Data Structures and Algorithms

Balancing Act

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Dr. Steven Halim for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Recording of modifications

- Currently, there are no modification on these contents.

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

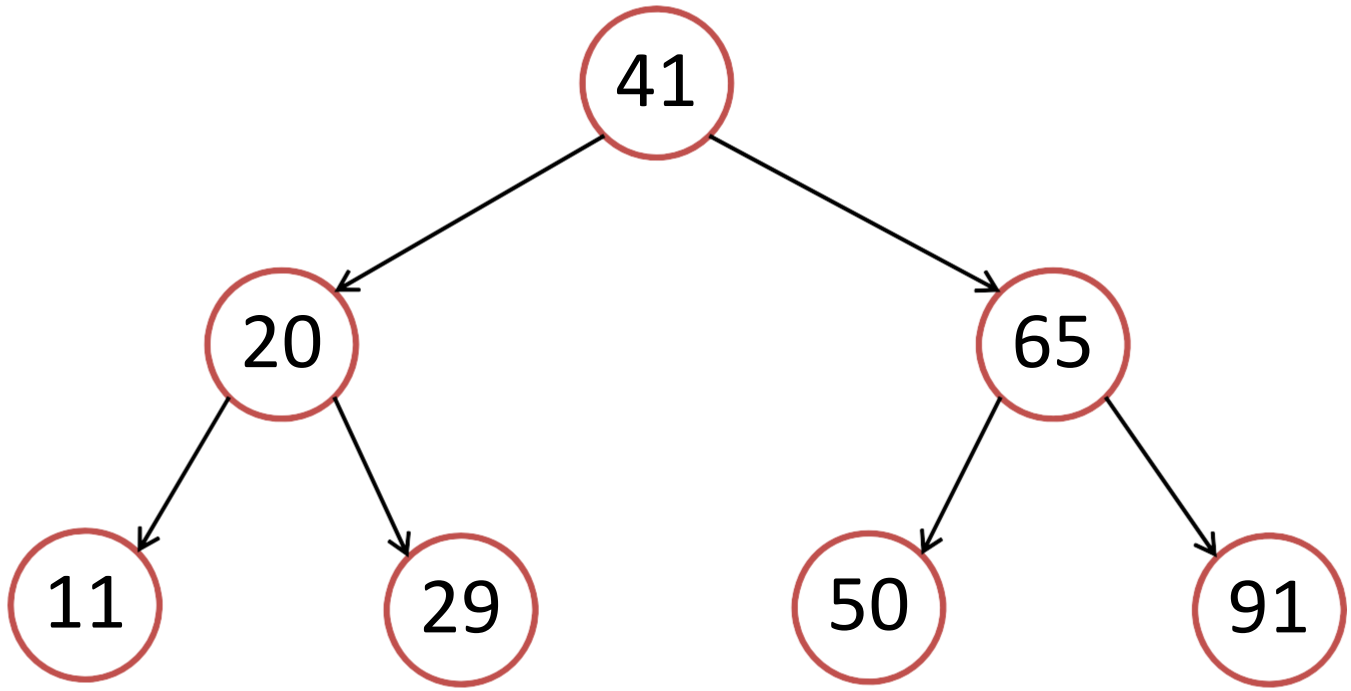
- Principle of “Height-Balanced”
- Keeping AVL Tree balanced via rotations
- Code is shown but not given (try this during PS2)

Eat All Foods In Moderation

Relation with CS2010 PS2: “The Baby Names Problem”

Reference in CP3 book: Page 43-47 + 380-382

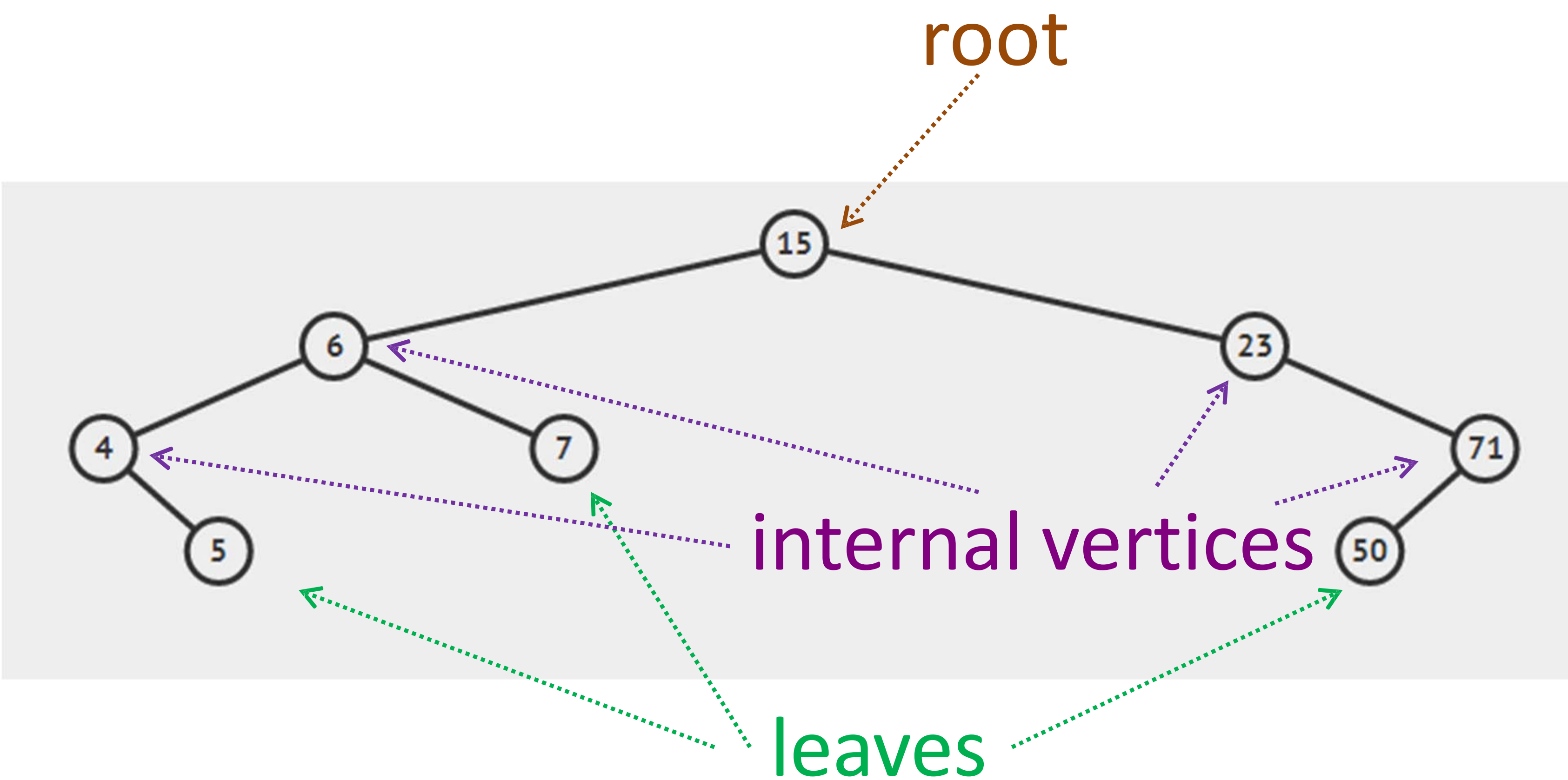
Binary Search Trees: Quick Review



- Vertex x has two children: **x.left**, **x.right** and one parent: **x.parent**
  - x.left/x.right/x.parent can be *null* for some vertices
- Vertex x has a key: **x.key**
- **BST Property**: all keys in left sub-tree < **x.key** < all keys in right sub-tree

BST Web-based Review

http://visualgo.net/bst.html



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 can be computed recursively:

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

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

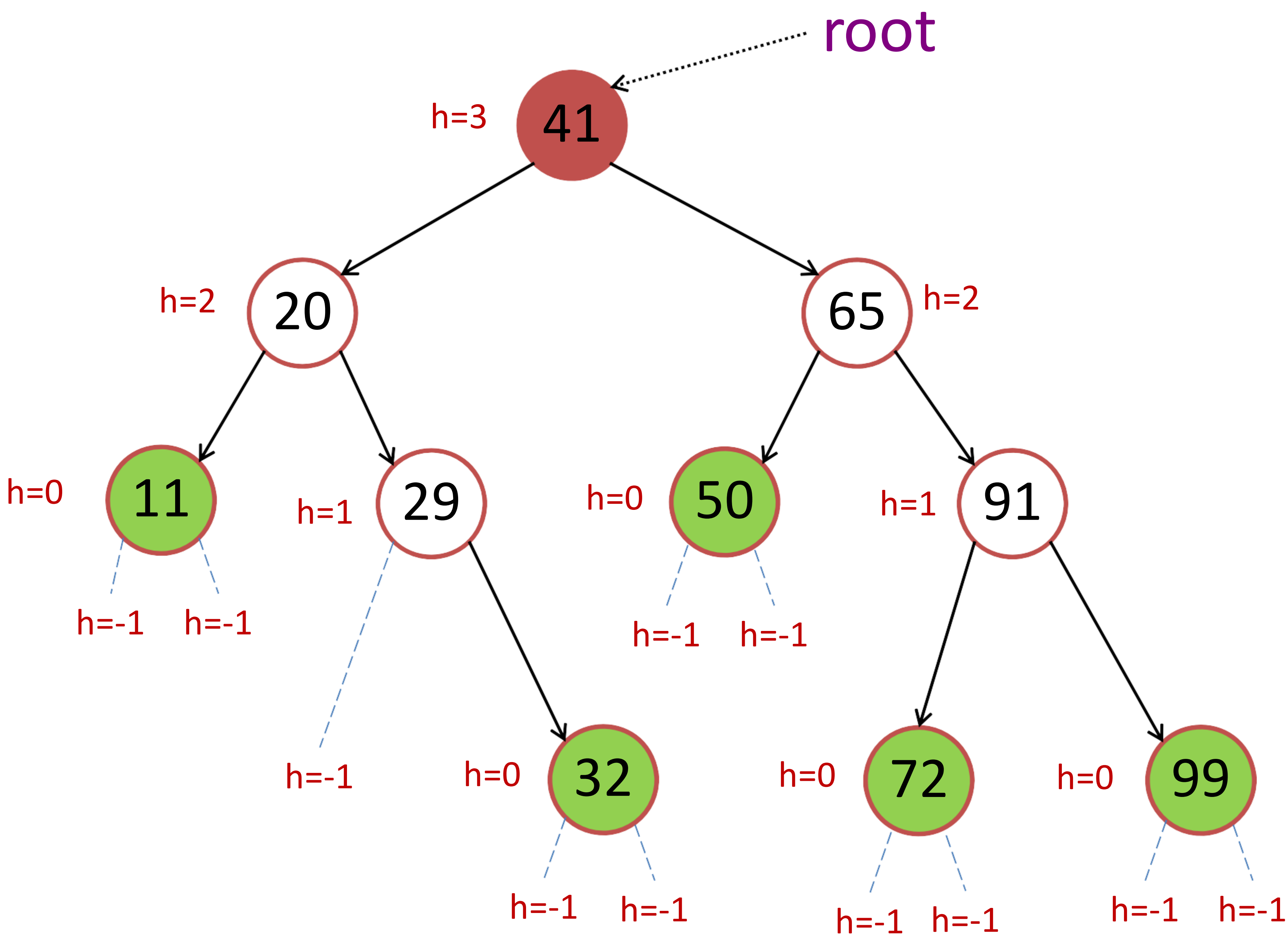
x.size = 0 (if x is an empty tree)

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

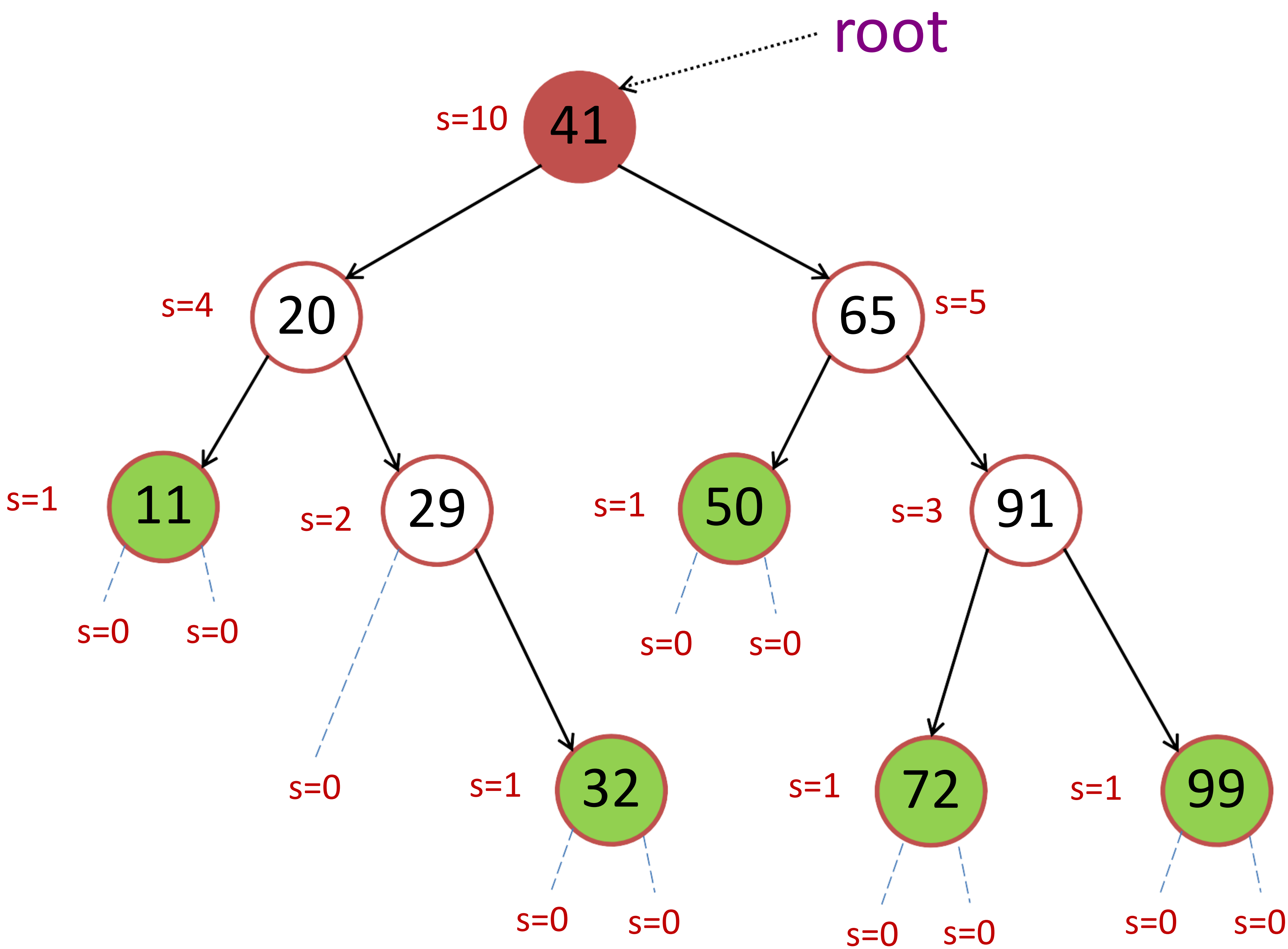
The height of the BST is thus: root.height

The size of the BST is thus: root.size

Binary Search Trees: Height (h)

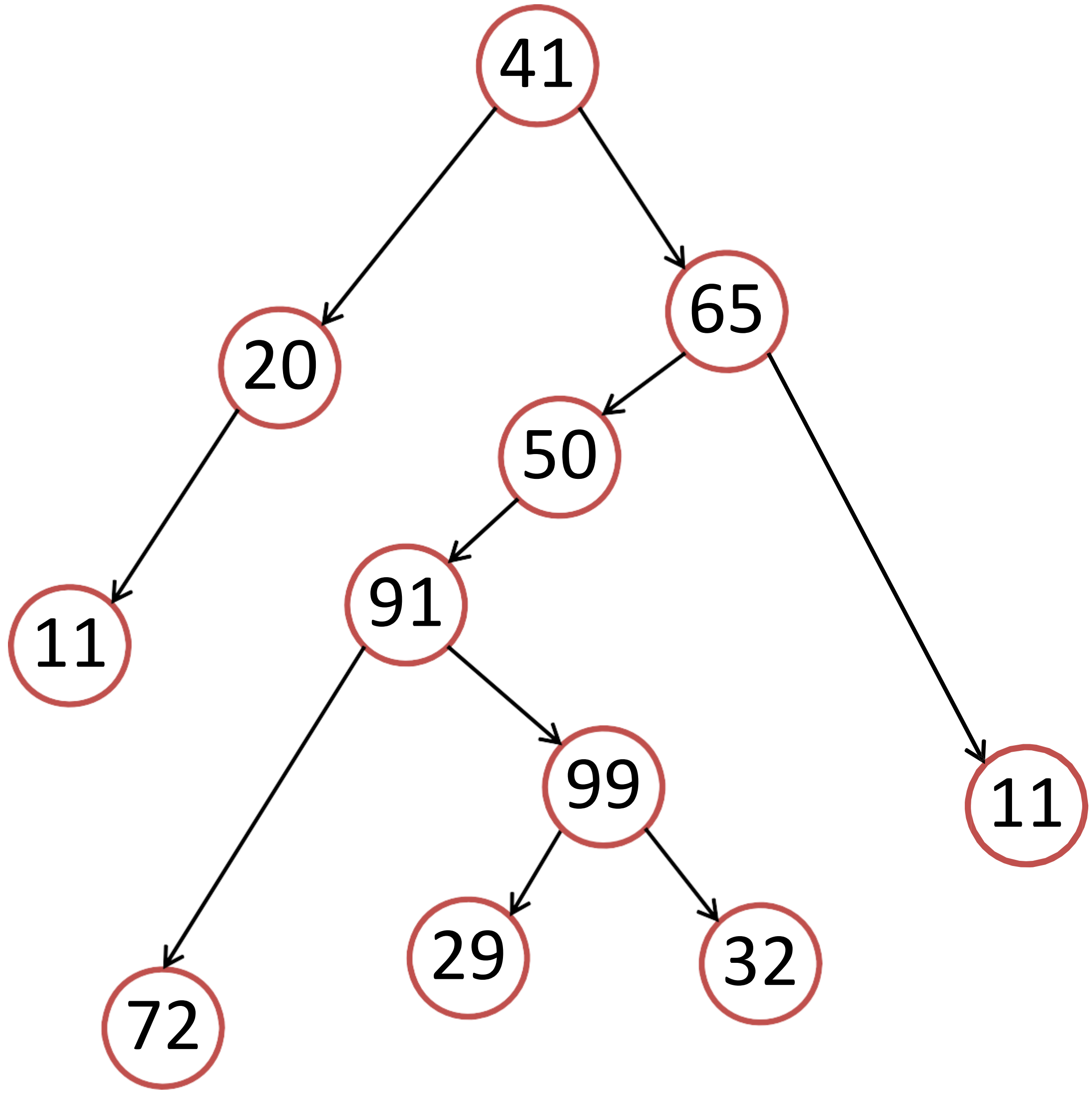


Binary Search Trees: Size (s)



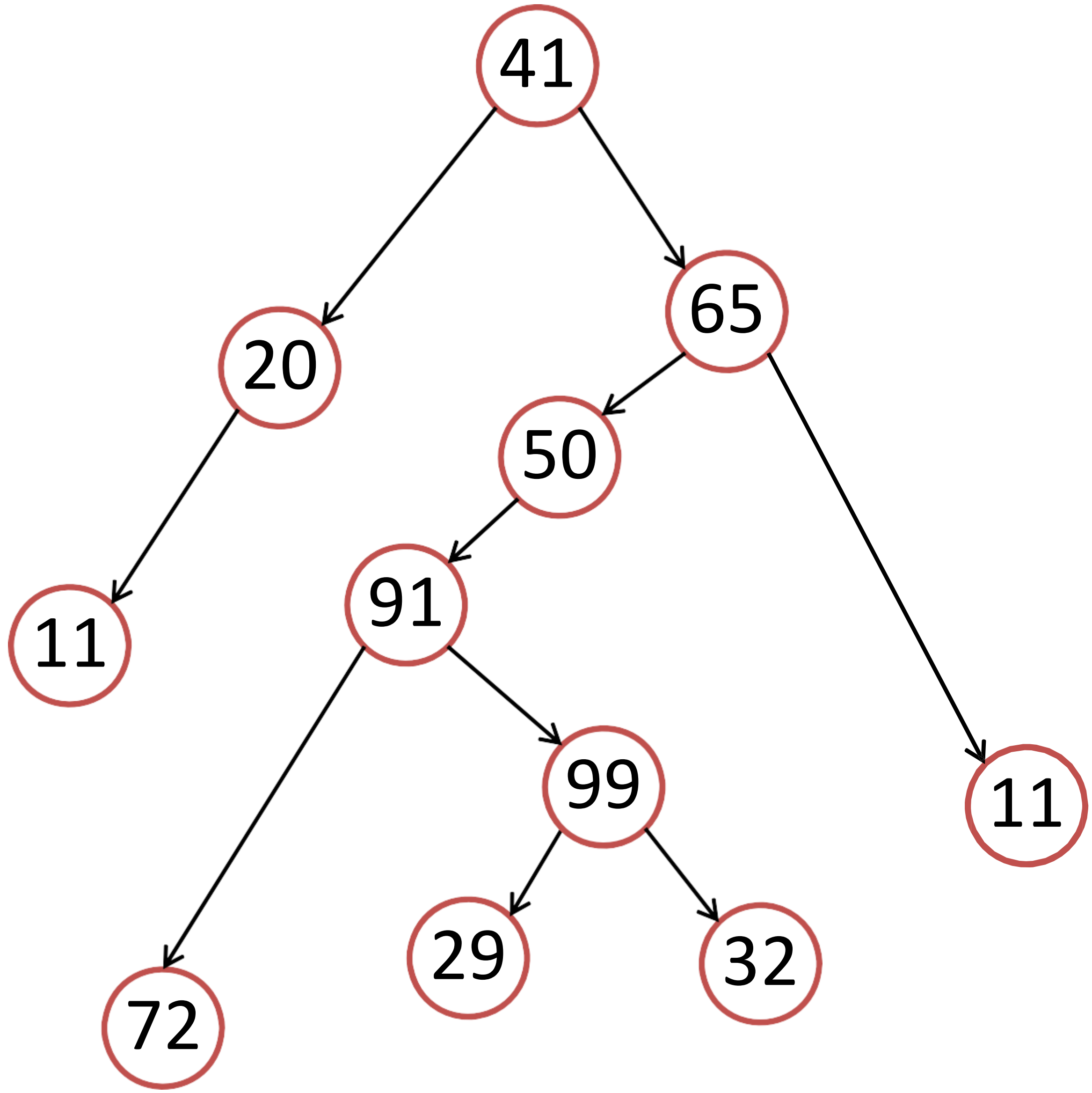
The height of this tree is?

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



Binary Search Tree: Summary

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

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

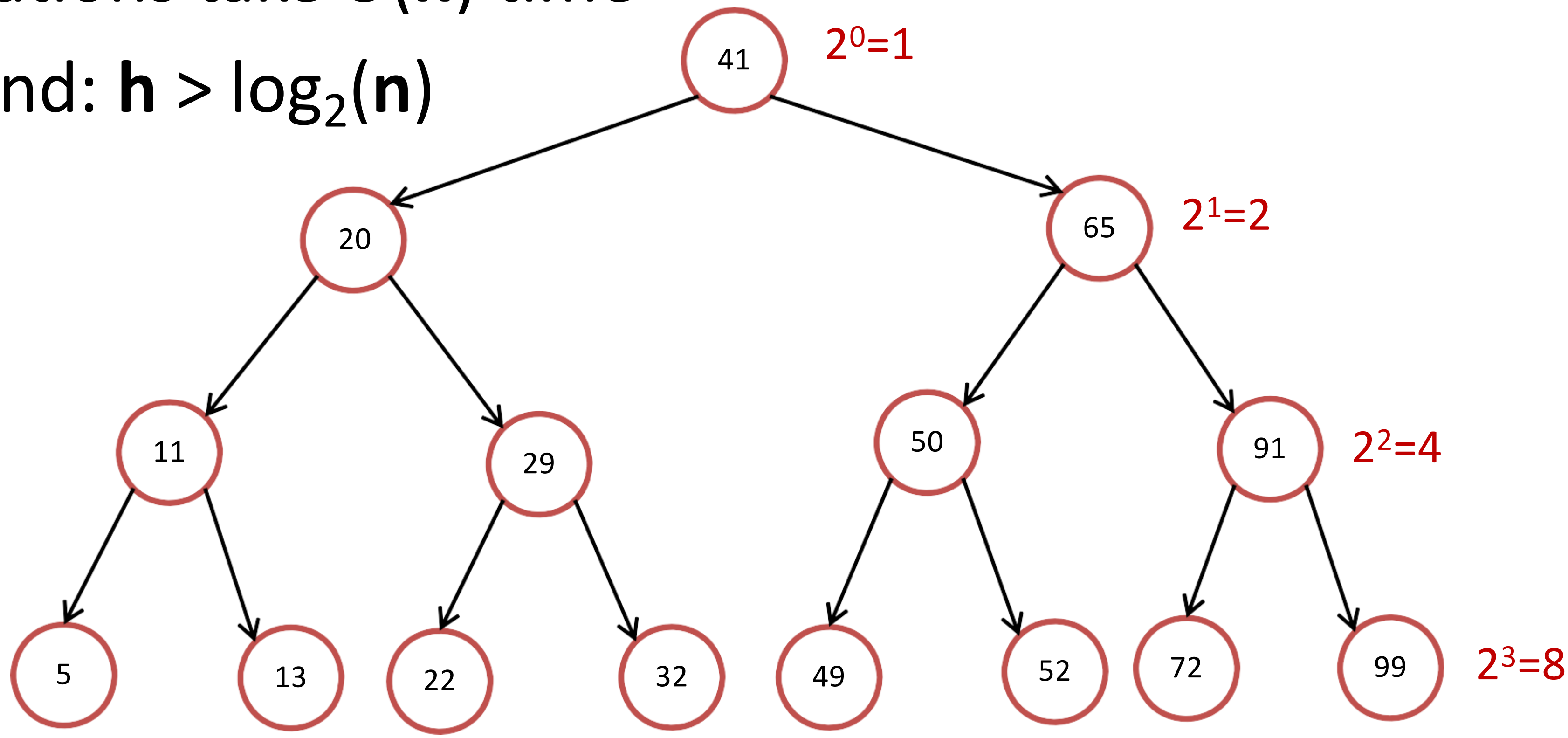
Query operations (the BST structure remains the same):

- 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 (discussed in tutorial)
- select/rank: ? (we have not discuss this yet)

The Importance of Being Balanced

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

Lower bound: $\mathbf{h} > \log_2(\mathbf{n})$



$n \leq 1 + 2 + 4 + \ldots + 2^h$

$\leq 2^0 + 2^1 + 2^2 + \ldots + 2^h < 2^{h+1}$ (sum of geometric progression)

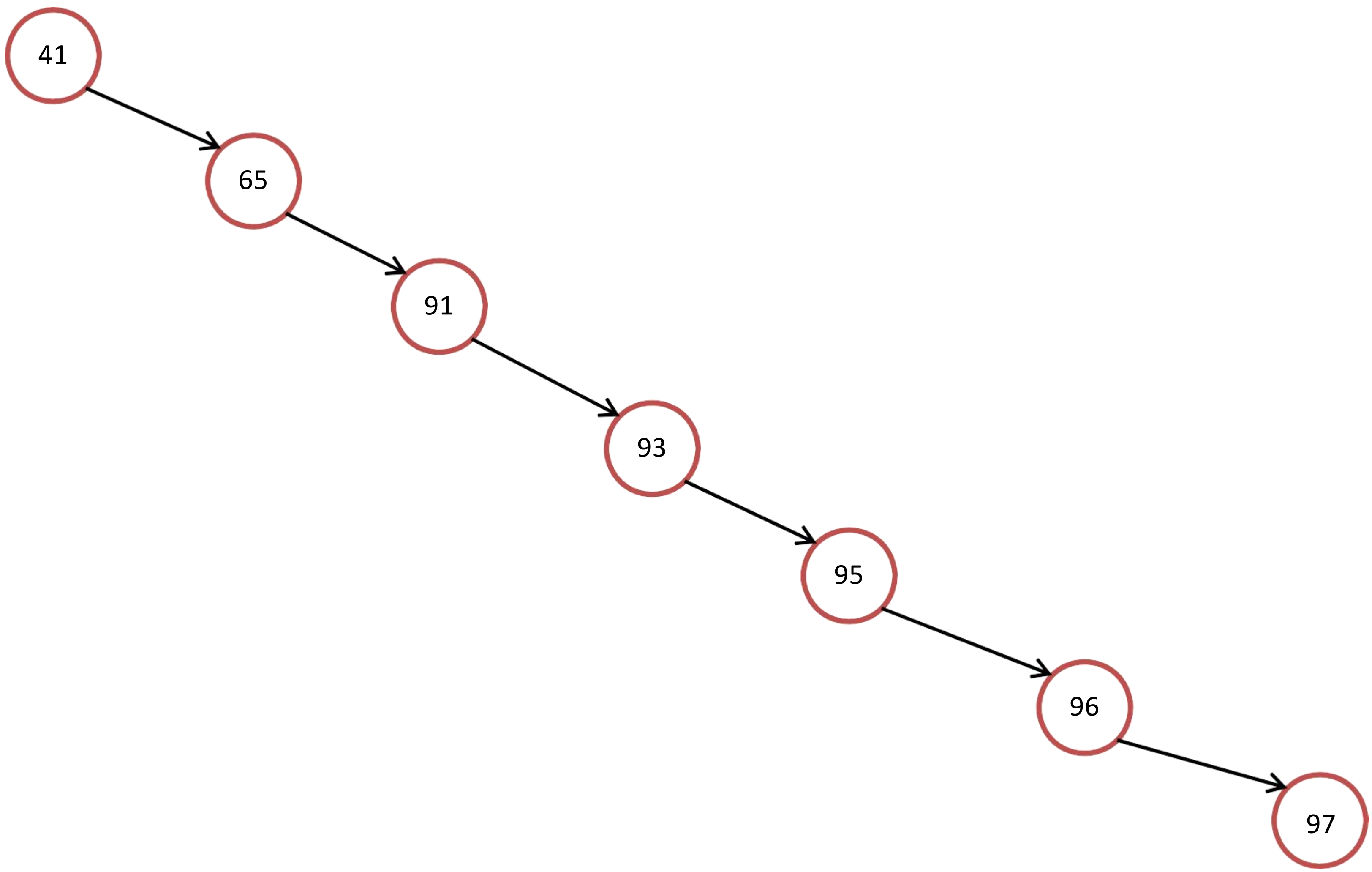
$\log_2(n) < \log_2(2^{h+1}) \Rightarrow \log_2(n) < (h+1) * \log_2(2) \Rightarrow h > \log_2(n) - 1$

$\Rightarrow h > \log_2(n)$

The Importance of Being Balanced

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

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



The Importance of Being Balanced

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

Combined bound: $\log_2(\mathbf{n}) < \mathbf{h} < \mathbf{n}$

$\log_2(\mathbf{n})$ versus **n** in picture (revisited with larger numbers):

n = 500

If we just stop at CS1020

$\log_2(\mathbf{n}) \sim 9$

After learning CS2010 ☺

If we just stop at CS1020

n = 1000

$\log_2(\mathbf{n}) \sim 10$

After learning CS2010 ☺

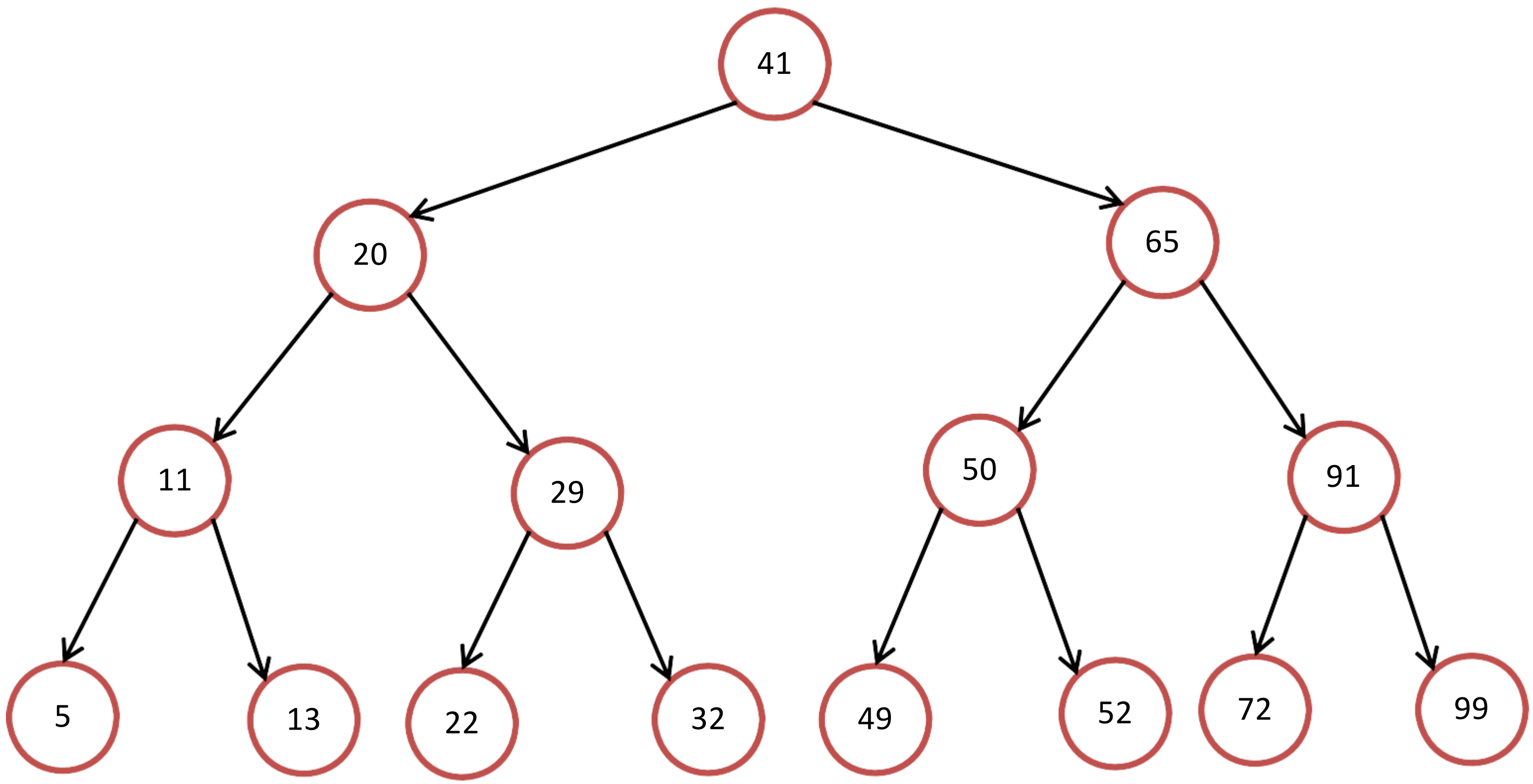
We say a BST is balanced if **h** = O(log **n**), i.e. O(c * log **n**)

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 (~53 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**:

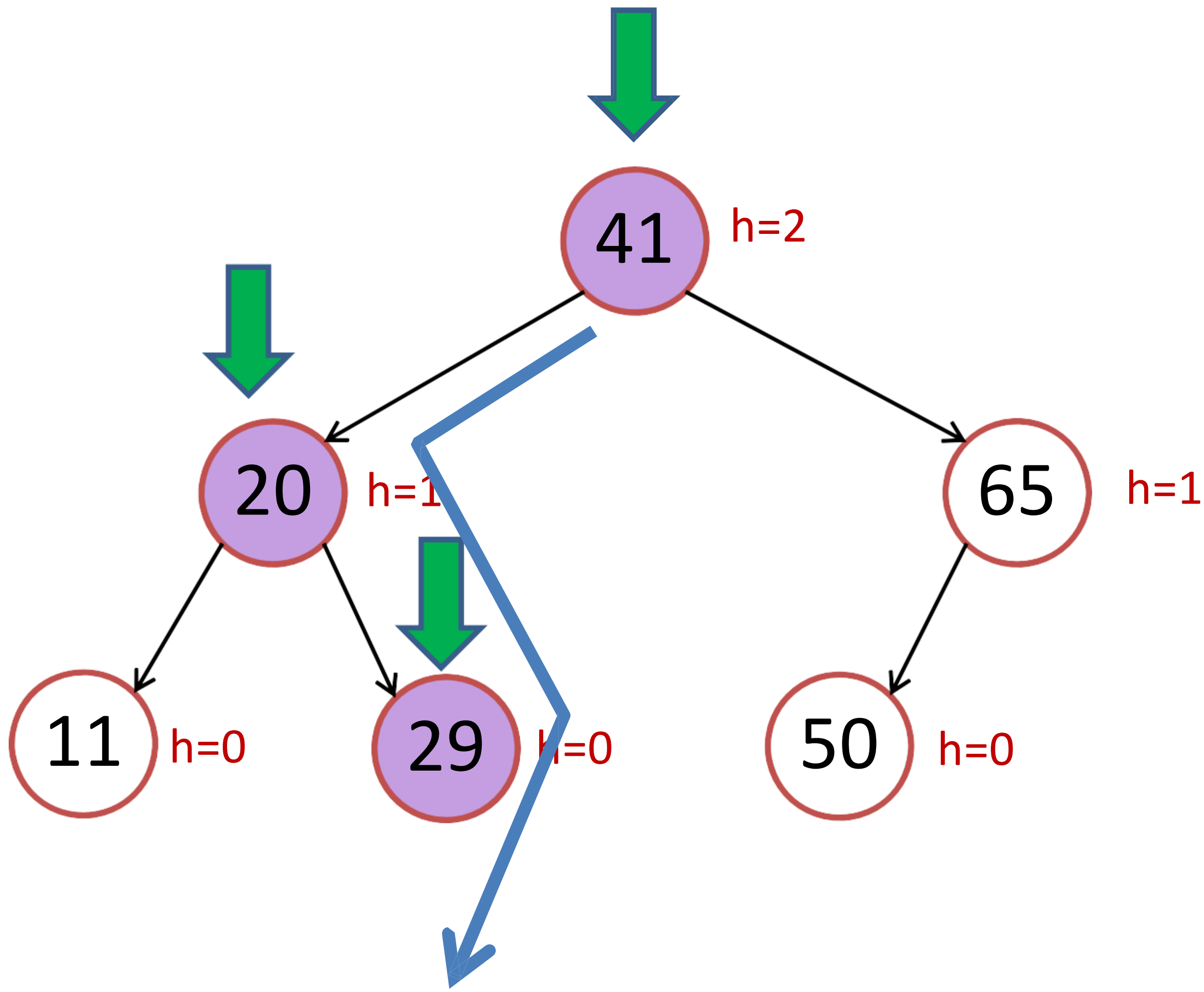
```
insert(x, v)
  // ... same as before ...
  x.height = max(x.left.height, x.right.height) + 1


// update height during deletion too (same as above)
```

Binary Search Trees

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

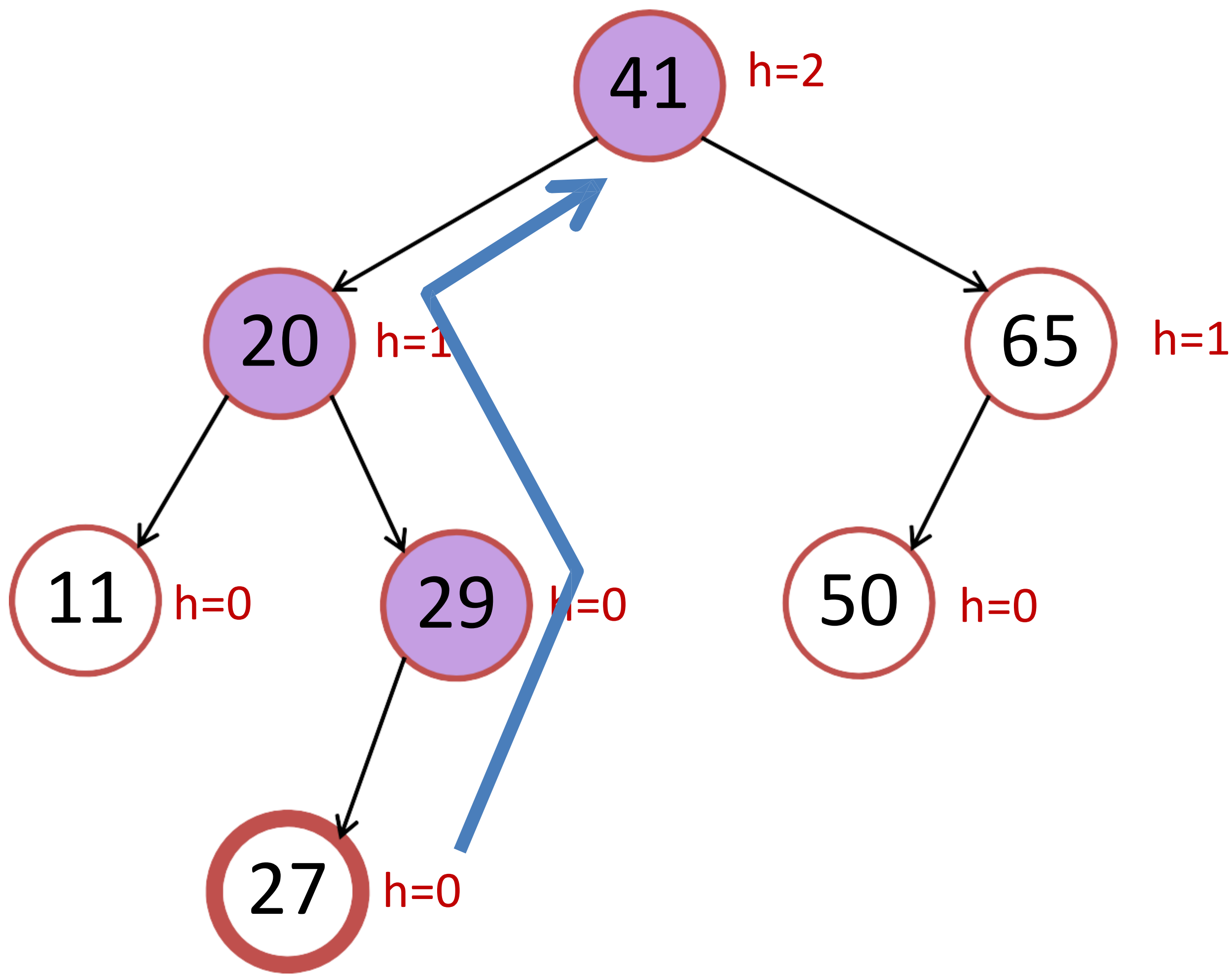
insert(27)



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

Binary Search Trees

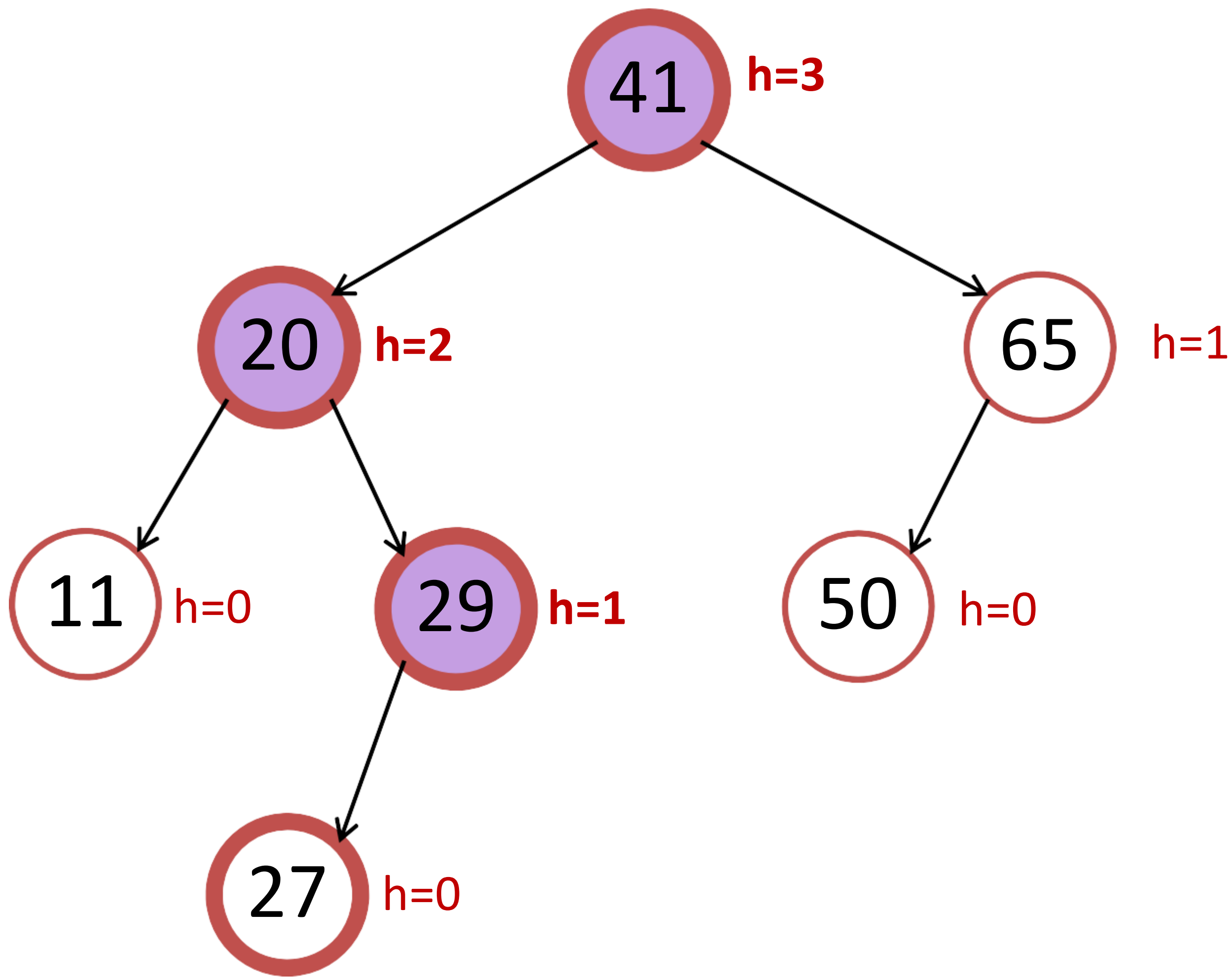
insert(27)



Binary Search Trees

insert(27)

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



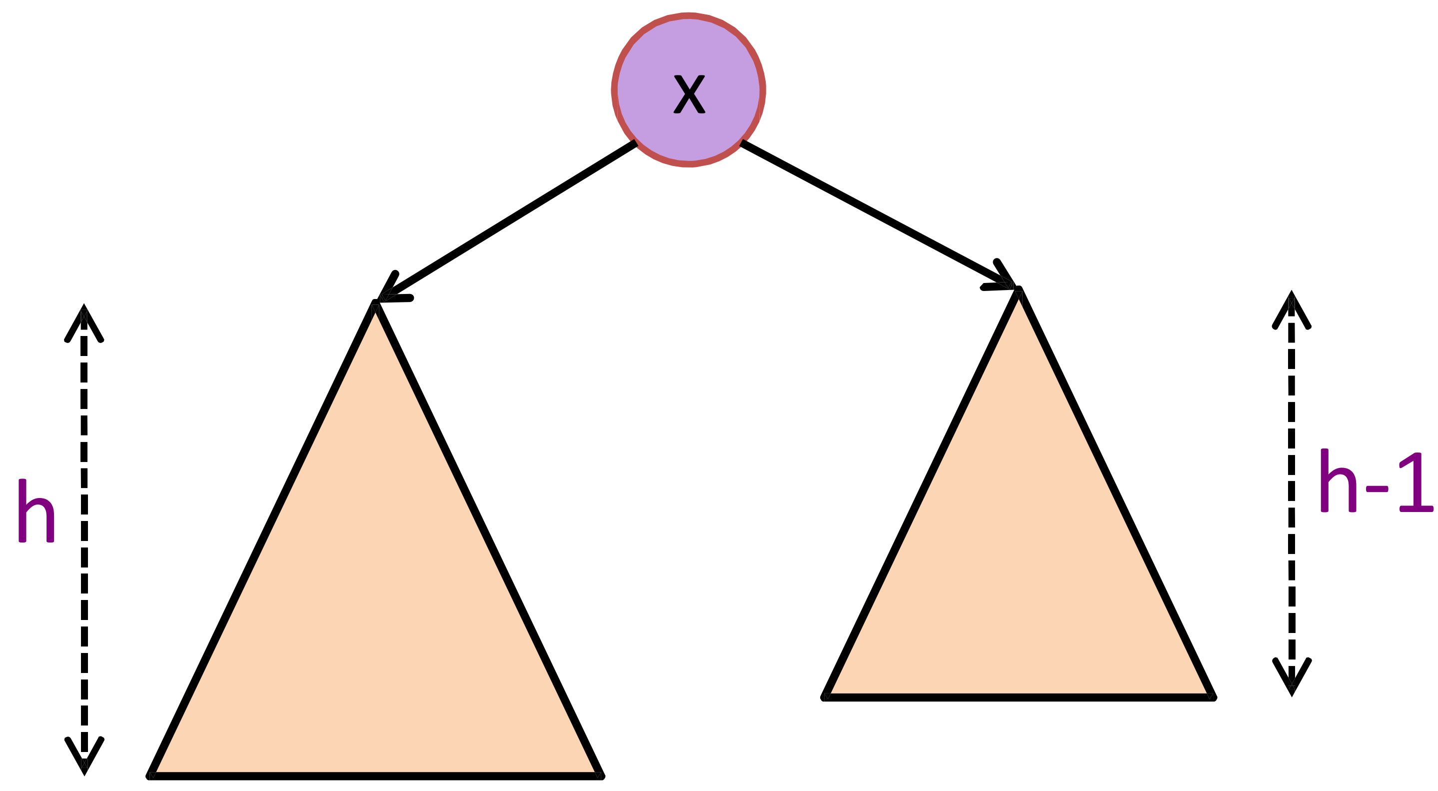
AVL Trees [Adelson-Velskii & Landis 1962]

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

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

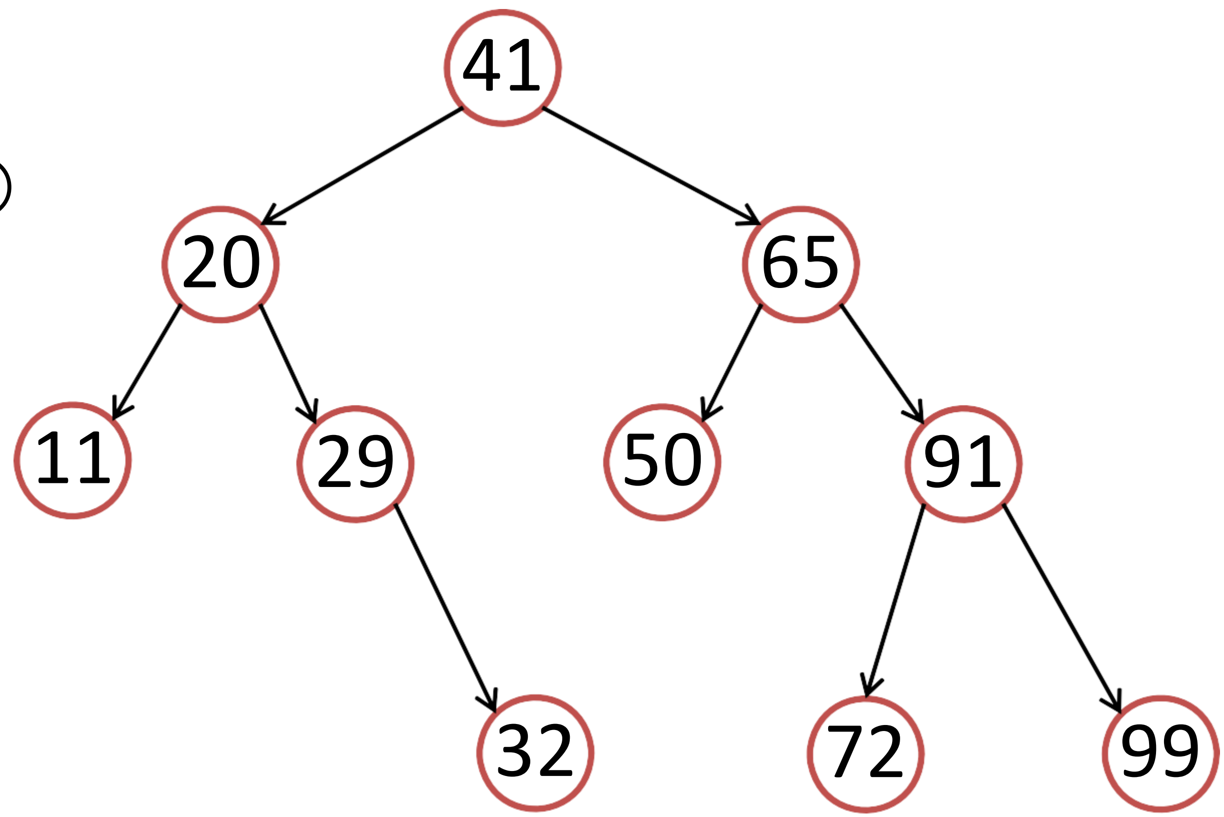
$$|x.left.height - x.right.height| \leq 1$$

An binary search tree is said to be <u>height balanced</u> if:
every vertex in the tree is height-balanced



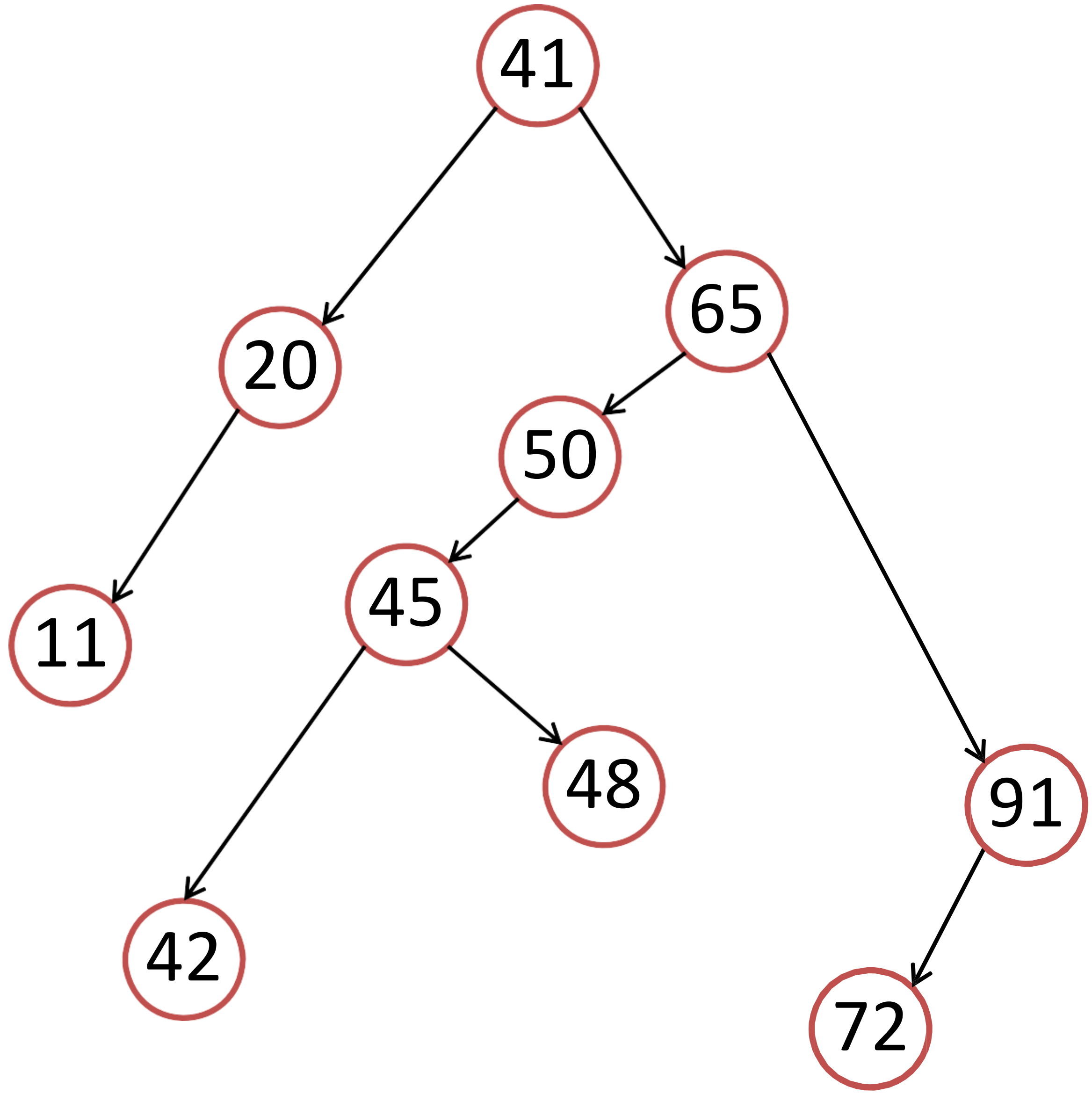
Is this tree height-balanced according to AVL?

1. Yes
2. No
3. I am confused... ☹



Is this tree height-balanced according to AVL?

1. Yes
2. No
3. I am confused... ☹



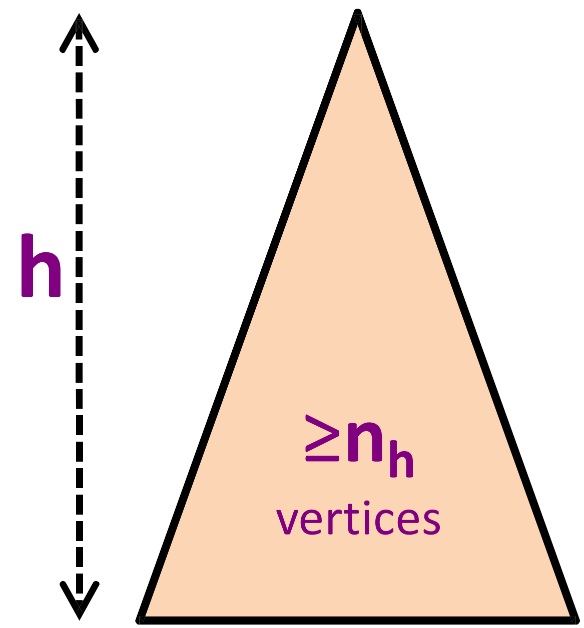
Height-Balanced Trees

Claim:

A height-balanced tree with **n** vertices has height **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 **h**



Height-Balanced Trees

Proof:

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

$n_h = 1 + n_{h-1} + n_{h-2}$

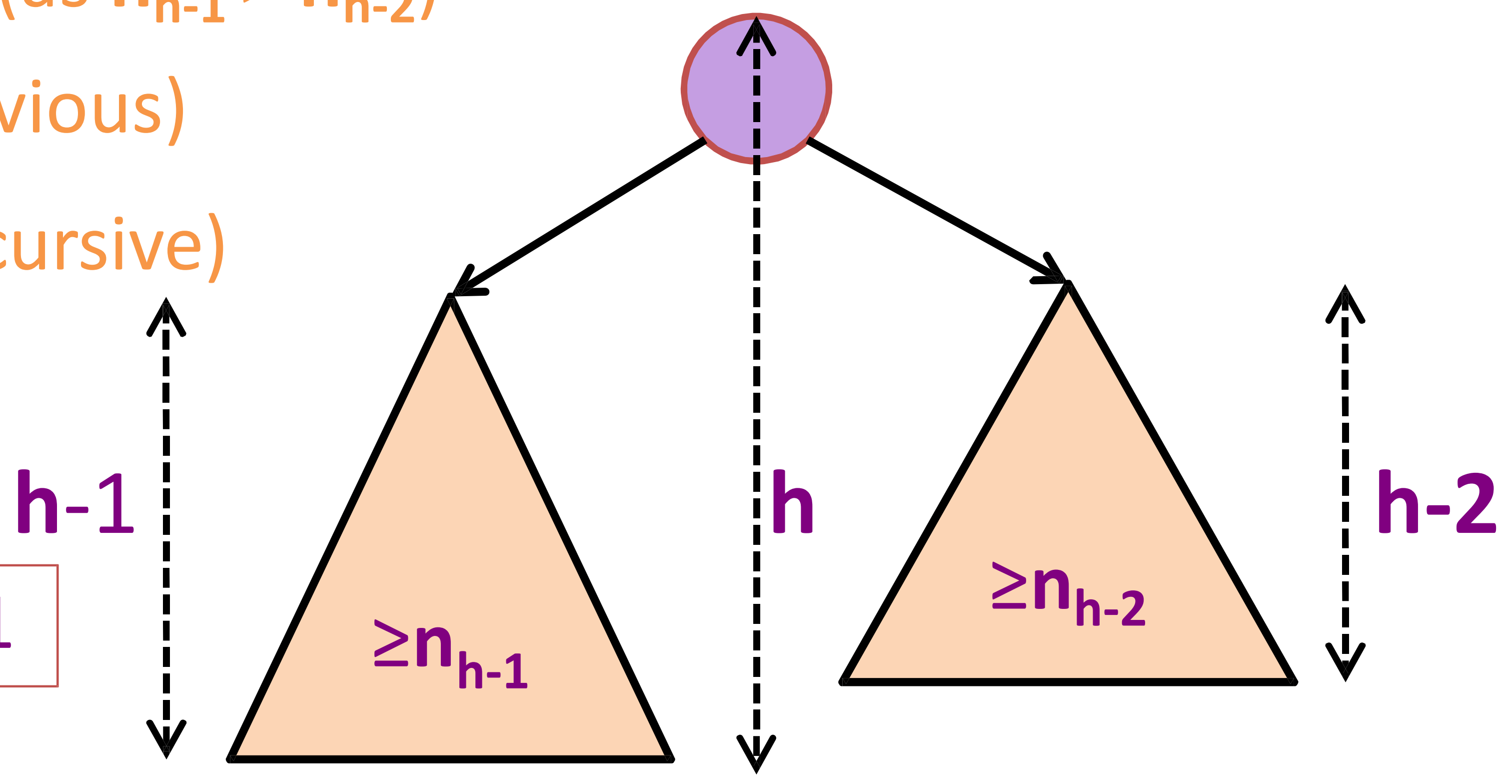
$n_h > 1 + 2n_{h-2}$ (as $n_{h-1} > n_{h-2}$)

$n_h > 2n_{h-2}$ (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

$n_h = 1 + n_{h-1} + n_{h-2}$

$n_h > 1 + 2n_{h-2}$

$n_h > 2n_{h-2}$

$= 4n_{h-4}$

$= 8n_{h-6}$

$= \ldots$

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 $\mathbf{n} \geq \mathbf{n_h}$

$\mathbf{n} \geq \mathbf{n_h} > 2^{\mathbf{h}/2}$

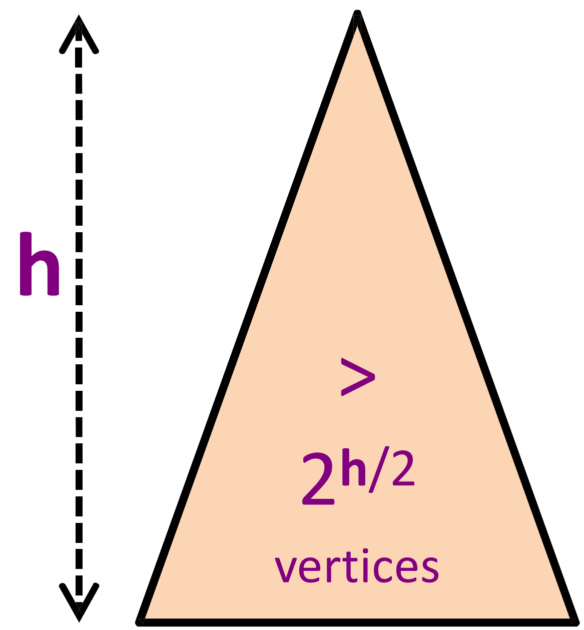
$\mathbf{n} > 2^{\mathbf{h}/2}$

$\log_2(\mathbf{n}) > \log_2(2^{\mathbf{h}/2})$ (log$_2$ on both side)

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

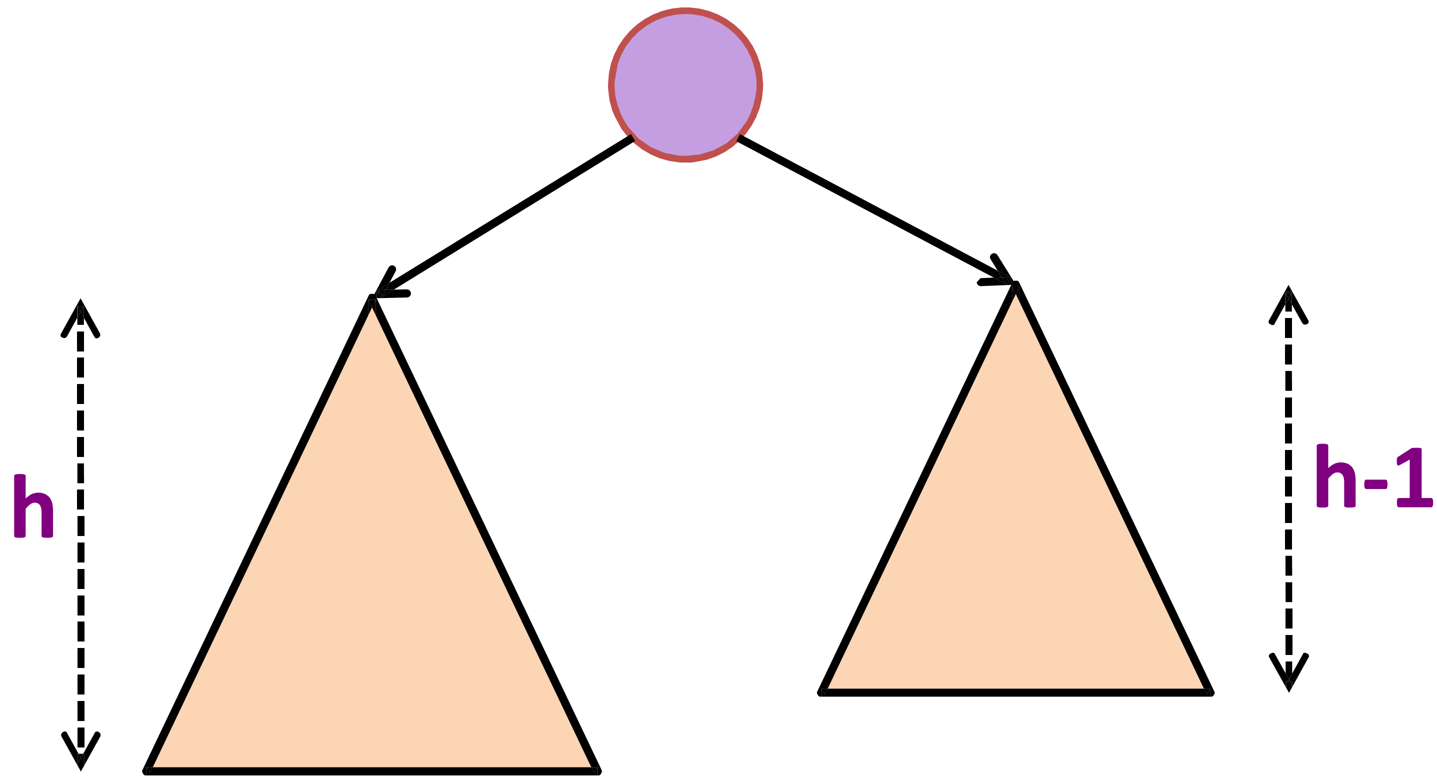
$2 * \log_2(\mathbf{n}) > \mathbf{h}$ or $\mathbf{h} < 2 * \log_2(\mathbf{n})$

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



AVL Trees [Adelson-Velskii & Landis 1962]

Step 3: Show how to maintain height-balance



Insertion to an AVL Tree

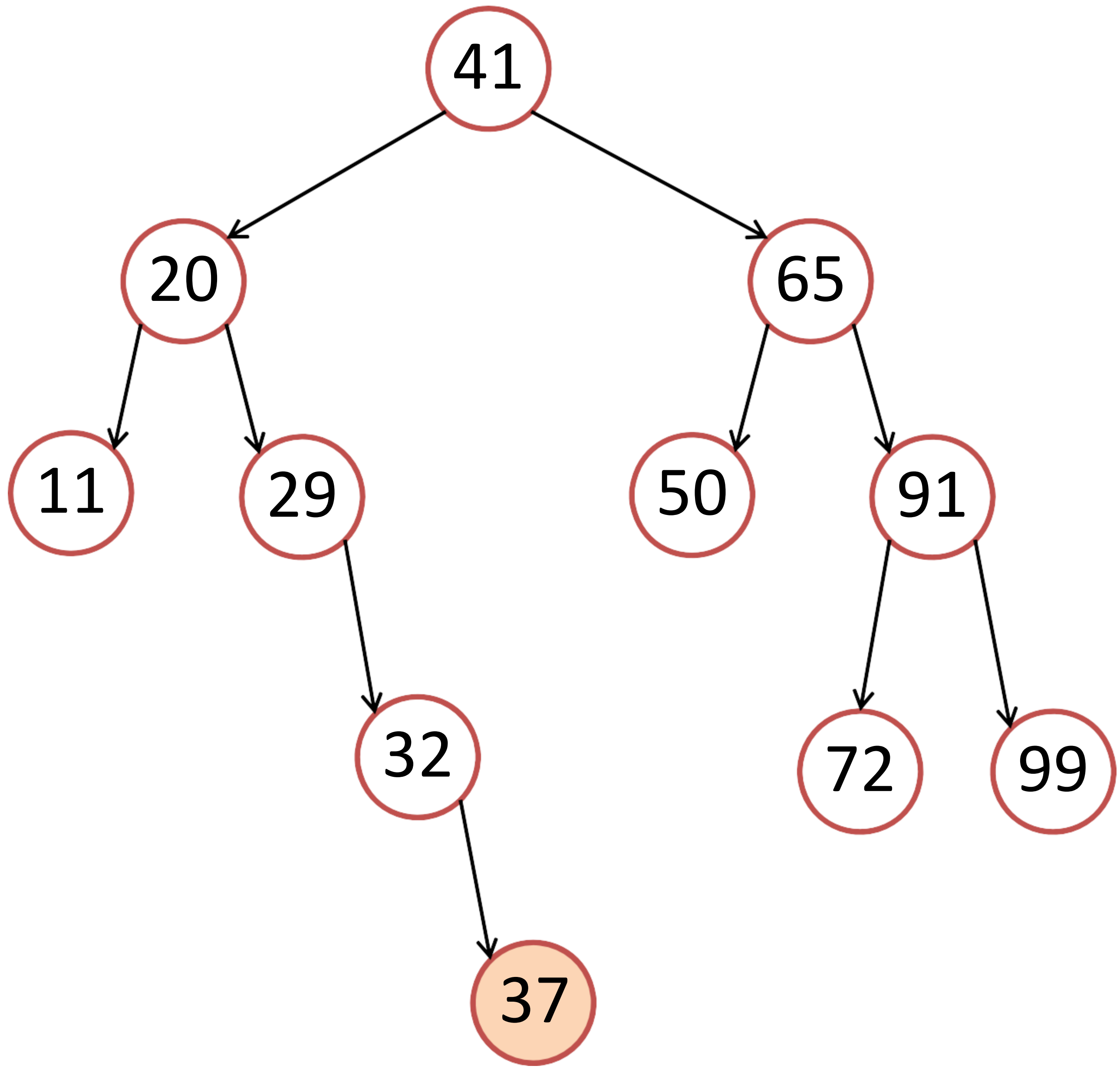
insert(37)

Initially balanced

But no longer balanced after Inserting 37

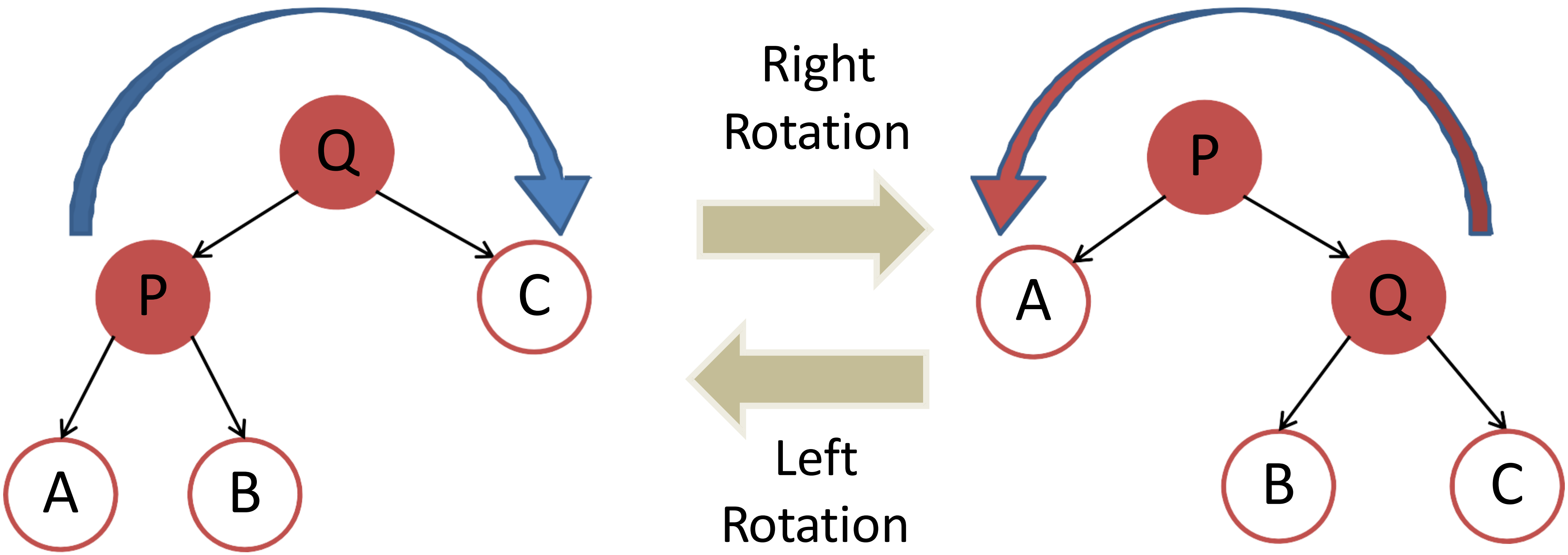
Need to rebalance!

But how?



"Infinite more" examples in VisuAlgo...

Tree Rotations



Rotations maintain ordering of keys

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

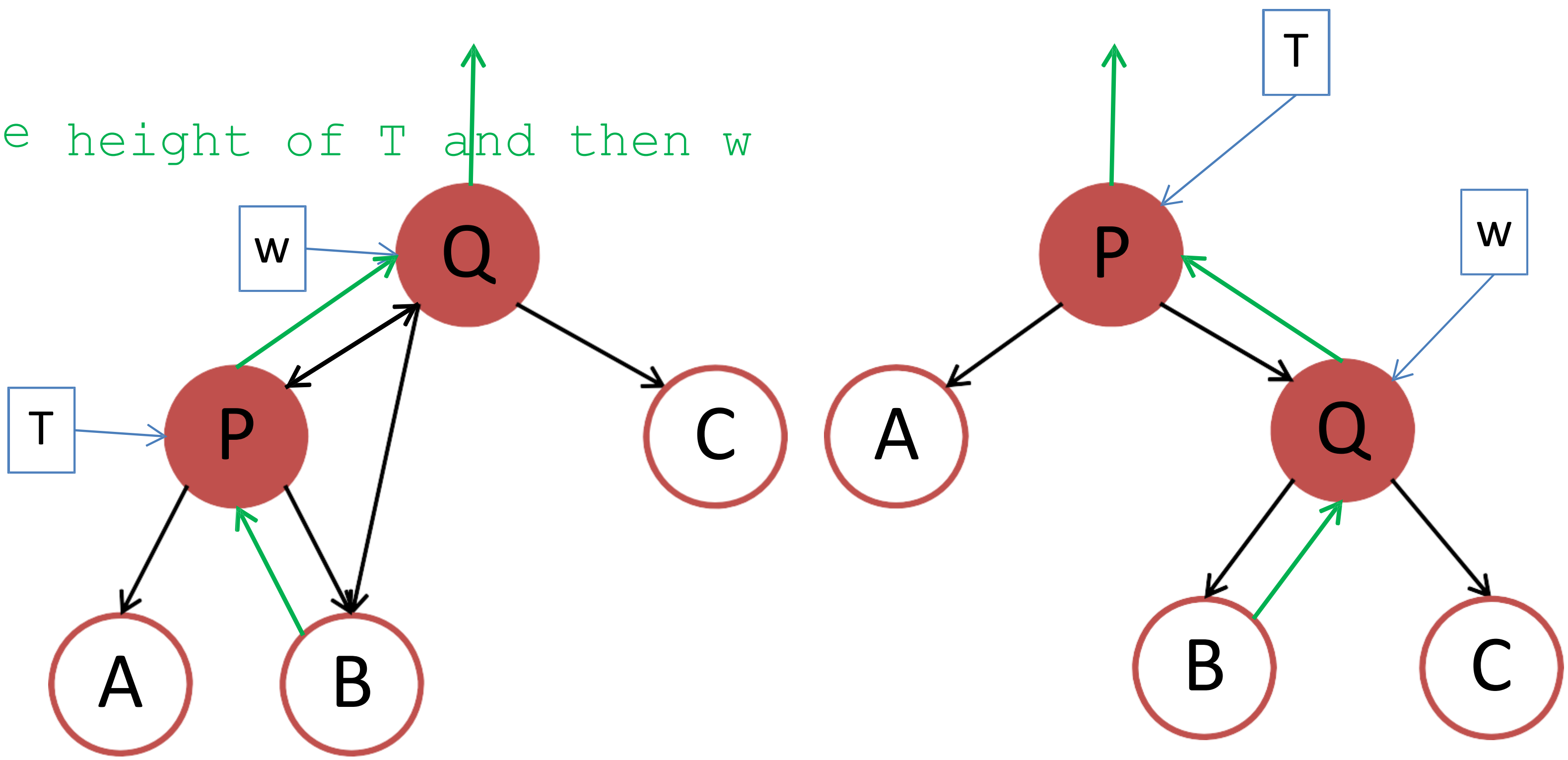
rotateRight requires a left child

rotateLeft requires a right child

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

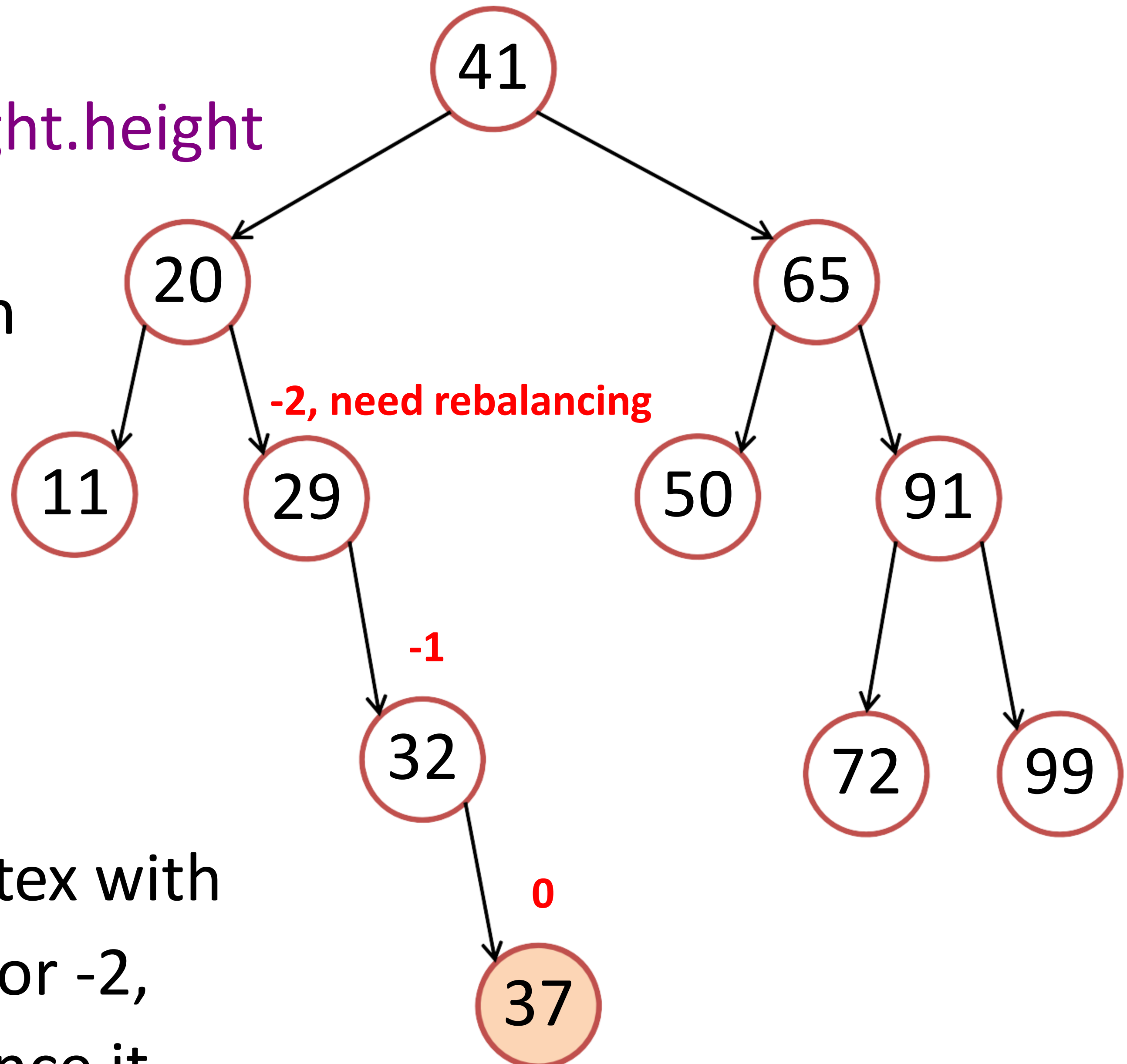
Balance Factor (bf(x))

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



Four Possible Cases

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

rightRotate(x)

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

leftRotate(x.left)

rightRotate(x)

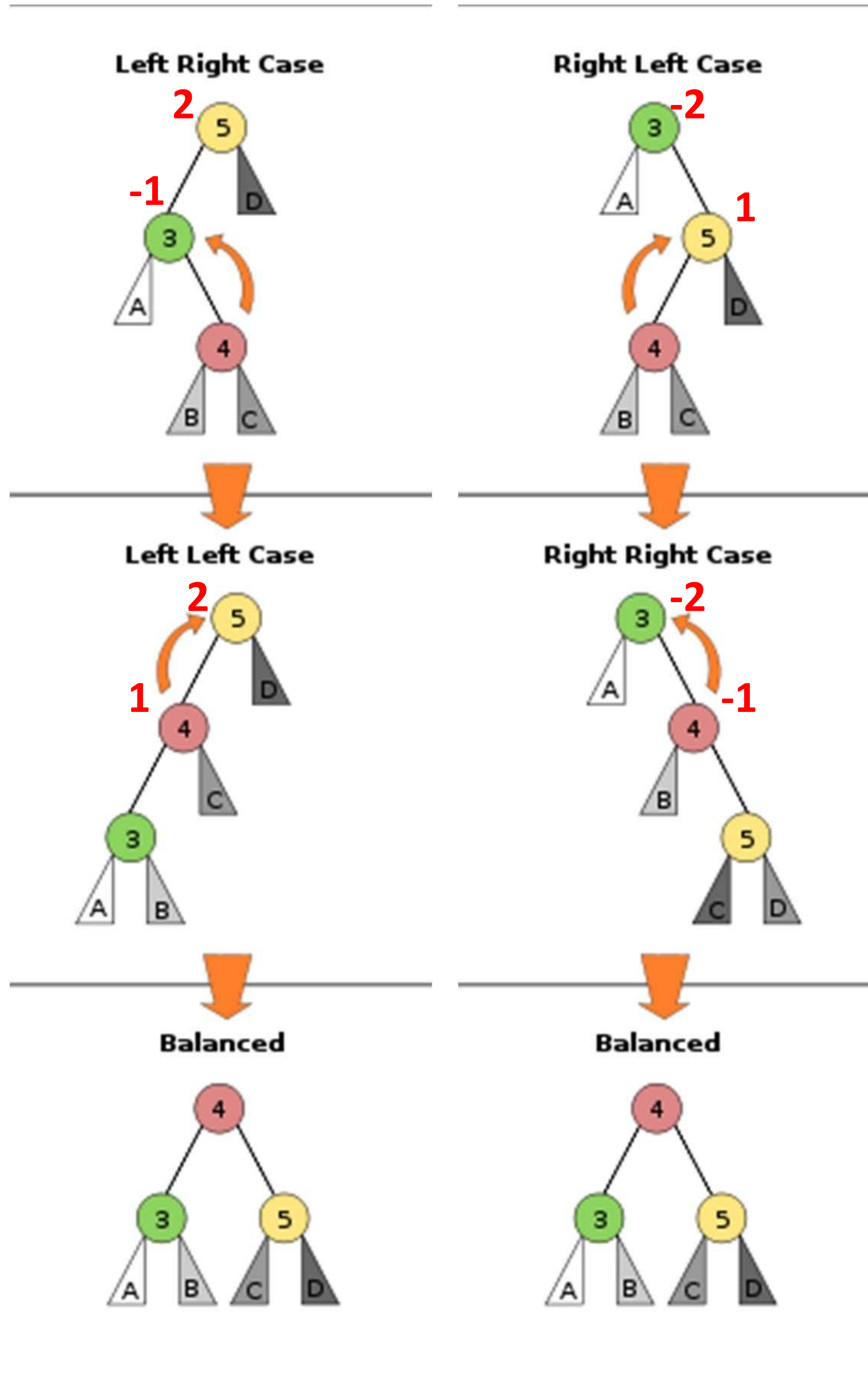
bf(x) = -2 and bf(x.right) = -1

leftRotate(x)

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

rightRotate(x.right)

leftRotate(x)

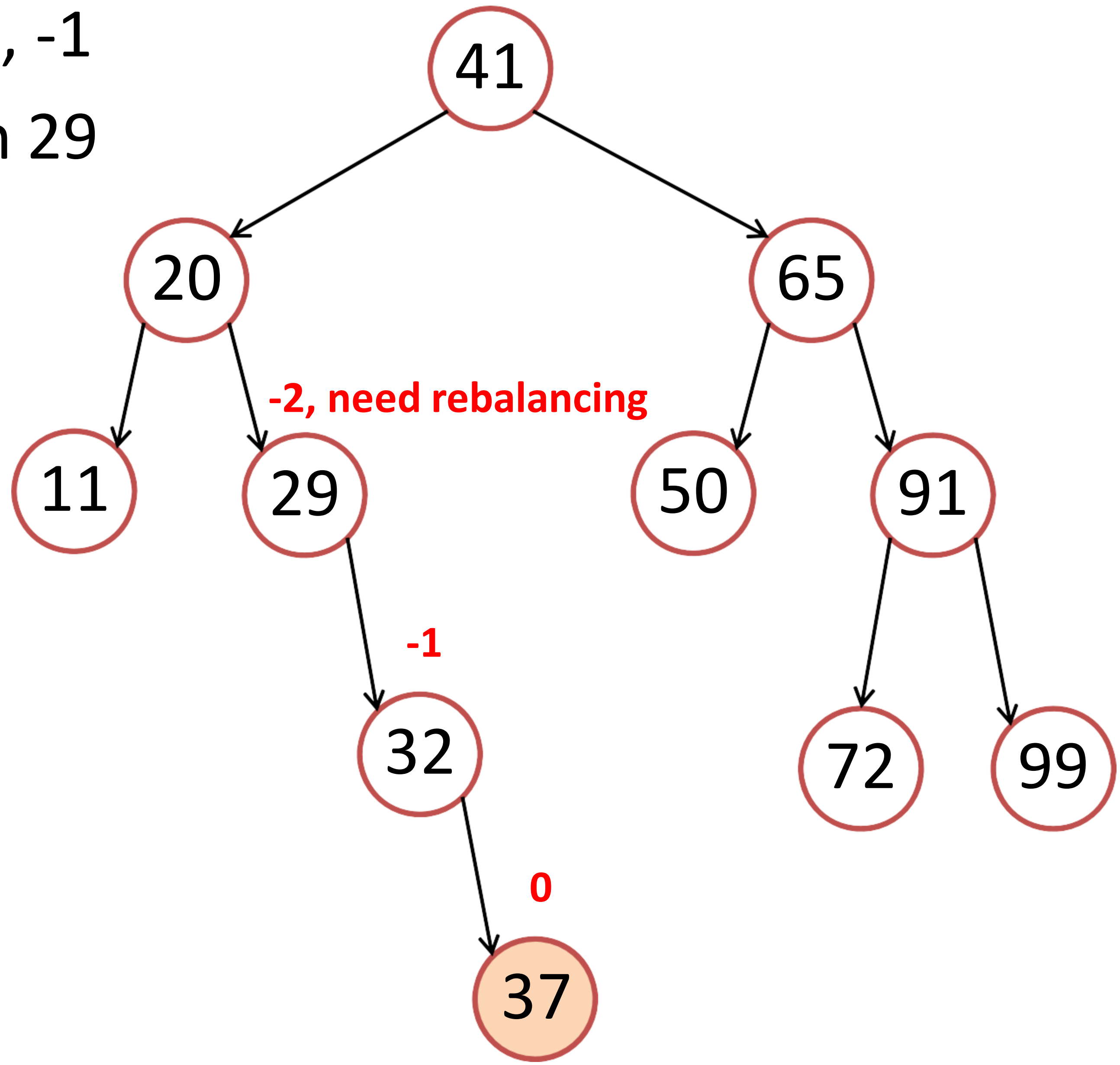


Pictures from Wikipedia

Rebalancing (1)

This is a case of -2, -1

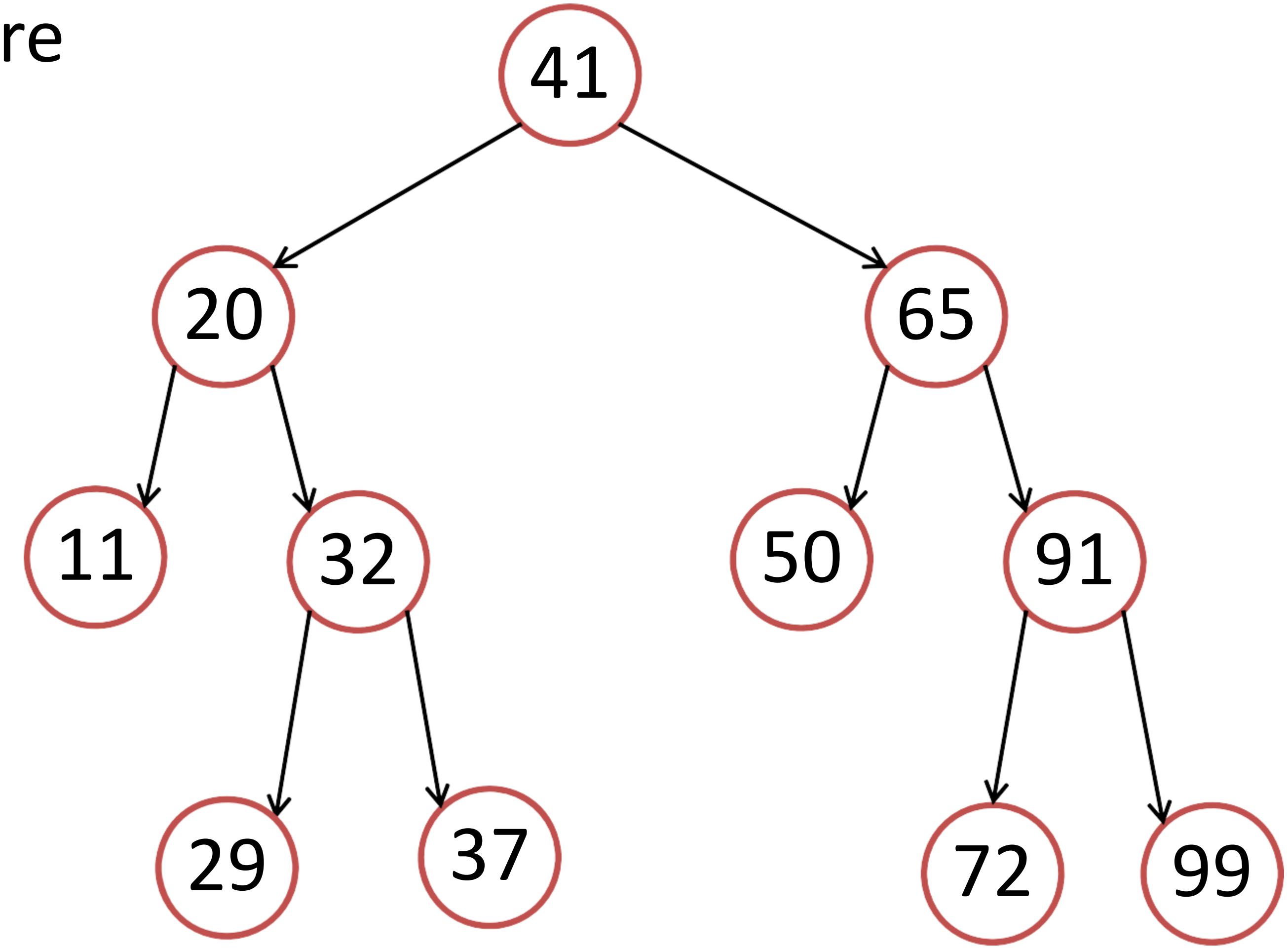
Do left rotate on 29



“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

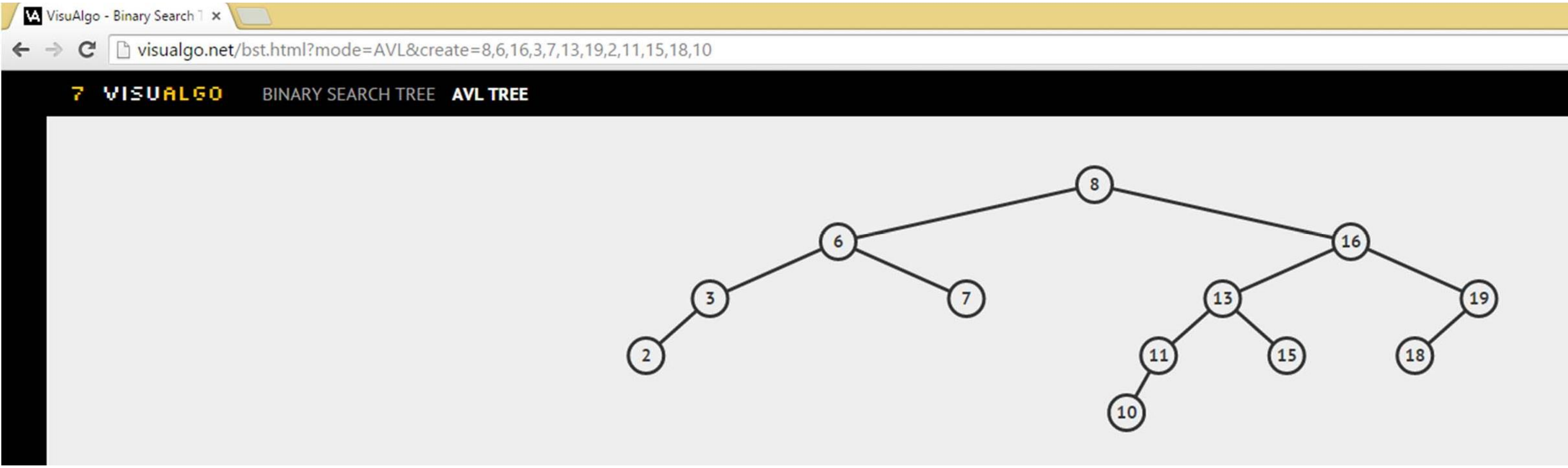
Deletion from an AVL Tree

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

Try: http://visualgo.net/bst.html?mode=AVL&create=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

The Implementation

Let's look *briefly* at AVLDemo.java

- The code is **NOT** given, it is asked in PS2 ☺
- So, I will only flash them

Introducing **Java Inheritance and Polymorphism**

Q: Do we have to use such long code every time we need a balanced BST ☹?

A: Fortunately no ☺, we can use Java API

- Details during your lab demo on Week 04

Balanced Search Trees

Many different flavors of balanced search trees

- AVL trees (Adelson-Velsii & 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)
- Treaps (Seidel and Aragon 1996)
- Skip Lists (Pugh 1989)

Balanced Search Trees

Red-Black Trees

- Every vertex is colored red or black
- All leaves are black
- A red vertex has only black children
- Every path from a vertex to any leaf contains the same number of black vertices.
- Rebalance using rotations on insert/delete

Balanced Search Trees

Skip Lists and Treaps

- Randomized data structures
- Random insertions → balanced tree
- Use randomness on insertion to maintain balance

Balanced Search Trees

Splay Trees

- On access (search or insert),
  move vertex to root (via rotations)
- Height can be linear!
- On average, O(log n) per operation (amortized)

Optimality?

- Cannot do better than O(log n) worst-case
- What about for specific access patterns
  (e.g., 10 searches in a row for value x)?

Now, after we learn balanced BST

No	Operation	Unsorted Array	Sorted Array	bBST
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)	O(log n)
8	NumYounger(age)	O(n log n)	O(log n)	????

NumYounger(age) = rank(age)-1

Now, how to get rank(v) efficiently?

This has not been discussed before

and will be revealed during the live lecture

–

Balanced BST

Summary:

- The Importance of Being Balanced
- Height Balanced Trees
- Tree Rotations
- AVL trees

Next Lecture:

- ADT Priority Queues
- Binary Heaps

CS2010R first meeting

- Tomorrow, Thu, 03 Sep 15, 6.00-6.30pm
- Note: **venue TBA**
- All R students must come
- Those who wants to get PS2E AC can also come ☺
- TA: Myself ☺