CS2040S – Data Structures and Algorithms

Lecture 12 – Balancing Act ~ AVL Tree

chongket@comp.nus.edu.sg

NUS
National University of Singapore
School of Computing

Outline

Binary Search Tree (BST): **A Quick Revision**

The Importance of a **Balanced** BST

- To keep **h** = O(log **N**)

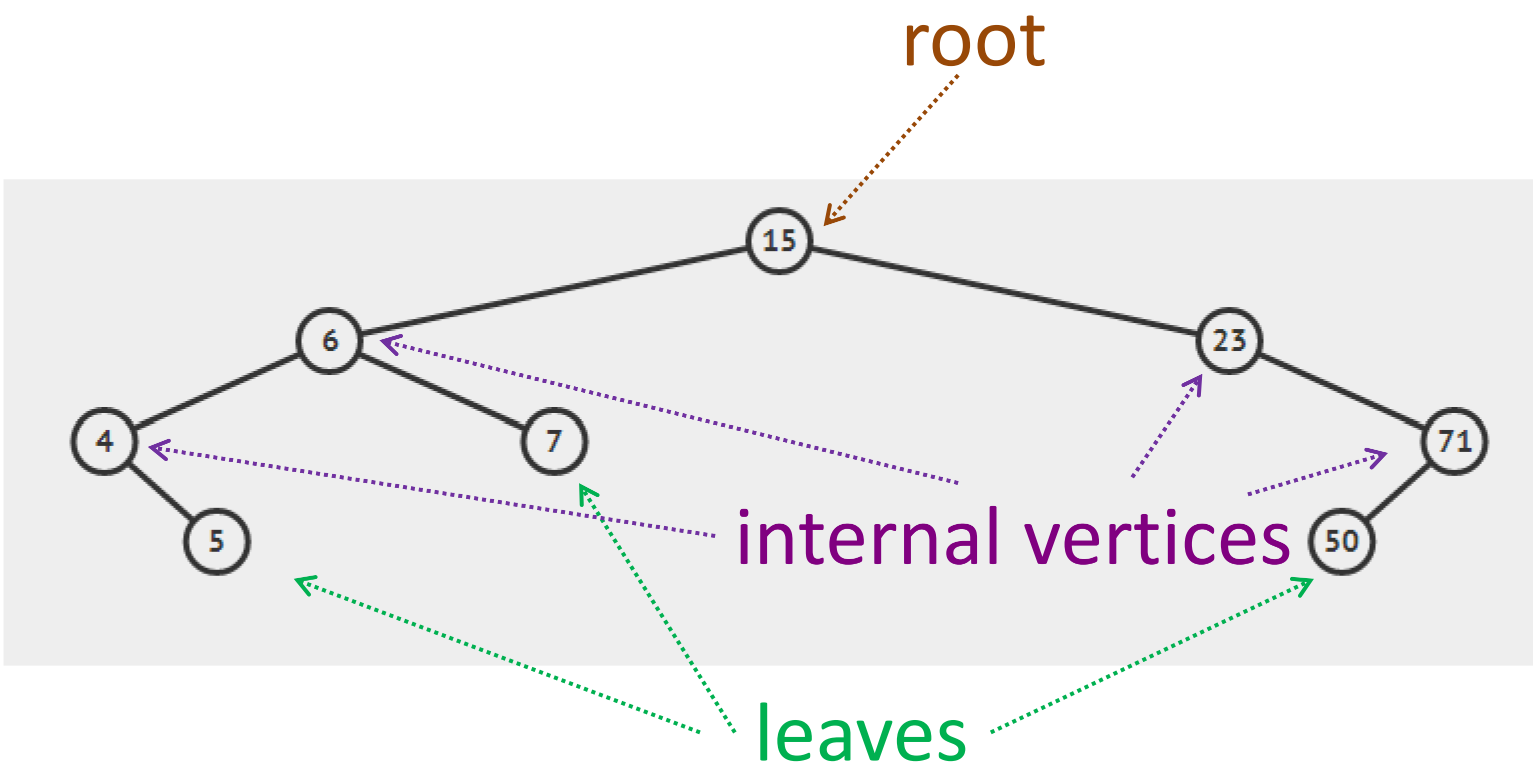
Adelson-Velskii Landis (AVL) Tree

- Principle of “Height-Balanced”
- Keeping AVL Tree balanced via rotations

Reference in CP4 book: Section 2.3.3

BST Web-based Review

[https://visualgo.net/**bst**](https://visualgo.net/bst)



Binary Search Tree: Summary

Operations that **modify** the BST (*dynamic* data structure):

- insert: O(**h**)
- delete: O(**h**)

Query operations (the BST structure remains the same):

- search: O(**h**)
- findMin, findMax: O(**h**)
- predecessor, successor: O(**h**)
- inorder traversal: O(**N**) – the only one that does not depend on **h**
  - PS: We also have preorder and postorder traversals for tree structure
- select/rank: ? (we have not discuss this yet)

More BST Attributes: Height and Size

Two more attributes at each BST vertex: Height and Size

Height: #edges on the path from this vertex to deepest leaf

Size: #vertices of the subtree rooted at this vertex

These values are recursively defined/computed:

x.height = -1 (if x is an empty tree)

x.height = max(x.left.height, x.right.height) + 1 (all other cases)

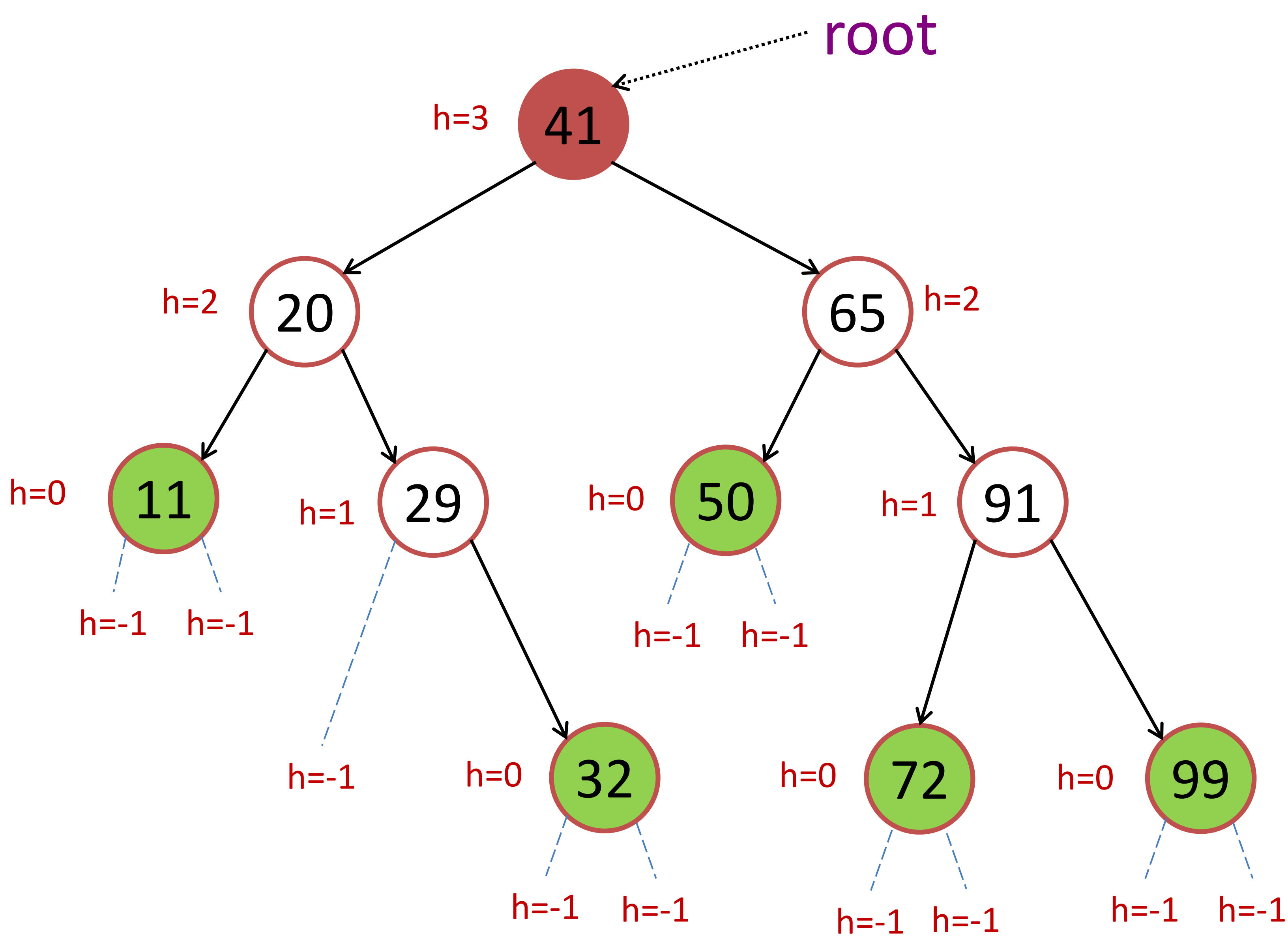
x.size = 0 (if x is an empty tree)

x.size = x.left.size + x.right.size + 1 (all other cases)

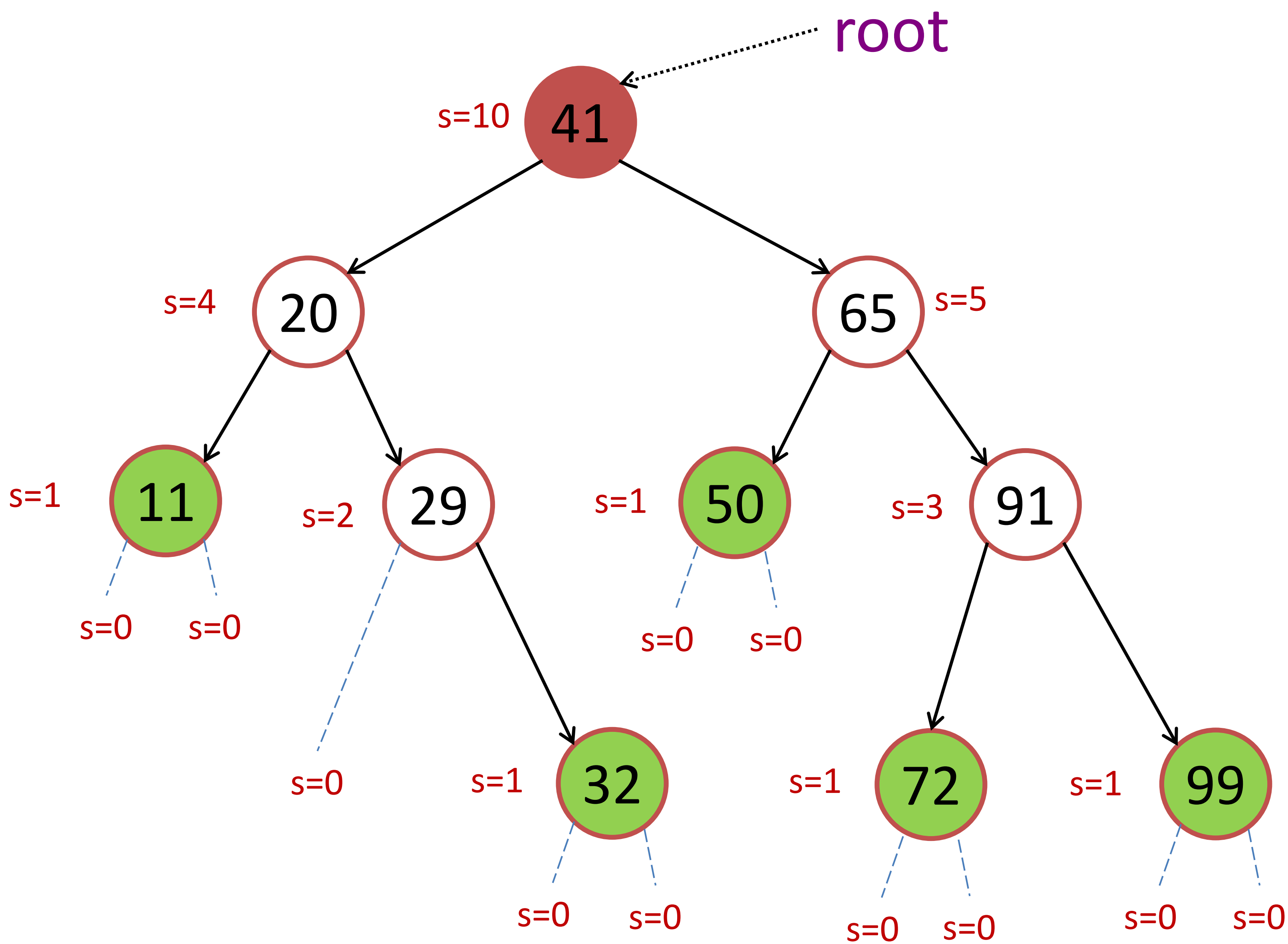
The height of the BST is thus: root.height

The size of the BST is thus: root.size

Binary Search Trees: Height (h)

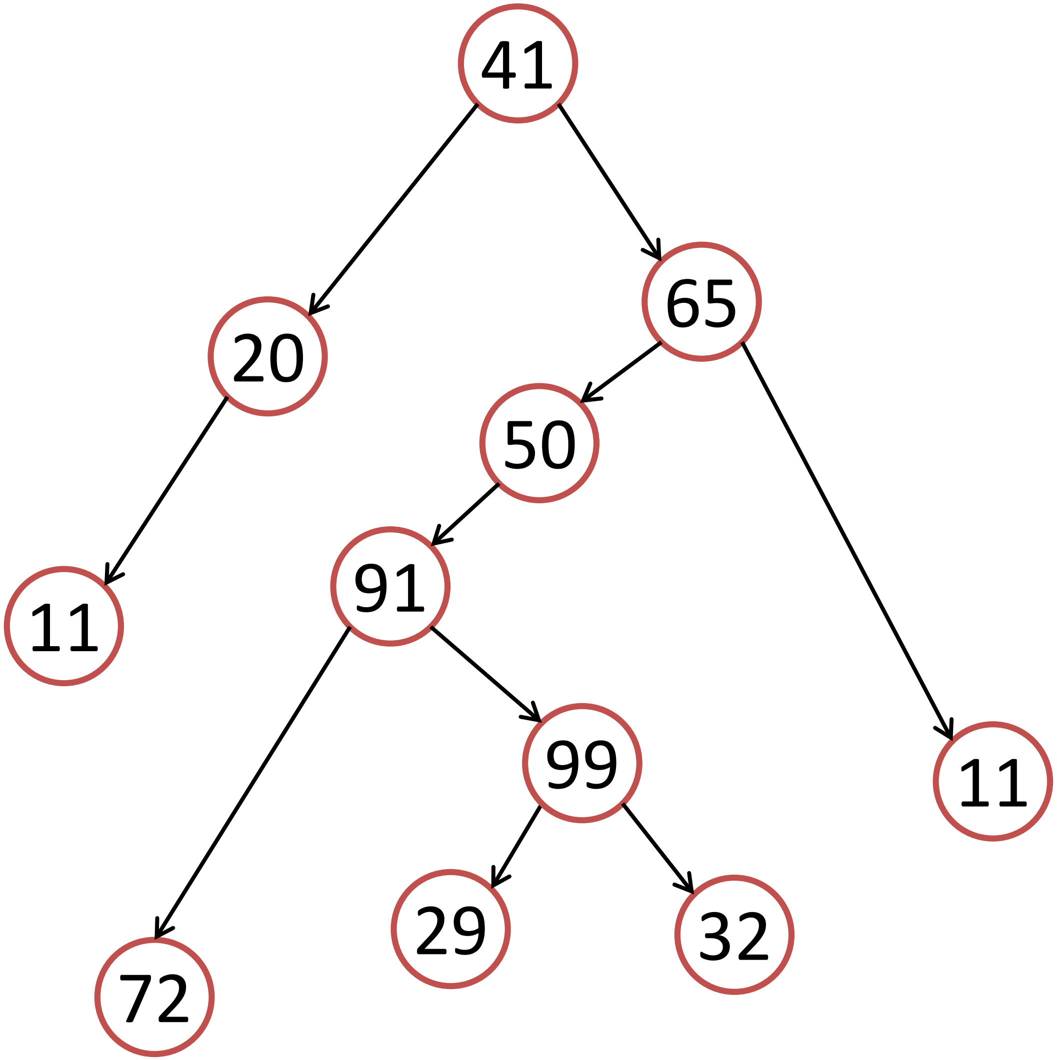


Binary Search Trees: Size (s)



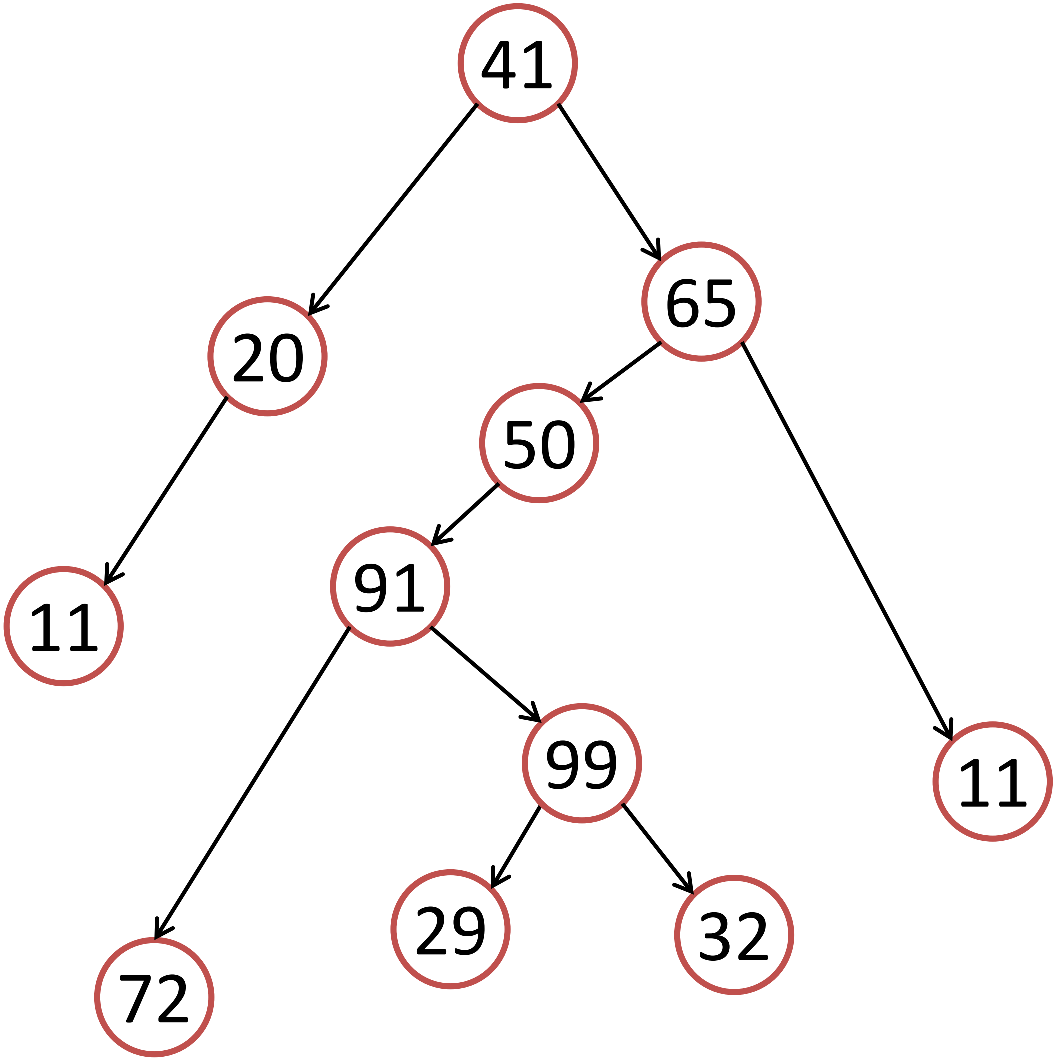
The height of this tree is?

1. 2
2. 4
3. 5
4. 6
5. 7
6. 42



The size of this tree is?

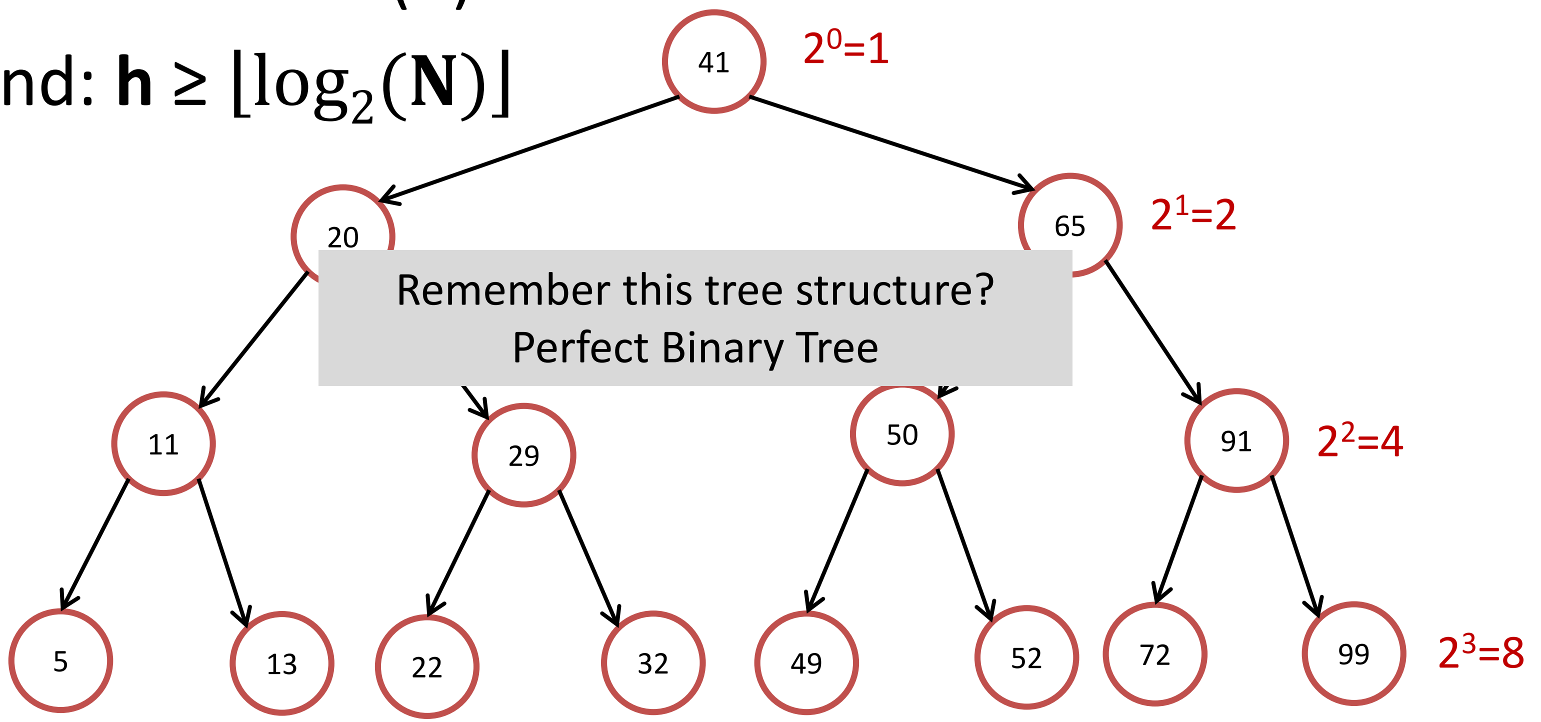
1. 10
2. 11
3. 12
4. 13
5. 14
6. 15



The Importance of Being Balanced

Most operations take O(**h**) time

Lower bound: $\mathbf{h} \geq \lfloor \log_2(\mathbf{N}) \rfloor$



$N = 1 + 2 + 4 + \ldots + 2^h = 2^0 + 2^1 + 2^2 + \ldots + 2^h$

$= 2^{h+1} - 1 < 2^{h+1}$ (sum of geometric progression)

$\log_2(N) < \log_2(2^{h+1}) \Rightarrow \log_2(N) < (h+1) * \log_2(2) \Rightarrow h > \log_2(N)-1$

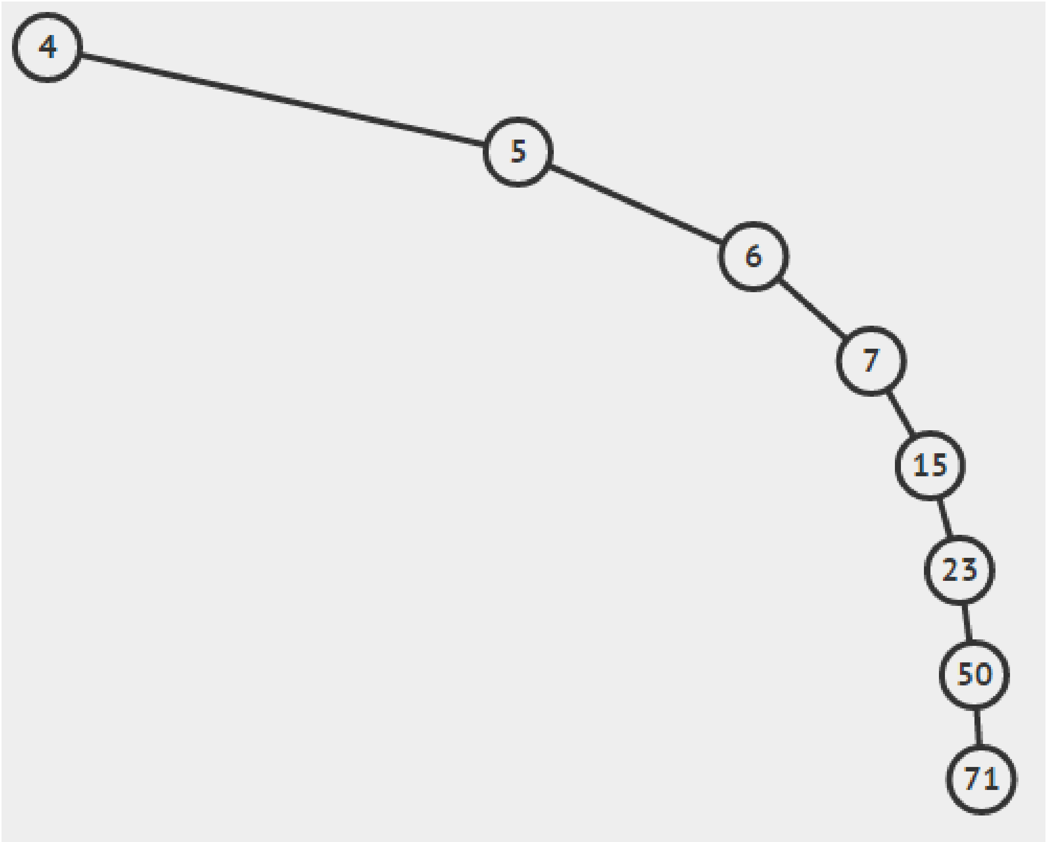
$\Rightarrow h \geq \lfloor \log_2(\mathbf{N}) \rfloor$

The Importance of Being Balanced

Most operations take O(**h**) time

Upper bound: **h** ≤ **N**-1 ➔ **h** < **N**

Remember this tree structure?
The worst case for BST…



The Importance of Being Balanced

Most operations take O(**h**) time

Combined bound: $\lfloor \log_2(\mathbf{N}) \rfloor \leq \mathbf{h} < \mathbf{N}$

$\log_2$(**N**) versus **N** in picture (revisited with larger numbers):

N = 500

$\log_2$(**N**) ~ 9 After learning CS2040S ☺

N = 1 000

$\log_2$(**N**) ~ 10 After learning CS2040S ☺

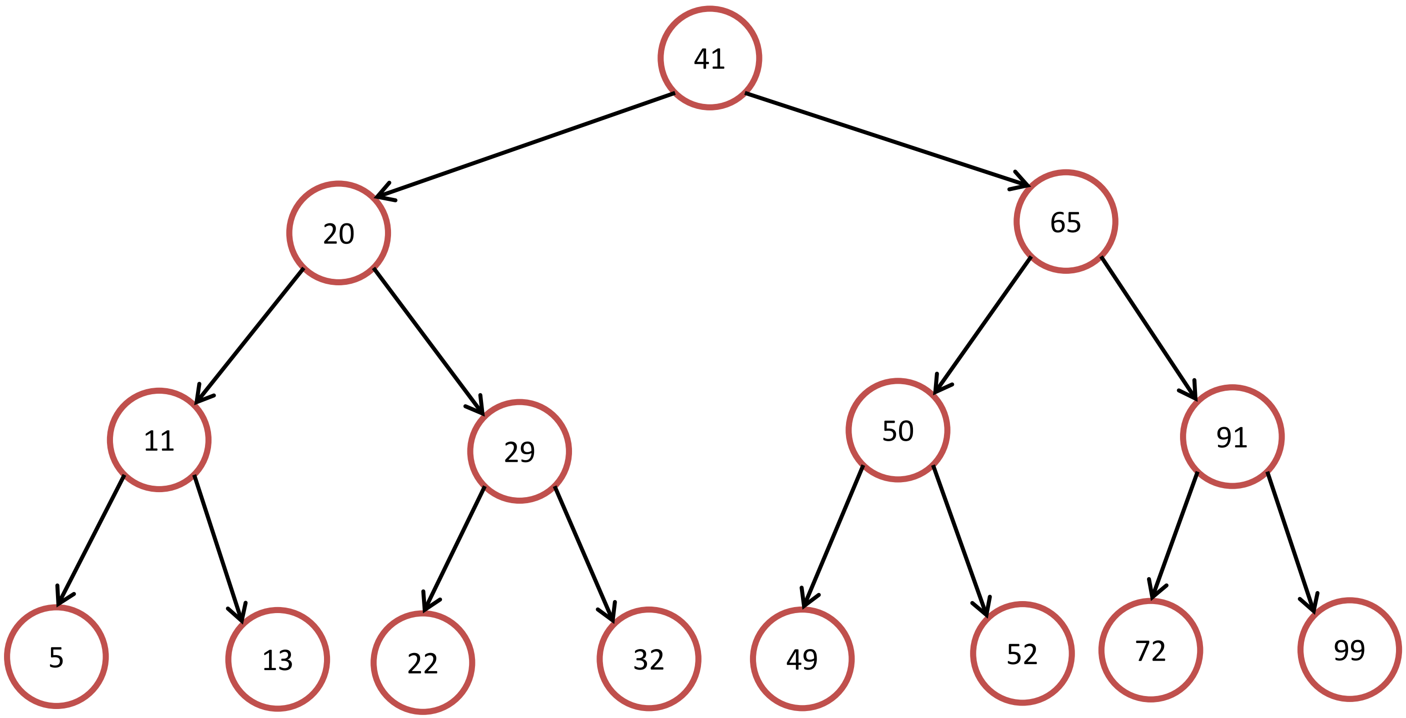
We say a BST is balanced if **h** = O(log **N**), i.e. **c** * log **N**

On a balanced BST, all operations run in O(log **N**) time

The Importance of Being Balanced

Example of a perfectly balanced BST:

- This is hard to achieve though…



The Importance of Being Balanced

How to get a balanced tree:

- Define a good property of a tree
- Show that if the good property holds, then the tree is **balanced**
- After every insert/delete, make sure the good property still holds
  - If not, fix it!

Adelson-Velskii & Landis, 1962 (~57 years ago… :O)

Can be a little bit frustrating if you are not comfortable with recursion

Hang on…

AVL TREES

AVL Trees [Adelson-Velskii & Landis 1962]

Step 1: Augment (i.e. add more information)

In every vertex x, we also store its height: **x.height**

(Note that x already has: **x.left**, **x.right**, **x.parent**, and **x.key**)

During insertion and deletion, we *also* update **height**:

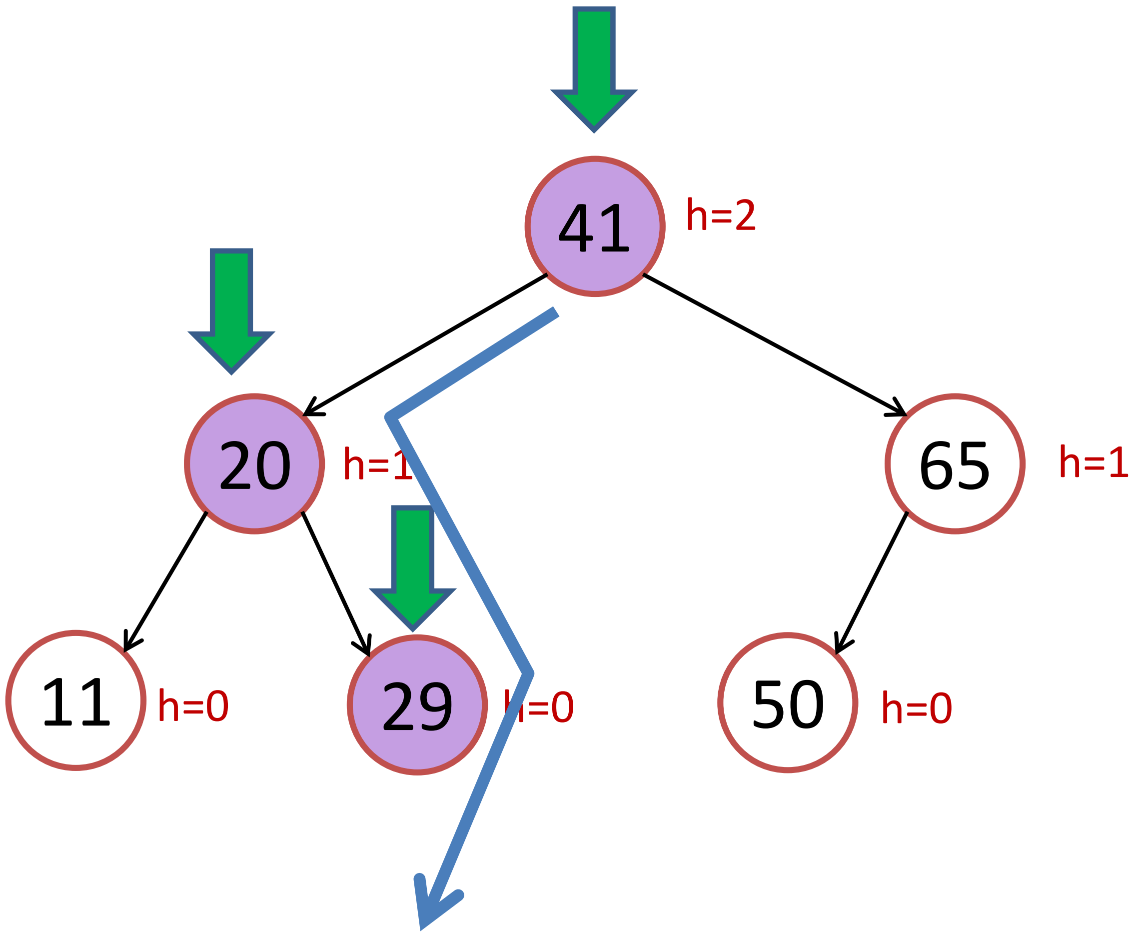
```
insert(x, v)
  // ... same as before ...
  x.height = max(x.left.height, x.right.height) + 1

// update height during deletion too (same as above)
// update on attribute size can be done similarly
```

Binary Search Trees

Height of empty trees are ignored in this illustration (all -1)

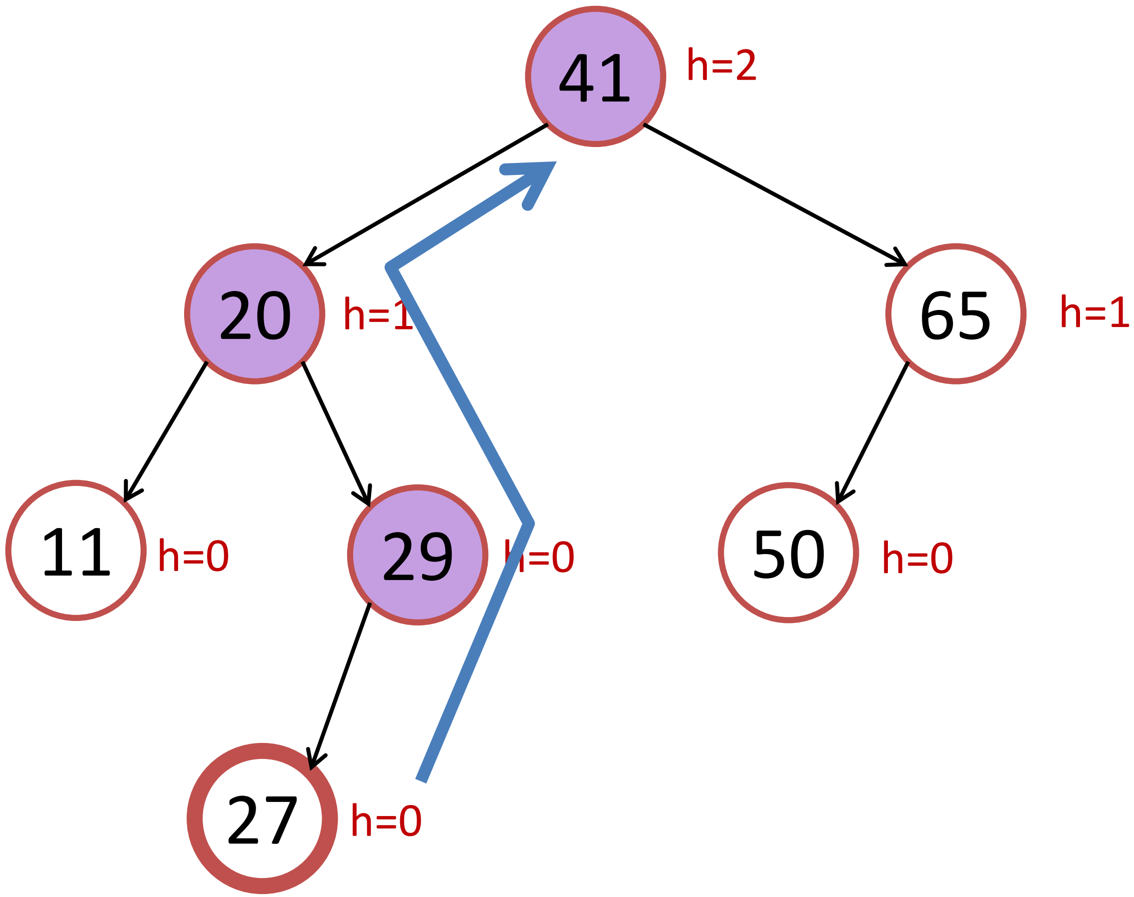
insert(27)



Height information during insertion/deletion is not shown in VisuAlgo (yet)

Binary Search Trees

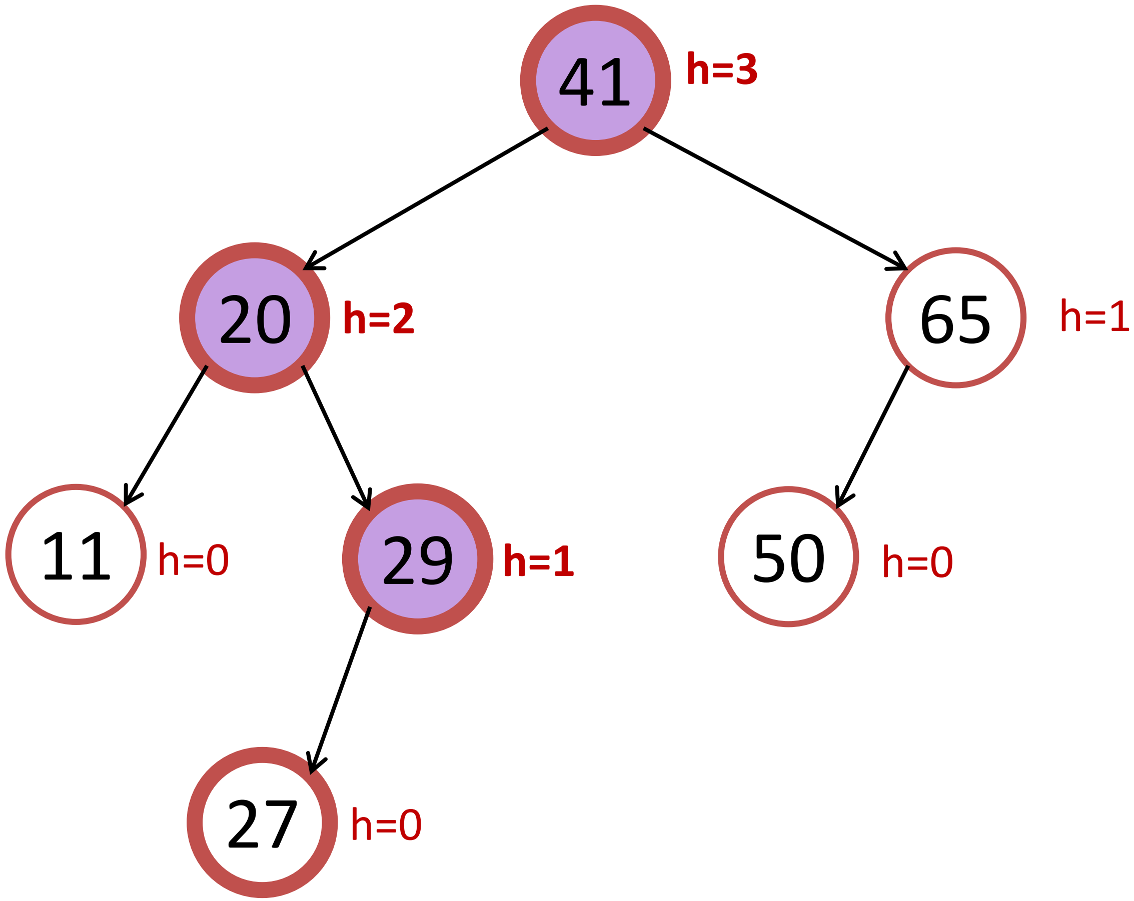
insert(27)



Binary Search Trees

insert(27)

Notice that only vertices along the insertion path may have their height attribute updated...



AVL Trees [Adelson-Velskii & Landis 1962]

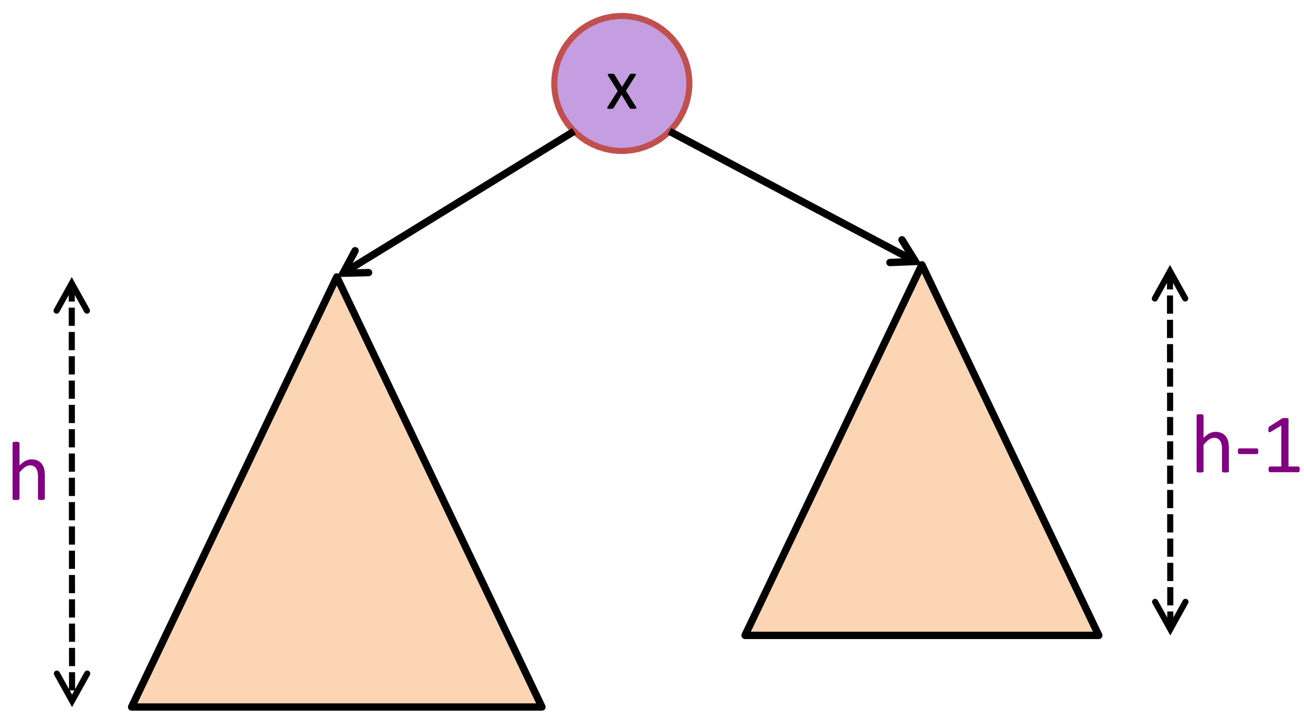
Step 2: Define Invariant (something that will not change)

A vertex x is said to be <u>height-balanced</u> if:

$$|x.left.height - x.right.height| \leq 1$$

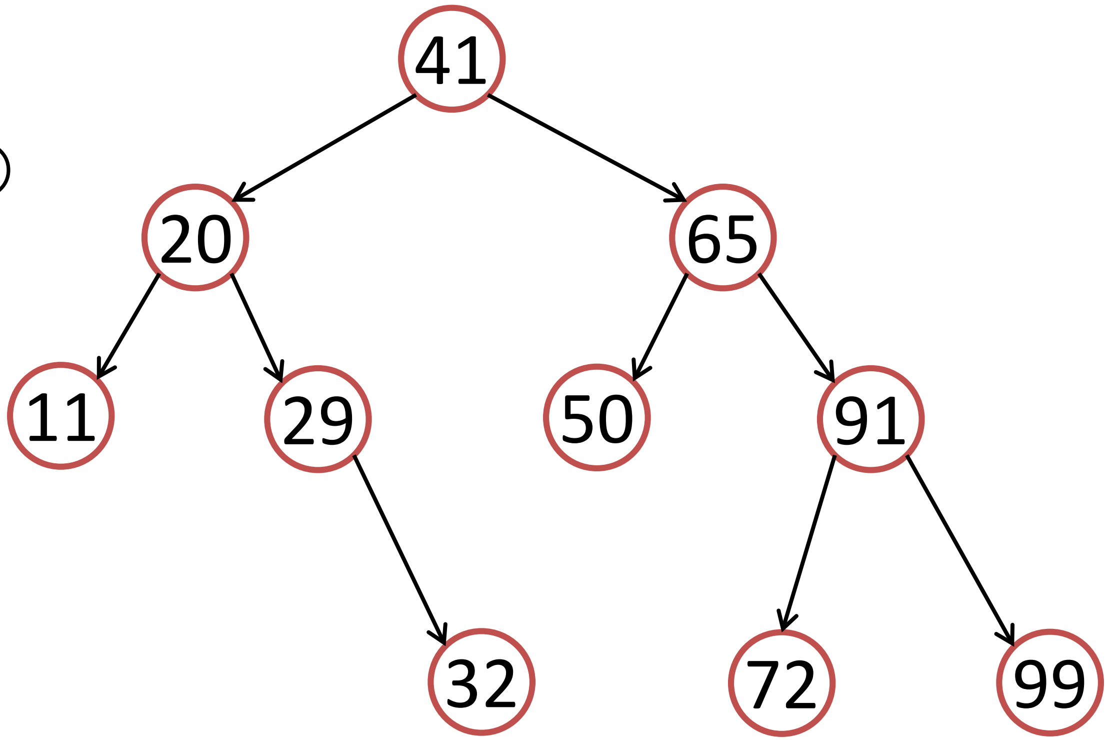
A binary search tree is said to be <u>height balanced</u> if:

every vertex in the tree is height-balanced



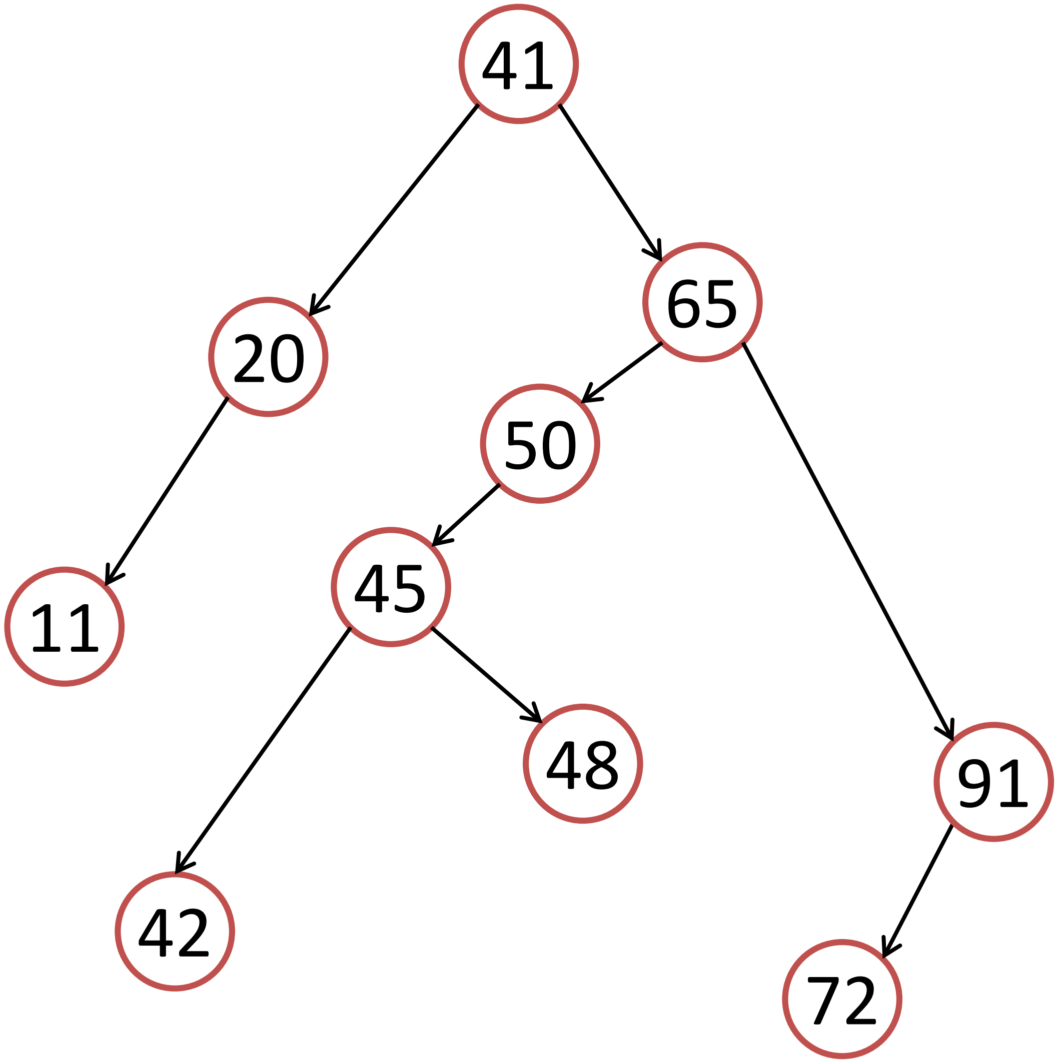
Is this tree height-balanced according to AVL?

1. Yes
2. No
3. I am confused… ☹



Is this tree height-balanced according to AVL?

1. Yes
2. No
3. I am confused… ☹



Height-Balanced Trees

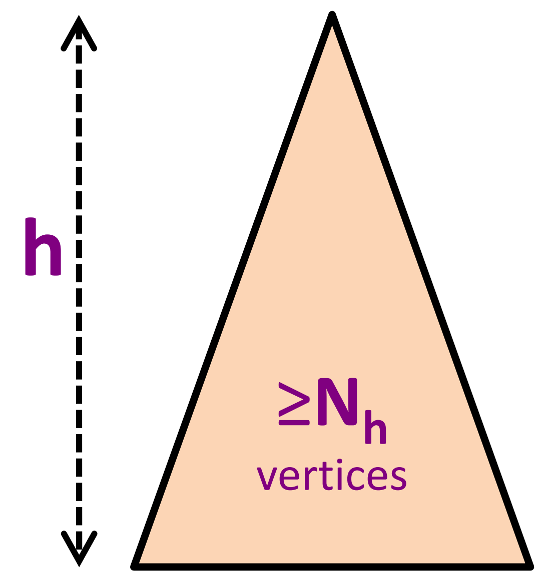
Claim:

A height-balanced tree with $\mathbf{N}$ vertices has height $\mathbf{h} < 2 * \log_2(\mathbf{N})$

Proof (do **not** be scared):

Let $\mathbf{N_h}$ be the minimum number of vertices in a height-balanced tree of height $\mathbf{h}$

The actual number of vertices $\mathbf{N} \geq \mathbf{N_h}$



Height-Balanced Trees

Proof:

Let N_h be the minimum number of vertices in a height-balanced tree of height h

$N_h = 1 + N_{h-1} + N_{h-2}$

$N_h > 1 + 2N_{h-2}$ (as $N_{h-1} > N_{h-2}$)

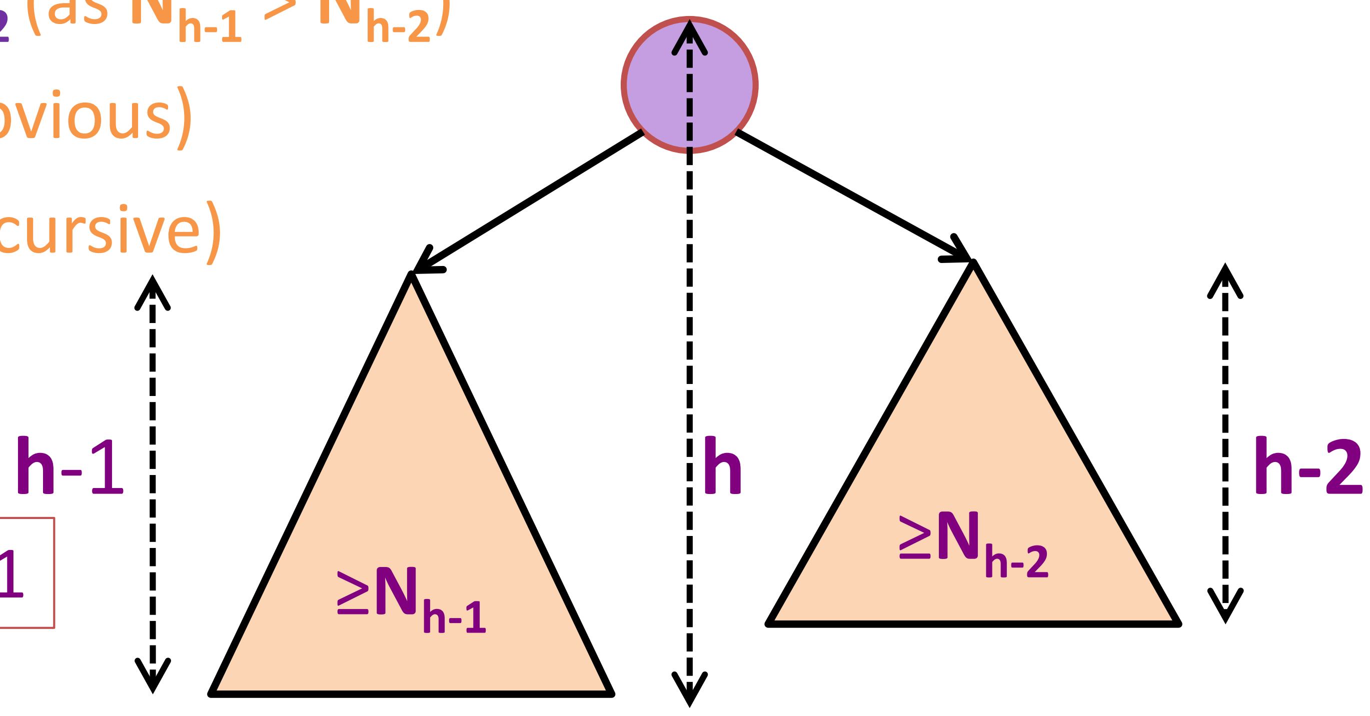
$N_h > 2N_{h-2}$ (obvious)

$= 4N_{h-4}$ (recursive)

$= 8N_{h-6}$

$= \ldots$

Base case: $N_0 = 1$



Height-Balanced Trees

Proof:

Let N_h be the minimum number of vertices in a height-balanced tree of height h

$N_h = 1 + N_{h-1} + N_{h-2}$

$N_h > 1 + 2N_{h-2}$

$$\left.\begin{aligned} N_h &> 2N_{h-2} \\ &> 4N_{h-4} \\ &> 8N_{h-6} \\ &> \ldots \end{aligned}\right\}$$

As each step we reduce **h** by 2,
Then we need to do this step **h**/2 times
to reduce **h** (assume **h** is even) to 0

Base case: $N_0 = 1$

$N_h > 2^{h/2} N_0$

$N_h > 2^{h/2}$

Height-Balanced Trees

Claim:

A height-balanced tree is balanced,
i.e. has height $\mathbf{h} = O(\log(\mathbf{N}))$

We have shown that: $\mathbf{N_h} > 2^{\mathbf{h}/2}$ and $\mathbf{N} \geq \mathbf{N_h}$

$\mathbf{N} \geq \mathbf{N_h} > 2^{\mathbf{h}/2}$

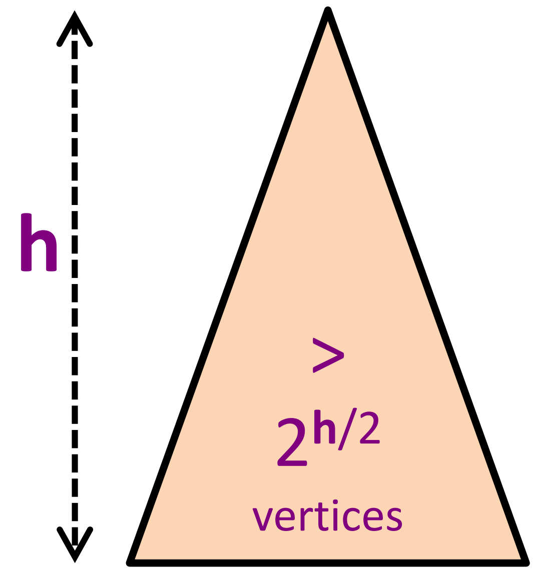
$\mathbf{N} > 2^{\mathbf{h}/2}$

$\log_2(\mathbf{N}) > \log_2(2^{\mathbf{h}/2})$ ($\log_2$ on both side)

$\log_2(\mathbf{N}) > \mathbf{h}/2$ (formula simplification)

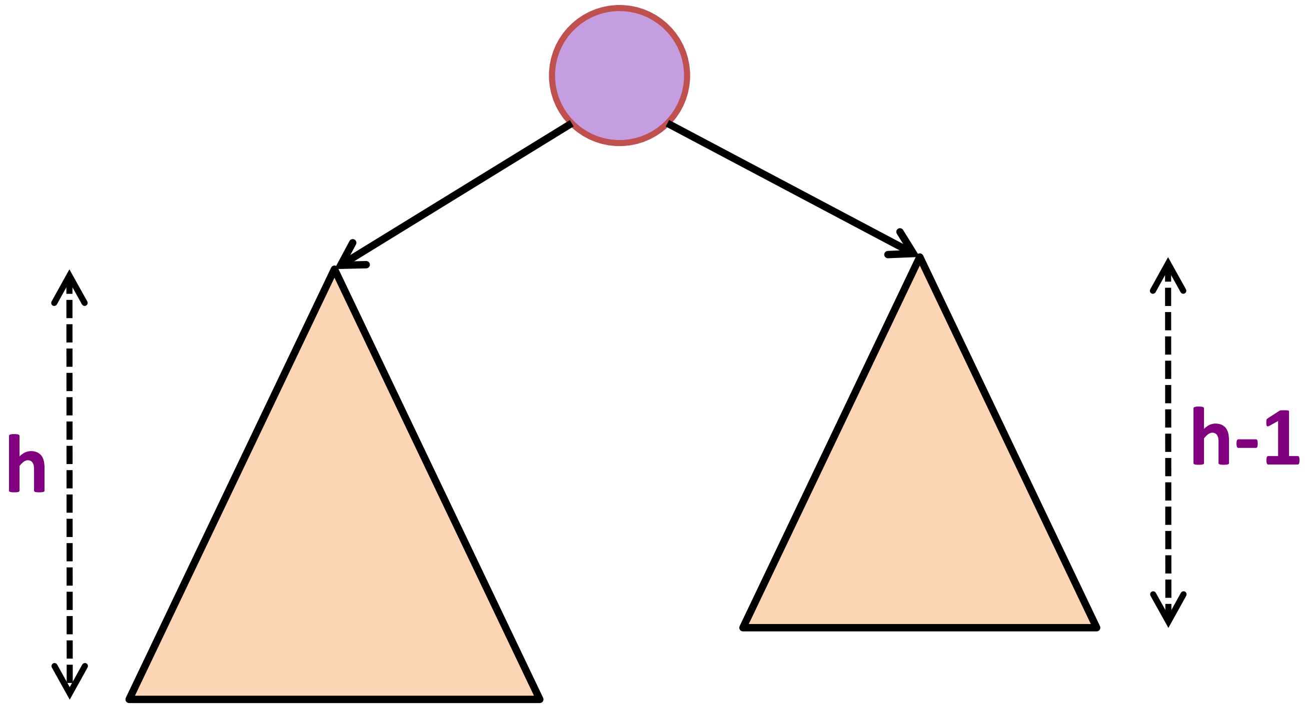
$2 * \log_2(\mathbf{N}) > \mathbf{h}$ or $\mathbf{h} < 2 * \log_2(\mathbf{N})$

$\mathbf{h} = O(\log(\mathbf{N}))$



AVL Trees [Adelson-Velskii & Landis 1962]

Step 3: Show how to maintain height-balance



Insertion to an AVL Tree

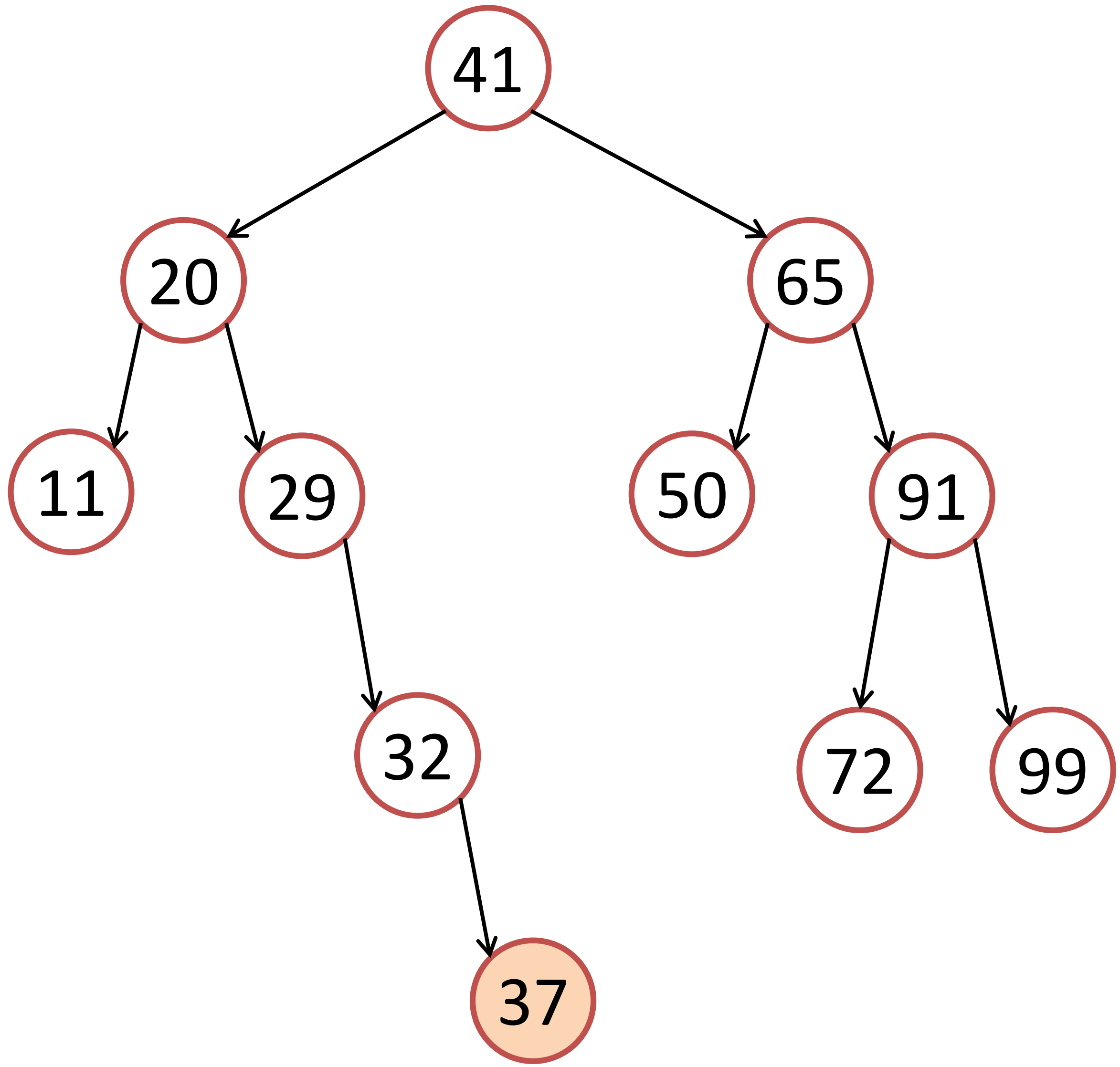
insert(37)

Initially balanced

But no longer balanced after Inserting 37

Need to rebalance!

But how?



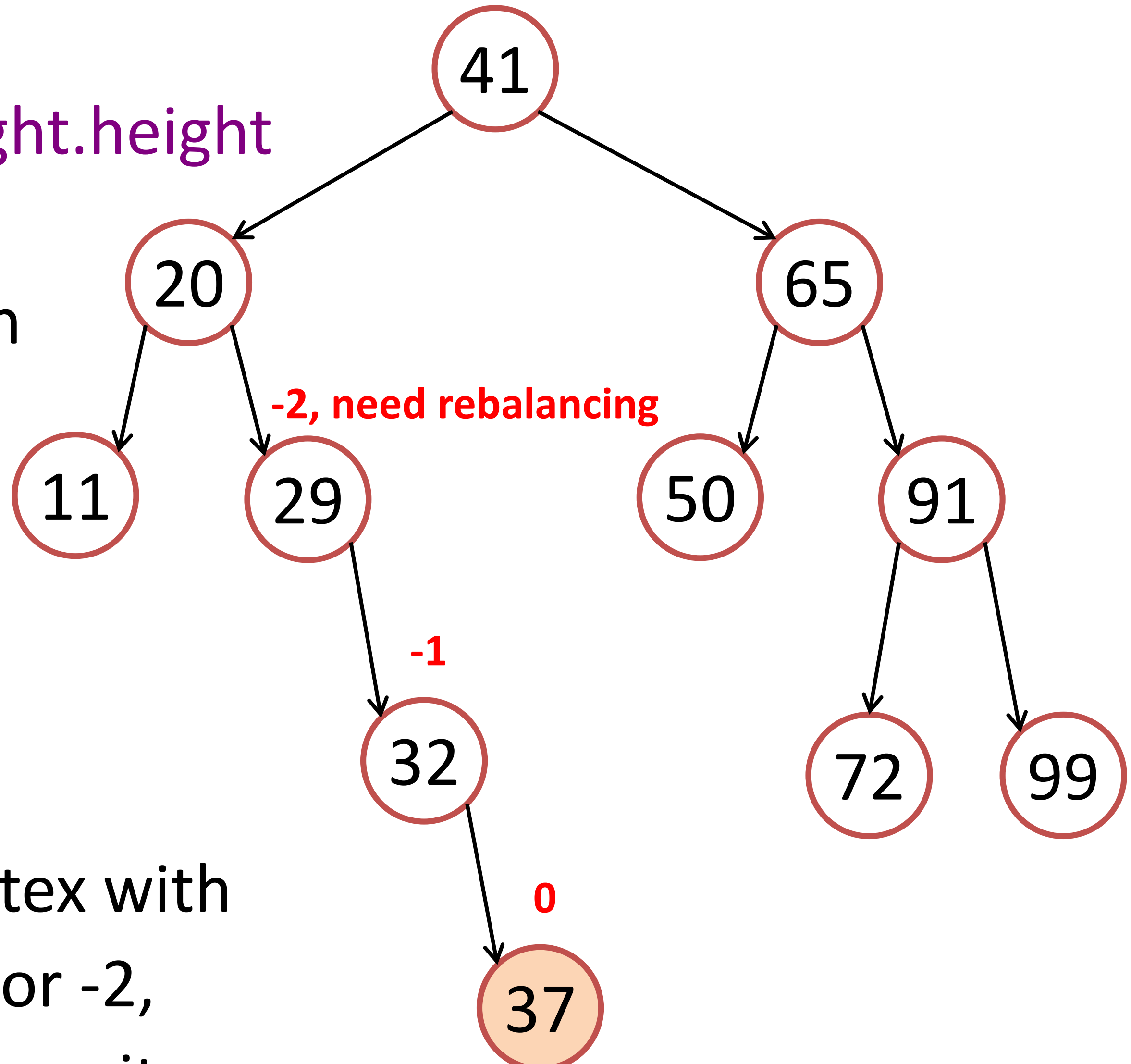
"Infinite more" examples in VisuAlgo…

Balance Factor (bf(x))

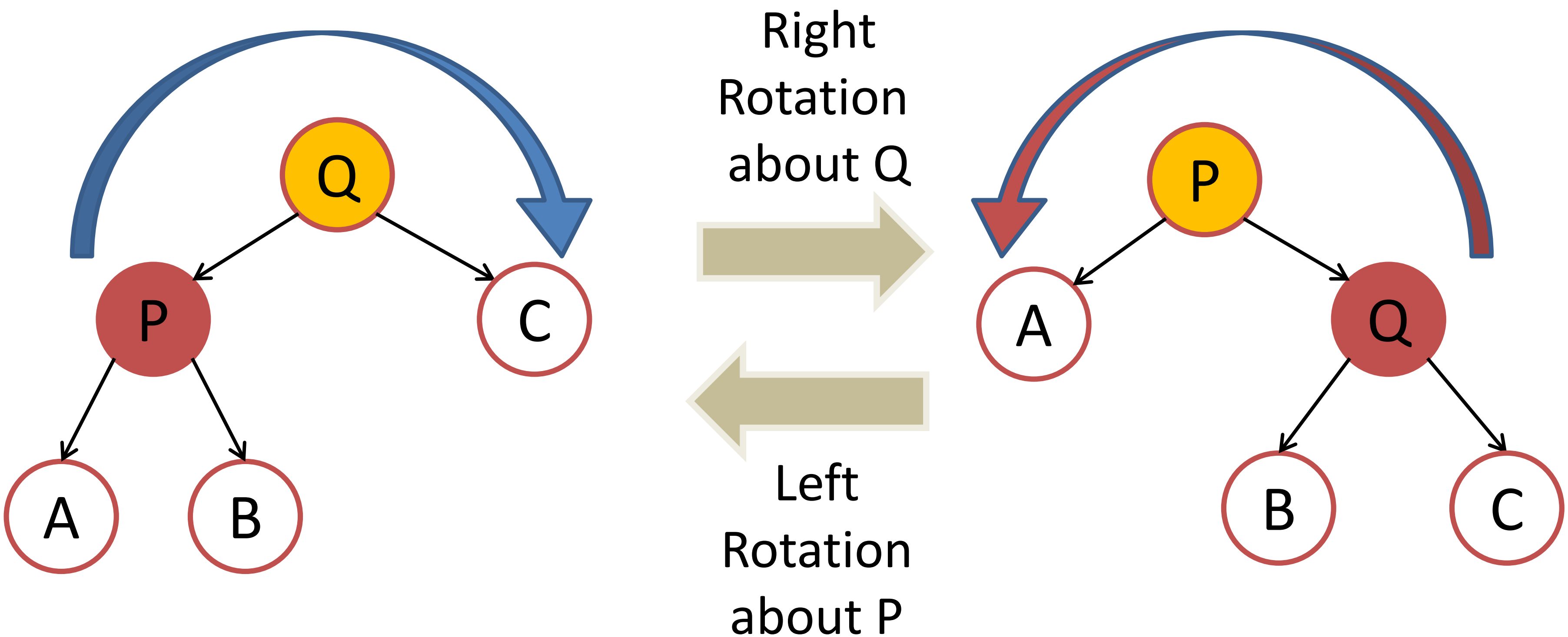
bf(x) =
x.left.height - x.right.height

From the insertion point, check the balance factor of each vertex up to the root

Once we have vertex with balance factor +2 or -2, we have to rebalance it



Tree Rotations (1)



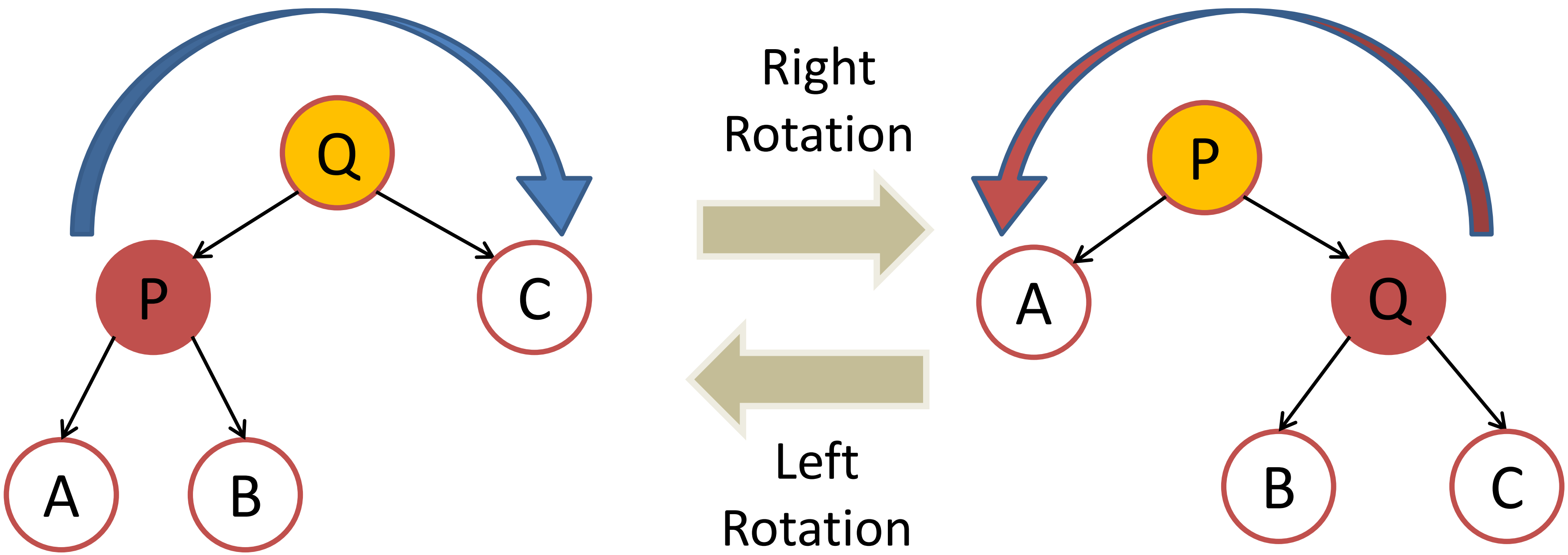
Right Rotation

- Need Q to have a left child P
- Make Q right child of P
- Other manipulations …

Left Rotation

- Need P to have a right child Q
- Make P left child of Q
- Other manipulations …

Tree Rotations (2)



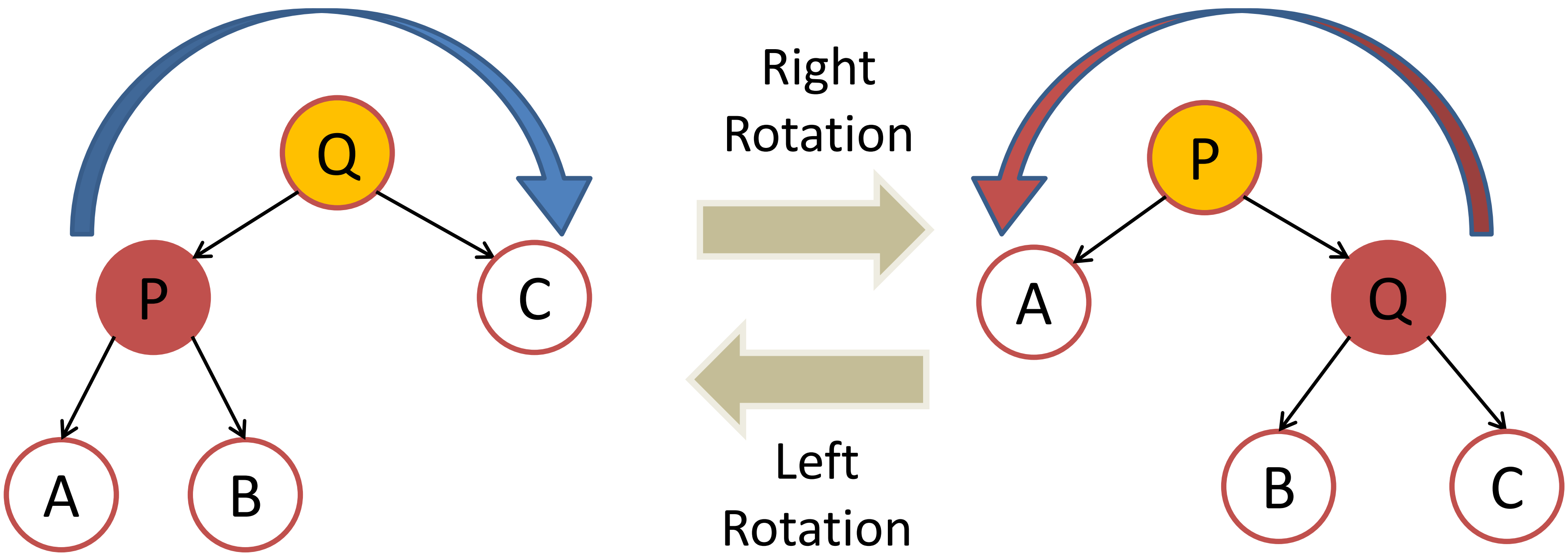
Right Rotation

- Need Q to have a left child P
- Make Q right child of P
- Make B (right child of P) left child of Q

Left Rotation

- Need P to have a right child Q
- Make P left child of Q
- Make B (left child of Q) right child of P

Tree Rotations (3)



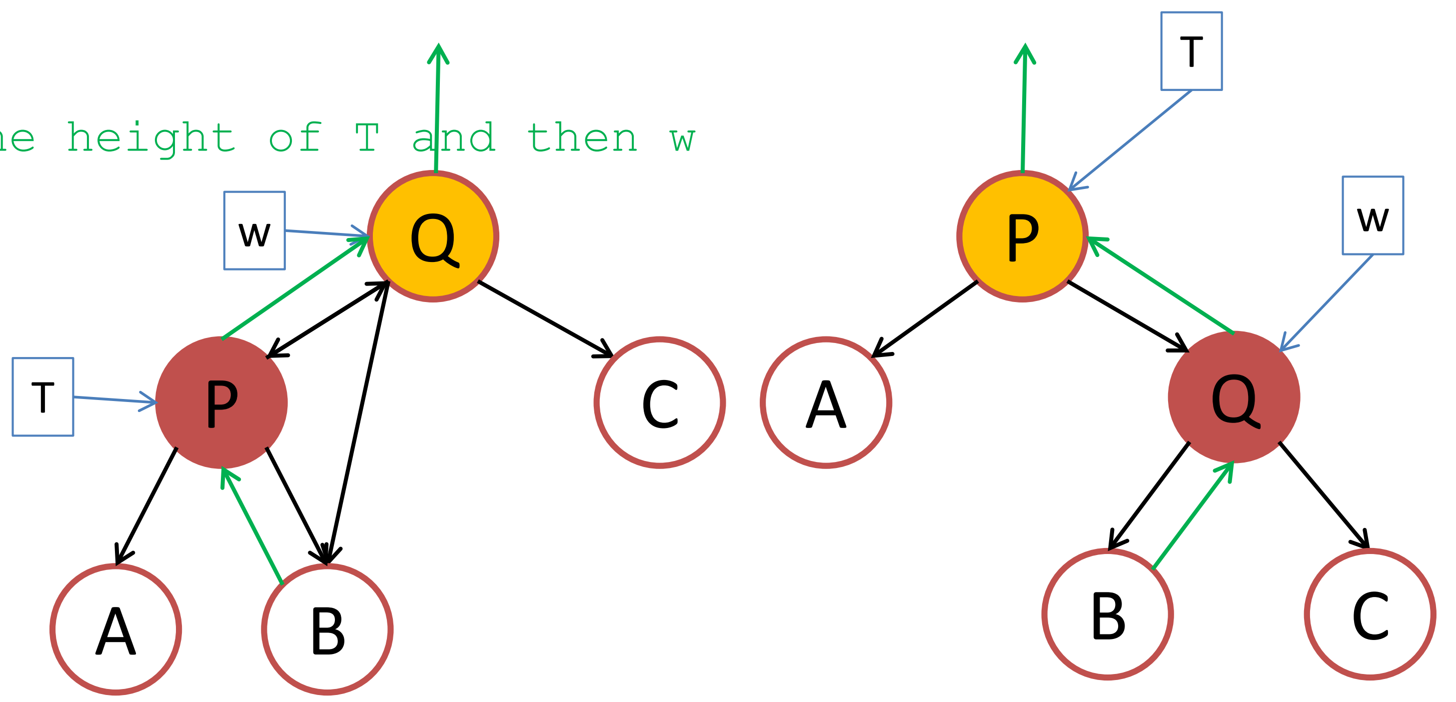
Rotations maintain ordering of keys

$\Rightarrow$ Maintains BST property *(see vertex B where $P \leq B \leq Q$)*

Tree Rotations Pseudo Code → O(1)

```
BSTVertex rotateLeft(BSTVertex T) // pre-req: T.right != null
   BSTVertex w = T.right
   w.parent = T.parent
   T.parent = w
   T.right = w.left
   if (w.left != null) w.left.parent = T
   w.left = T
   // Update the height of T and then w
   return w
```

`rotateRight` is the mirrored version of this pseudocode



This slide is can be confusing without the animation

Four Possible Cases

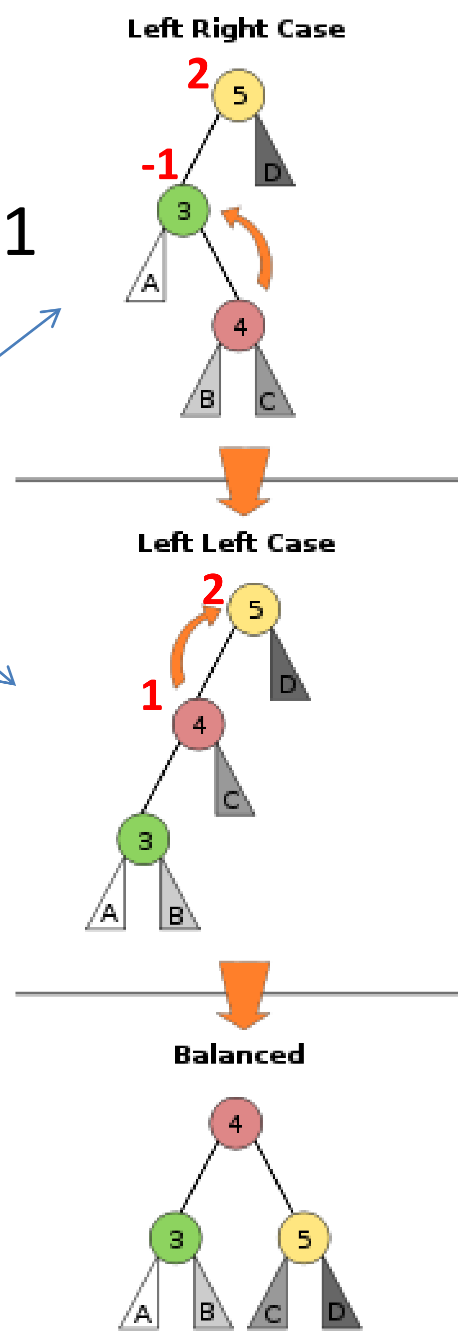
bf(x) = +2 and 0 <= bf(x.left) <= 1

rightRotate(x)

bf(x) = +2 and bf(x.left) = -1

leftRotate(x.left)

rightRotate(x)



Pictures from Wikipedia

Four Possible Cases

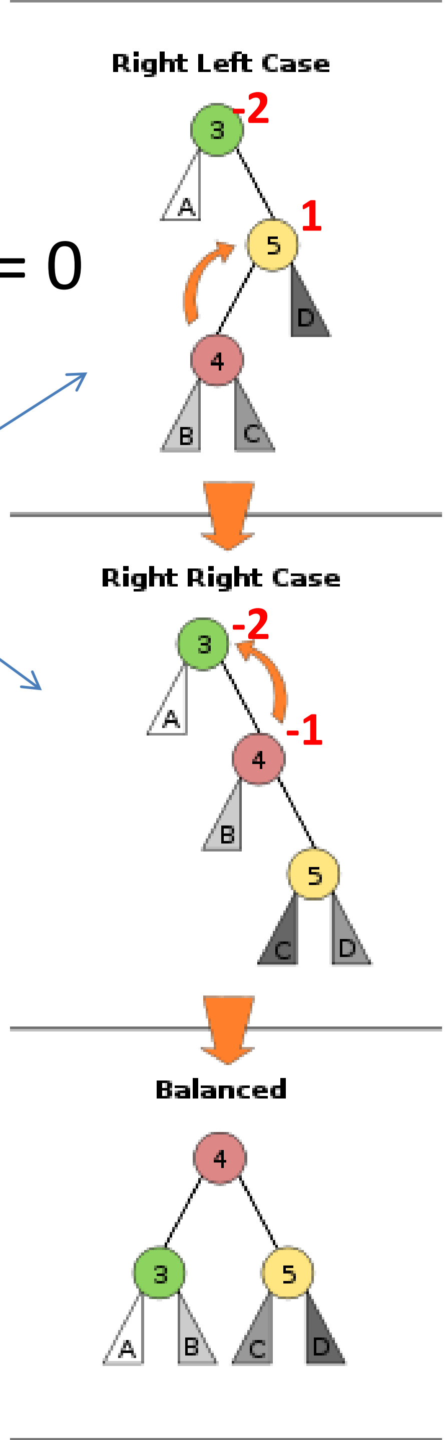
bf(x) = -2 and -1 <= bf(x.right) <= 0

leftRotate(x)

bf(x) = -2 and bf(x.right) = 1

rightRotate(x.right)

leftRotate(x)

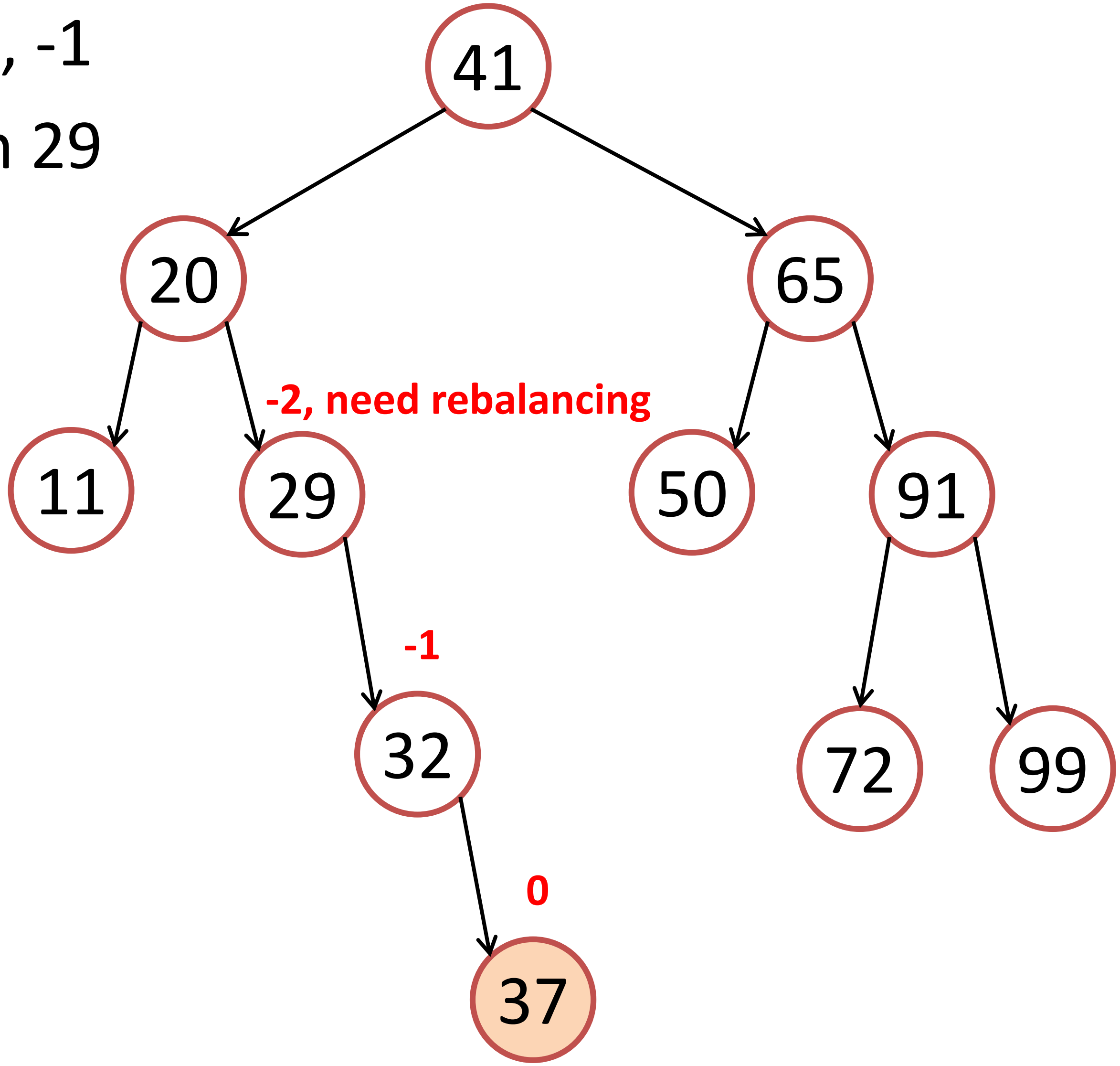


Pictures from Wikipedia

Rebalancing (1)

This is a case of -2, -1

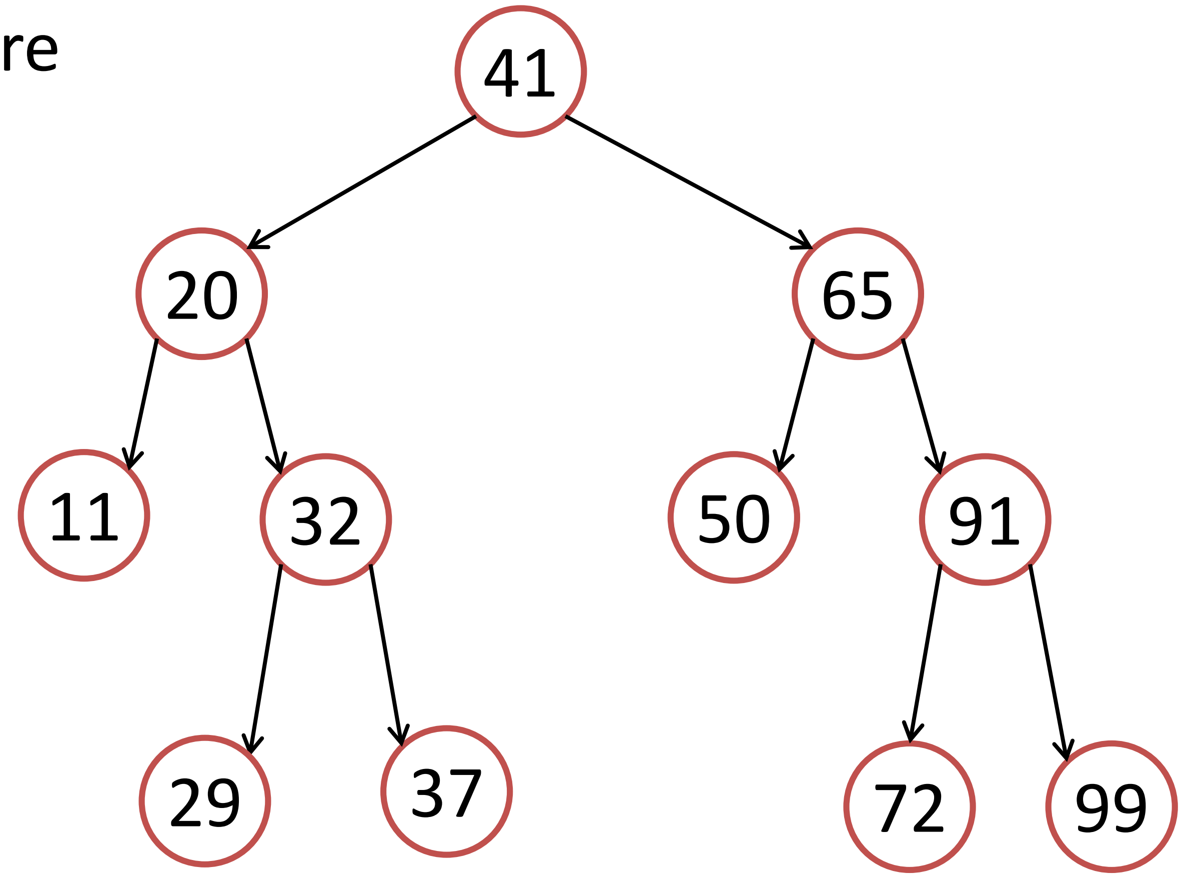
Do left rotate on 29



“Infinite more” examples in VisuAlgo…

Rebalancing (2)

Now all vertices are balanced again



“Infinite more” examples in [VisuAlgo AVL Tree Visualization]

Insertion to an AVL Tree

Summary:

- Just insert the key as in normal BST
- Walk up the AVL tree from the insertion point to root:
  - At every step, update height & check balance factor
  - If a certain vertex is out-of-balance (+2 or -2), use rotations to rebalance
    - During insertion to an AVL tree, you can only trigger one of the four possible rebalancing cases as shown earlier once !

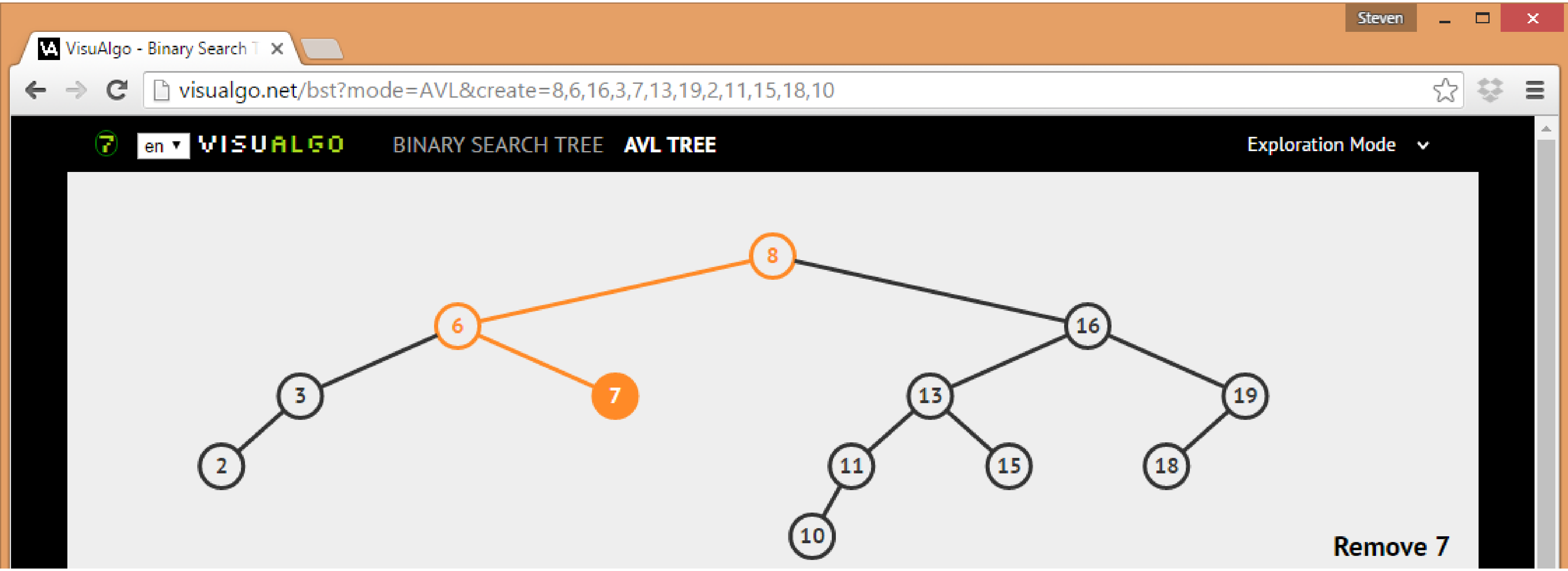
Deletion from an AVL Tree

Deletion is quite similar to Insertion:

- Just delete the key as in normal BST
- Walk up the AVL tree from the deletion point to root:
  - At every step, update height & check balance factor
  - If a certain vertex is out-of-balance (+2 or -2), use rotations to rebalance
    - The main difference compared to insertion into AVL tree is that you may trigger one of the four possible rebalancing cases several times, up to **h** = log **n** times :O, see this example (next slide)

AVL Tree Web-based Review

Create an AVL Tree using 8,6,16,3,7,13,19,2,11,15,18,10



Try **Remove (Delete)** vertex 7, it triggers **two (more than one)** rebalancing actions

Then try various **Insert operations** and notice that at most it will only trigger one (out of the four cases) of rebalancing actions

Balanced Search Trees

Many different flavors of balanced search trees

- AVL trees (Adelson-Velskii & Landis, 1962)
  - Discussed in this lecture…
- B-trees / 2-3-4 trees (Bayer & McCreight, 1972)
- BB[α] trees (Nievergelt & Reingold, 1973)
- Red-black trees (see CLRS 13)
- Splay trees (Sleator and Tarjan, 1985) ← Next Topic !
- Skip Lists (Pugh, 1989)
- Treaps (Seidel and Aragon, 1996)

Now, after we learn <u>balanced</u> BST

No	Operation	Unsorted Array	Sorted Array	<u>b</u>BST
1	Search(age)	O(**N**)	O(log **N**)	**O(log N)**
2	Insert(age)	O(1)	O(**N**)	**O(log N)**
3	FindOldest()	O(**N**)	O(1)	**O(log N)**
4	ListSortedAges()	O(**N** log **N**)	O(**N**)	O(**N**)
5	NextOlder(age)	O(**N**)	O(log **N**)	**O(log N)**
6	Remove(age)	O(**N**)	O(**N**)	**O(log N)**
7	GetMedian()	O(**N** log **N**)	O(1)	**????**
8	NumYounger(age)	O(**N** log **N**)	O(log **N**)	**????**

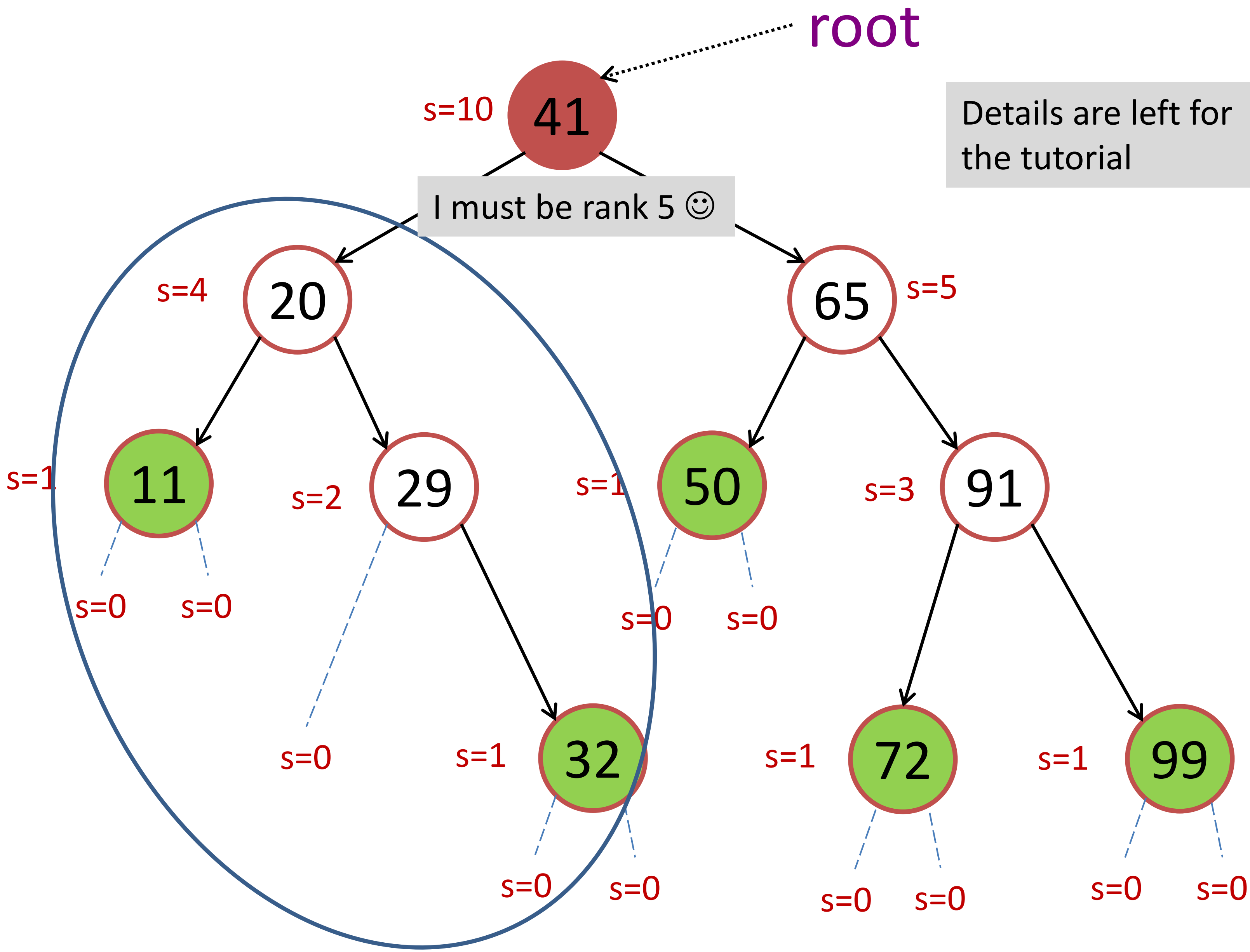
NumYounger(age) = rank(age)-1

Now, how to get rank(v) efficiently?

–

Binary Search Trees: Size (s)

Details are left for the tutorial



Balanced BST

Summary:

- The Importance of Being Balanced
- Height Balanced Trees – AVL Trees
- Tree Rotations to re-balance the Tree