>: עבור h=0 ו- h=1 הטענה מתקיימת:

f_3 -1= 2-1=1

f_4 -1= 3-1=2

<u>צעד</u>: נניח שהטענה נכונה לכל $h' < h$.

$$|F_h| = |F_{h-1}| + |F_{h-2}| + 1 = (f_{h+2} - 1) + (f_{h+1} - 1) + 1 = f_{h+3} - 1$$

חסם לגובה עץ AVL – הוכחה (4)

אם כן, עבור עץ AVL בעל n צמתים וגובה h מתקיים :

$$n \geq |F_h|$$

$$n \geq |F_h| = f_{h+3} - 1 = \frac{\Phi^{h+3} - \overline{\Phi}^{h+3}}{\sqrt{5}} - 1 \geq \frac{\Phi^{h+3}}{\sqrt{5}} - 2$$

$$\sqrt{5}(n+2) \geq \Phi^{h+3}$$

$$h + 3 \leq \log_\Phi\left(\sqrt{5}(n+2)\right)$$

$$h \leq \log_\Phi(n+2) + \log_\Phi\left(\sqrt{5}\right) - 3 = O(\log n)$$

ניתן אף להראות (עם עוד טיפה מאמץ):

$$h \leq \log_\Phi n \sim 1.44 \log_2 n$$

Insertion Into AVL

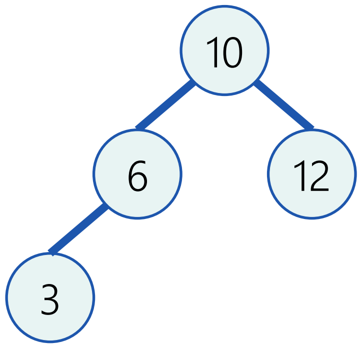
How to Keep It Balanced?

Fixing After Insertion
Rotations

What can we do if an insertion created “AVL criminals”?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:



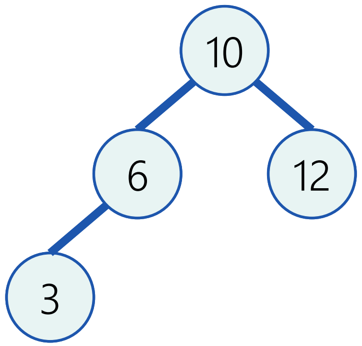
Fixing After Insertion

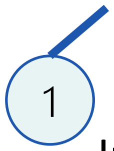
Rotations

What can we do if an insertion created “AVL criminals”?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:





Insert (1)

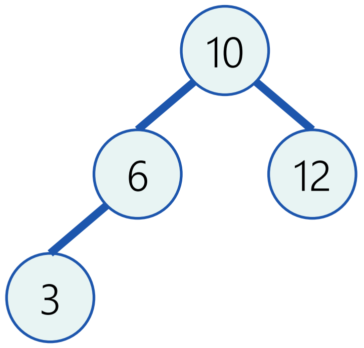
Fixing After Insertion

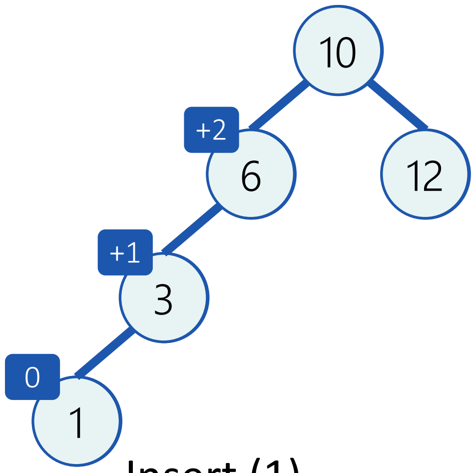
Rotations

What can we do if an insertion created “AVL criminals”?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:





Insert (1)

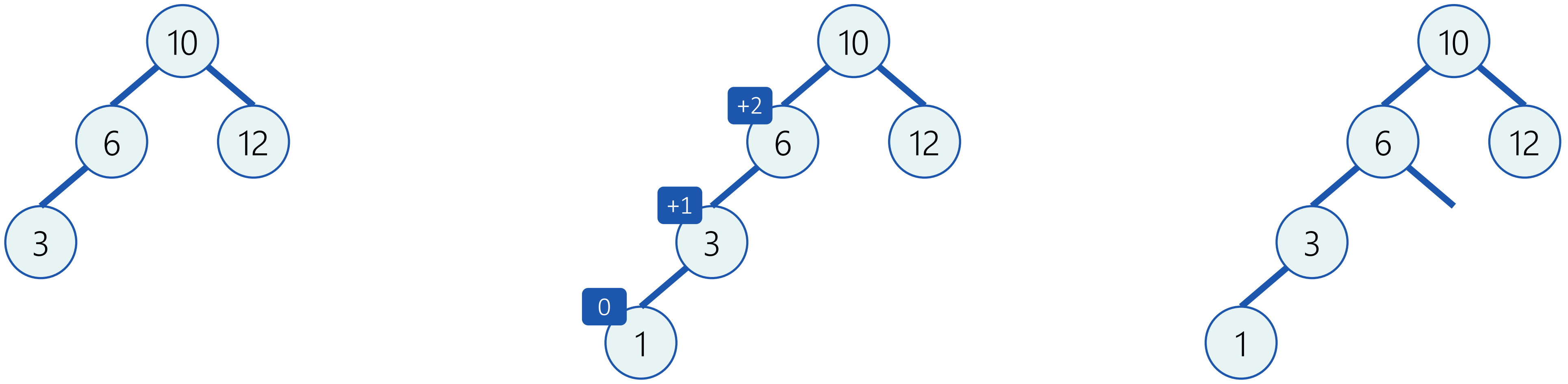
Fixing After Insertion

Rotations

What can we do if an insertion created “AVL criminals”?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:



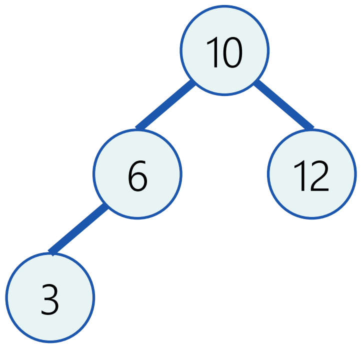
Fixing After Insertion

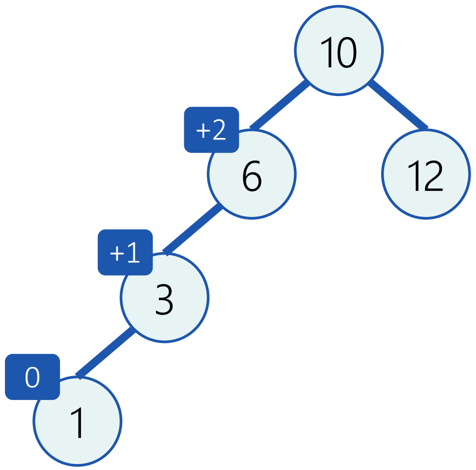
Rotations

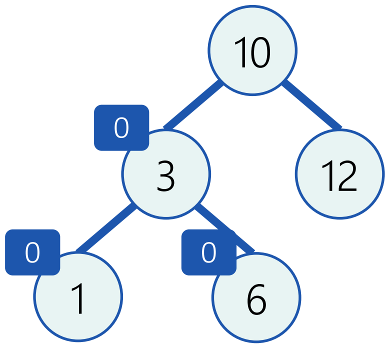
What can we do if an insertion created “AVL criminals”?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:







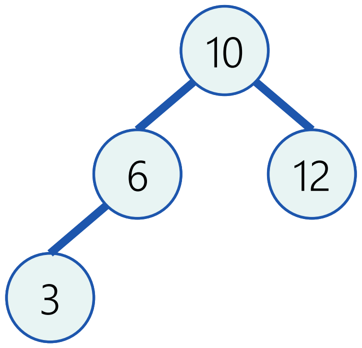
Fixing After Insertion

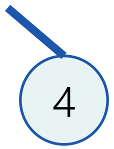
Rotations

What can we do if an insertion created "AVL criminals"?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:





Insert (4)

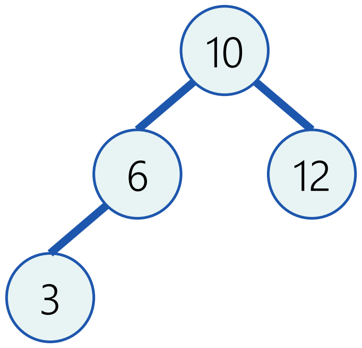
Fixing After Insertion

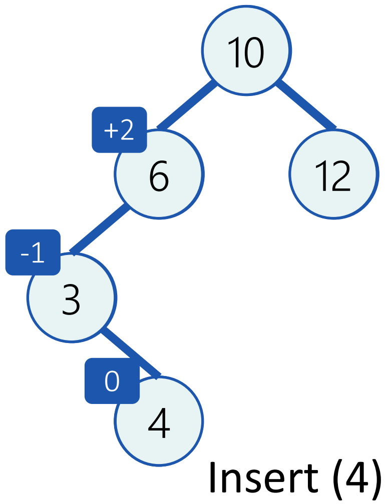
Rotations

What can we do if an insertion created “AVL criminals”?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:





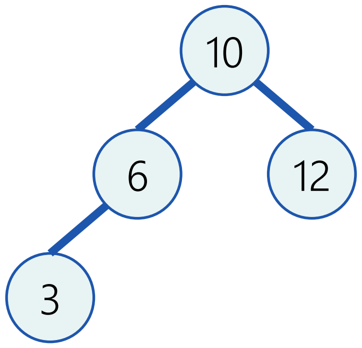
Fixing After Insertion

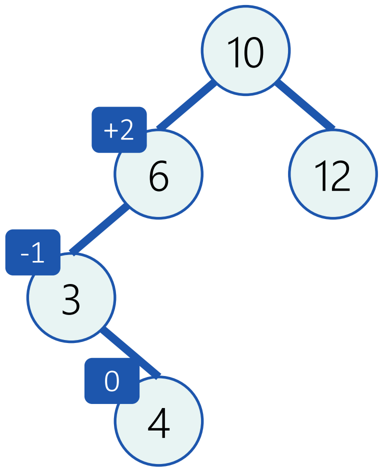
Rotations

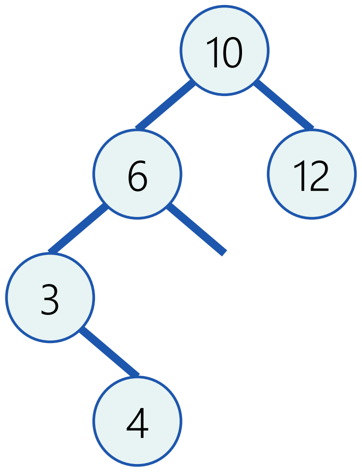
What can we do if an insertion created “AVL criminals”?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:







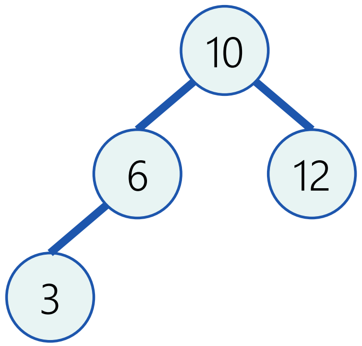
Fixing After Insertion

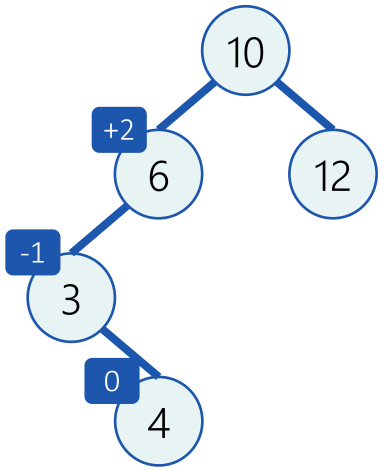
Rotations

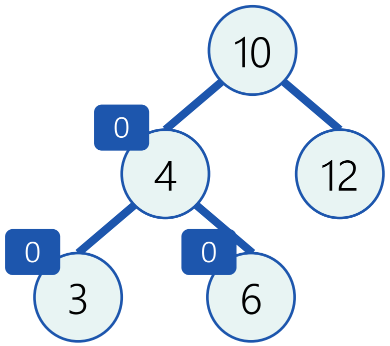
What can we do if an insertion created "AVL criminals"?

Rotation: change of a few pointers in order to balance the height difference.

Here are some simple examples:







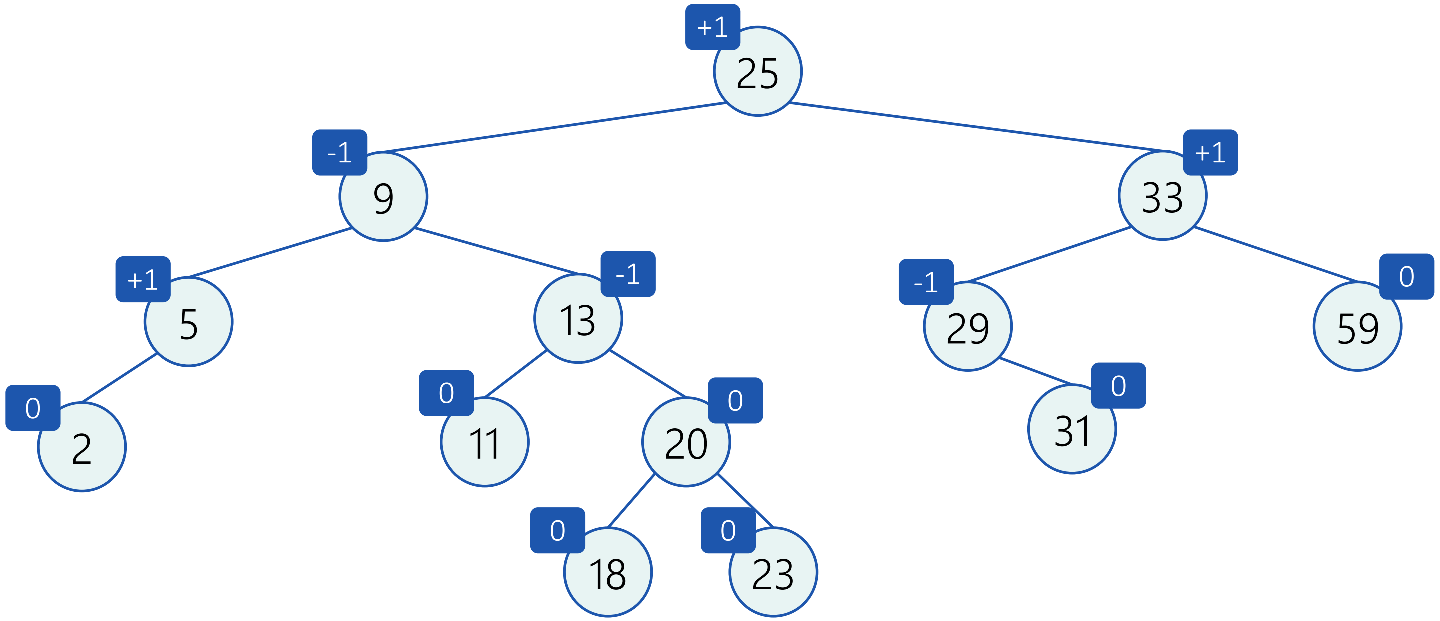
Comic relief

One more simple example: https://vt.tiktok.com/ZS8DsmBew

Fixing After Insertion

Reminder: $BF(v) = height(v.left) - height(v.right)$

In a valid AVL tree, each node v satisfies: $|BF(v)| \leq 1$

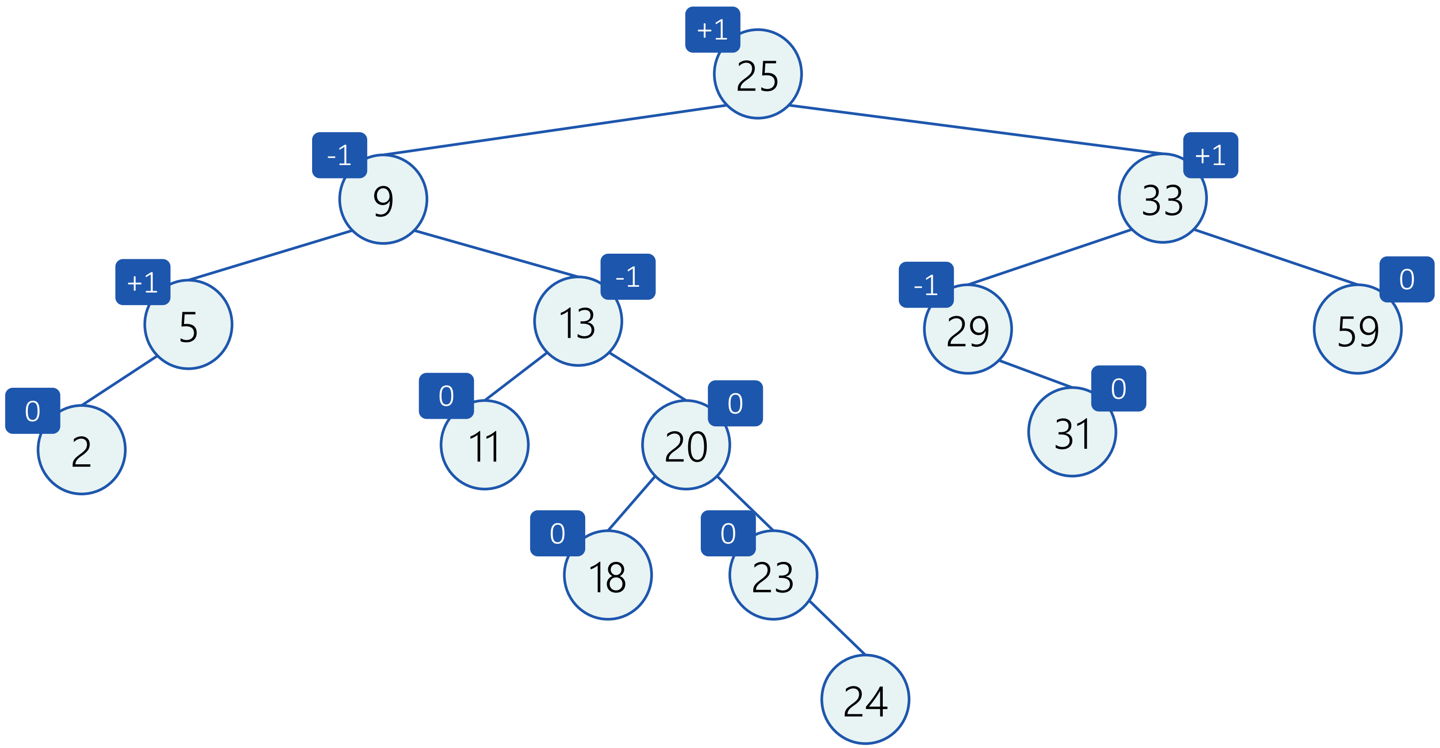


Fixing After Insertion

Observation 1

Assume we inserted 24 into the tree.

What are the possible BF values?

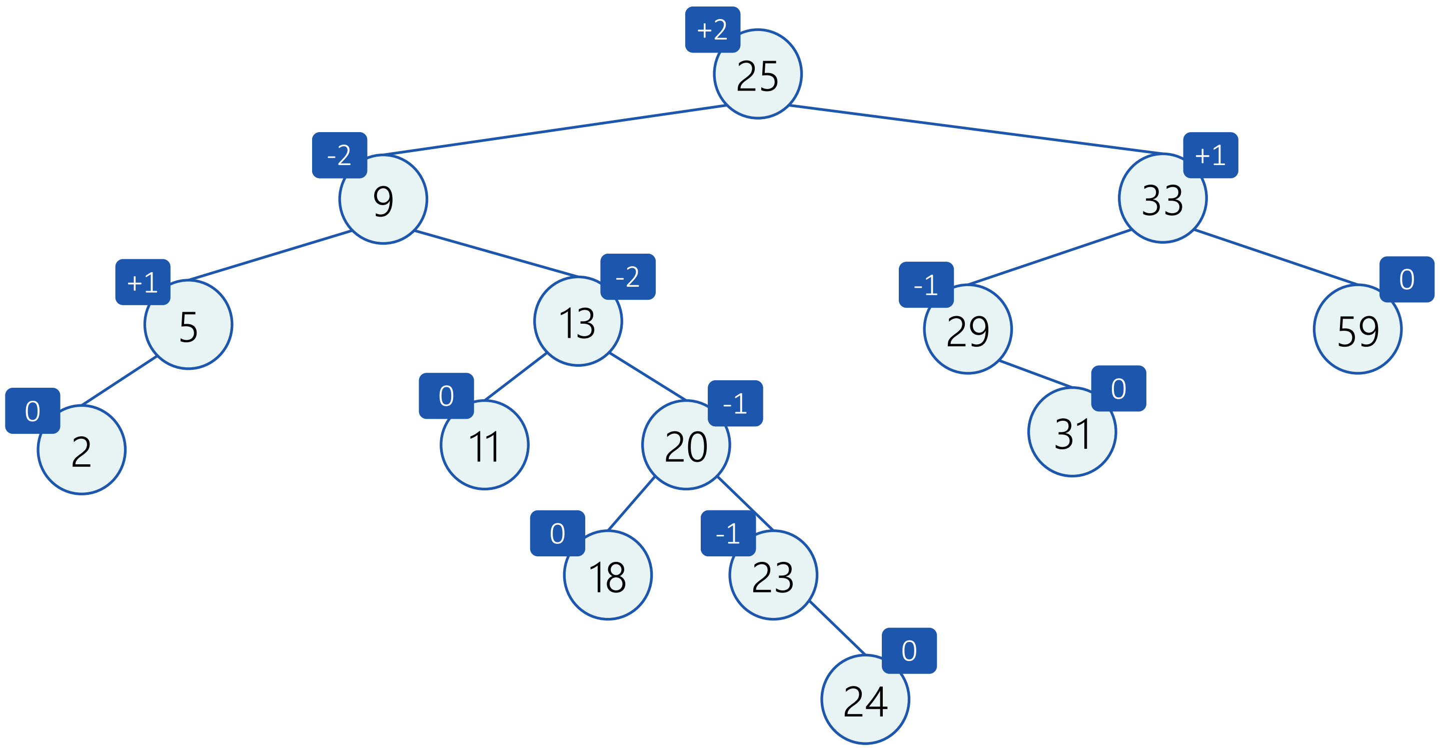


Fixing After Insertion

Observation 1

Assume we inserted 24 into the tree.

What are the possible BF values?



Fixing After Insertion

Observation 1

Assume we inserted 24 into the tree.

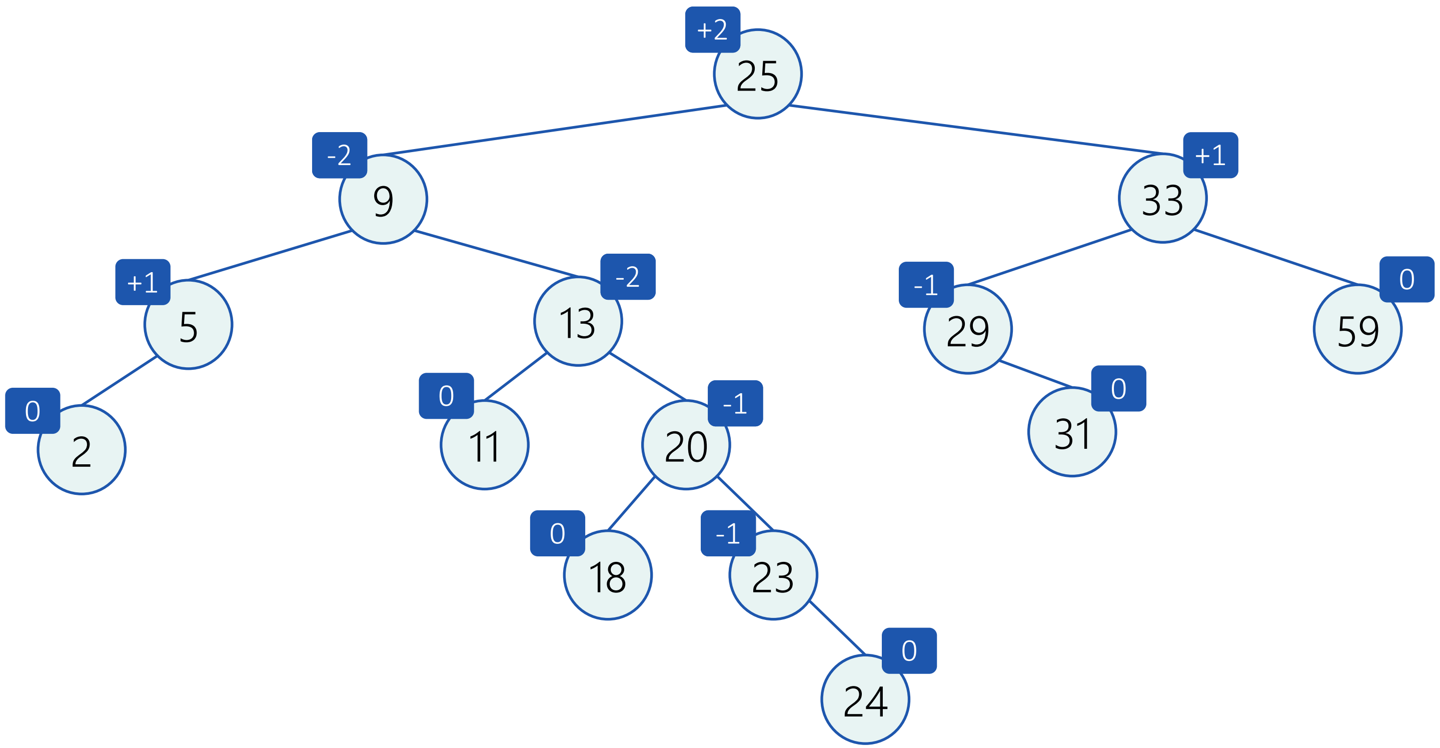
What are the possible BF values?

Observation 1:
After an insertion, the new balance factors are between -2 and +2, because they can change at most by ± 1.

Fixing After Insertion

Observation 2

Which nodes might change their BF?



Fixing After Insertion
Observation 2

Which nodes might change their BF?

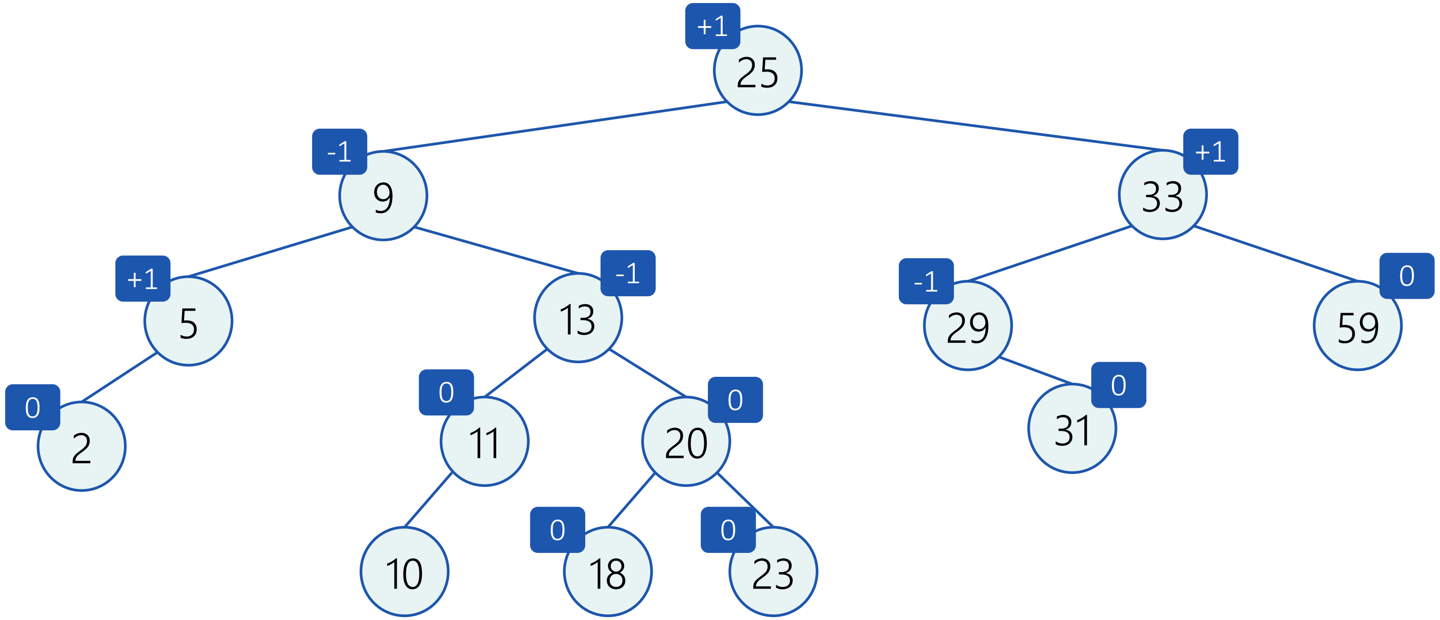
Observation 2:
The only nodes whose balance may have been changed are the nodes on the path from the root to the new node.

Fixing After Insertion

Observation 3

Do all nodes on the path from the root to the new node necessarily change their BF?

For example, inserting 10:

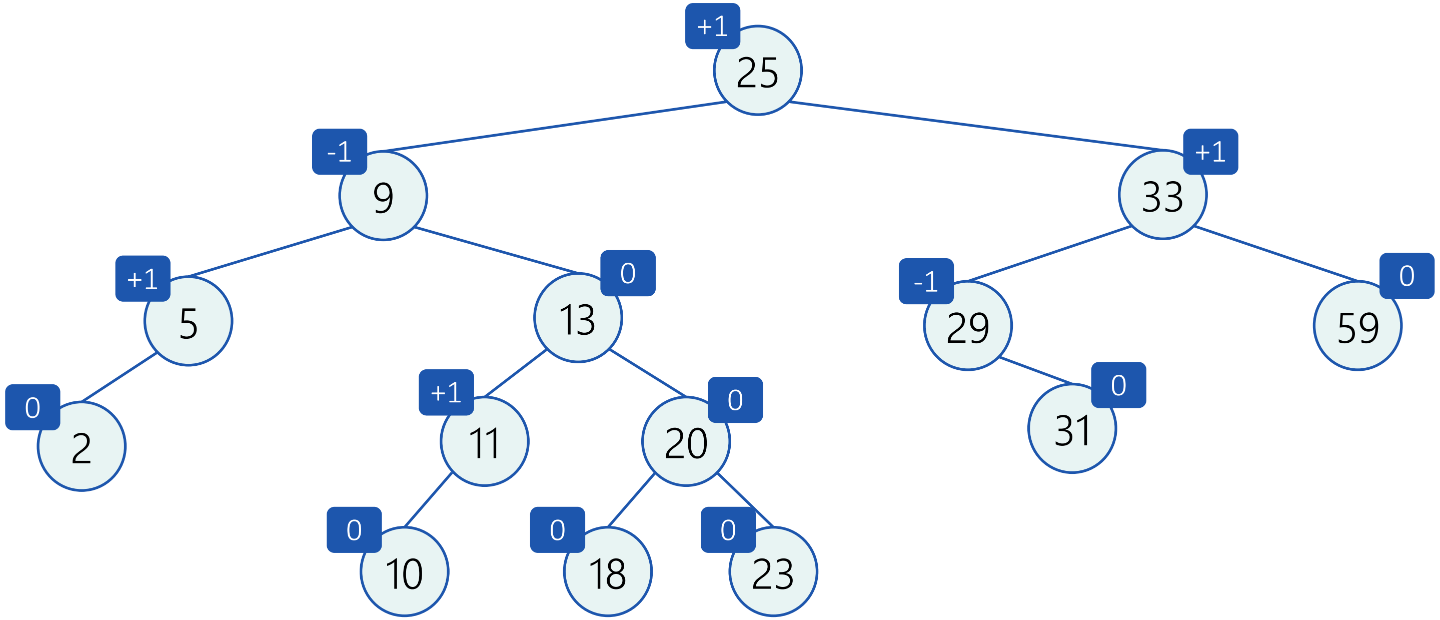


Fixing After Insertion

Observation 3

Do all nodes on the path from the root to the new node necessarily change their BF?

For example, inserting 10:



Fixing After Insertion

Observation 3

Do all nodes on the path from the root to the new node necessarily change their BF?

Observation 3:
If there's a node on the above-mentioned path, whose **height hasn't changed** after the insertion, then the **BFs** of the nodes **above** it haven't changed.

The Insertion Algorithm

The Algorithm

AVL-Insert(T, z)

1. insert z as usual (as in a BST)
2. let y be the parent of the inserted node.
3. while $y \neq Null$ do:

   3.1. compute $BF(y)$*

   3.2. if $|BF(y)| < 2$ and y's height hasn't changed: terminate Obs. 3

   3.3. else if $|BF(y)| < 2$ and y's height changed: go back to stage 3 with y's parent Obs. 2

   3.4. else ($|BF(y)| = 2$): perform a rotation and go back to stage 3 with y's parent

Obs. 1

*Requires maintaining additional information at each node (its height). We will refer to this topic later.

The Insertion Algorithm

Time Complexity

AVL-Insert(T, z)

1.	Insertion into a BST	$O(h+1)$
2.	Go up to the root to spot “AVL criminals”	$O(h+1)$
3.	Rotations	$O(1)\times O(h+1)$
		$O(h+1) = O(logn)$

Rotations

Four Rotations to Rule Them All

Part A

Fixing After Insertion

Rotations

What is the "criminal" BF?

Fixing After Insertion

Rotations

What is the “criminal” BF?

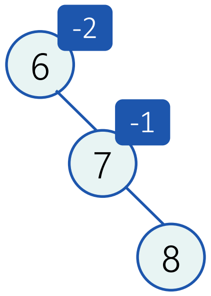
-2	+2

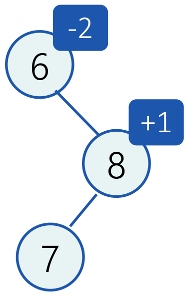
Fixing After Insertion

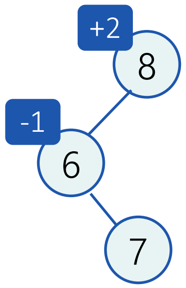
Rotations

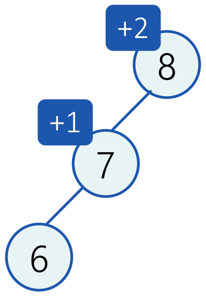
What is the “criminal” BF?

-2	+2
What is the BF of the **right** son?	What is the BF of the **left** son?







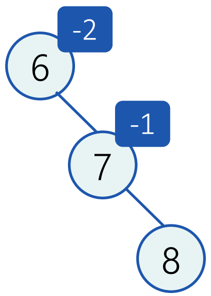


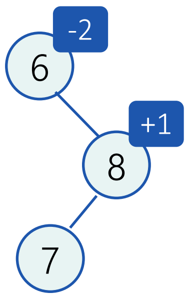
Fixing After Insertion

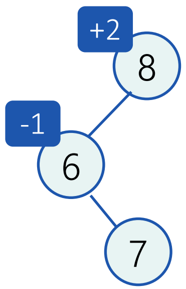
Rotations

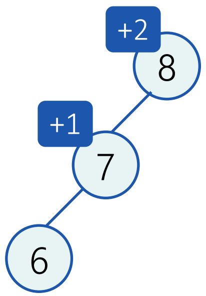
What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1	+1	-1	+1









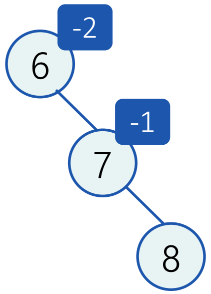
Fixing After Insertion

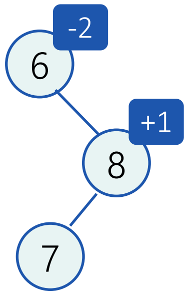
Rotations

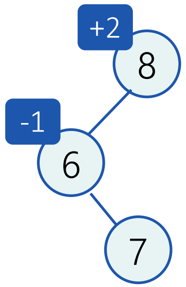
What is the “criminal” BF?

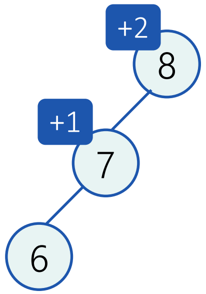
-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1	-1	+1

Before rotation

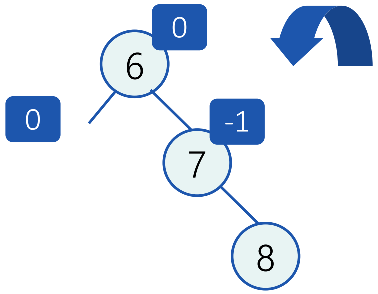








After rotation



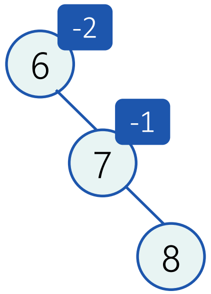
Fixing After Insertion

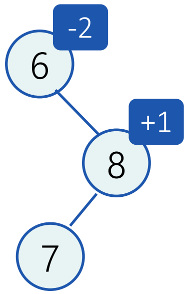
Rotations

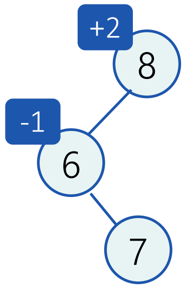
What is the “criminal” BF?

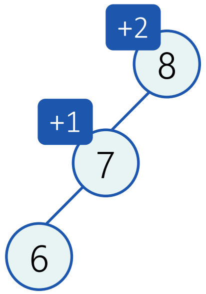
-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1	-1	+1 Right rotation

Before rotation

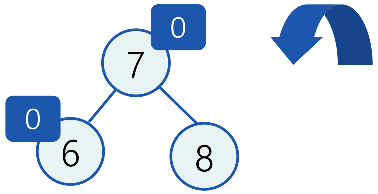


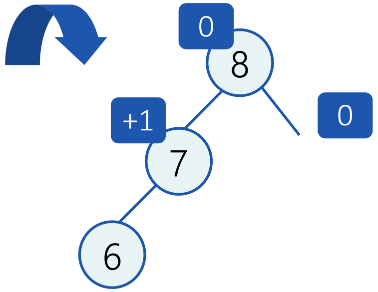






After rotation





Fixing After Insertion

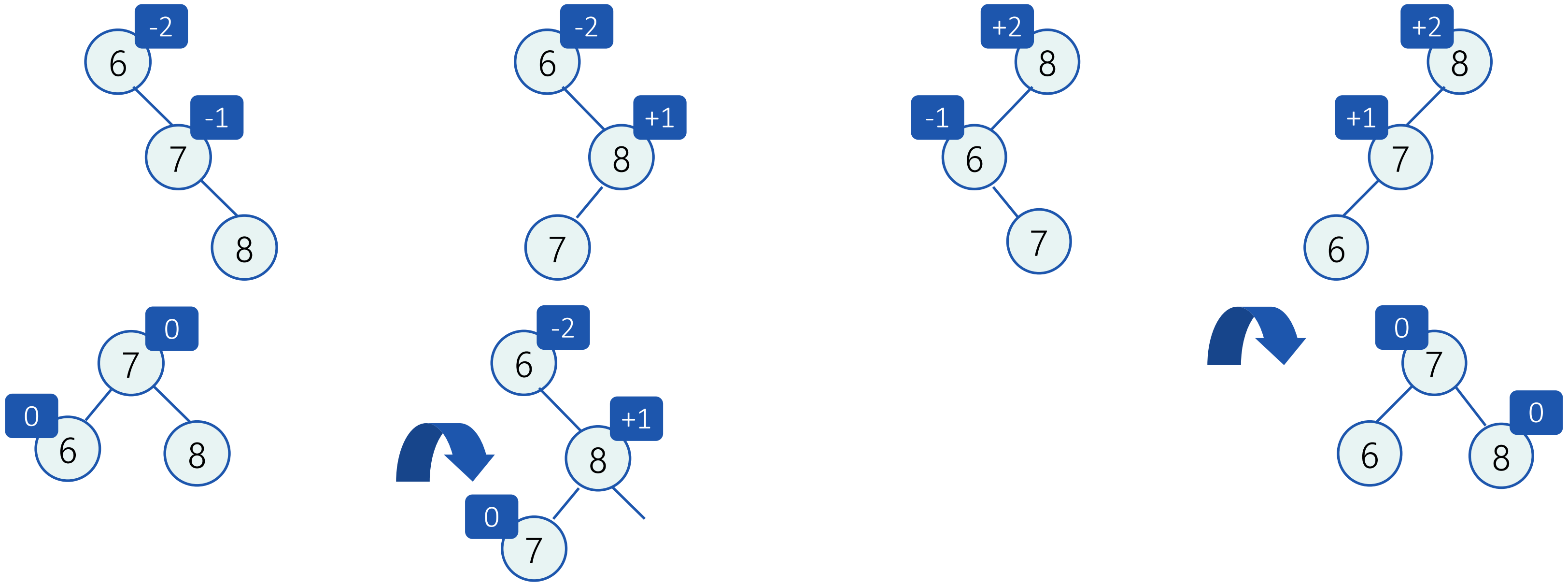
Rotations

What is the “criminal” BF?

-2		+2	
What is the BF of the right son?		**What is the BF of the left son?**	
-1 Left rotation	+1 Right then left rotation	-1	+1 Right rotation

Before rotation

After rotation



Fixing After Insertion

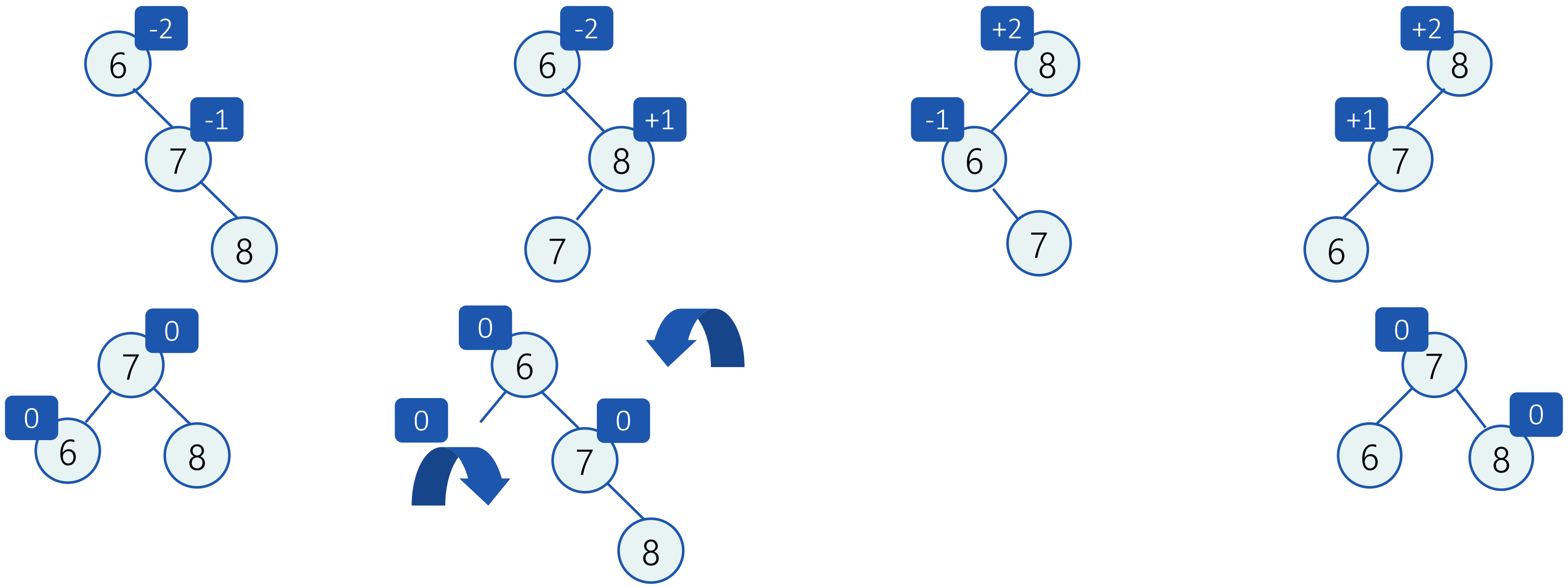
Rotations

What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1 Right then left rotation	-1	+1 Right rotation

Before rotation

After rotation



Fixing After Insertion

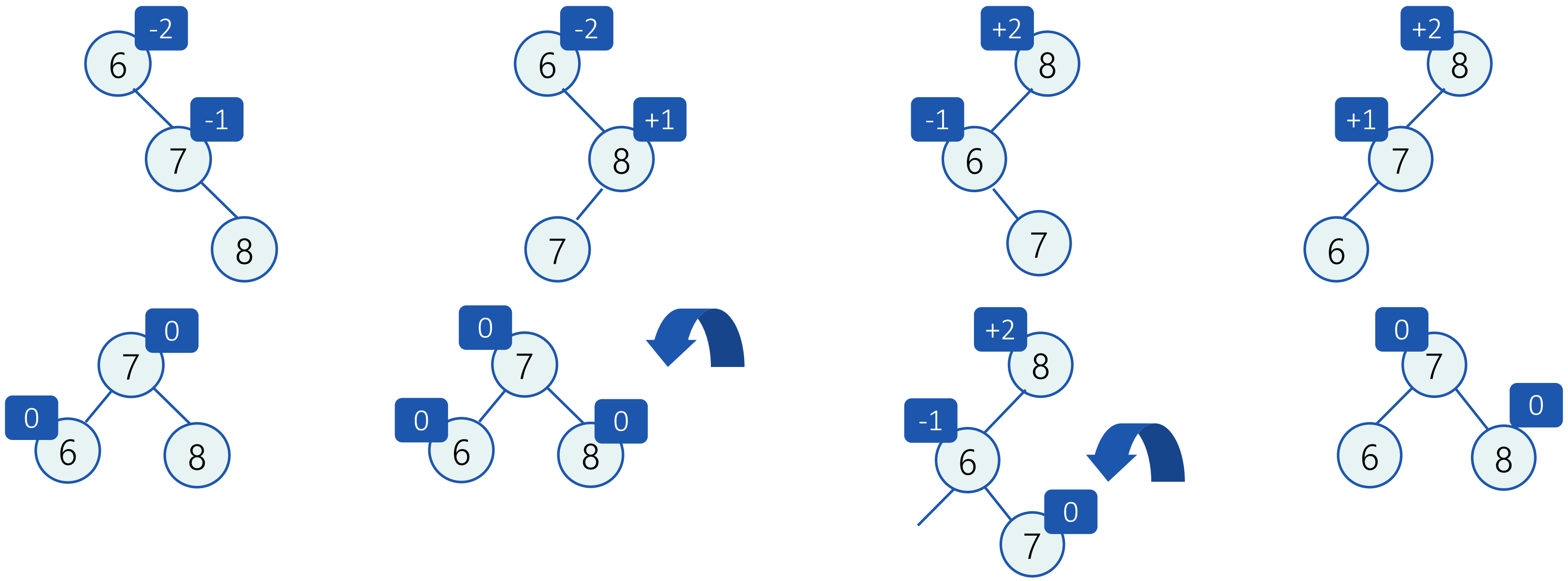
Rotations

What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1 Right then left rotation	-1 Left then right rotation	+1 Right rotation

Before rotation

After rotation



Fixing After Insertion

Rotations

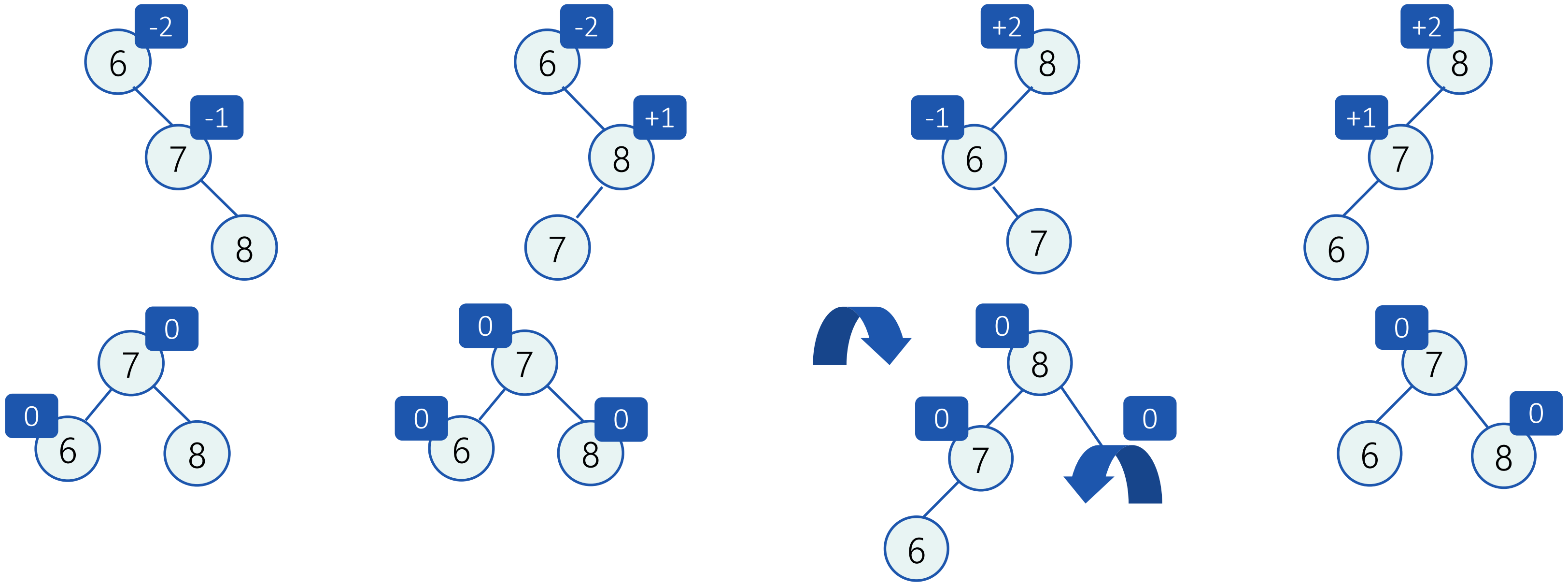
What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1 Right then left rotation	-1 Left then right rotation	+1 Right rotation

Why can’t the son BF be **0**?

Before rotation

After rotation



Rotations

Four Rotations to Rule Them All

Part B

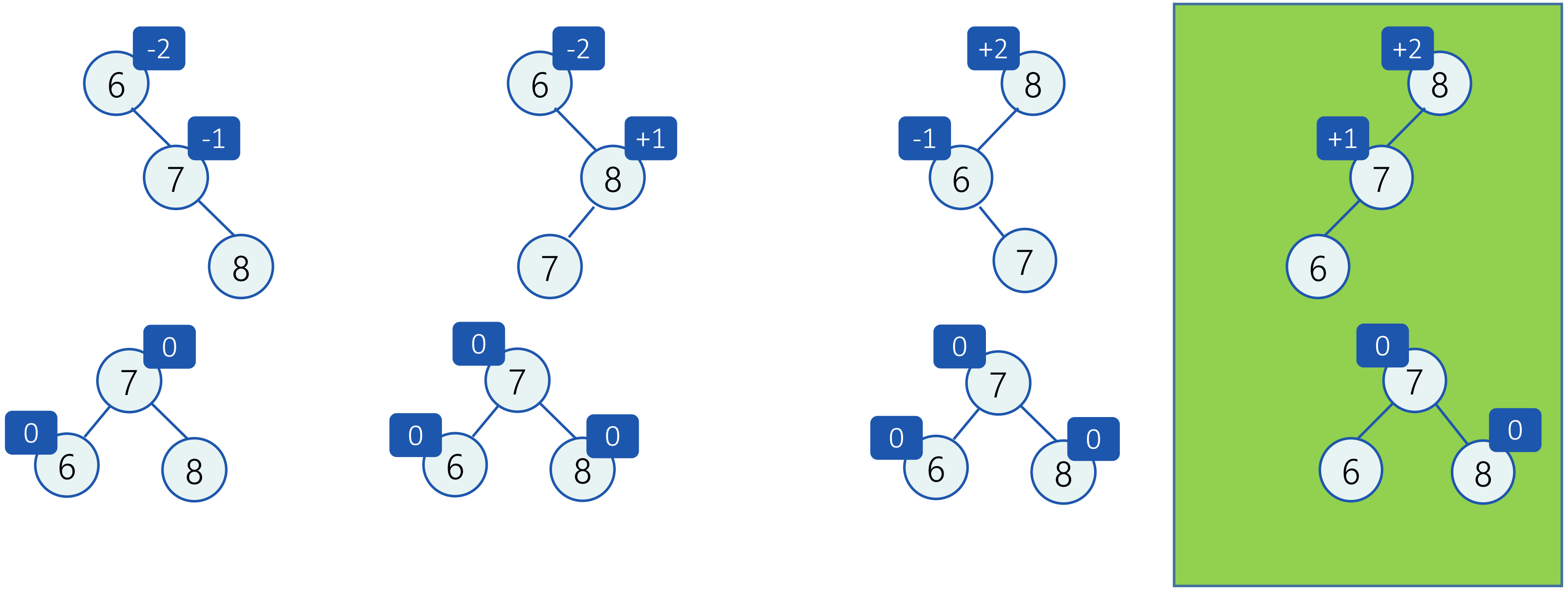
Fixing After Insertion
Rotations

What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1 Right then left rotation	-1 Left then right rotation	+1 Right rotation

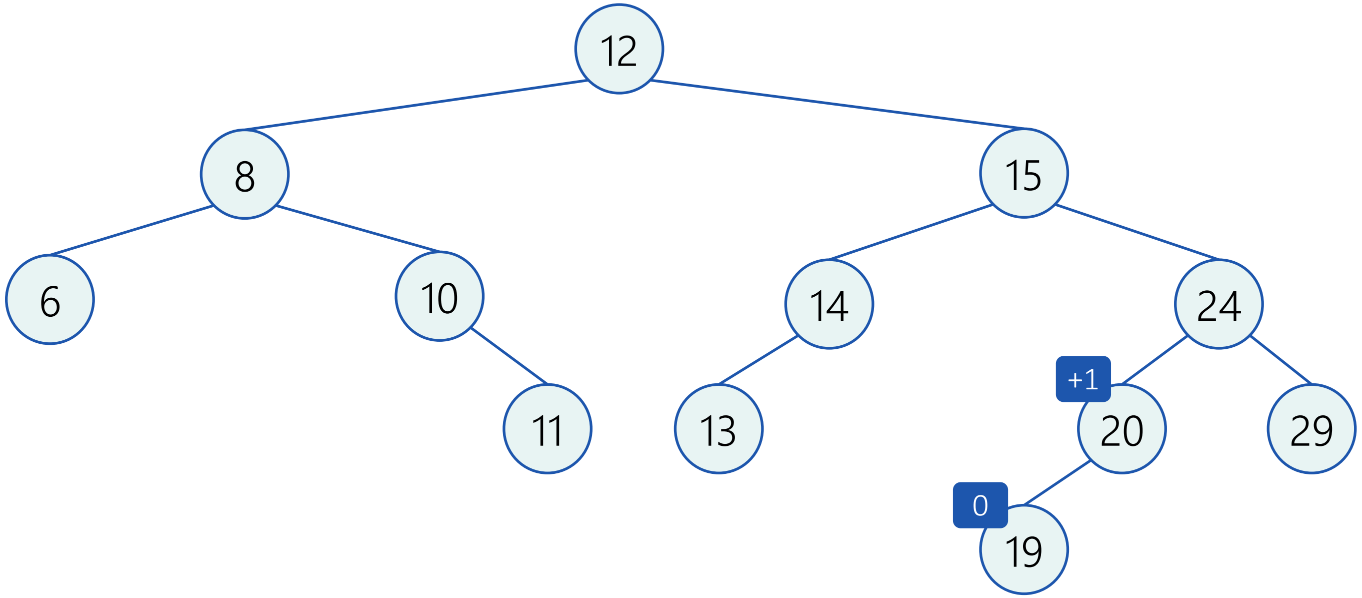
Before rotation

After rotation



Right Rotation

Example



Before insertion

Insert 18

Temporarily after insertion

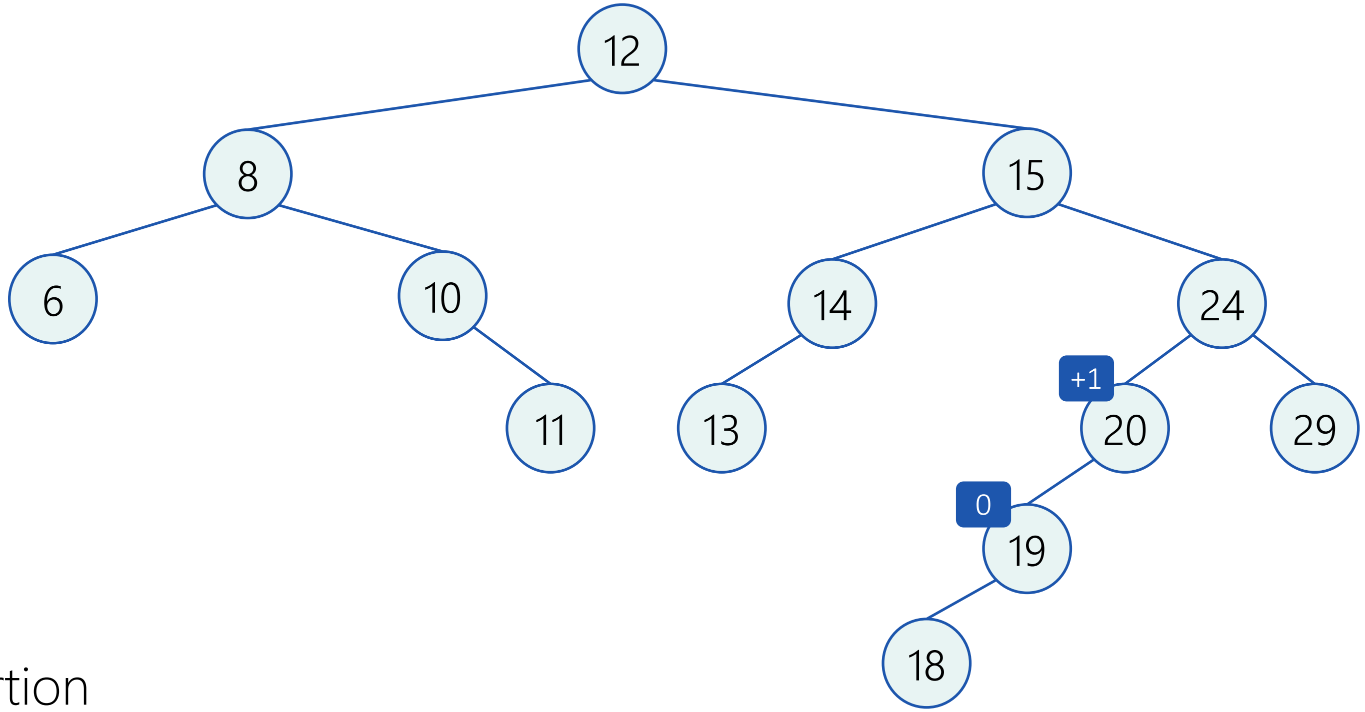
Rotate Right

After Rotation

[Interactive](https://visualgo.net) (https://visualgo.net)

Right Rotation

Example



Before insertion

Insert 18

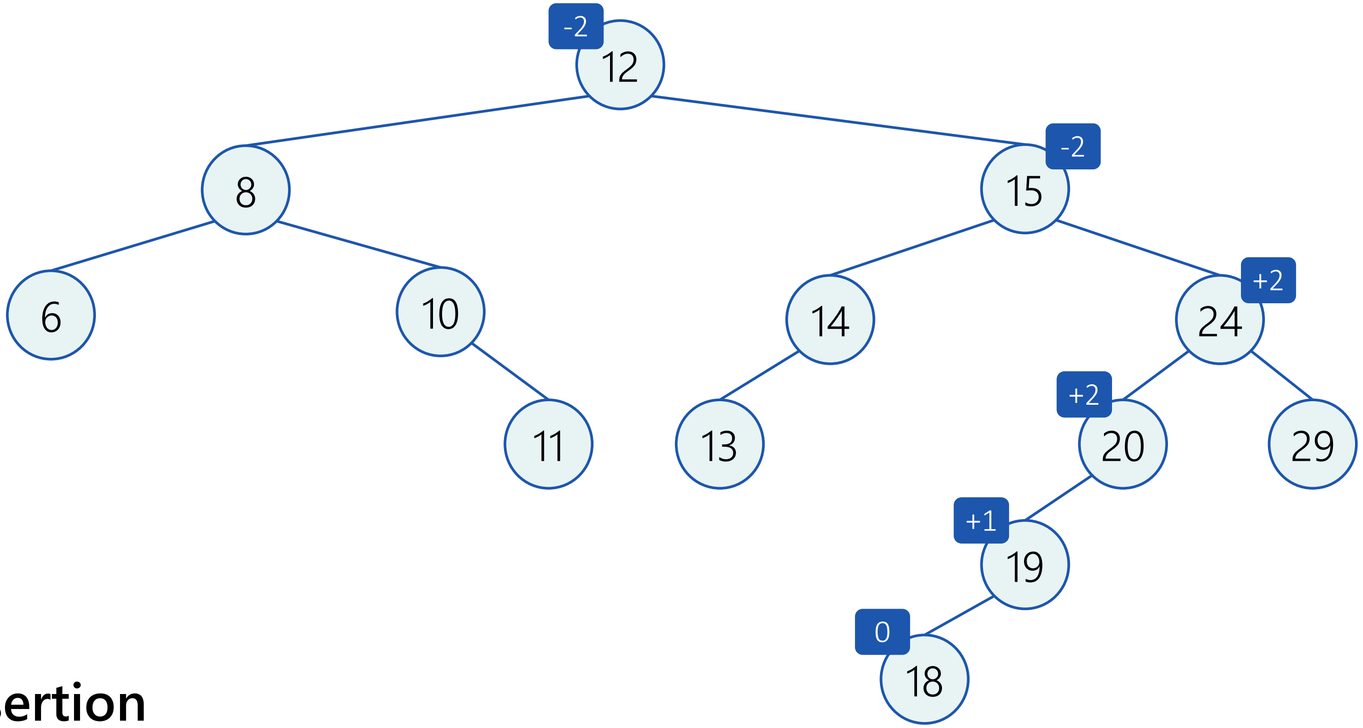
Temporarily after insertion

Rotate Right

After Rotation

Right Rotation

Example



Before insertion

Insert 18

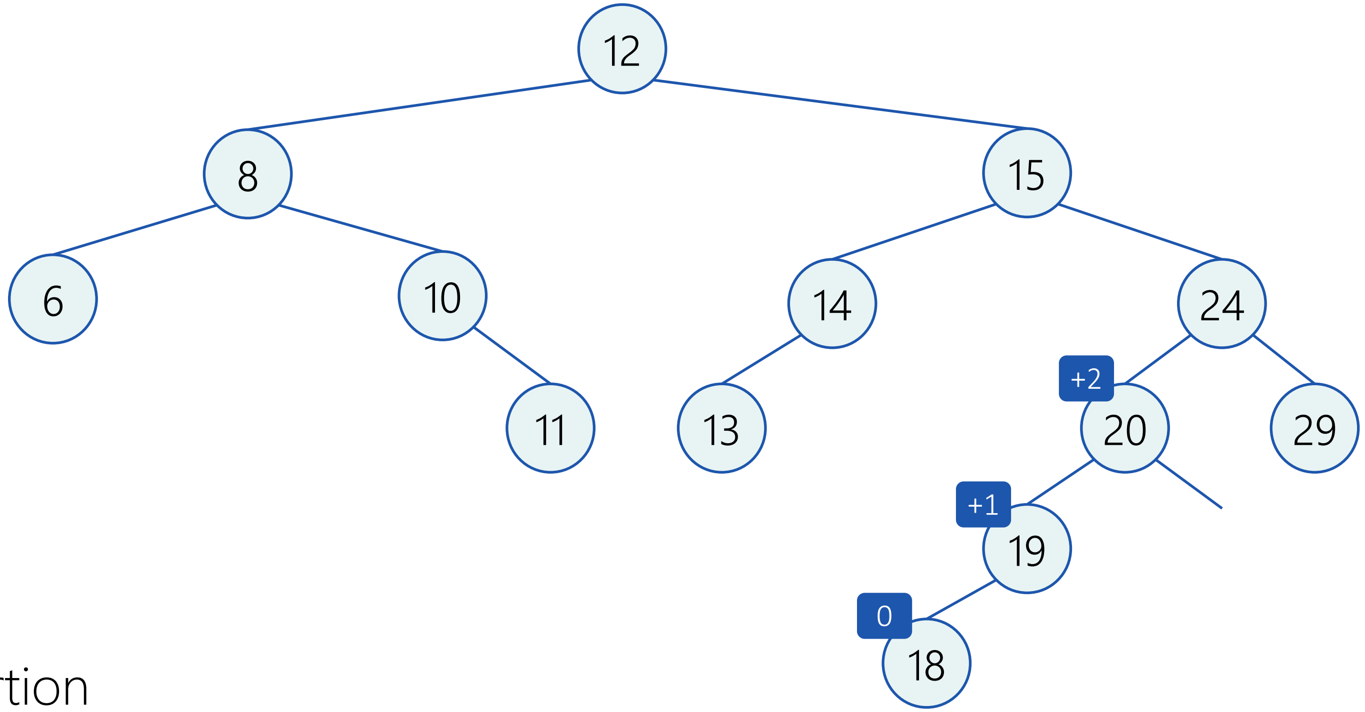
Temporarily after insertion

Rotate Right

After Rotation

Right Rotation

Example



Before insertion

Insert 18

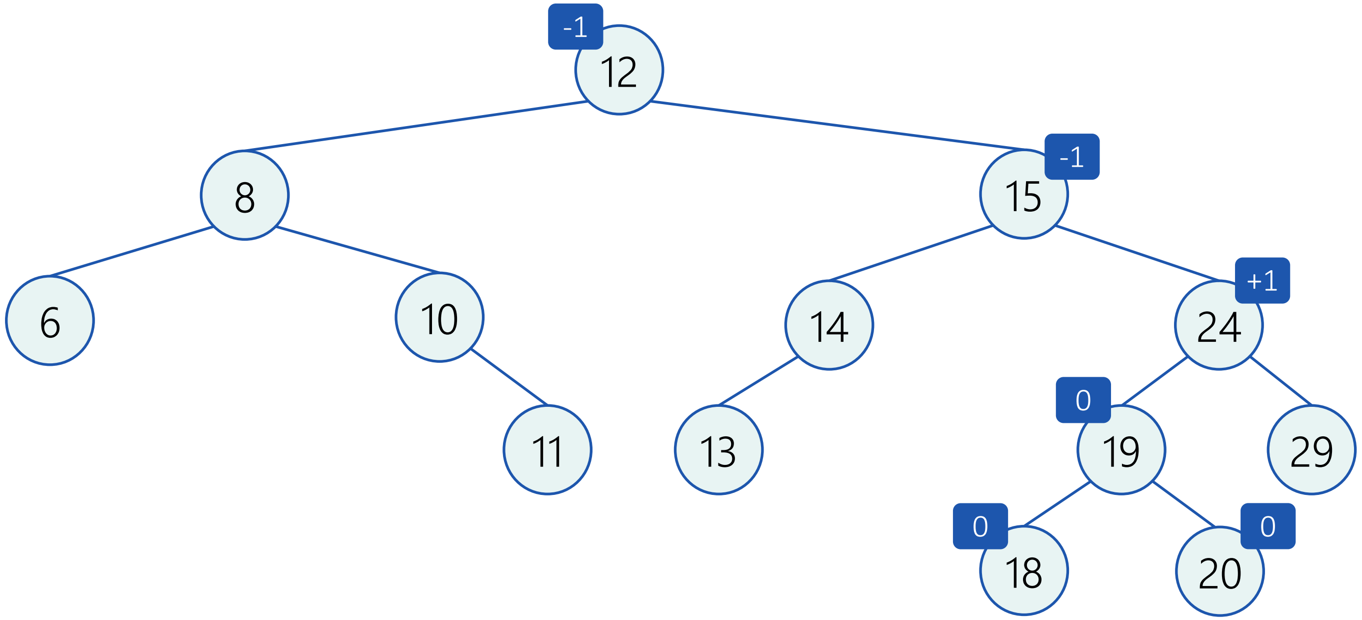
Temporarily after insertion

Rotate Right

After Rotation

Right Rotation

Example



Before insertion

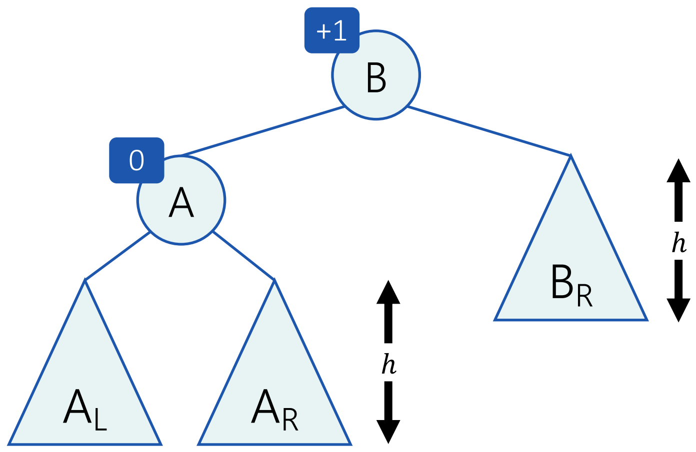
Insert 18

Temporarily after insertion

Rotate Right

After Rotation

Right Rotation



Before insertion

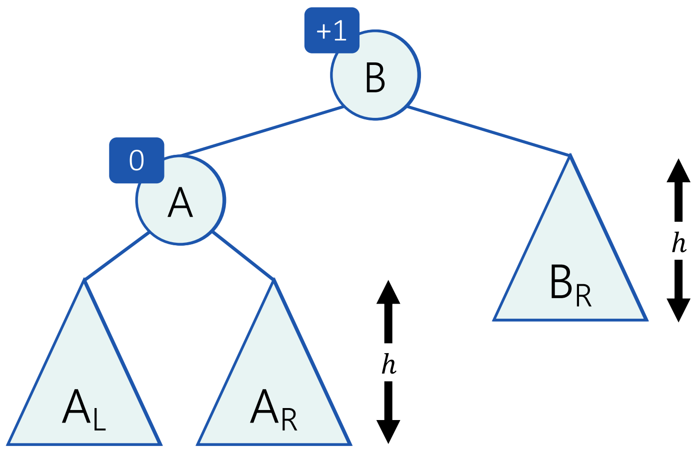
Insert v

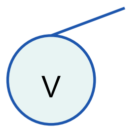
Temporarily after insertion

Rotate right

After rotation

Right Rotation





Before insertion

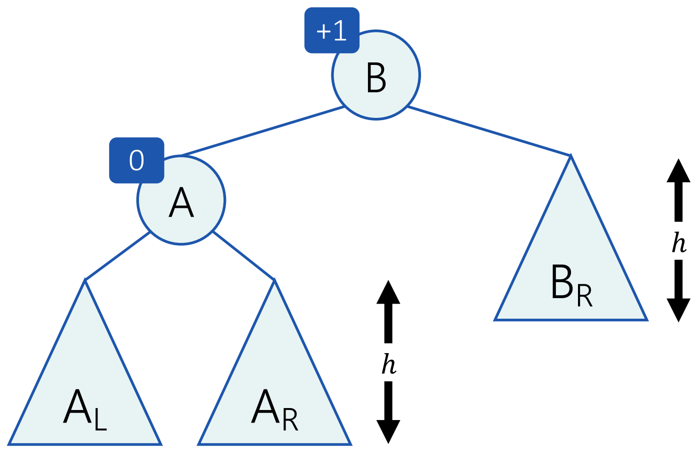
Insert v

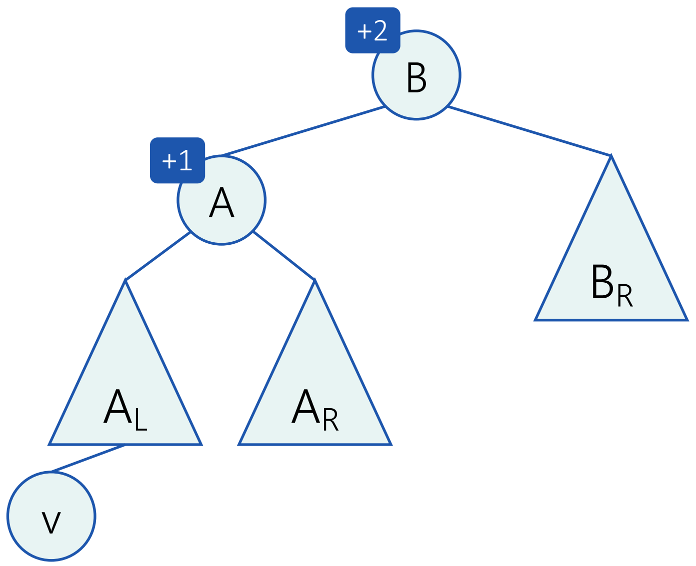
Temporarily after insertion

Rotate right

After rotation

Right Rotation





Before insertion

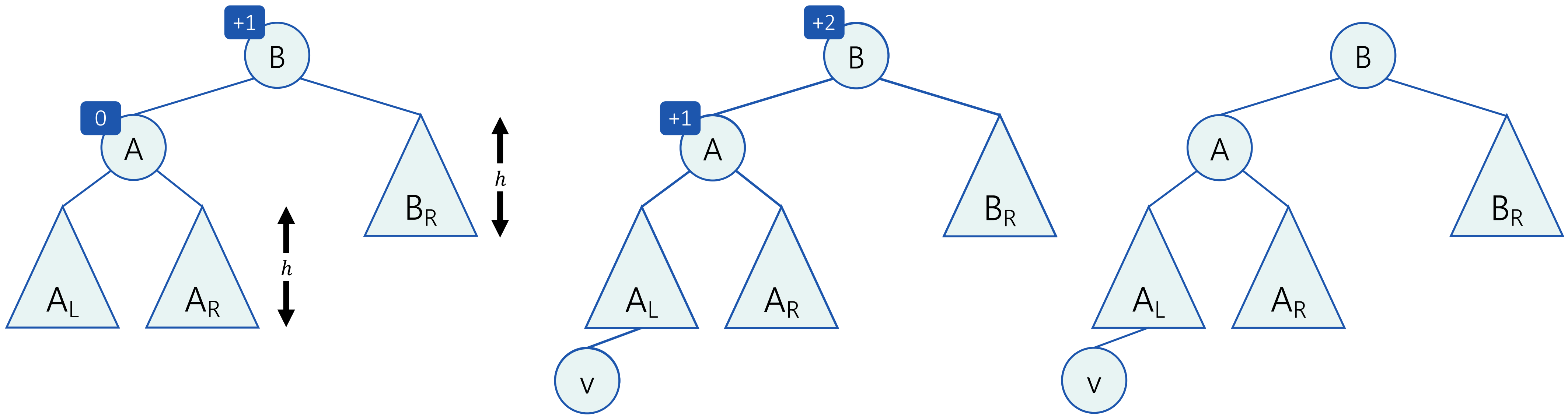
Insert v

Temporarily after insertion

Rotate right

After rotation

Right Rotation



Before insertion

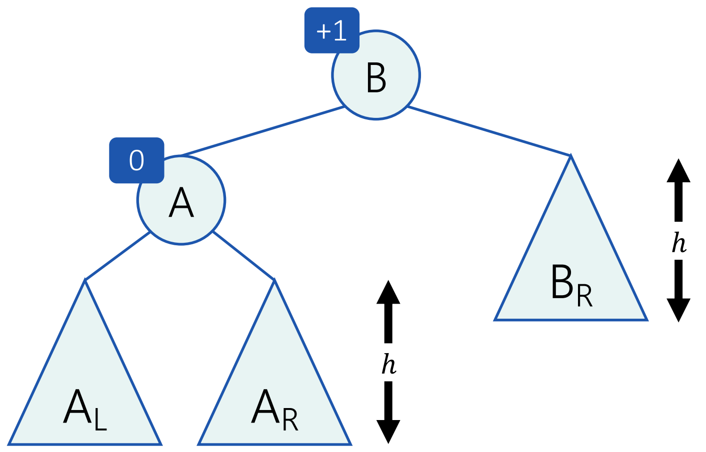
Insert v

Temporarily after insertion

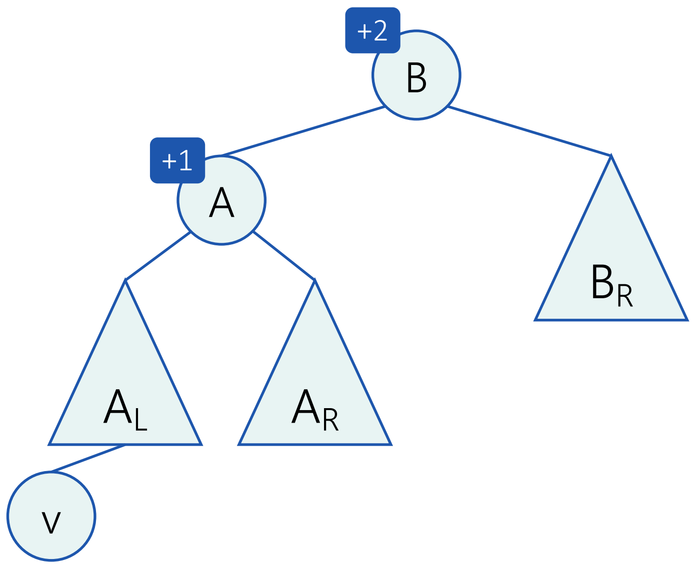
Rotate right

After rotation

Right Rotation

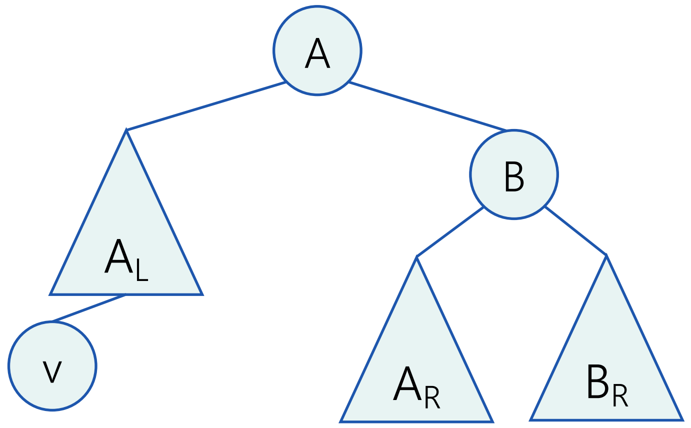


Before insertion



Insert v

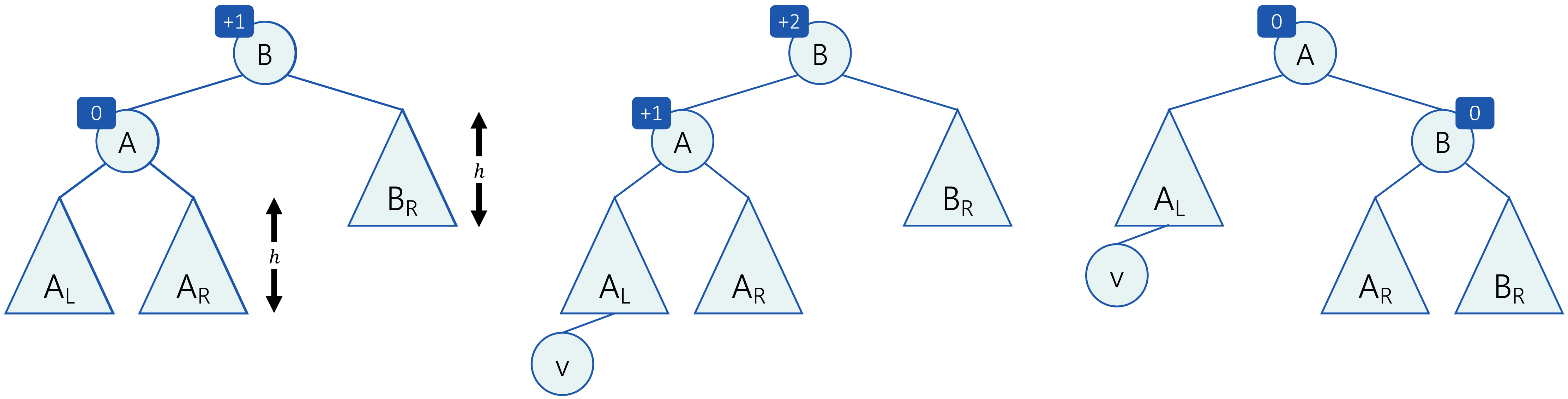
Temporarily after insertion



Rotate right

After rotation

Right Rotation

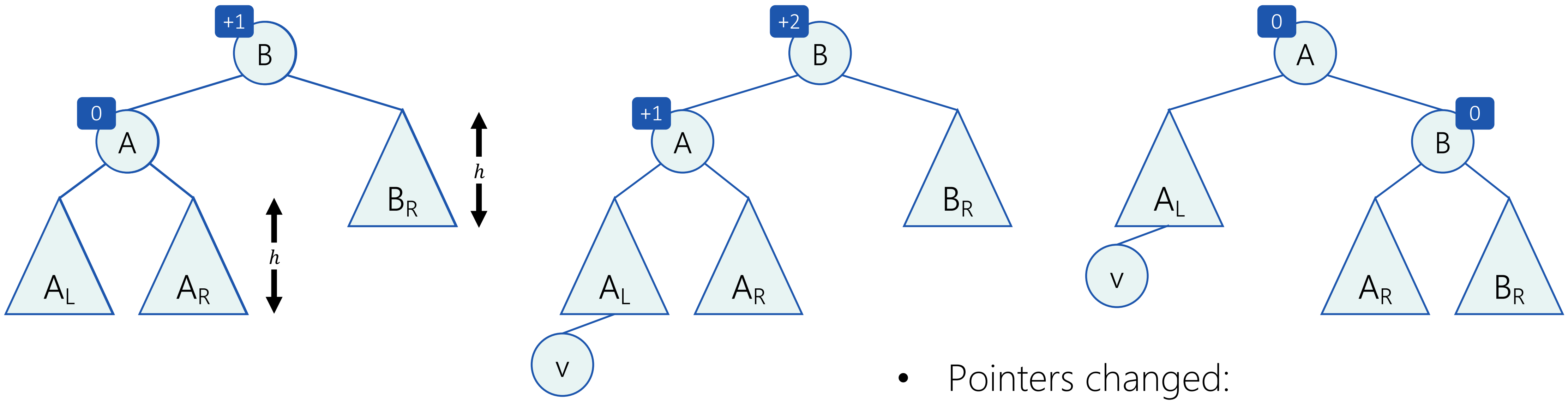


Before insertion

Insert v
Temporarily after insertion

Rotate right
After rotation

Right Rotation



- The search tree property is preserved
- The violation of balance was fixed in this subtree
- Left rotation is symmetric

- Pointers changed:
  - $B.left \leftarrow A.right$
  - $B.left.parent \leftarrow B$
  - $A.right \leftarrow B$
  - $A.parent \leftarrow B.parent$
  - $A.parent.left/right \leftarrow A$
  - $B.parent \leftarrow A$

Fixing After Insertion

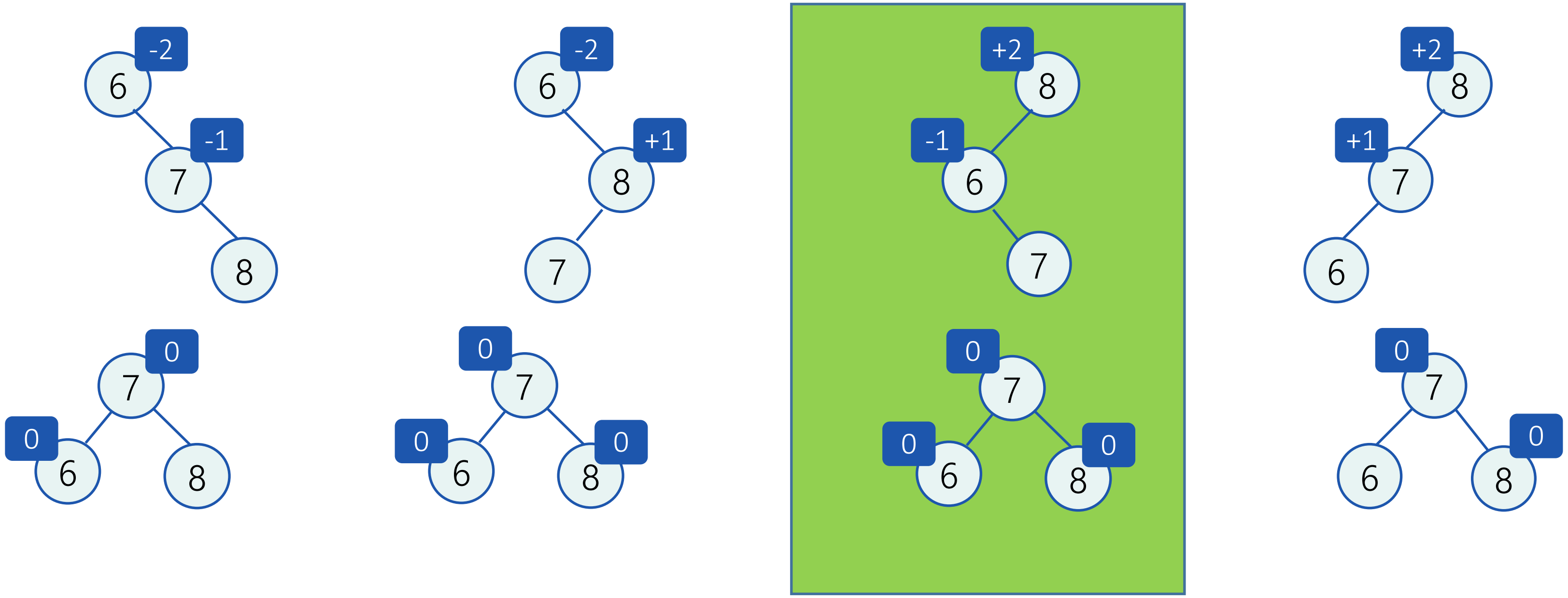
Rotations

What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1 Right then left rotation	-1 Left then right rotation	+1 Right rotation

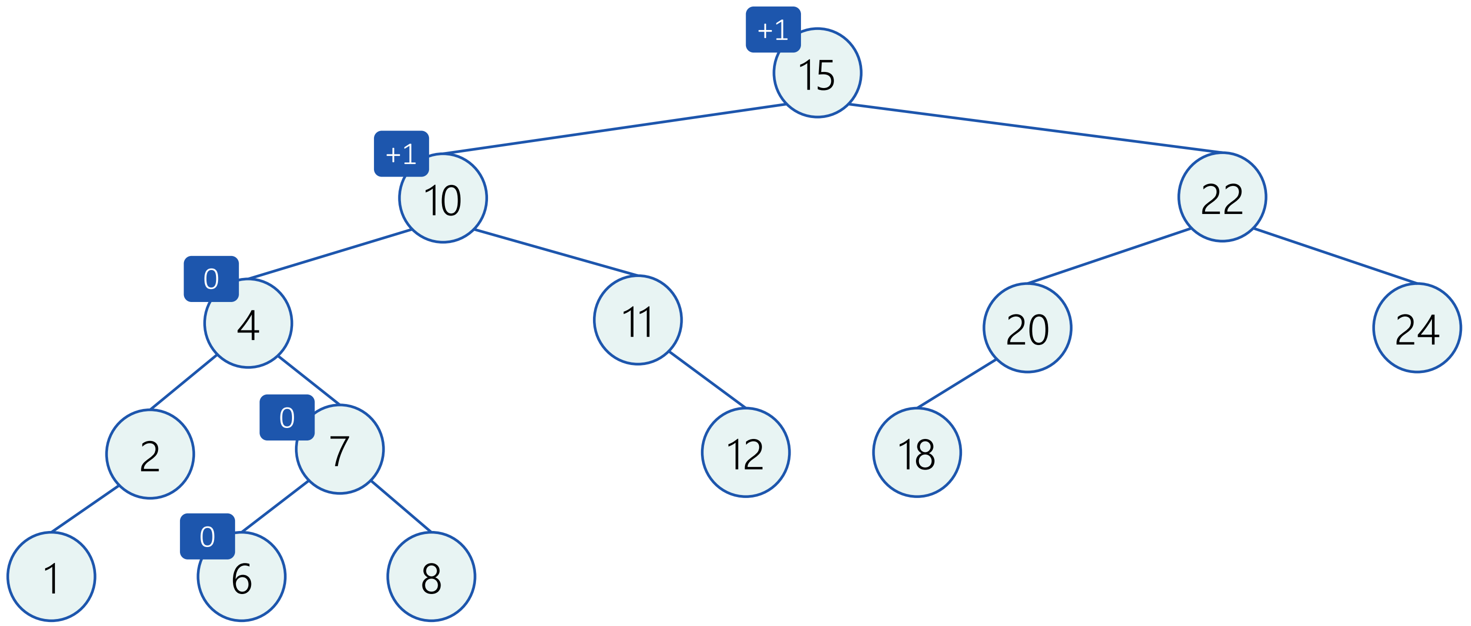
Before rotation

After rotation



Left Then Right Rotation

Example



Before insertion

Insert 5

Temporarily after insertion

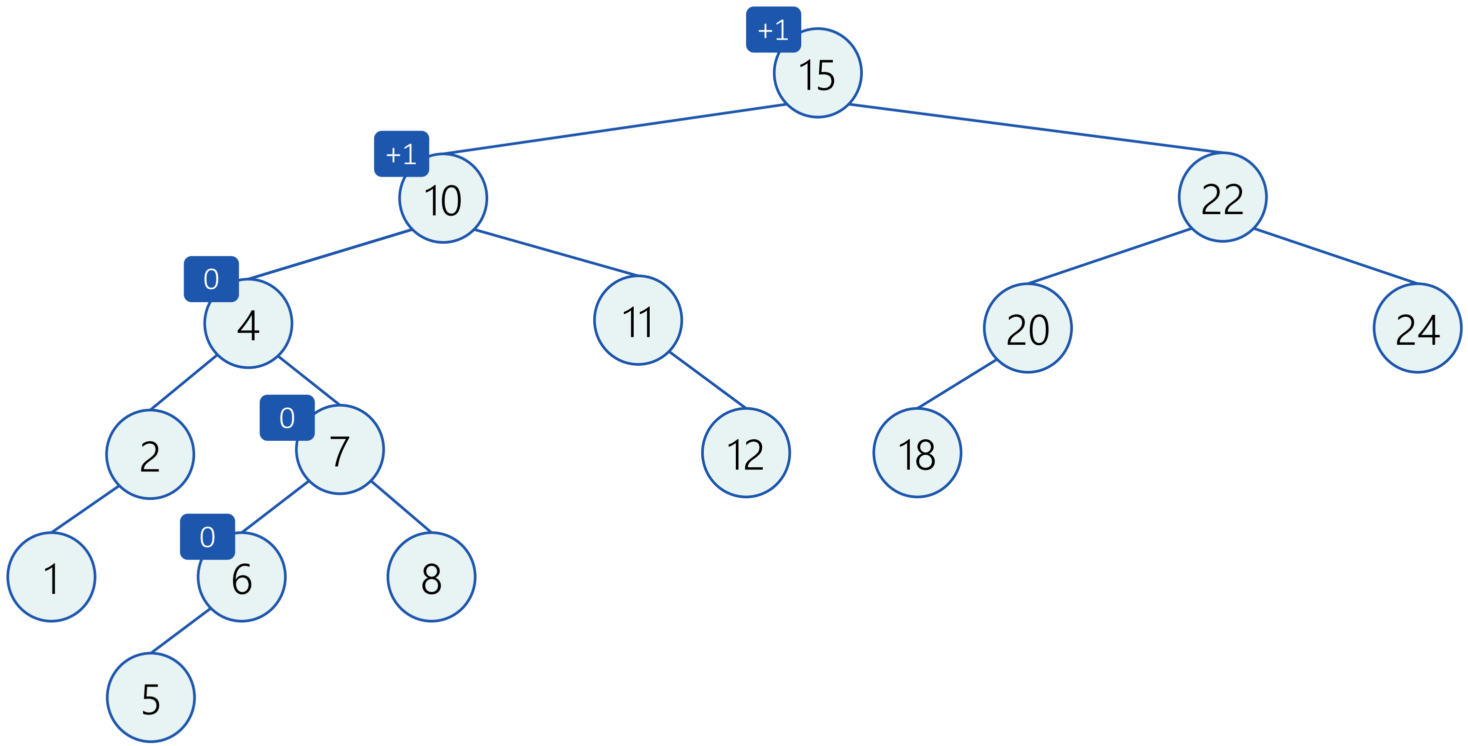
Rotate left then right

After rotation

Interactive (https://visualgo.net)

Left Then Right Rotation

Example



Before insertion

Insert 5

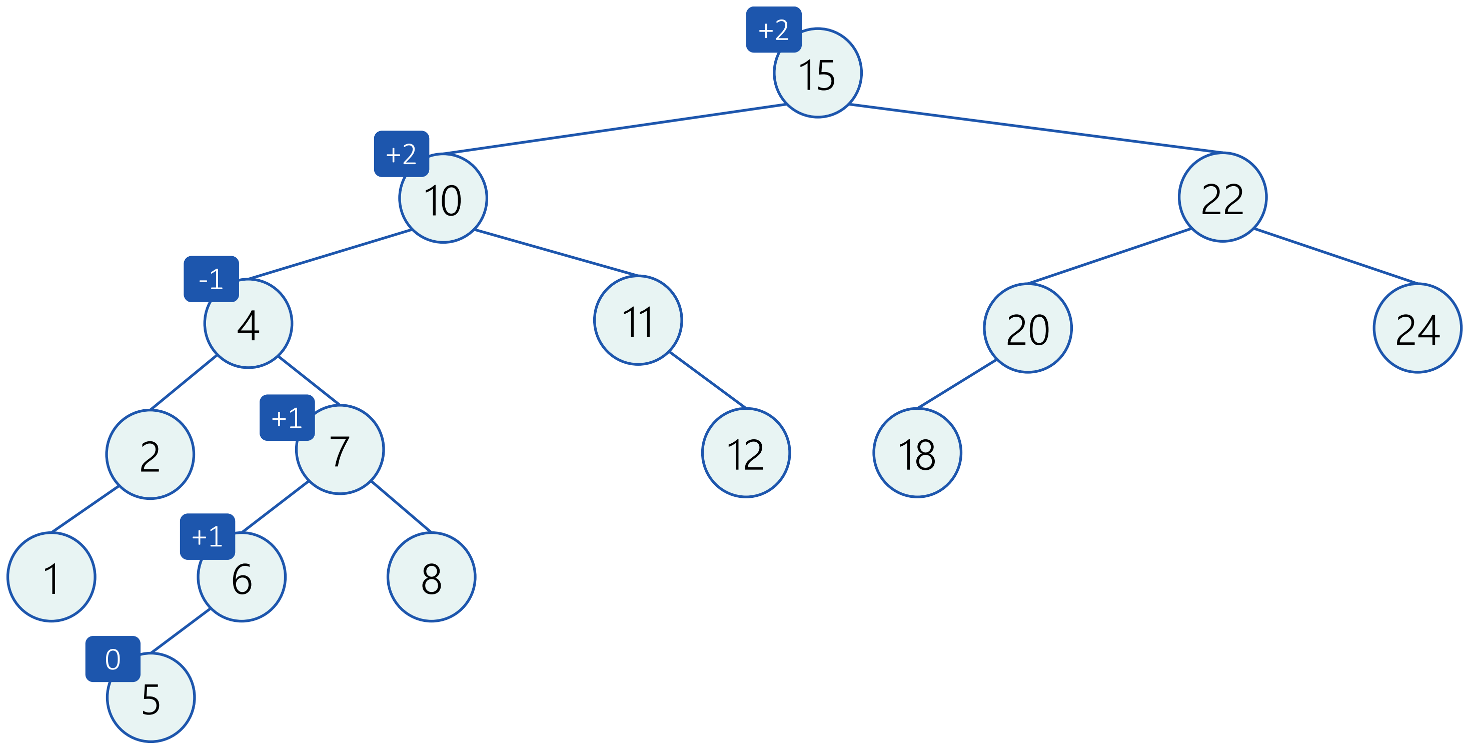
Temporarily after insertion

Rotate left then right

After rotation

Left Then Right Rotation

Example



Before insertion

Insert 5

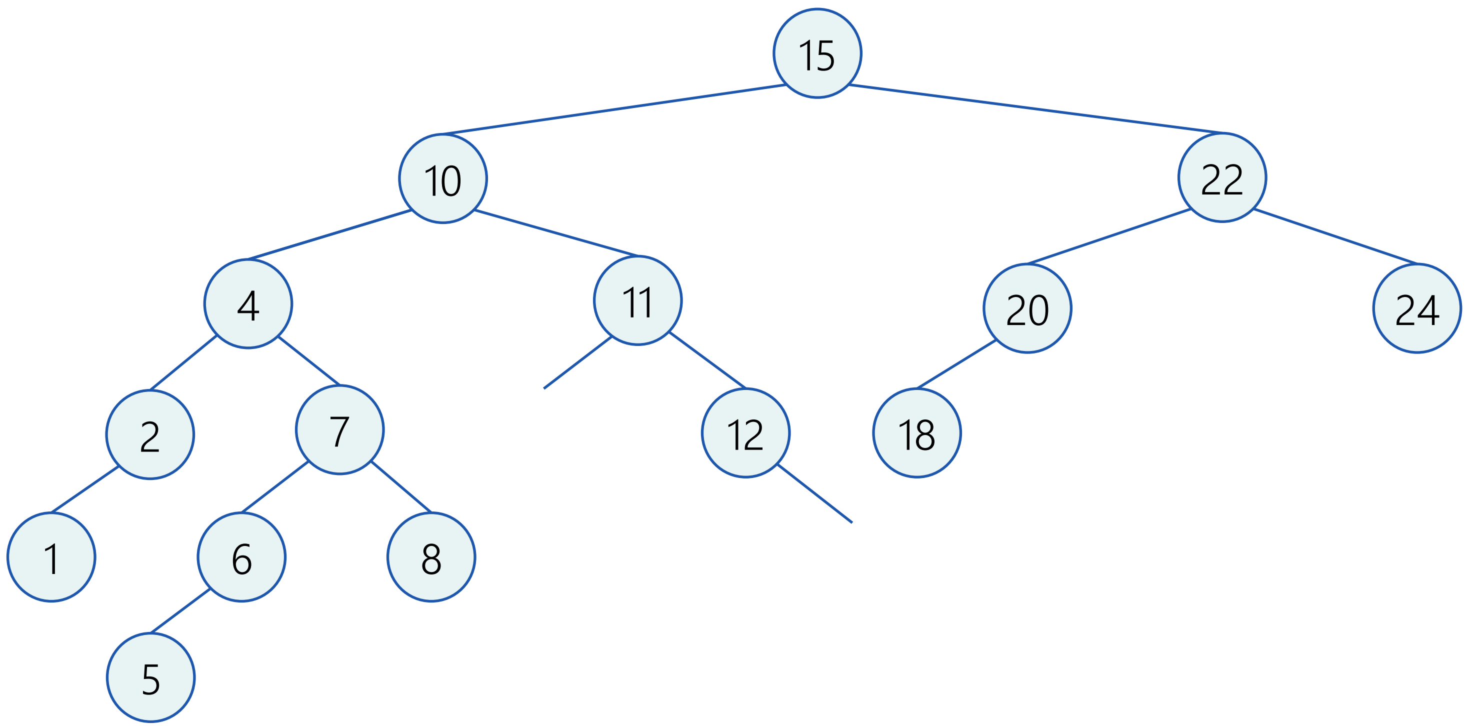
Temporarily after insertion

Rotate left then right

After rotation

Left Then Right Rotation

Example



Before insertion

Insert 5

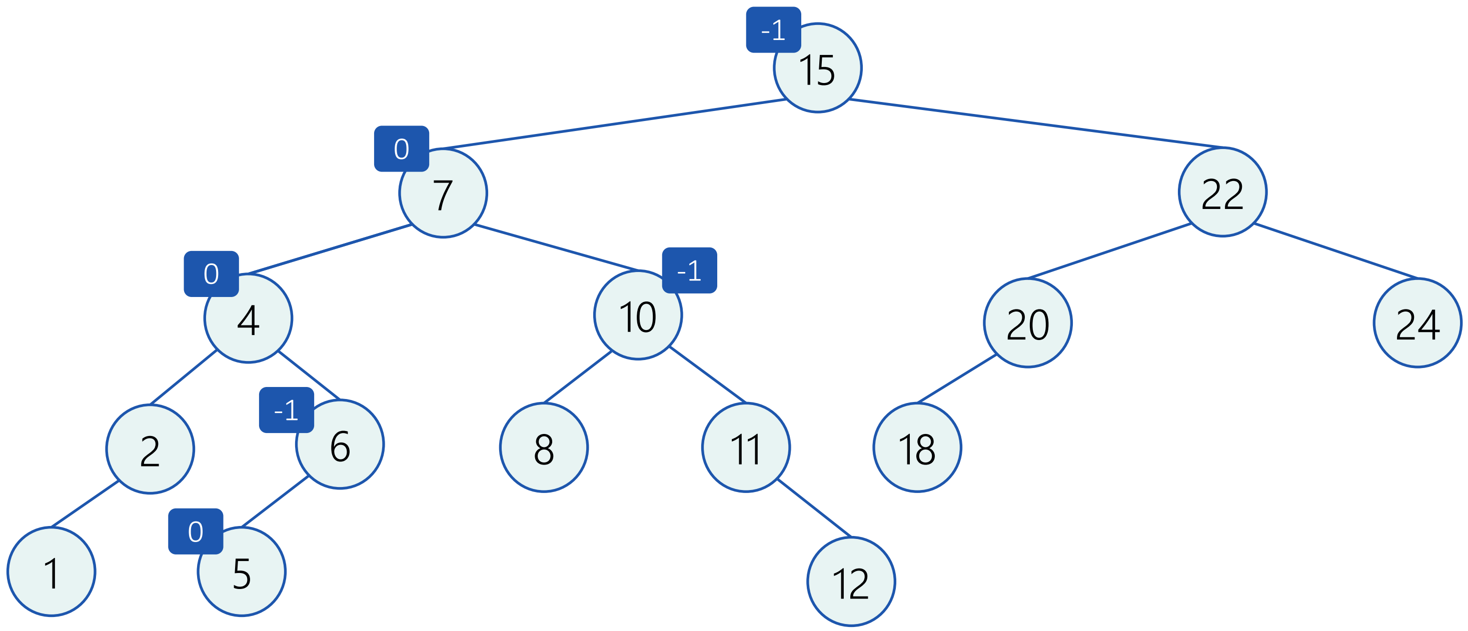
Temporarily after insertion

Rotate left then right

After rotation

Left Then Right Rotation

Example



Before insertions

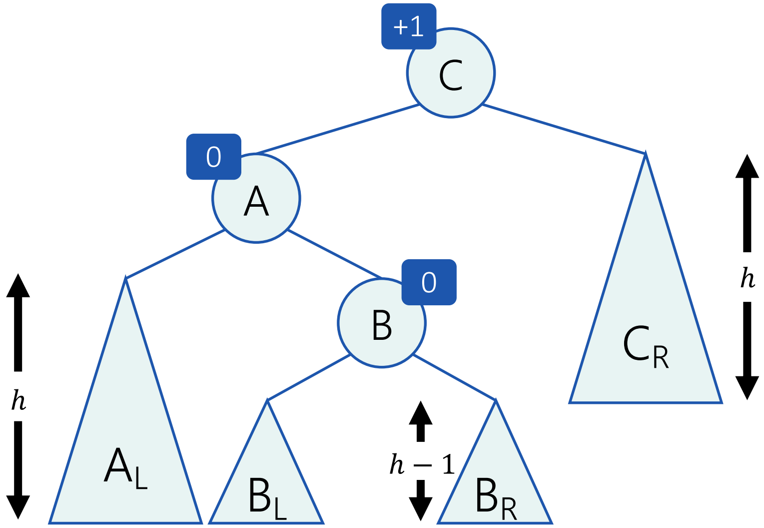
Insert 5

Temporarily after insertion

Rotate left then right

After rotation

Left Then Right Rotation



Before insertion

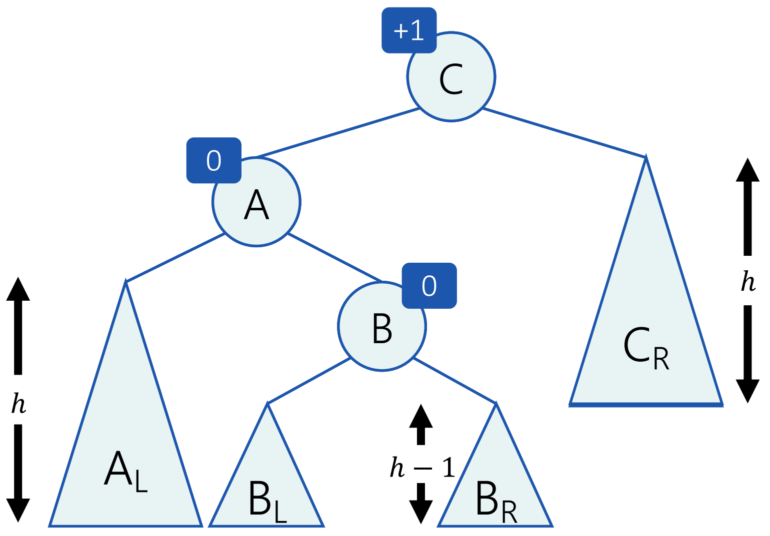
Insert v

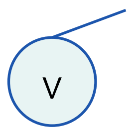
Temporarily after insertion

Rotate left then right

After rotation

Left Then Right Rotation





Before insertion

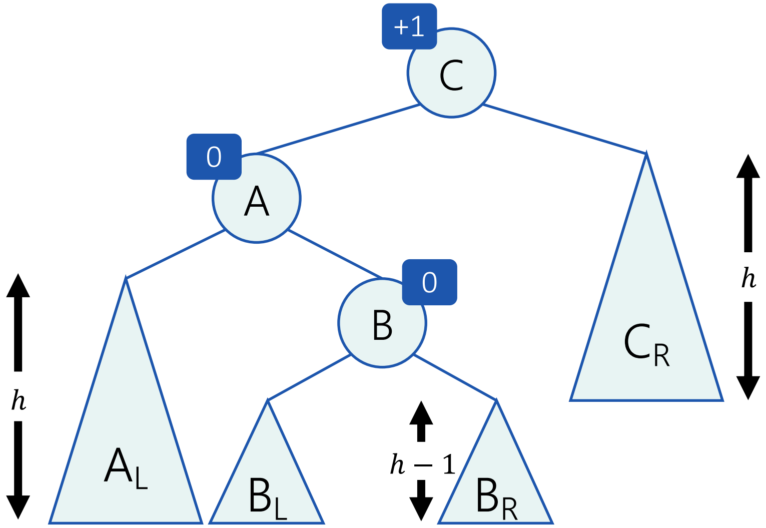
Insert v

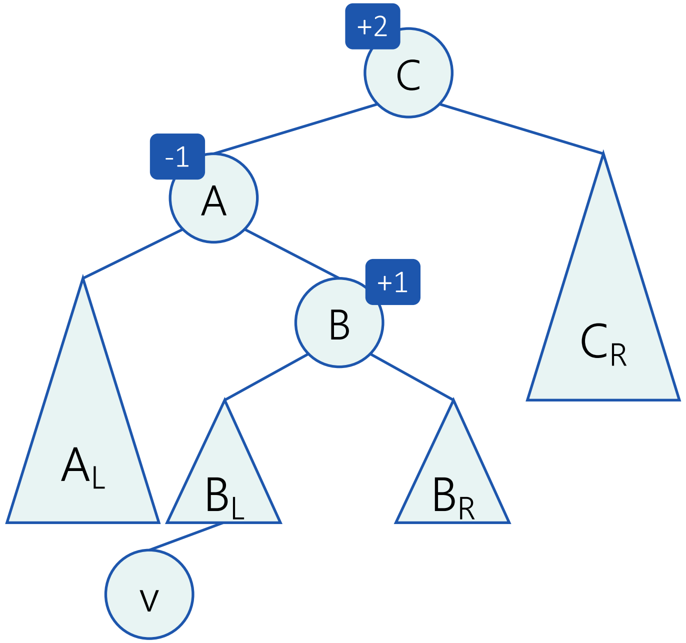
Temporarily after insertion

Rotate left then right

After rotation

Left Then Right Rotation





Before insertion

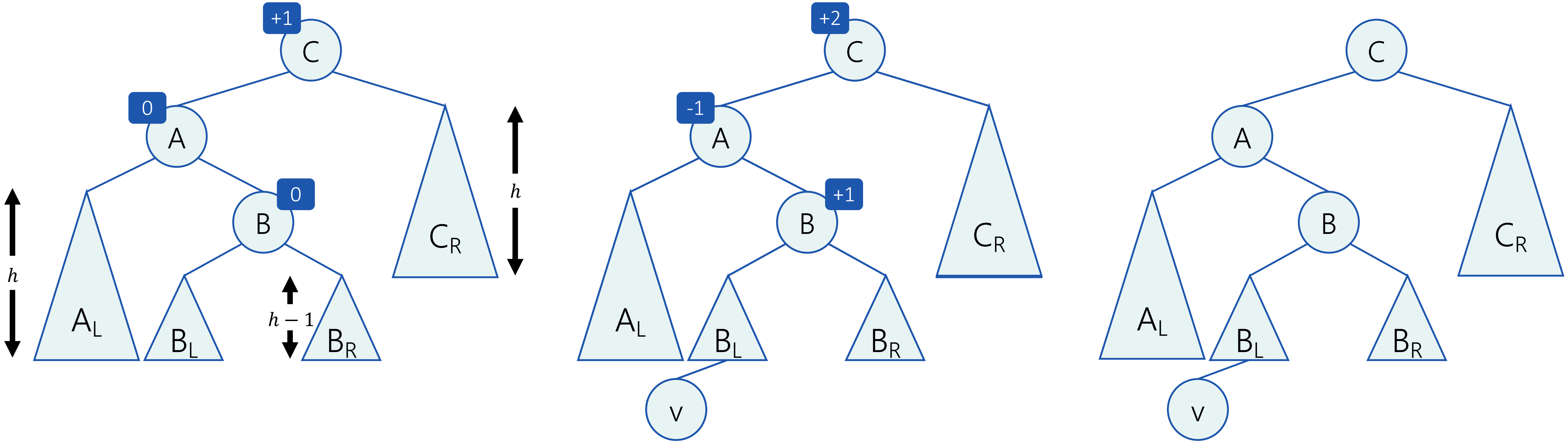
Insert v

Temporarily after insertion

Rotate left then right

After rotation

Left Then Right Rotation

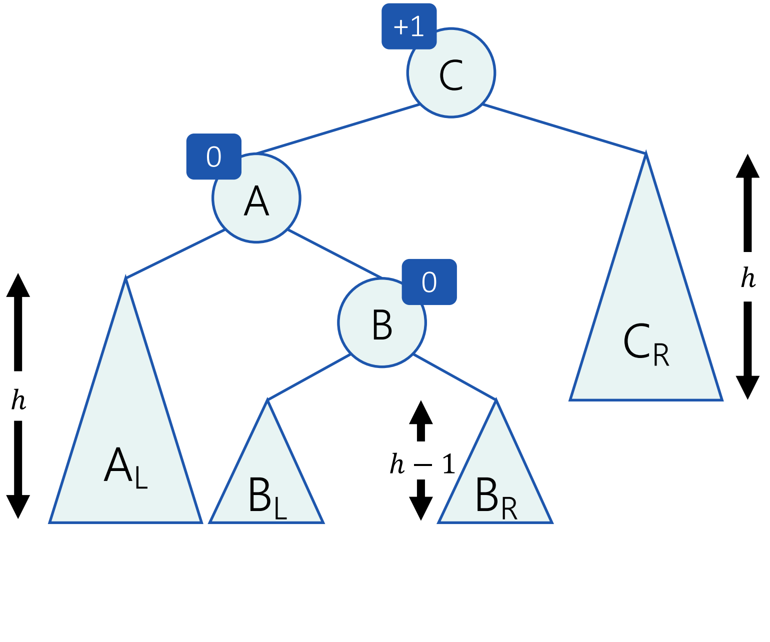


Before insertion

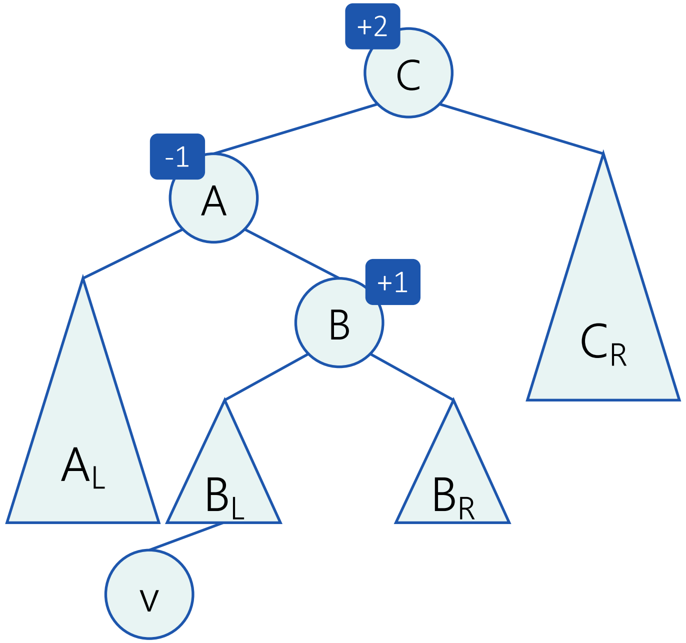
Insert v
Temporarily after insertion

Rotate left then right
After rotation

Left Then Right Rotation

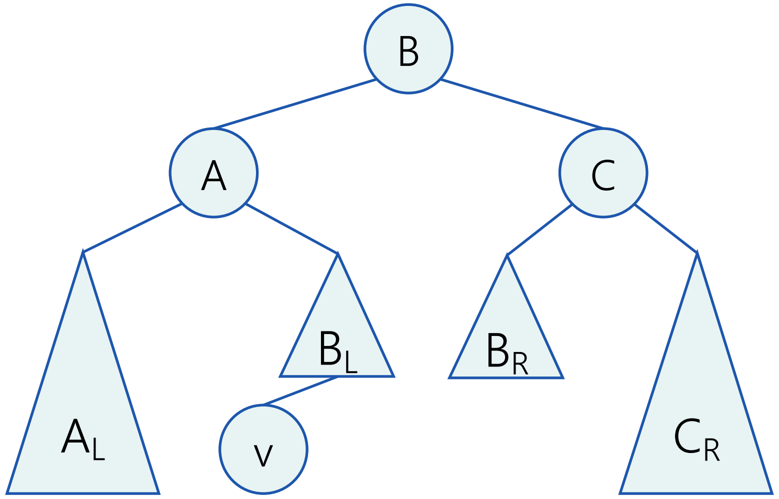


Before insertion



Insert v

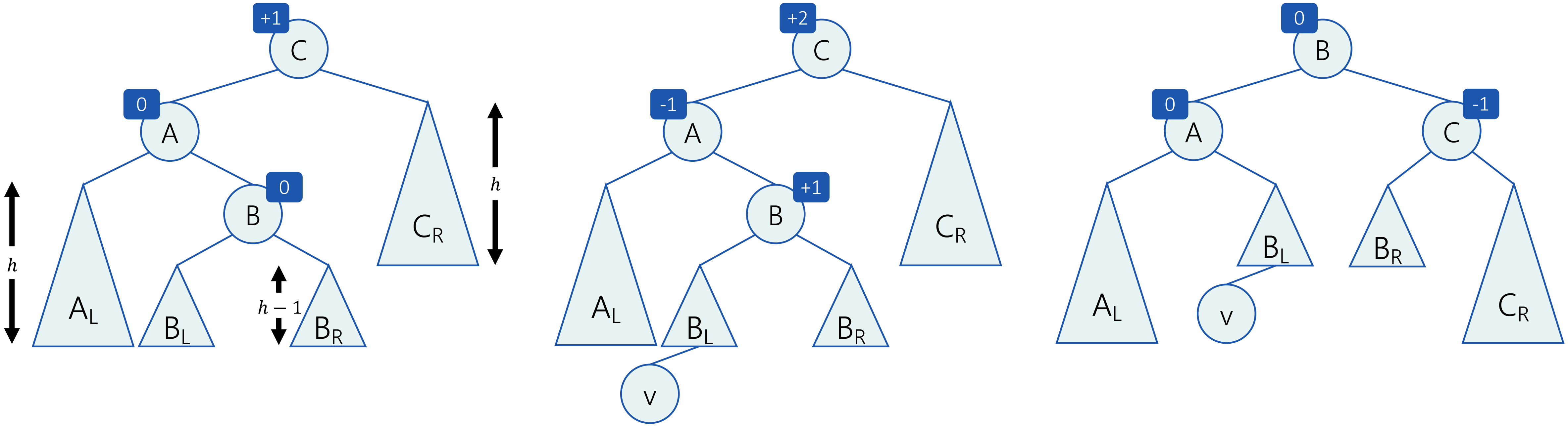
Temporarily after insertion



Rotate left then right

After rotation

Left Then Right Rotation

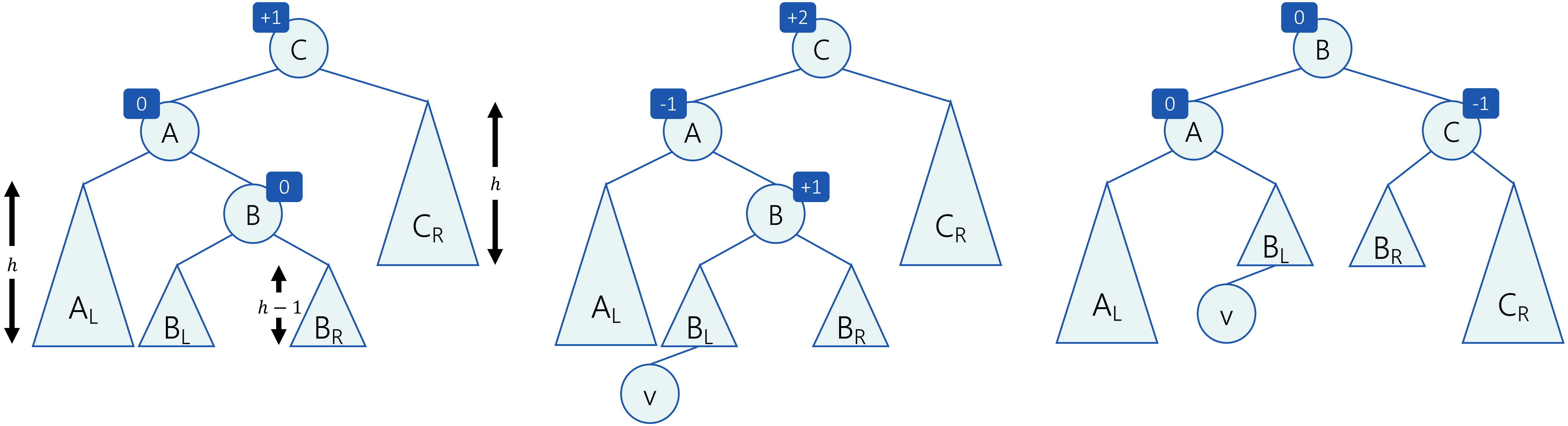


Before insertion

Insert v
Temporarily after insertion

Rotate left then right
After rotation

Left Then Right Rotation

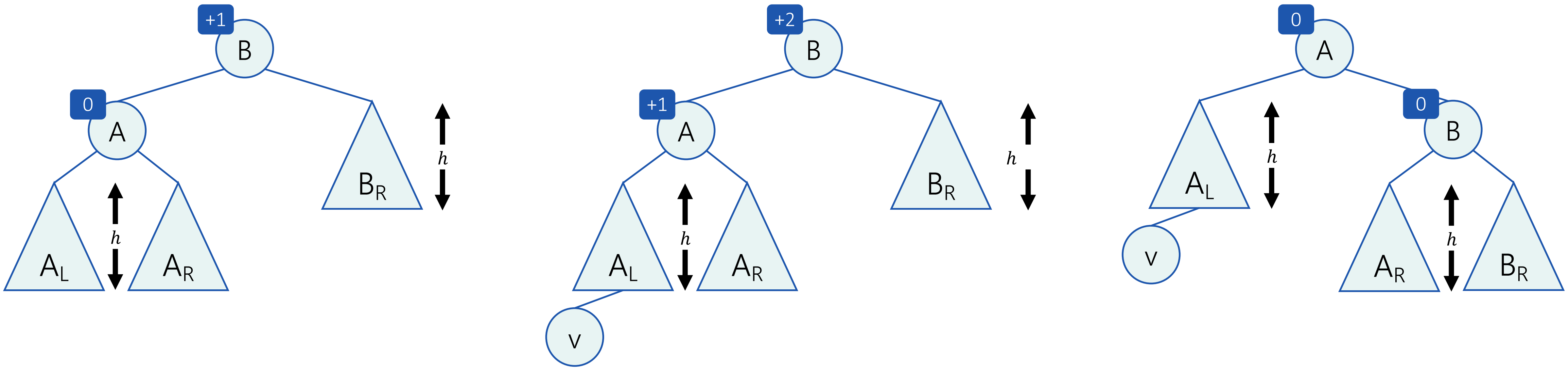


- The search tree property is preserved
- The violation of balance was fixed in this subtree
- Right then left rotation is symmetric

AVL Insertion:
One Rotation At Most

Number of Rotations After Insertion

Right Rotation (Left Rotation is symmetric):



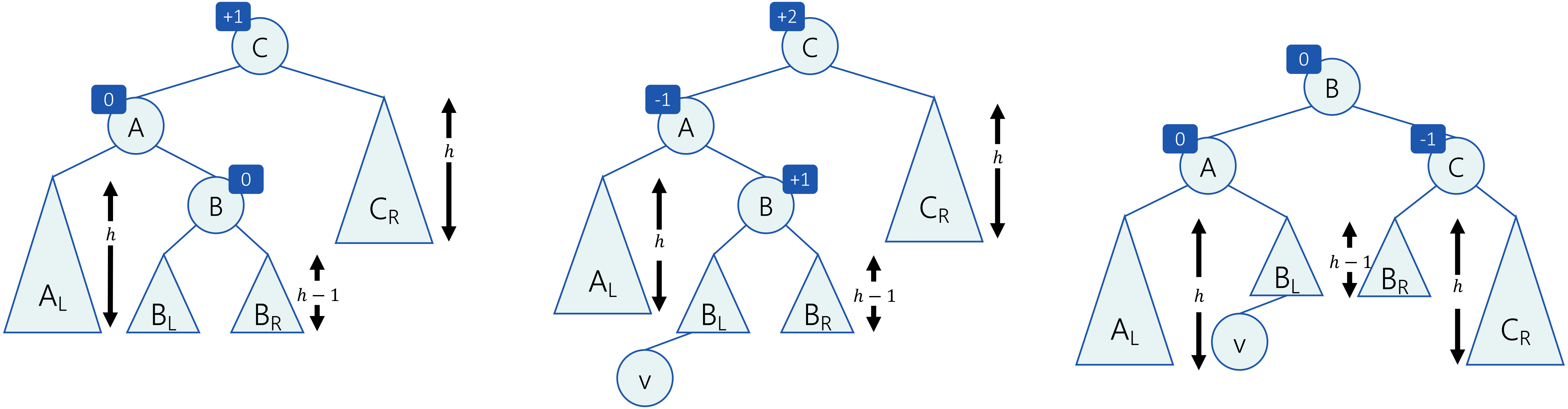
Before insertion
$height(B) = h + 2$

Right after insertion

After rotation
$height(A) = h + 2$

Number of Rotations After Insertion

Left Then Right Rotation (Right Then Left is symmetric):



Before insertion

$height(C) = h + 2$

Right after insertion

After rotation

$height(B) = h + 2$

Number of Rotations After Insertion

Conclusion:

After insertion, it takes one rotation at most in order to fix the tree.

Proof:

Rotation after insertion always restores the height of the sub-tree prior to the insertion.

The Insertion Algorithm

The Algorithm

AVL-Insert(T, z)

1. insert z as usual (as in a BST)
2. let y be the parent of the inserted node.
3. while $y \neq Null$ do:

   3.1. compute $BF(y)$*

   3.2. if $|BF(y)| < 2$ and y's height hasn't changed: terminate

   3.3. else if $|BF(y)| < 2$ and y's height changed: go back to stage 3 with y's parent

   3.4. else ($|BF(y)| = 2$): perform a rotation and go back to stage 3 with y's parent

*Requires maintaining additional information at each node. We will refer to this topic later.

The Insertion Algorithm

The Algorithm

AVL-Insert(T, z)

1. insert z as usual (as in a BST)
2. let y be the parent of the inserted node.
3. while $y \neq Null$ do:

   3.1. compute $BF(y)$*

   3.2. if $|BF(y)| < 2$ and y's height hasn't changed: terminate

   3.3. else if $|BF(y)| < 2$ and y's height changed: go back to stage 3 with y's parent

   3.4. else ($|BF(y)| = 2$): perform a rotation and terminate

*Requires maintaining additional information at each node. We will refer to this topic later.

Deletion From AVL

How to Keep It Balanced?

Fixing After Deletion

Rotations

- Deletion may also cause balance factor violation
- “Criminals” may appear on the path from the deleted node to the root
    - Deleted node = physically deleted (not necessarily the element removed)

- The 4 types of rotations are also used here to fix the violation.

Fixing After Insertion

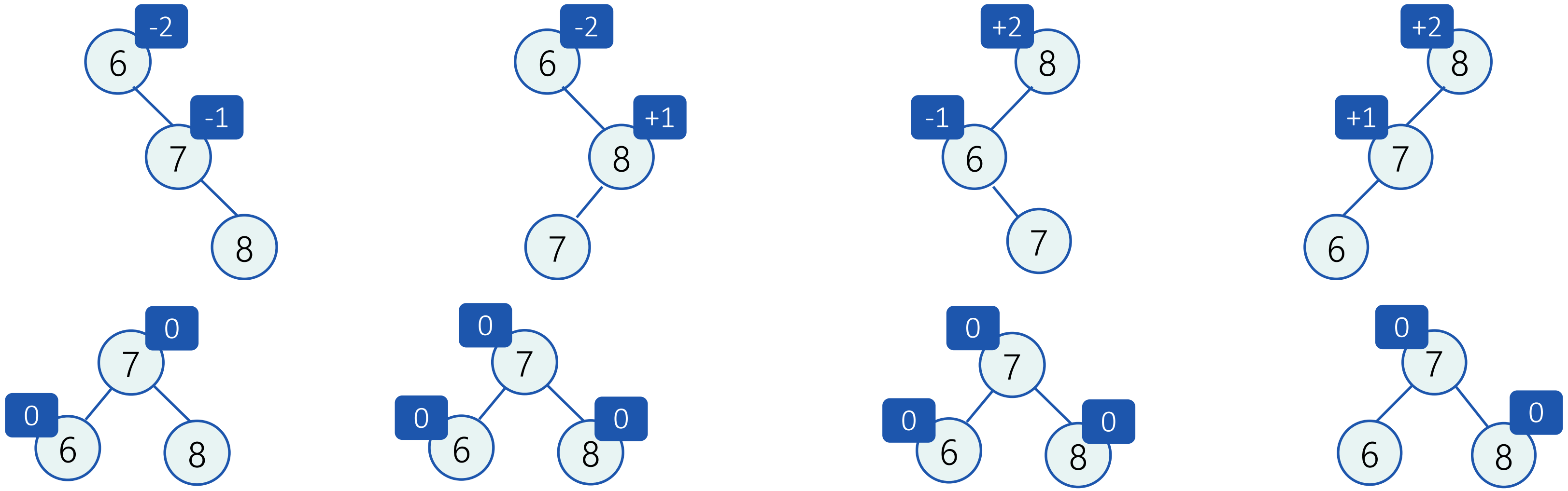
Rotations

What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 Left rotation	+1 Right then left rotation	-1 Left then right rotation	+1 Right rotation

Before rotation

After rotation



Fixing After Deletion

Rotations

What is the “criminal” BF?

-2		+2	
What is the BF of the **right** son?		What is the BF of the **left** son?	
-1 or 0 Left rotation	+1 Right then left rotation	-1 Left then right rotation	+1 or 0 Right rotation

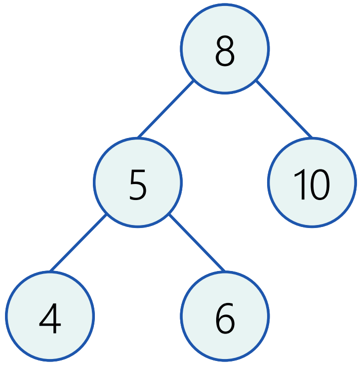
Before rotation

After rotation

Fixing After Deletion

Differences Between Insertion and Deletion

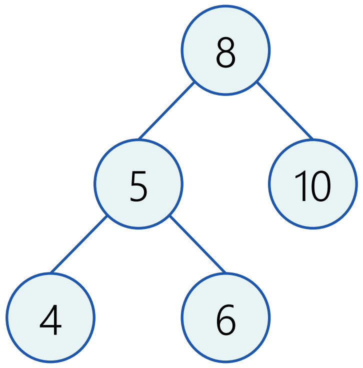
In deletion, there may be cases that cannot occur in insertion.
For example: criminal with BF=+2 whose left son has BF=0:

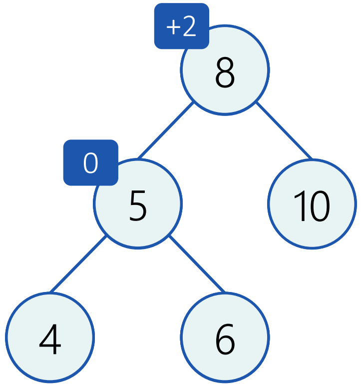


Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, there may be cases that cannot occur in insertion.
For example: criminal with BF=+2 whose left son has BF=0:



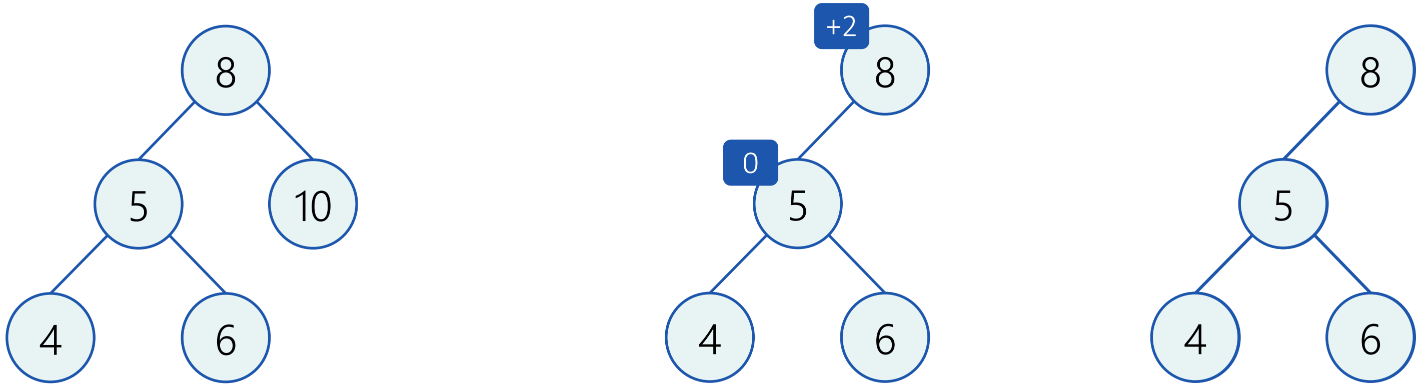


Delete 10
Rotate right

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, there may be cases that cannot occur in insertion.
For example: criminal with BF=+2 whose left son has BF=0:

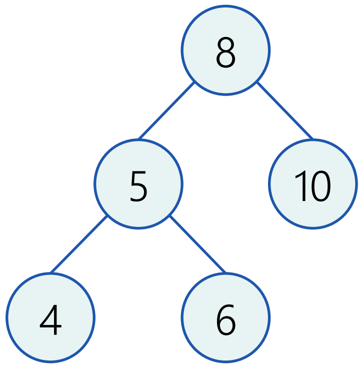


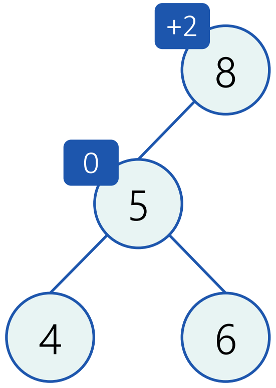
Delete 10
Rotate right

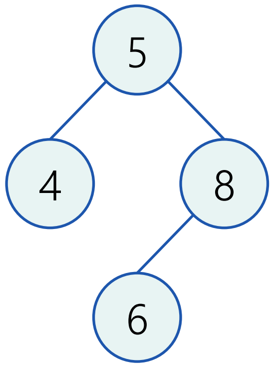
Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, there may be cases that cannot occur in insertion.
For example: criminal with BF=+2 whose left son has BF=0:





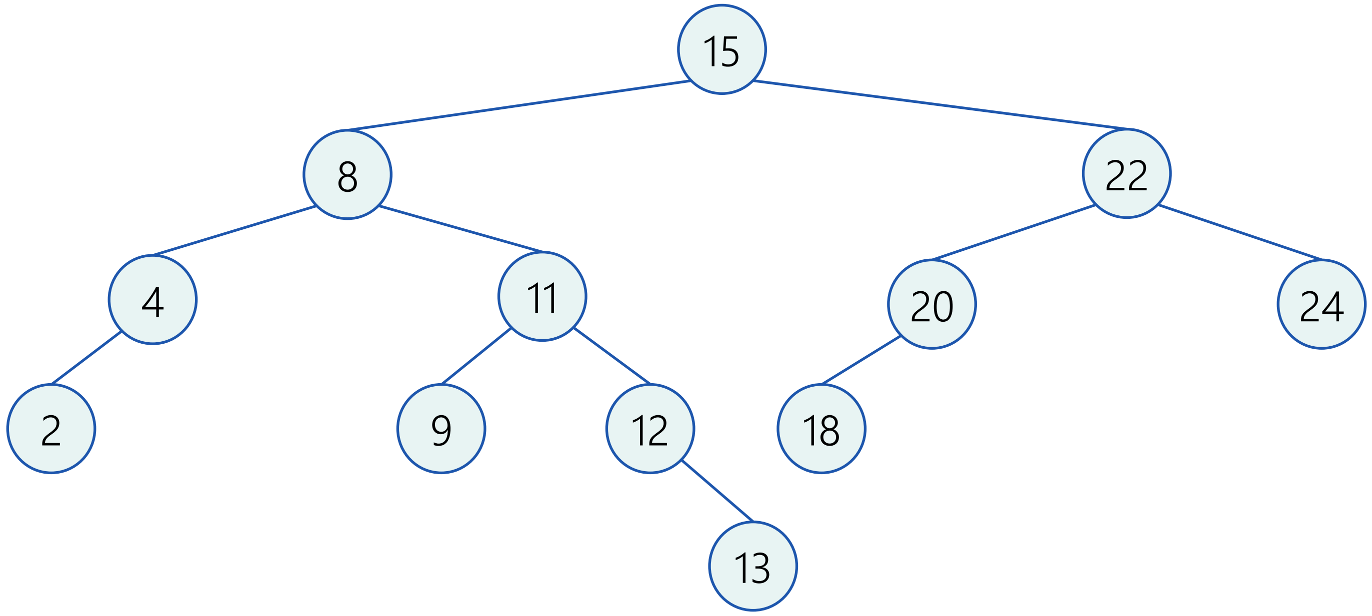


Delete 10
Rotate right

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, more than one rotation is possible
(one rotation may be performed at each level of the tree)



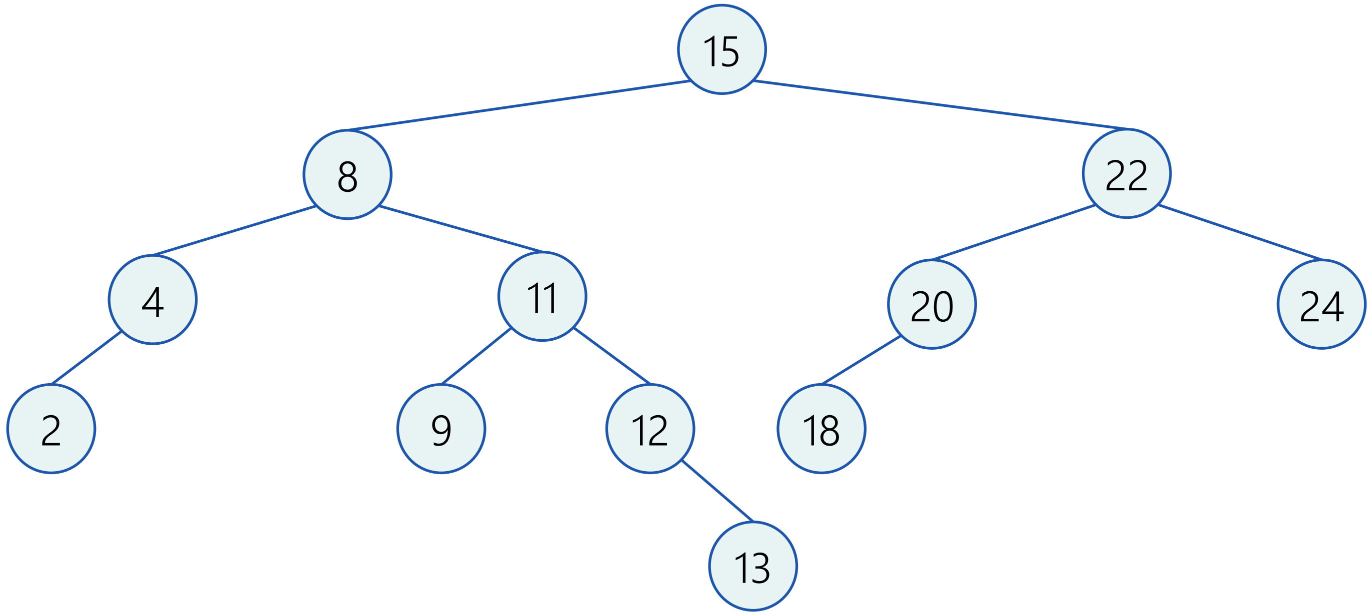
Delete 24
Rotate right
Rotate left then right

Interactive (https://visualgo.net)

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, more than one rotation is possible
(one rotation may be performed at each level of the tree)

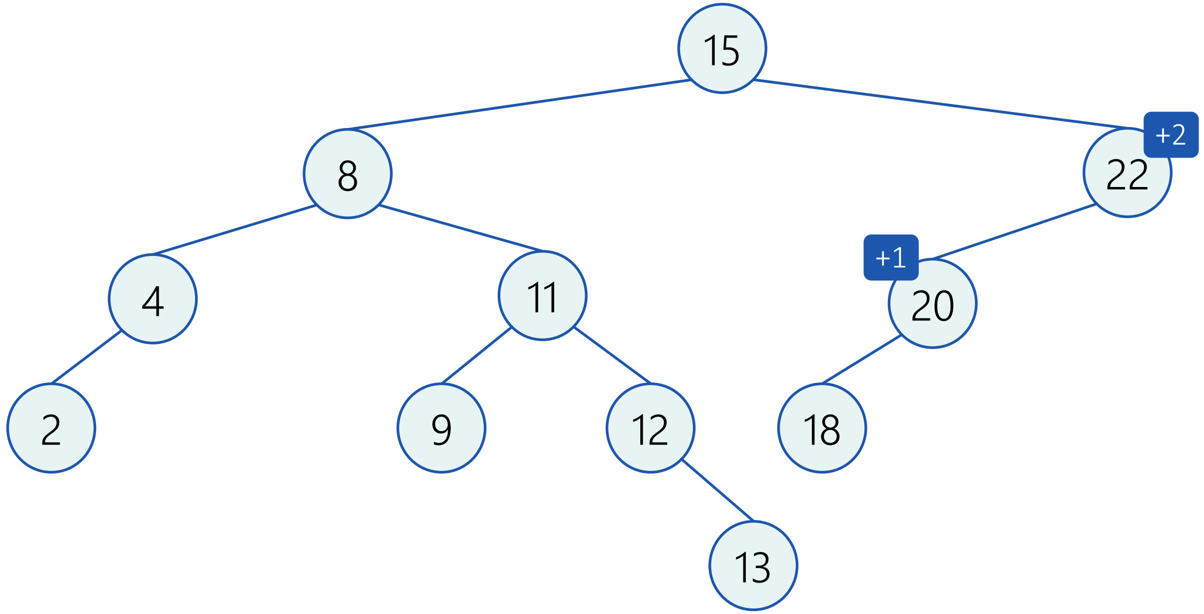


Delete 24
Rotate right
Rotate left then right

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, more than one rotation is possible
(one rotation may be performed at each level of the tree)

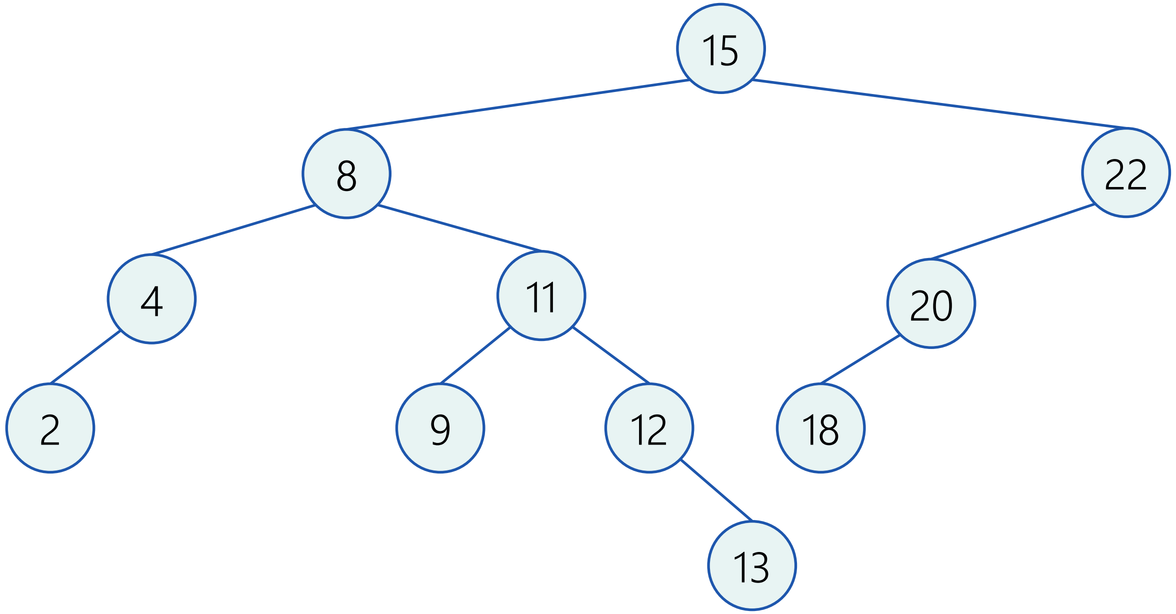


Delete 24
Rotate right
Rotate left then right

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, more than one rotation is possible
(one rotation may be performed at each level of the tree)

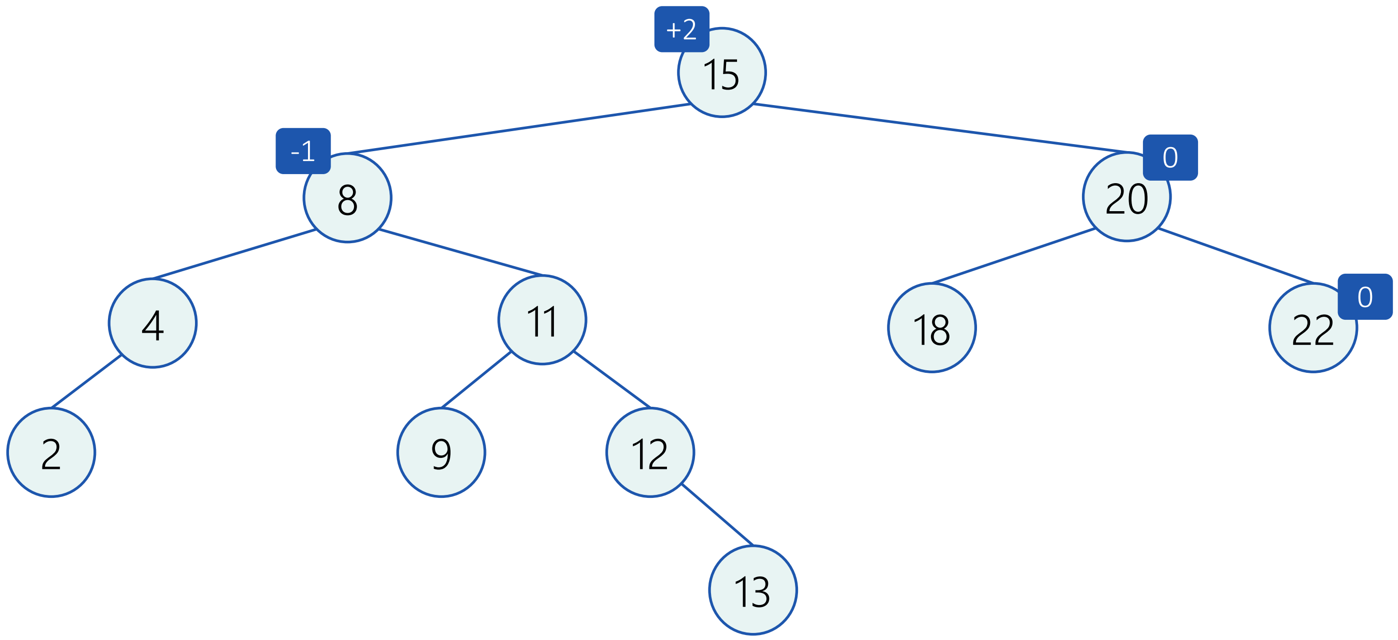


Delete 24
Rotate right
Rotate left then right

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, more than one rotation is possible
(one rotation may be performed at each level of the tree)

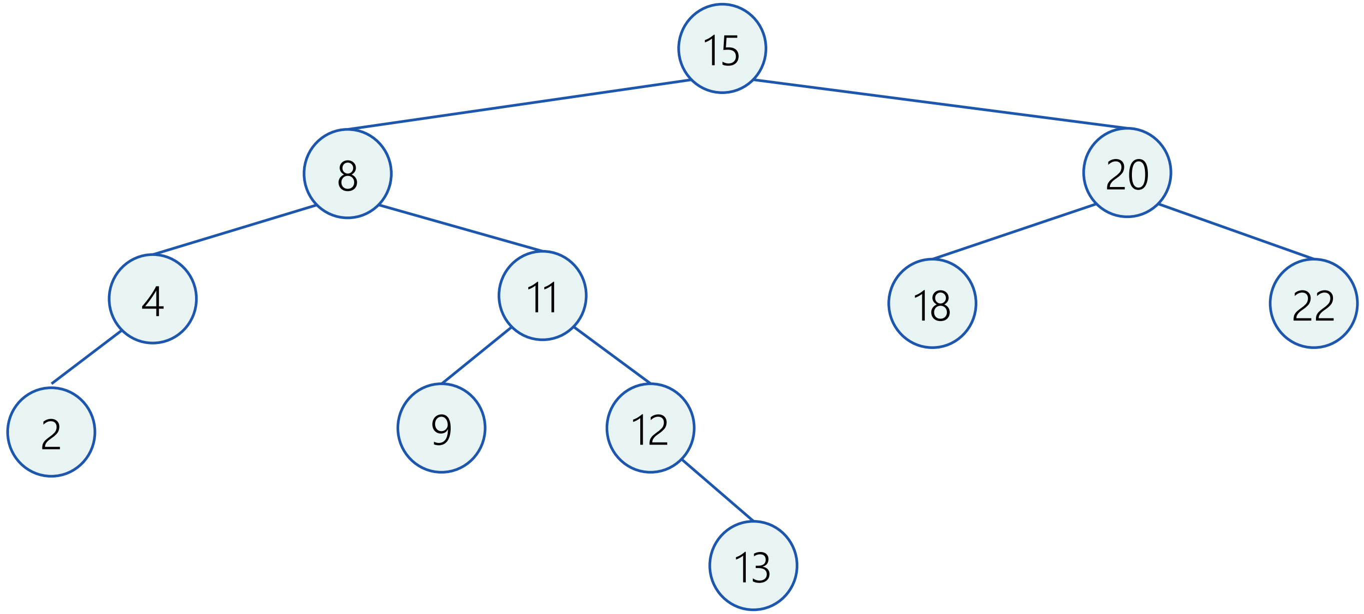


Delete 24
Rotate right
Rotate left then right

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, more than one rotation is possible
(one rotation may be performed at each level of the tree)

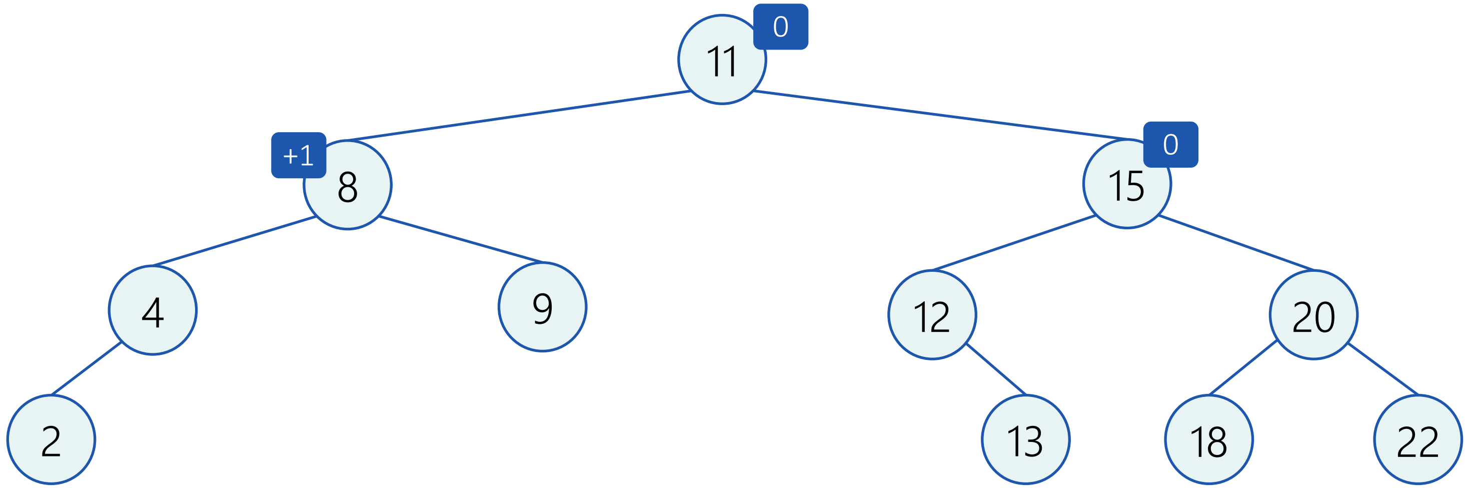


Delete 24
Rotate right
Rotate left then right

Fixing After Deletion

Differences Between Insertion and Deletion

In deletion, more than one rotation is possible
(one rotation may be performed at each level of the tree)



Delete 24
Rotate right
Rotate left then right

Fixing After Deletion

The Algorithm

AVL-Delete(T, z)

1. delete z as usual (as in a BST).
2. Let y be the parent of the (physically) deleted node.
3. while $y \neq Null$ do:

   3.1. compute $BF(y)$*

   3.2. if $|BF(y)| < 2$ and y's height hasn't changed: terminate

   3.3. else if $|BF(y)| < 2$ and y's height changed: go back to stage 3 with y's parent

   3.4. else ($|BF(y)| = 2$): perform a rotation and go back to stage 3 with y's parent

*Requires maintaining additional information at each node. We will refer to this topic later.

The Deletion Algorithm

Time Complexity

AVL-Delete(T, z)

1. Deleting from a BST $O(h+1)$
2. At most one rotation at each level $O(h+1)$

$O(h+1) \;=\; O(\log n)$

Recommended Animations on the Web

- https://people.ksp.sk/~kuko/gnarley-trees/Balance.html

  Explains rotations

- https://people.ksp.sk/~kuko/gnarley-trees/AVL.html

  Explains AVL, accompanied by a nice summary

- https://visualgo.net/en/bst?mode=AVL

  Explains AVL

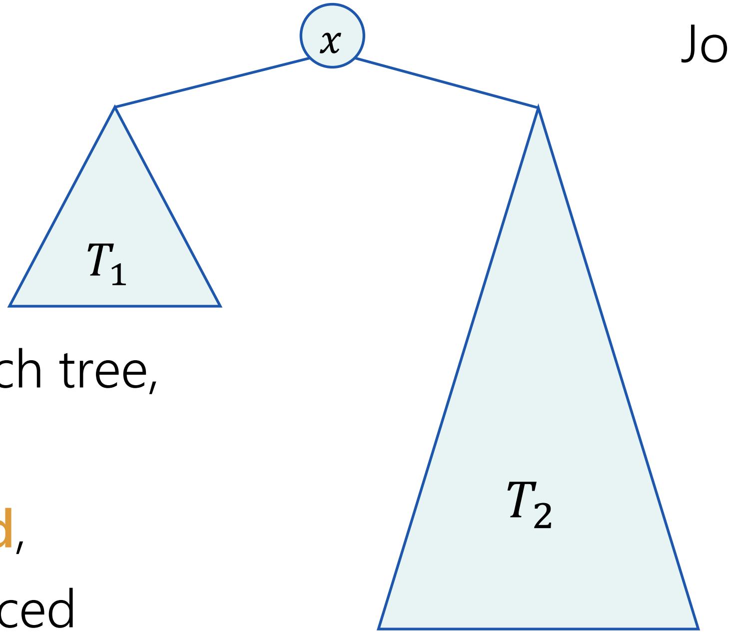
Joining and Splitting AVL Trees

Joining Trees

Joining Two BST's

Suppose that for all nodes $x_1 \in T_1$ and $x_2 \in T_2$
x_1.key < x.key < x_2.key

"$T_1 < x < T_2$"



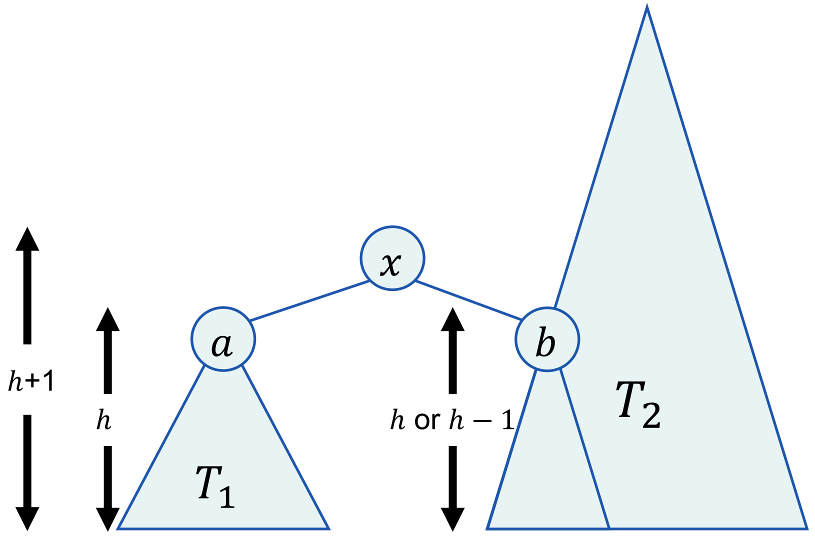
Join(T_1,x,T_2) in $O(1)$

The tree formed is a valid search tree,
but
may be **very unbalanced**,
even if T_1 and T_2 are balanced

Joining Two AVL Trees Efficiently, Maintaining Balance

Join(T_1,x,T_2) when "$T_1 < x < T_2$"

Assume $height(T_1) \leq height(T_2)$



Idea: $x.right$ will be a subtree of T_2 of similar height as T_1

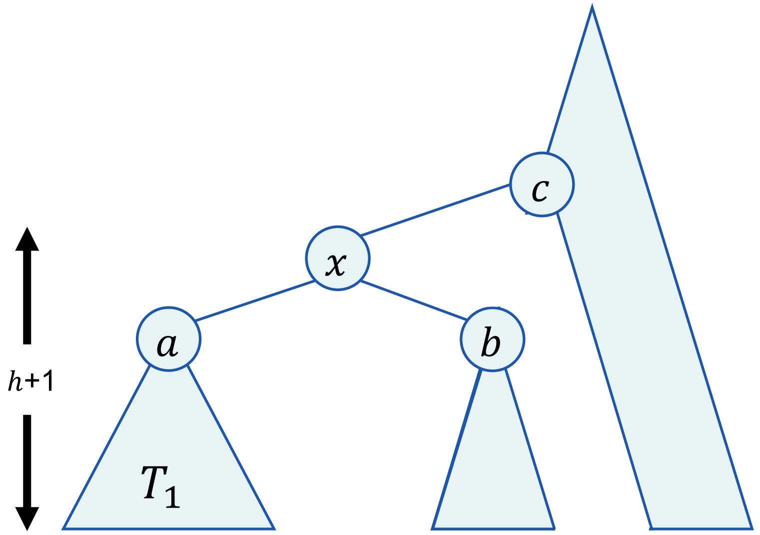
Denote $height(T_1) = h$

b – first vertex on the left spine of T_2 with $height \leq h$

Joining Two AVL Trees Efficiently, Maintaining Balance

Join(T_1,x,T_2) when "$T_1 < x < T_2$"

Assume $height(T_1) \leq height(T_2)$



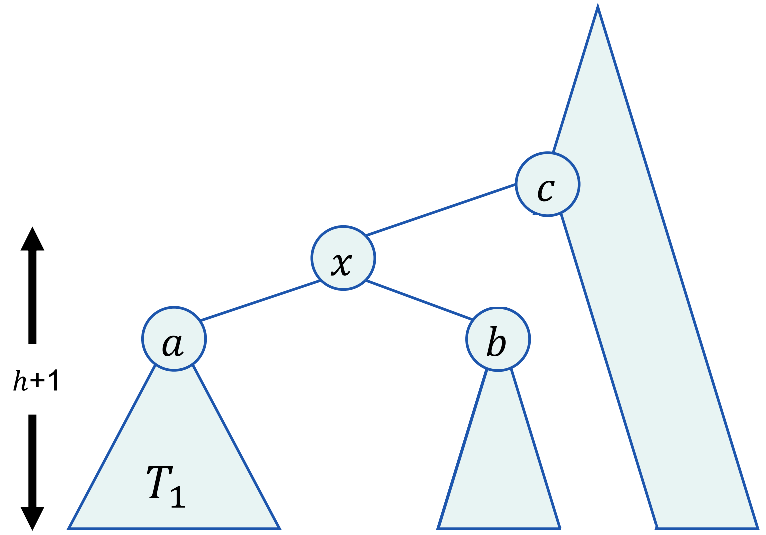
Attach x to b's former parent (denoted c)

Do rebalancing from x upwards, if needed (when?)

Joining Two AVL Trees Efficiently, Maintaining Balance

Join(T_1,x,T_2) when "$T_1 < x < T_2$"

Assume $height(T_1) \leq height(T_2)$



$O(\log n)$ time

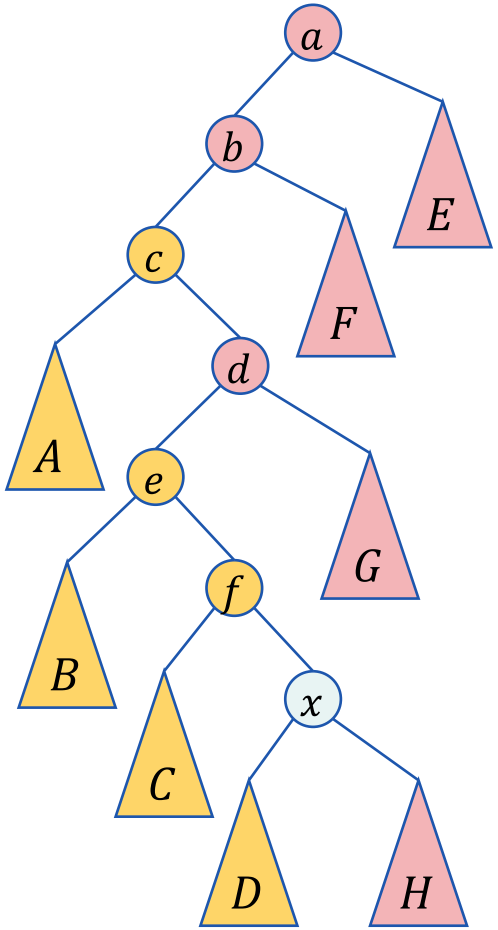
And even $O(height(T_2) - height(T_1) + 1)$ time

(if heights maintained explicitly, so we can **find b** while going **down**)

Splitting Trees

Splitting BST (by Joins)

Given a BST and a node x,
we want to split the tree into T_1, T_2
such that "$T_1 < x < T_2$"



Splitting BST (by Joins)

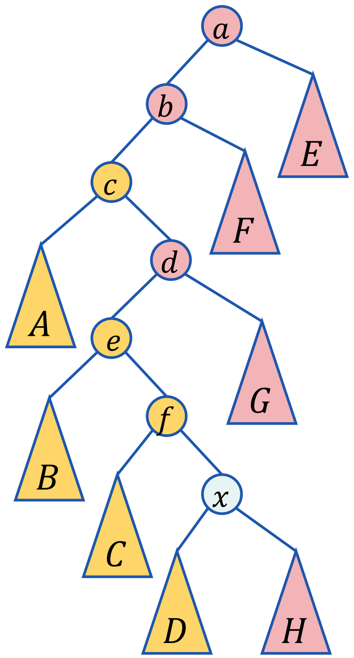
Given a BST and a node x,
we want to split the tree into T_1, T_2
such that "$T_1 < x < T_2$"

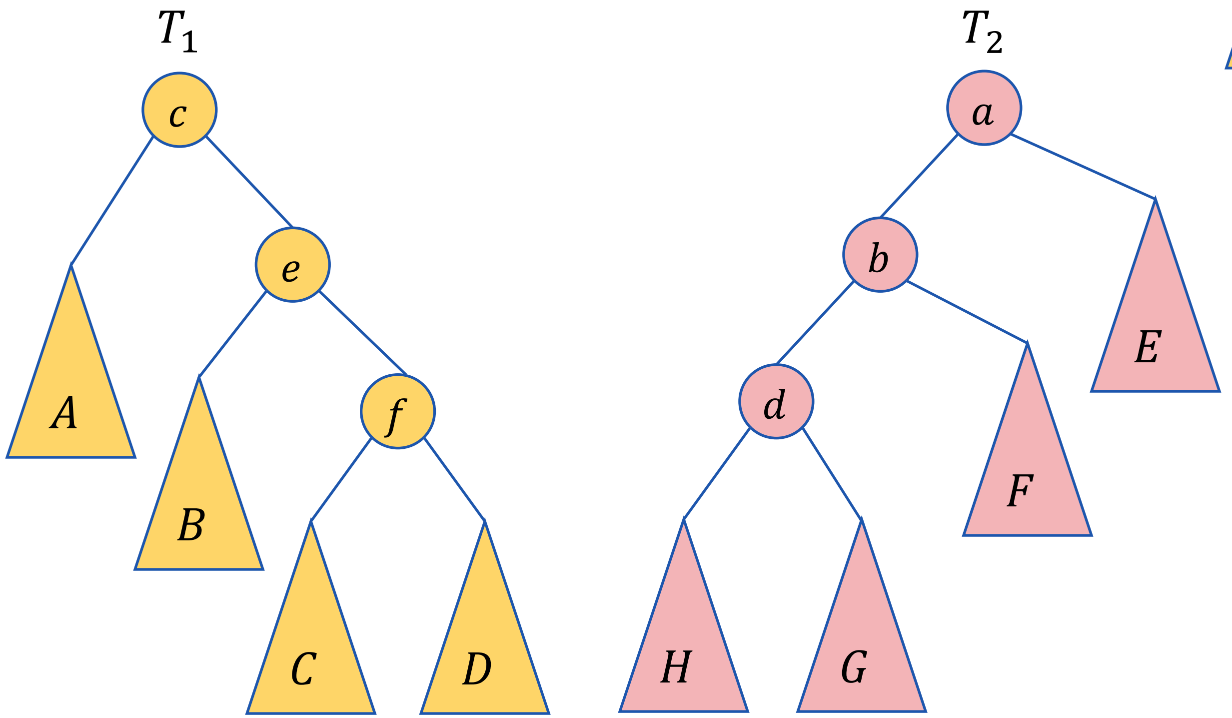
Time:

$O(h)$ for BST

$O(\log^2 n)$ for AVL

(naïve analysis)





$T_1 = \text{Join}(A, c, \text{Join}(B, e, \text{Join}(C, f, D)))$ $T_2 = \text{Join}(\text{Join}(\text{Join}(H, d, G), b, F), a, E)$

Splitting AVL (by Efficient Joins)

Tighter Analysis

Recall each join really takes only O(height difference+1)

To generate each of T_1, T_2,

we need to make a telescopic join series: $(\ldots(((t_1, t_2), t_3), t_4), \ldots, t_k)$

$$time = O\left(\sum_{i=2}^{k} |height(t_i) - height(join(t_1, \ldots, t_{i-1}))| + 1\right) \qquad = O(\log n)$$