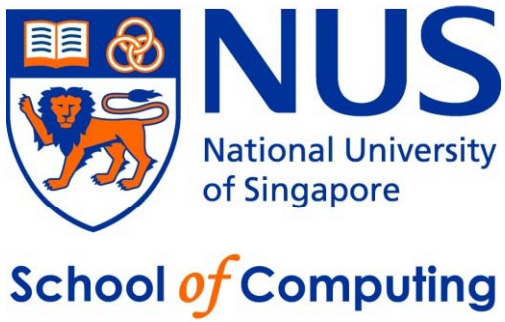
CS2010 – Data Structures and Algorithms II

Lecture 03 – Keeping Everything in Balance

stevenhalim@gmail.com



BuzzCity-SoC Clicks Fraud Detection Challenge 2012

NUS National University of Singapore
School of Computing

BUZZCITY

Call for Contestants

Pay-per-click is a popular payment model for Internet advertising where the ***advertisers*** contract a ***syndicator*** to distribute their advertisements to the ***publishers***. Pay-per-click is subjected to abuses by malicious publishers through ***click fraud*** where the malicious publishers intimate legitimate users, or mislead the users to generate clicks that do not have genuine interest in the advertisements' content.

By entering into this contest, you are taking up the challenge in solving a real-life problem: deriving an automated method that identifies malicious publishers from large volume of clicks history. The data (with certain fields anonymized for privacy protection) are provided by BuzzCity, a global advertising network.

Visit
www.comp.nus.edu.sg/~clickfrd/
for details.

Important Dates

15 Aug to 15 Oct 2012	Registration
29 Aug 2012	Briefing
19 Jan 2013	Announcement of winners

You do not need to print these, just FYI

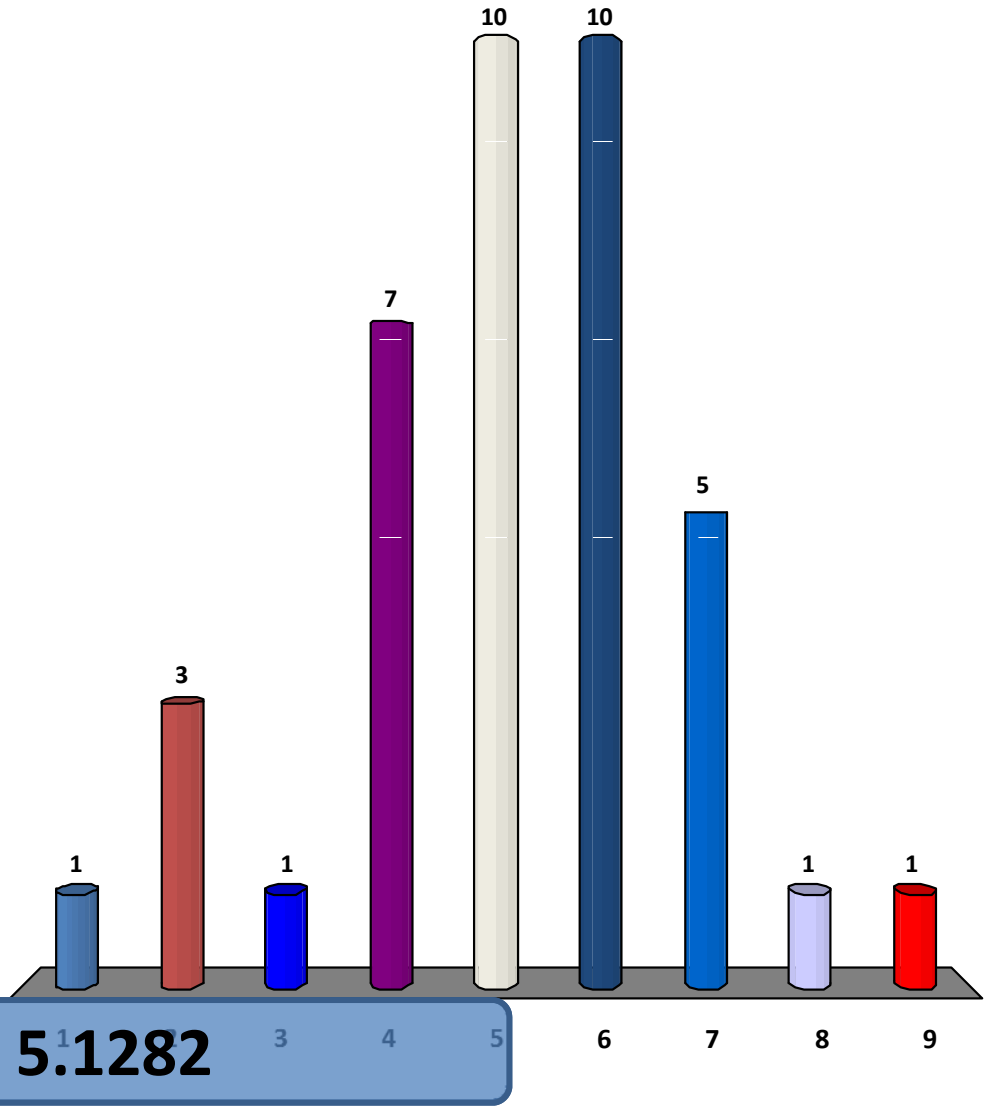
CENSUS SLIDES FROM WEEK02

‘[‘ (or ‘]’) means that endpoint is included (closed)

Your Age:

1. [24 … ∞)
2. [23 … 24)
3. [22 … 23)
4. [21 … 22)
5. [20 … 21)
6. [19 … 20)
7. [18 … 19)
8. [17 … 18)
9. [0 … 17)

‘(‘ (or ‘)‘) means that endpoint is **not** included (open)

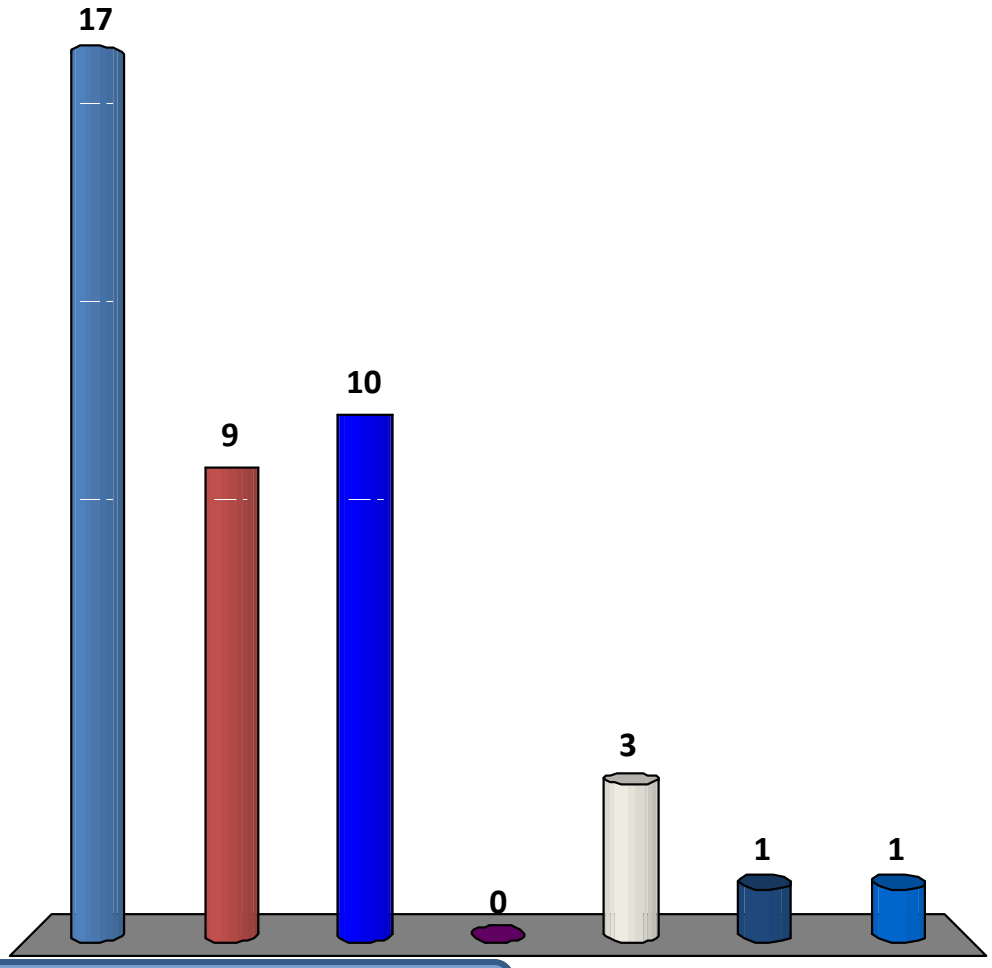


Mean = 5.1282

Your Major:

1. Computer Science (CS)
2. Communications and Media (C&M)
3. Computer Engineering (CEG/CEC)
4. Comp. Biology (CB)
5. Information System (IS)
6. Science Maths (SCI)
7. None of the above :O

Three main majors: 69 CS, 33 C&M, 17 CEG
More than half of CEG students have clickers, whereas less than one third of CS and C&M students have it...

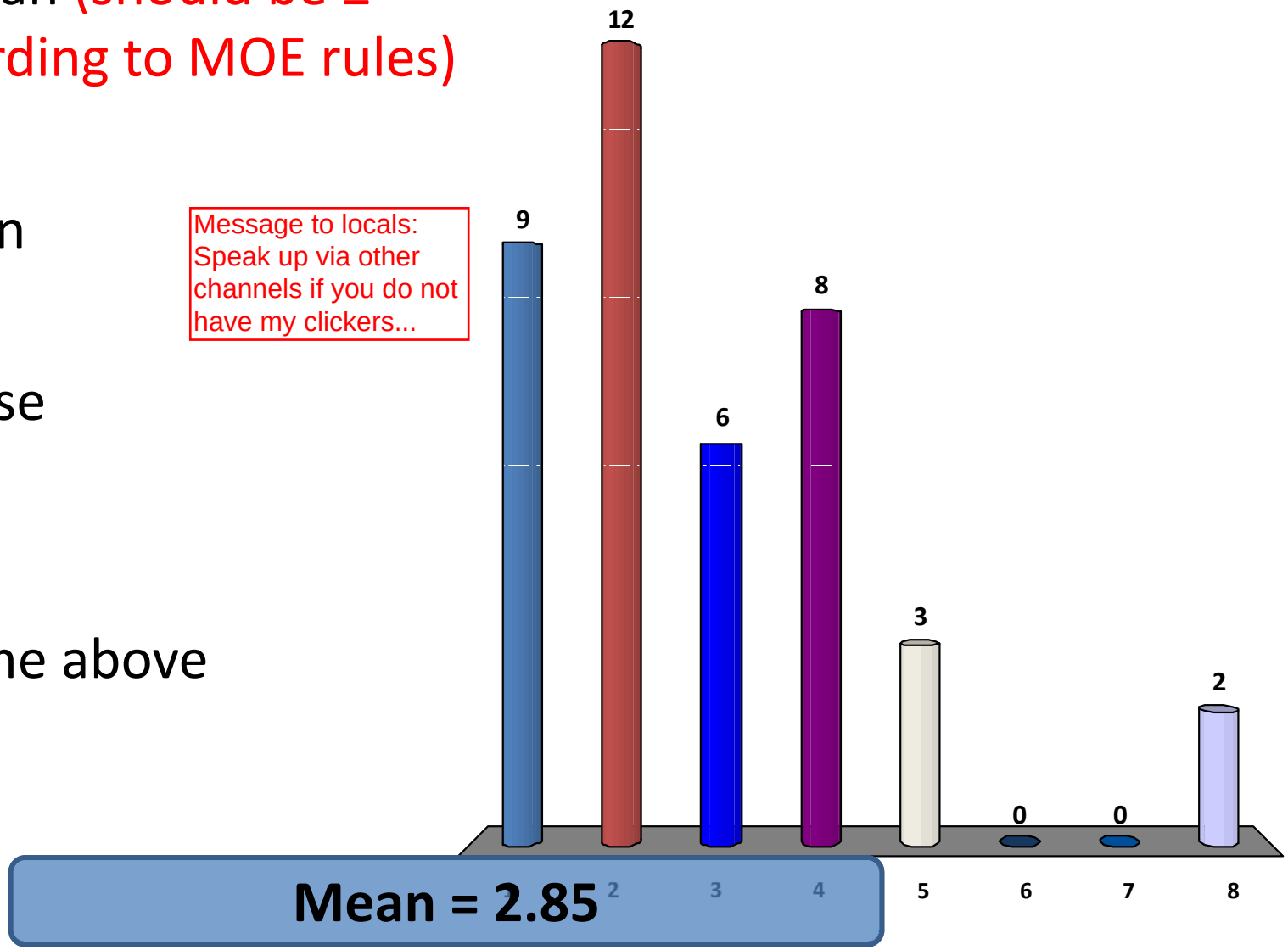


Mean = 2.2683

Your Nationality:

1. Singaporean (should be ≥ 70% according to MOE rules)
2. Chinese
3. Indonesian
4. Indian
5. Vietnamese
6. Malaysian
7. European
8. None of the above (tell me)

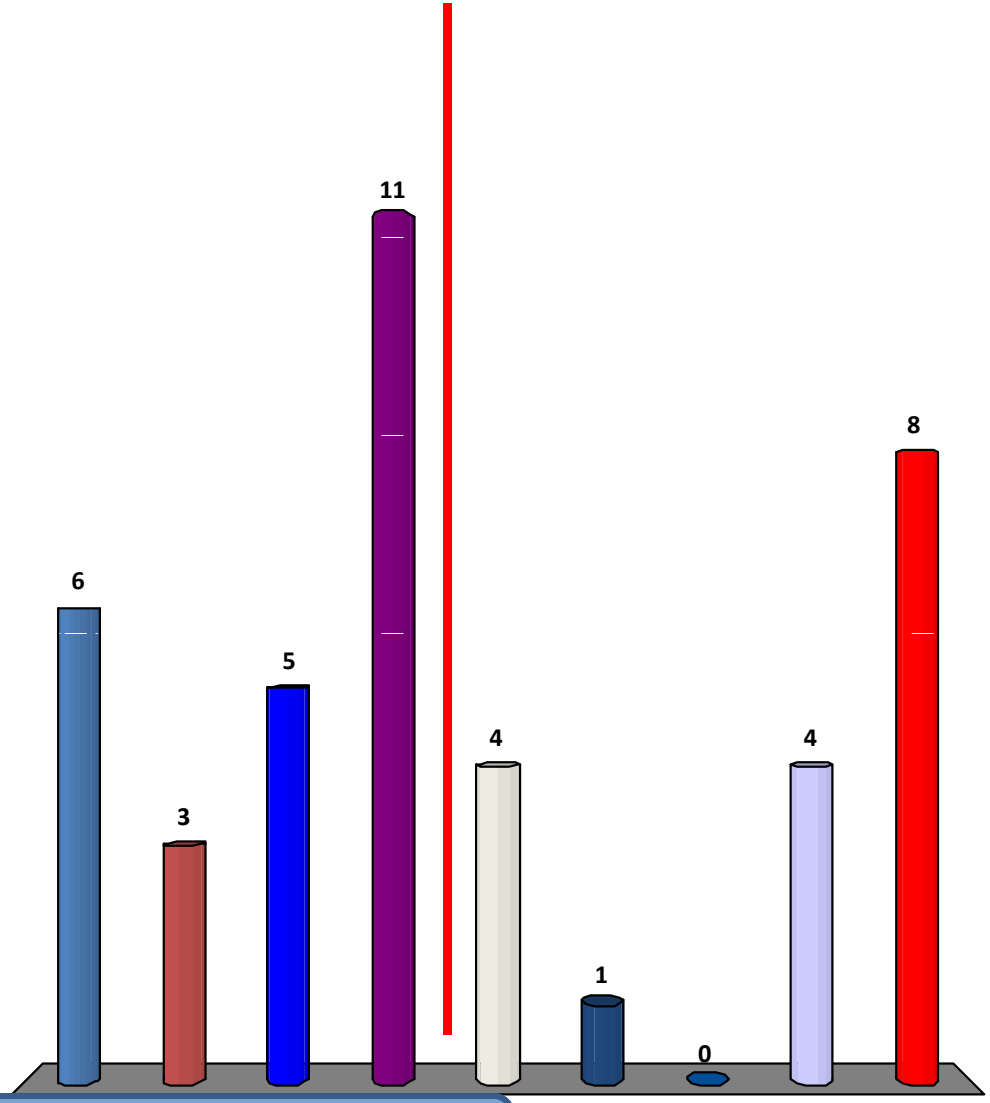
Message to locals: Speak up via other channels if you do not have my clickers...



Your CAP:

(remember, the clicker system is anonymous!)

1. [4.5 … 5.0]
2. [4.25 … 4.5)
3. [4.0 … 4.25)
4. [3.75 … 4.0)
5. [3.5 … 3.75)
6. [3.25 … 3.5)
7. [3.0 … 3.25)
8. [0.0 … 3.00)
9. I do not want to tell

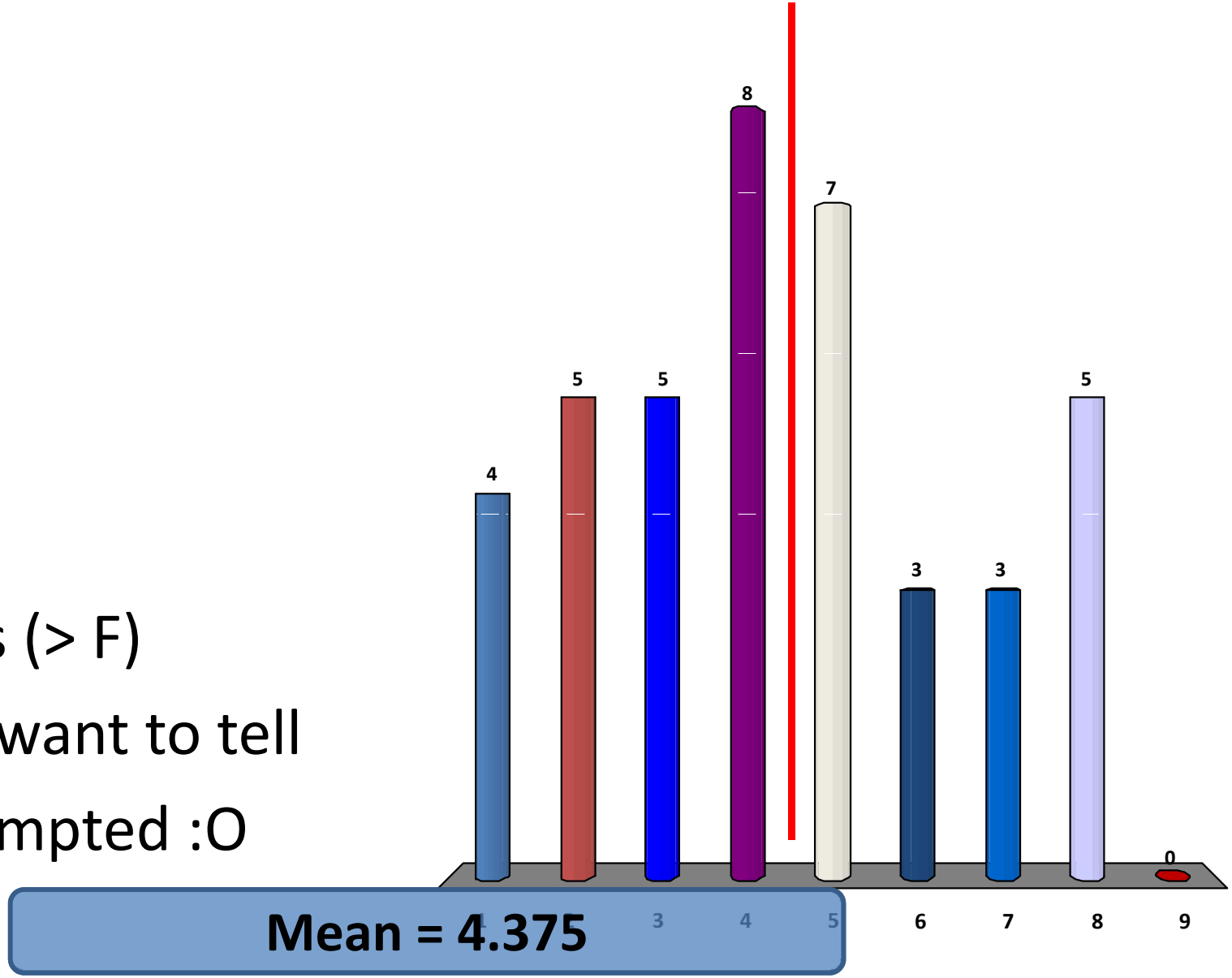


Mean = 4.7857

My CS1020 Grade

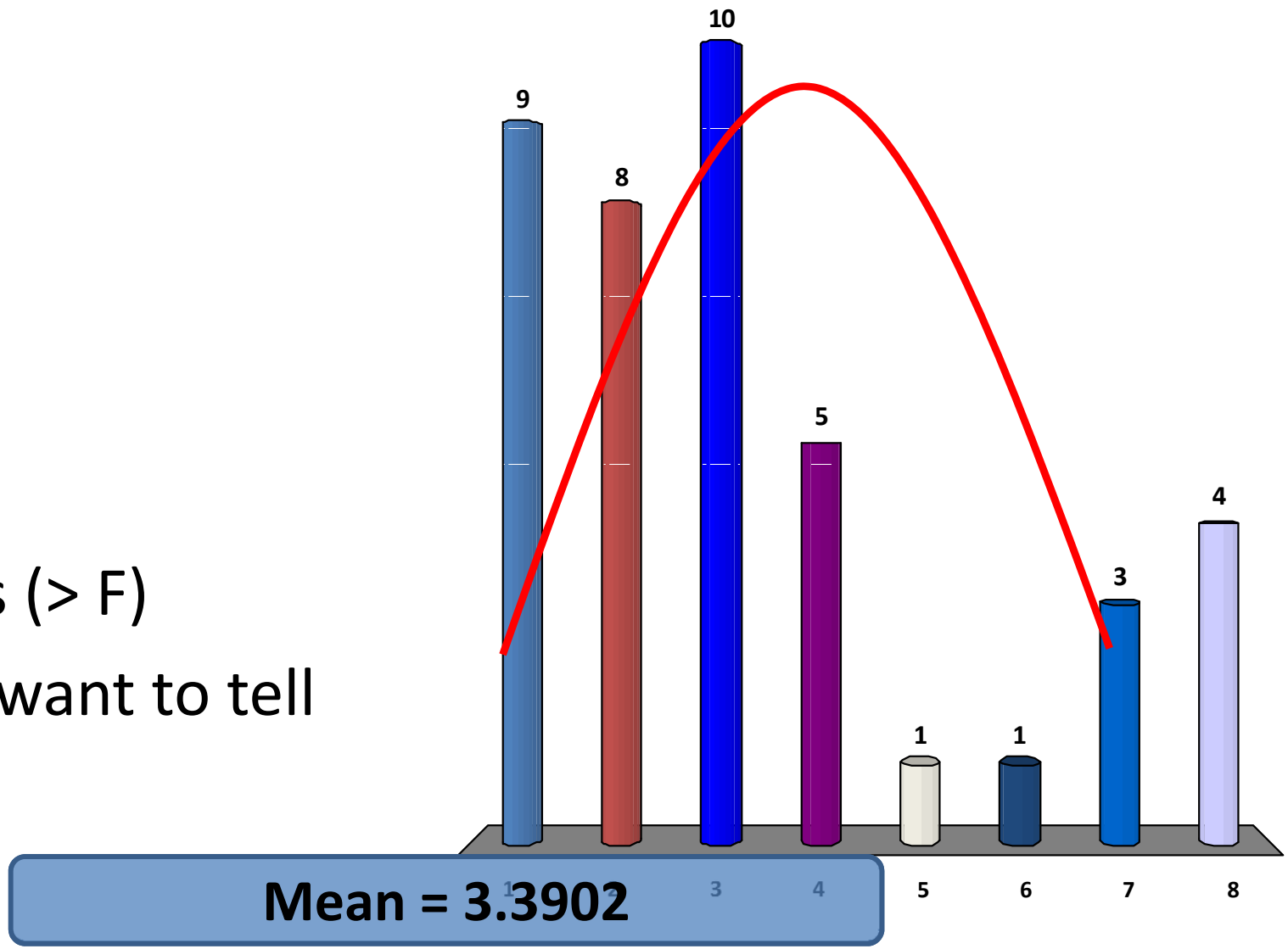
(I already know your CS1231 profile last week)

1. A+
2. A
3. A-
4. B+
5. B
6. B-
7. Just pass (> F)
8. I do not want to tell
9. I am exempted :O



Your Grade Expectation for CS2010

1. A+
2. A
3. A-
4. B+
5. B
6. B-
7. Just pass (> F)
8. I do not want to tell



Admin Slide

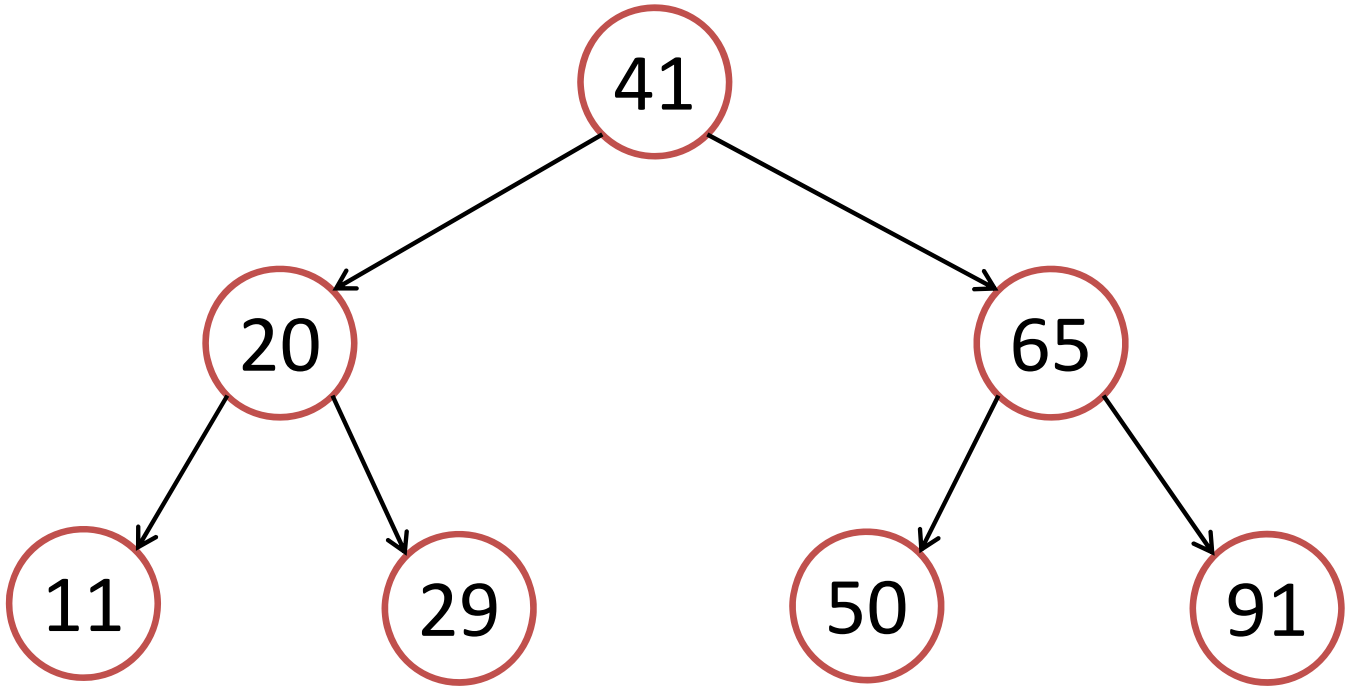
- This slide will be replaced on Tue 28 Aug 2012
- You do not have to print this slide

- Tutorial and Lab Bidding Status Update
  - Tutorial and Labs starts THIS WEEK (Week03)
- Game System Registration Check
  - Ask students to download PS1 on Tue 28 Aug 2012, 10am

Outline

- What are we going to learn in this lecture?
  - Binary Search Tree (BST): A Revision
    - Using clickers, of course ☺
    - A quick look at **BSTDemo.java, BST.java, BSTVertex.java**
  - 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
    - Codes is shown but not given (try this on your own during PS1)
  - Reference in CP 2.5 book: Page 42-43

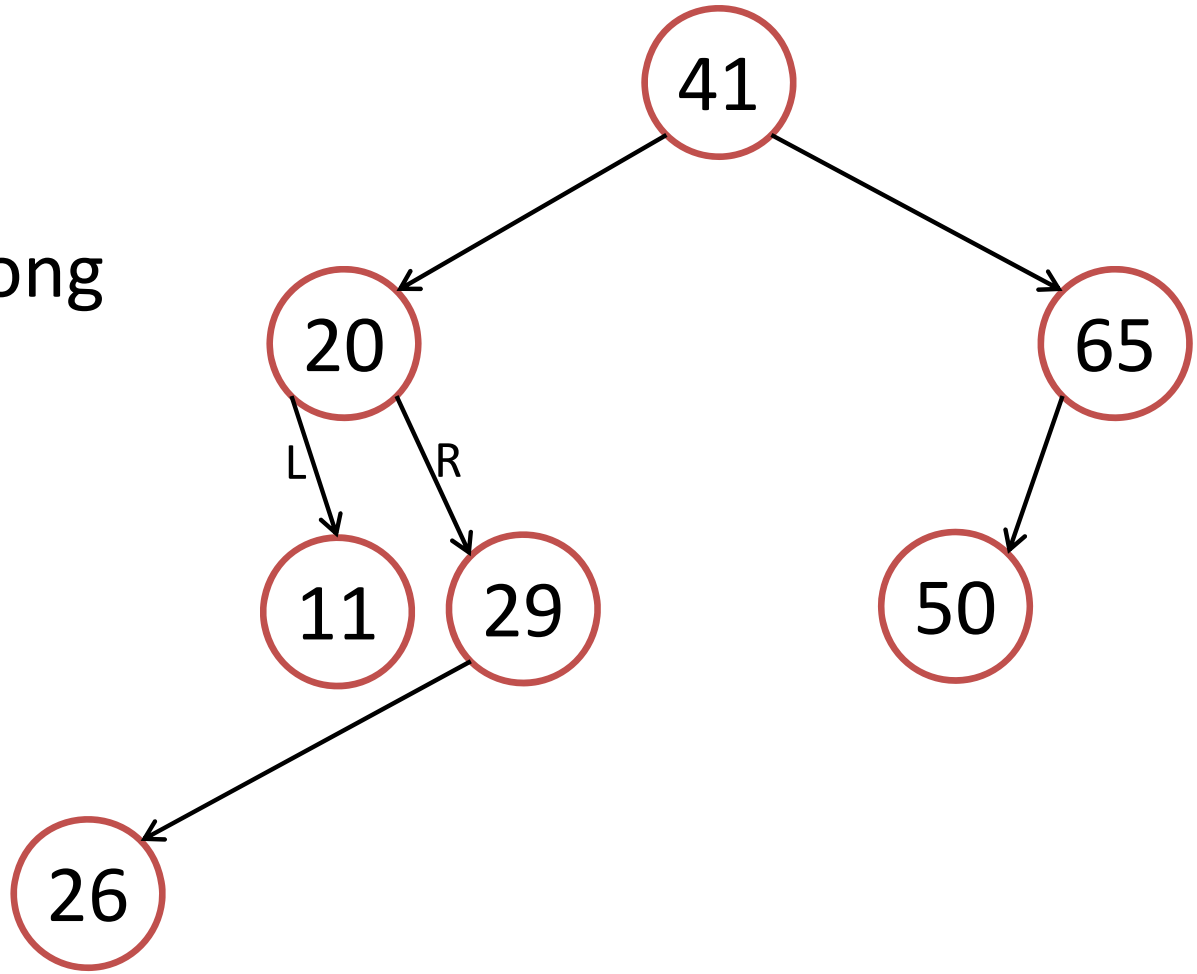
Binary Search Trees: Quick Review

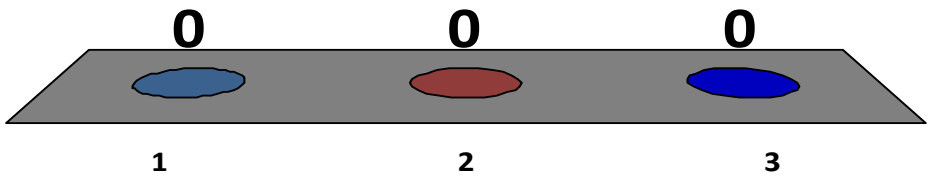


- Vertex x has two children: **x.left**, **x.right** and one parent: **x.parent**
- Vertex x has a key: **x.key**
- **BST Property**: all keys in left sub-tree < **x.key** < all keys in right sub-tree

Is this a binary search tree?

1. Yes
2. No
3. Your drawing is wrong

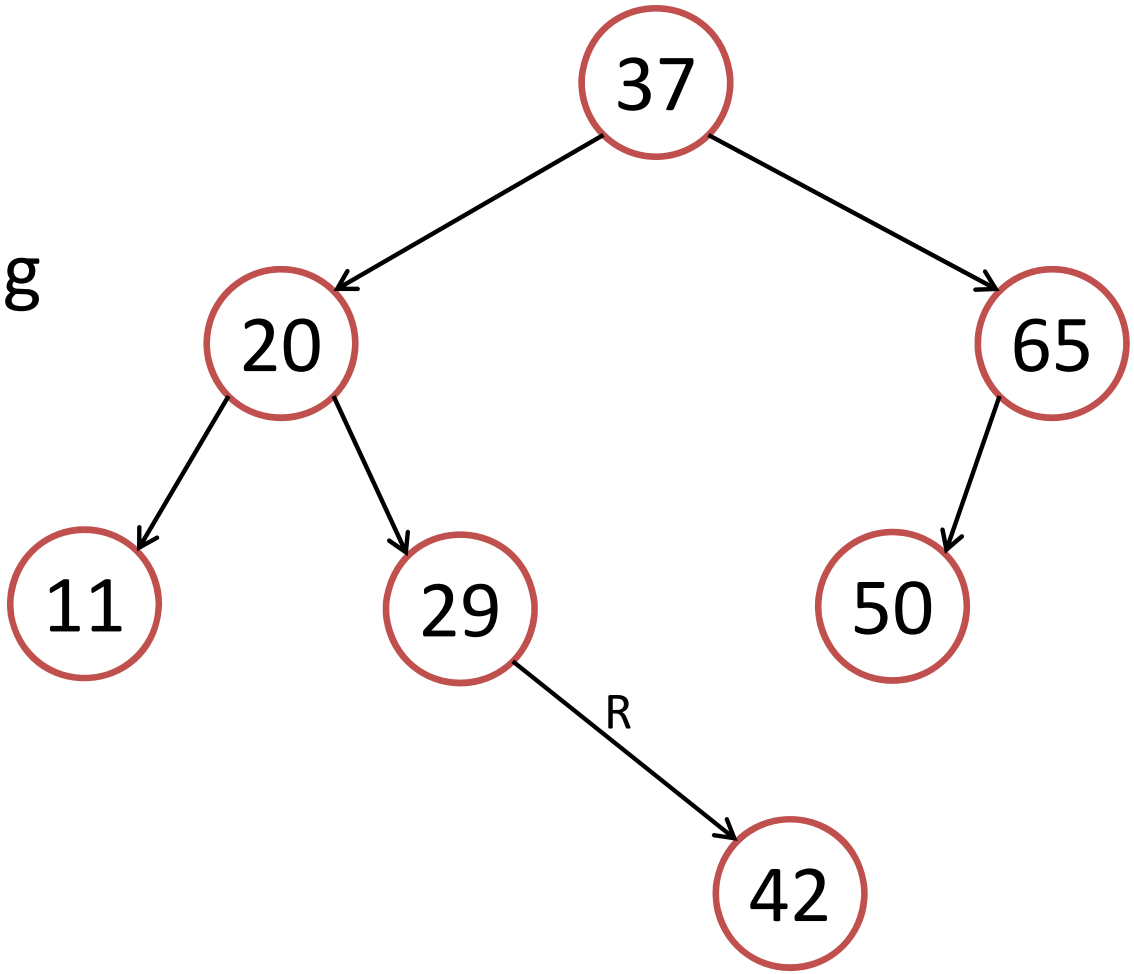


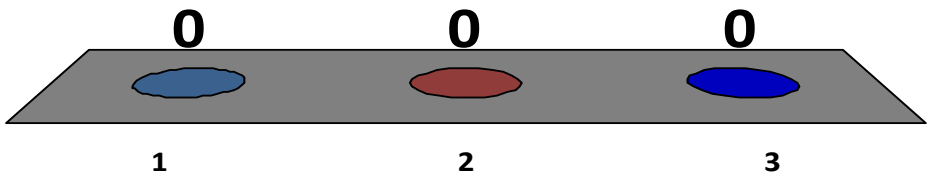


0 of 120

Is this a binary search tree?

1. Yes
2. No
3. Your drawing is wrong

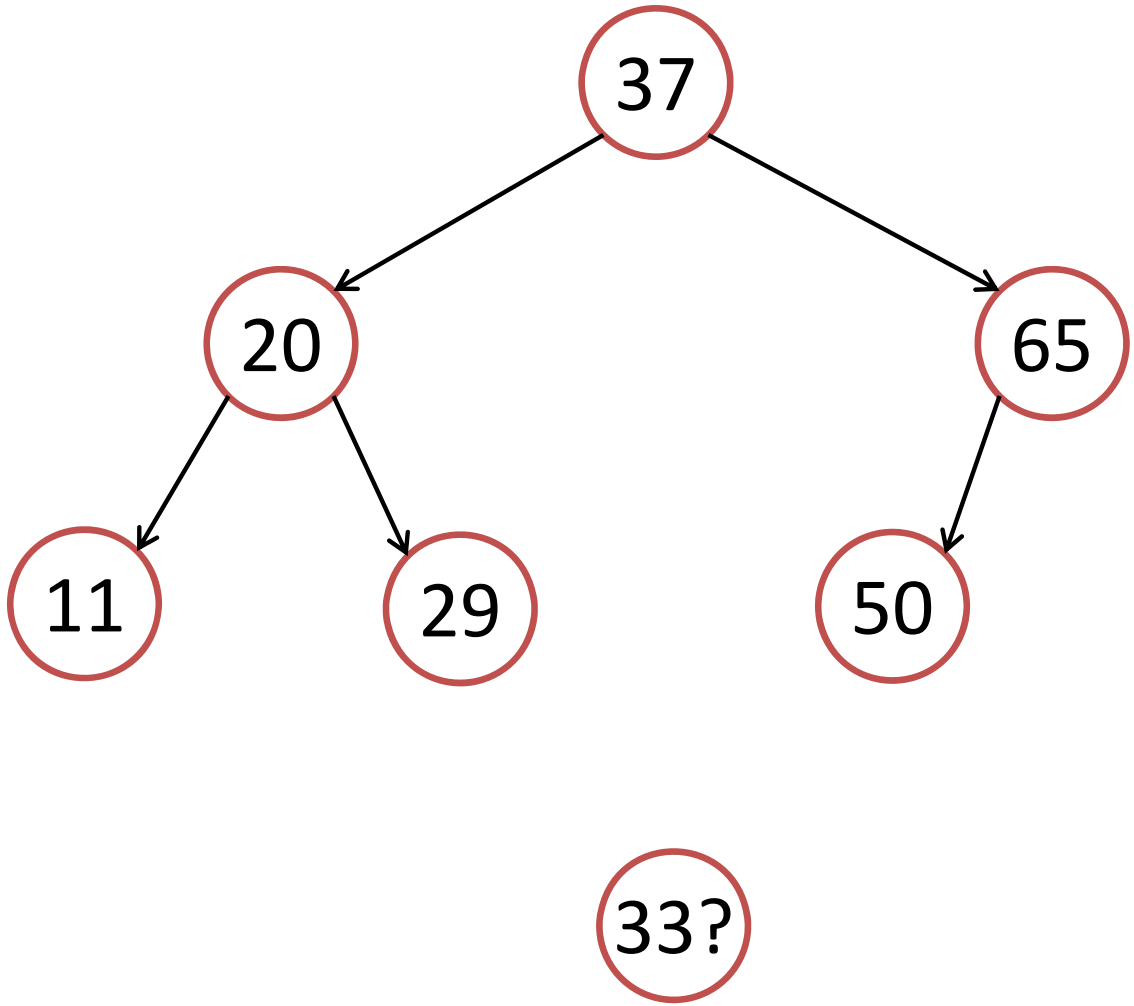


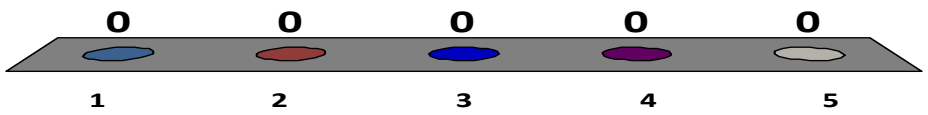


0 of 120

If I insert new value 33 into this BST, where will it be inserted?

1. On the left of 29
2. On the right of 29
3. On the left of 50
4. On the right of 65
5. None of the above

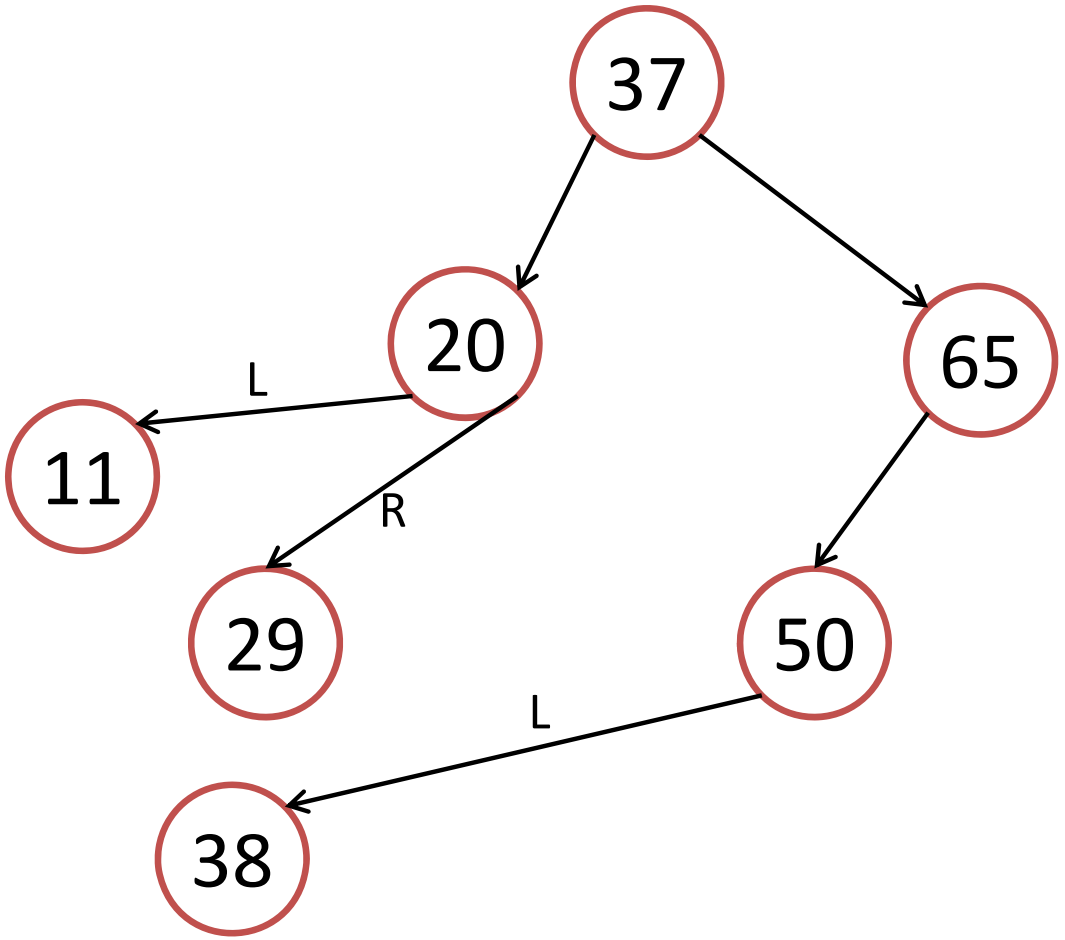


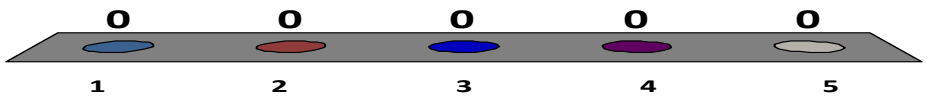


0 of 120

Which vertex is the successor of 37?

1. No successor
2. 65
3. 50
4. 38
5. Your drawing is confusing…

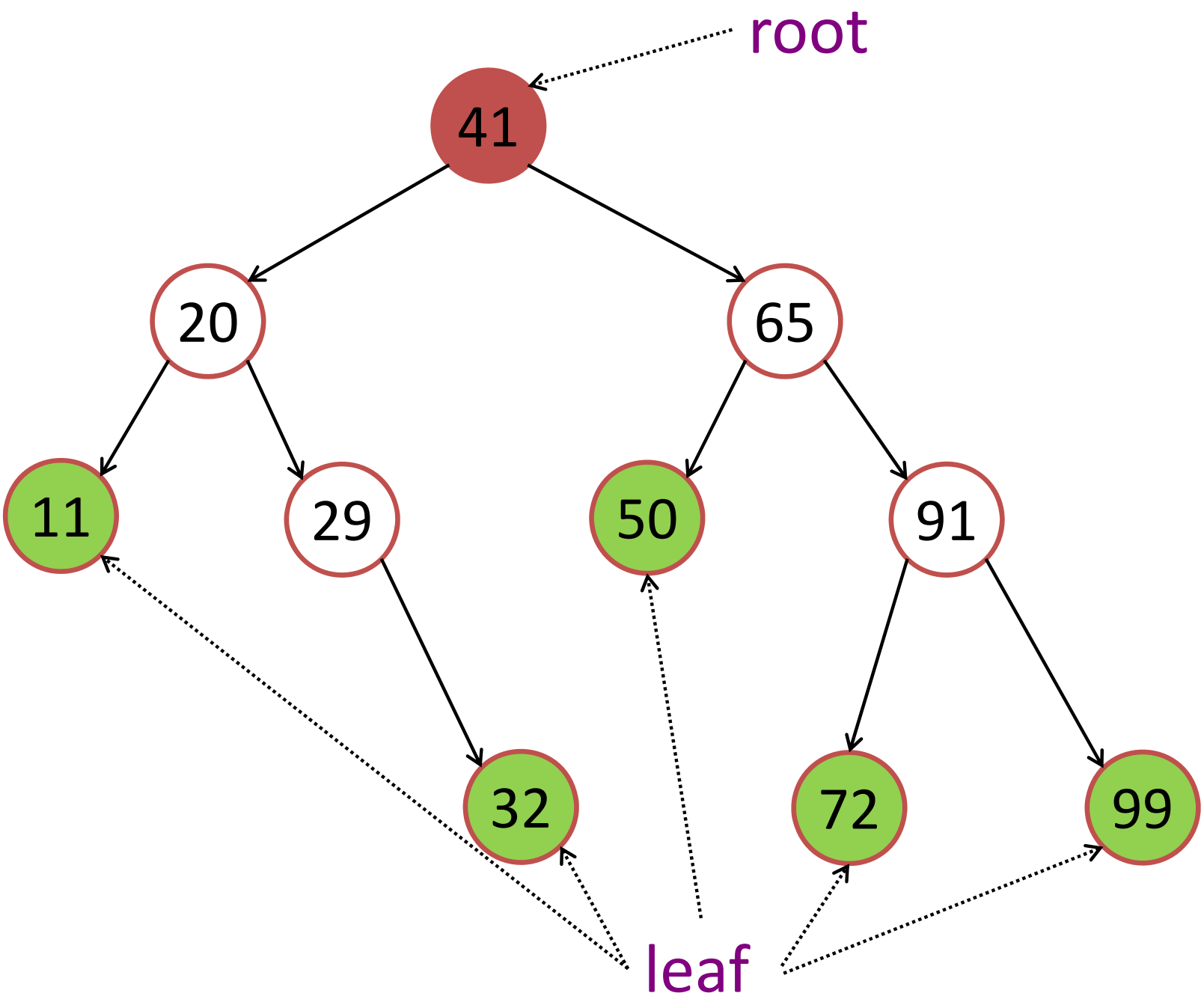




Binary Search Trees: Quick Review

Let’s take a look at BSTDemo.java, BST.java, and BSTVertex.java

Binary Search Trees: Quick Review



Binary Search Trees: Height

New attribute at each BST vertex: Height

The number of edges on the path from this vertex to deepest leaf

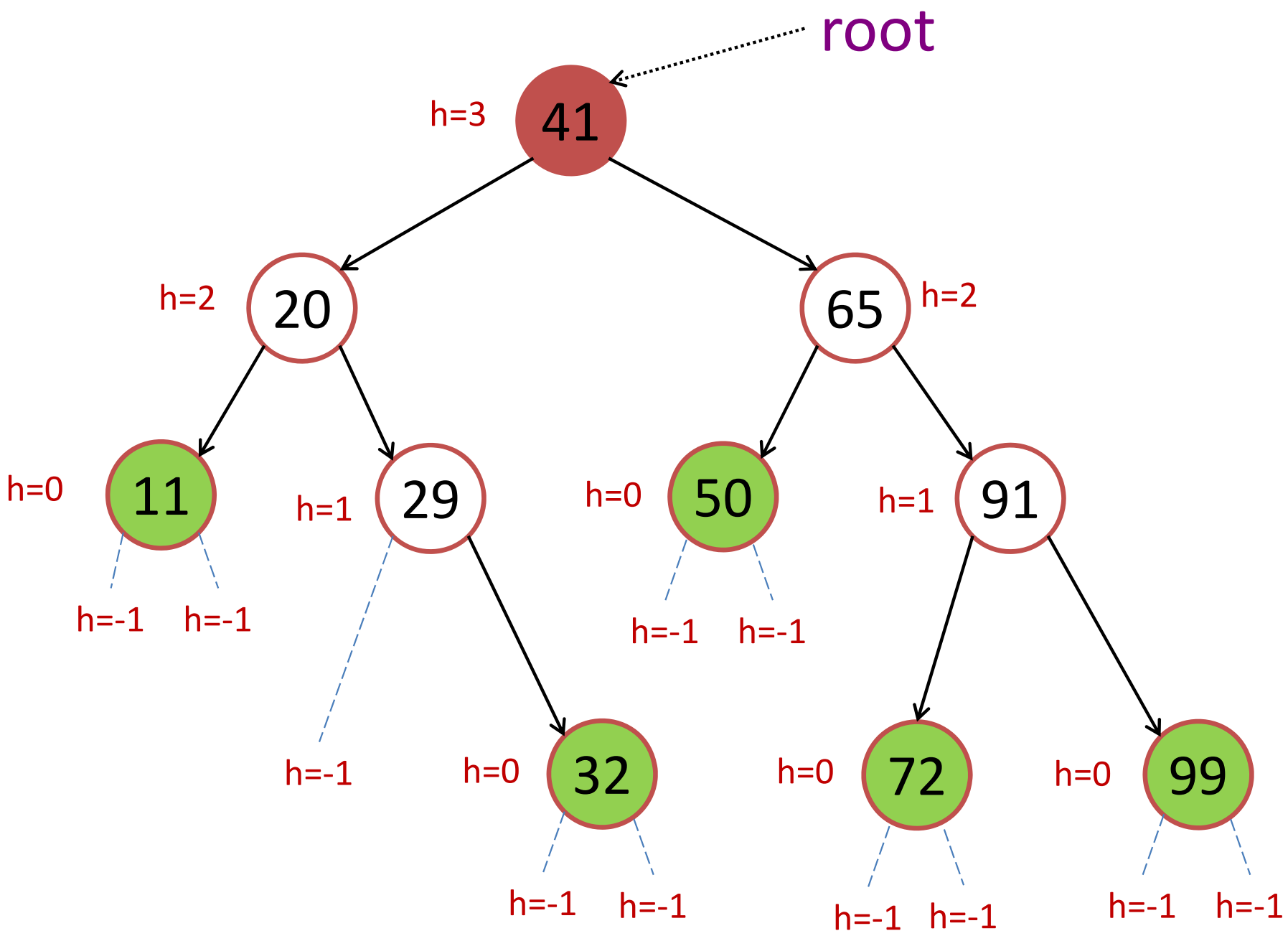
These values can be computed recursively:

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

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

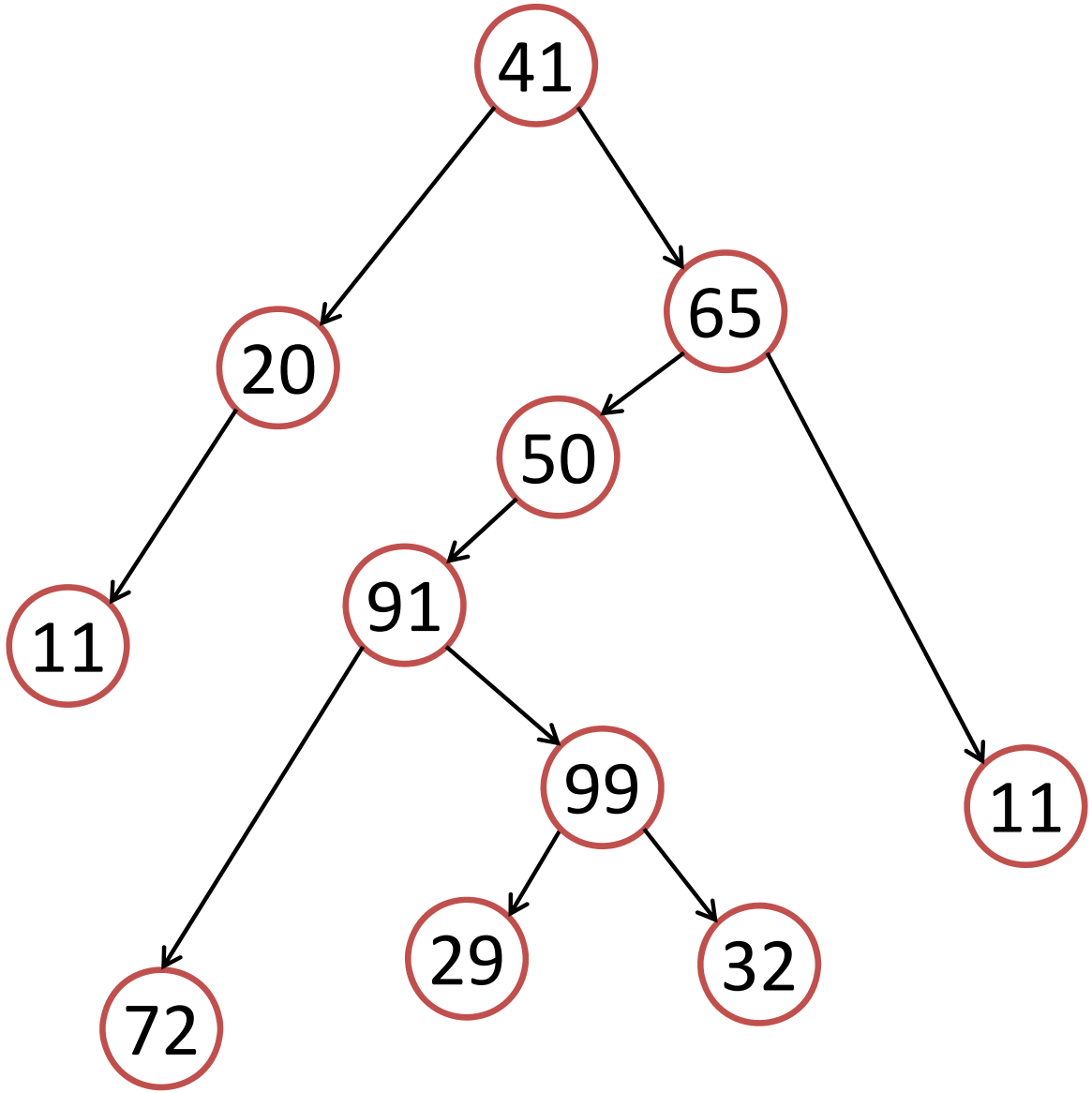
The height of the BST is thus: root.height

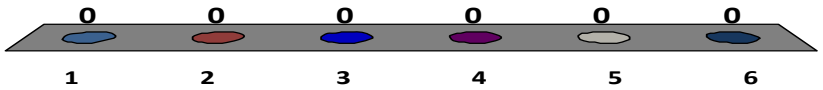
Binary Search Trees: Height (h)



The height of this tree is?

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





0 of 120

Binary Search Tree: Summary

Operations that modify the BST:

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

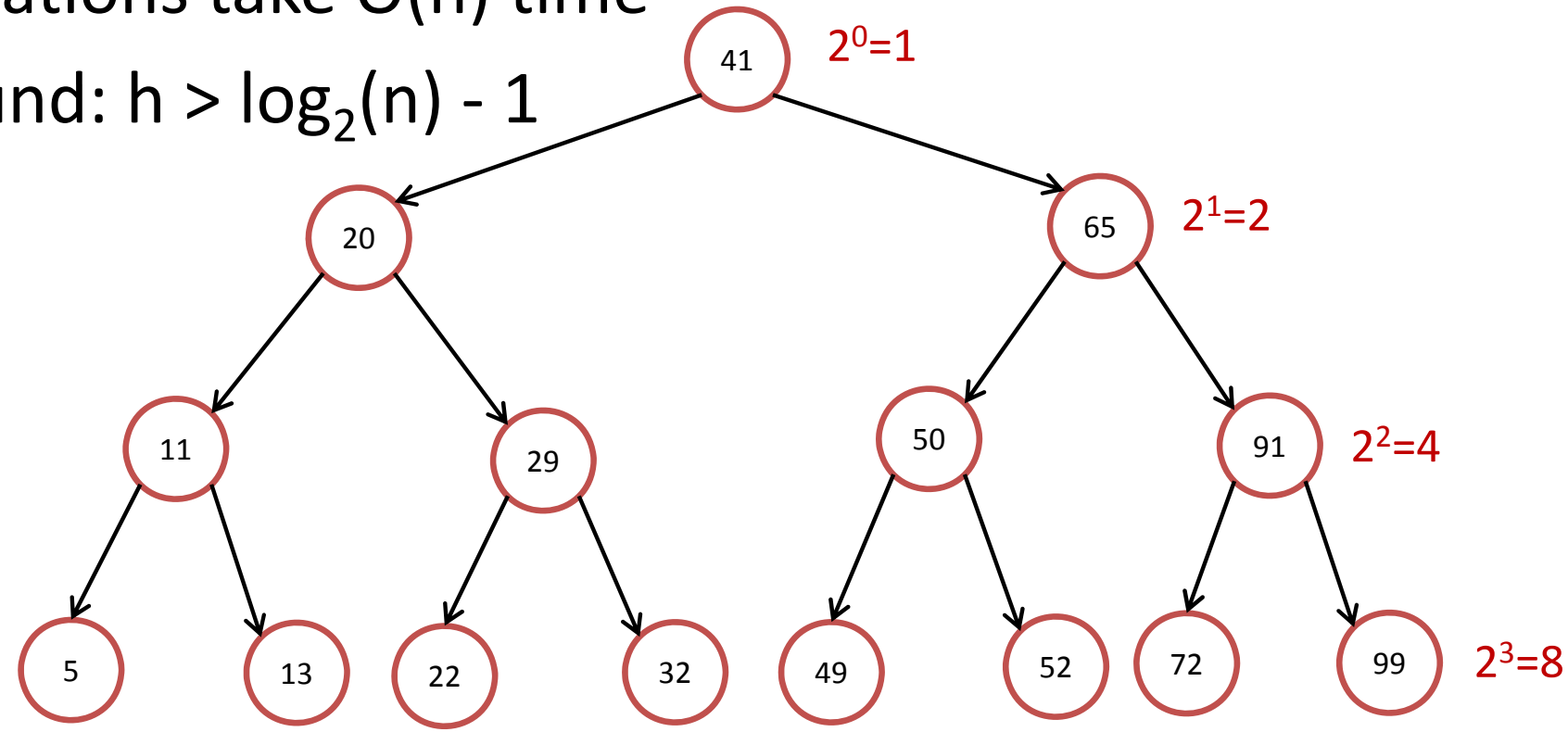
Query Operations:

- search: O(h)
- findMin, findMax: O(h)
- predecessor, successor: O(h)
- inorder traversal: O(n) – the only one that does not depend on **h**

The Importance of Being Balanced

Most operations take O(h) time

Lower bound: $h > \log_2(n) - 1$



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

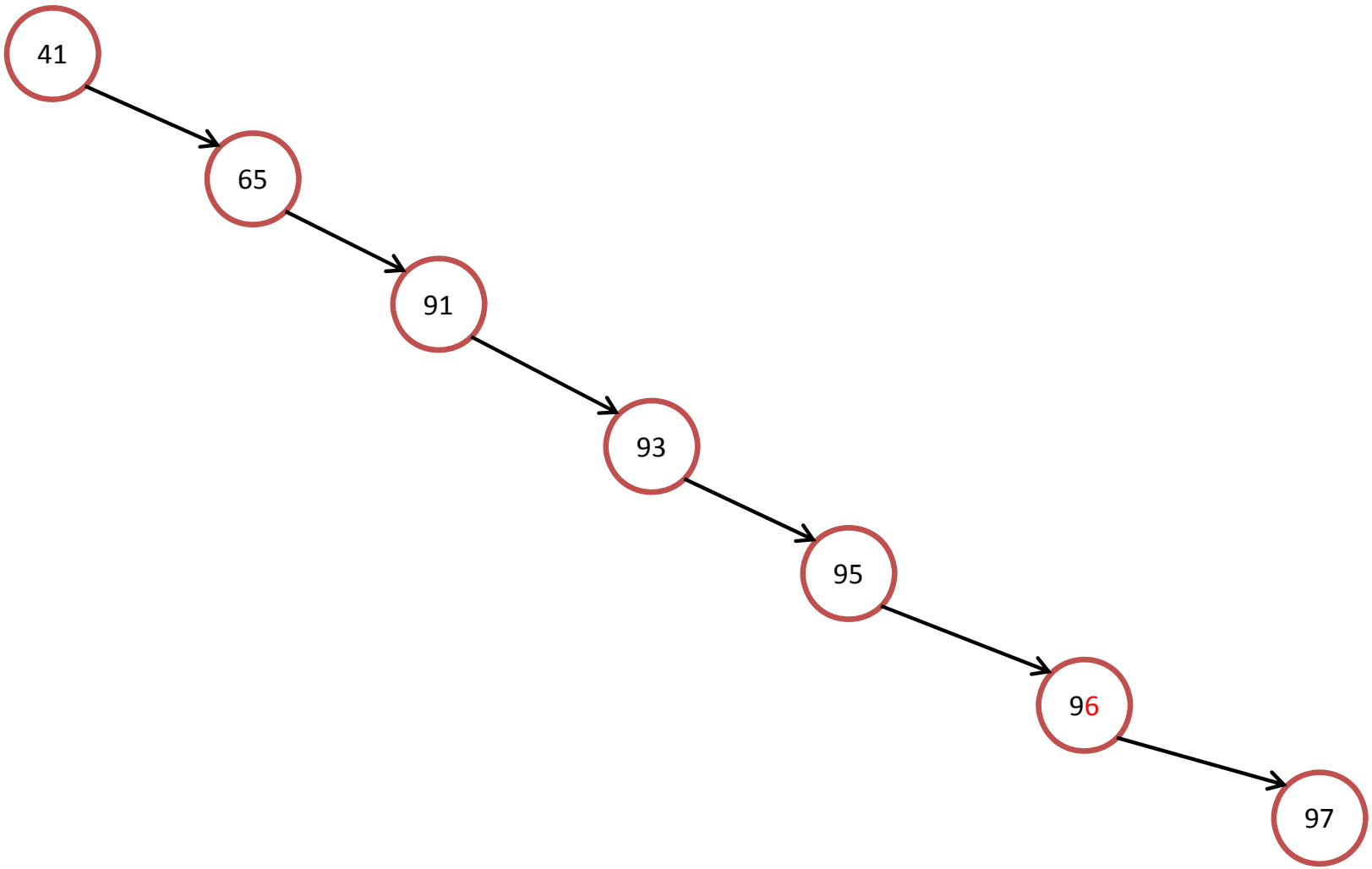
$\leq 2^0 + 2^1 + 2^2 + \ldots + 2^h < 2^{h+1}$ (sum of geometric progression)

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

The Importance of Being Balanced

Most operations take O(h) time

Upper bound: h < n

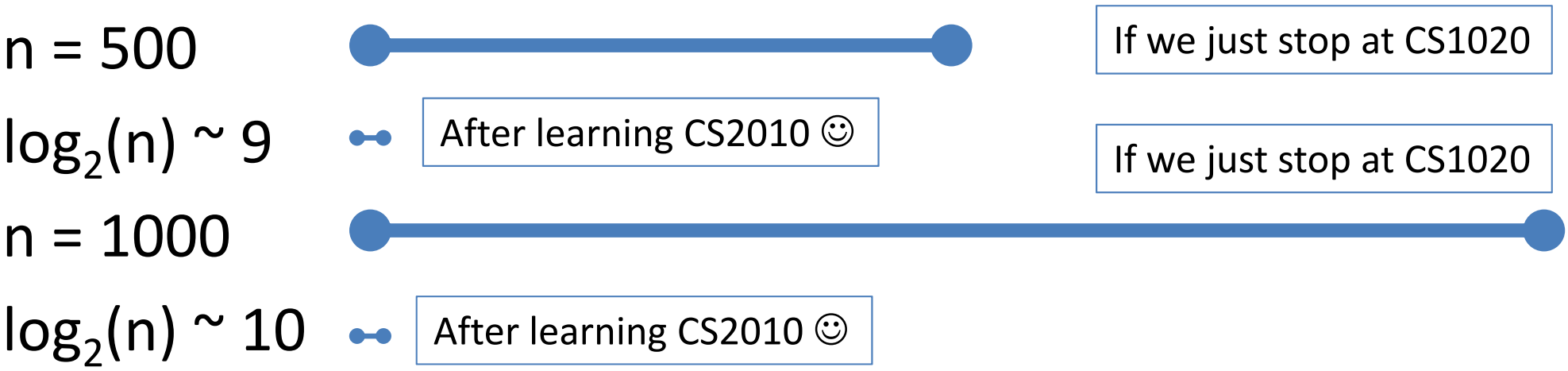


The Importance of Being Balanced

Most operations take O(h) time

Combined bound: $\log_2(n) - 1 < h < n$

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



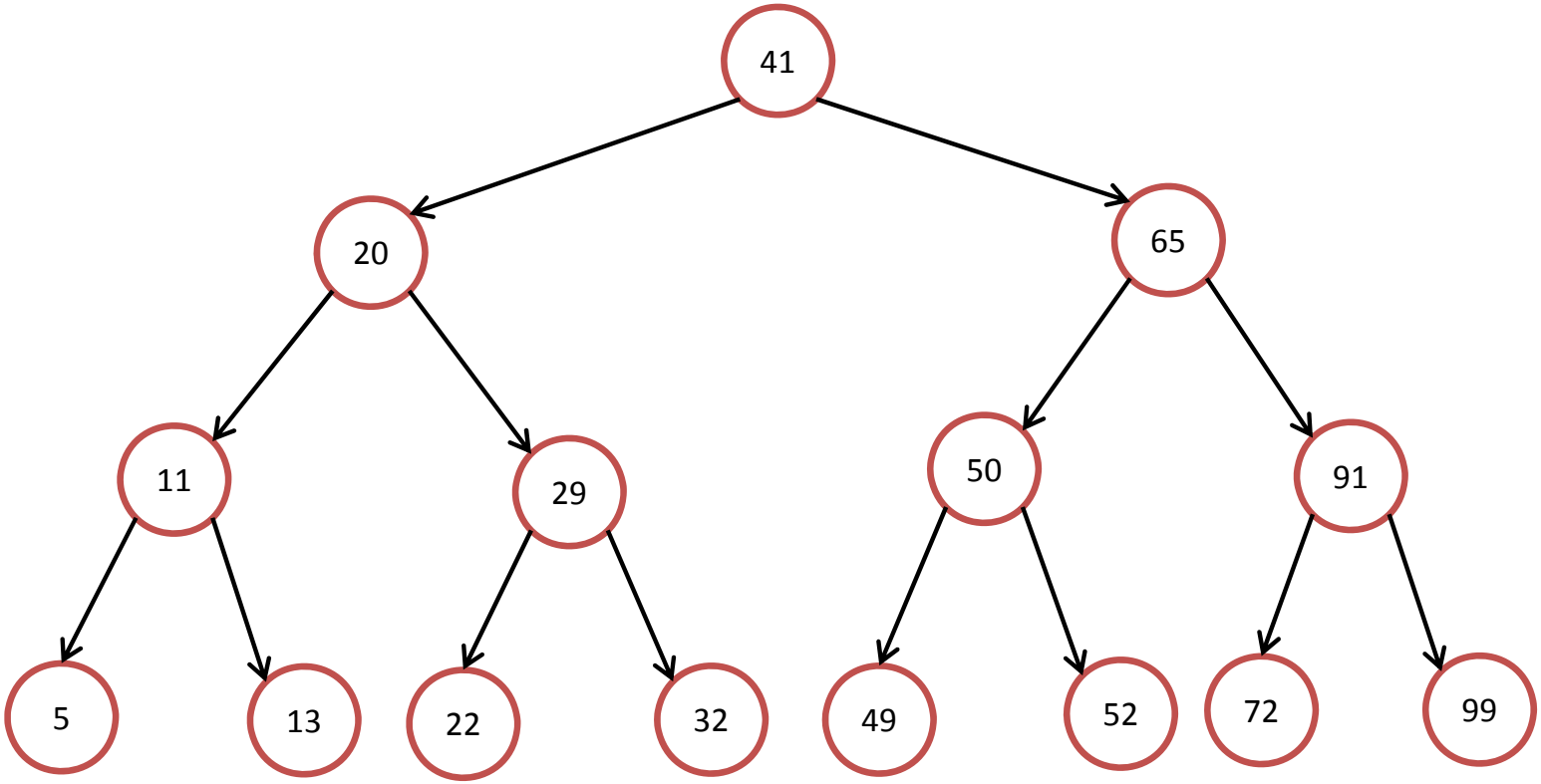
We say a BST is balanced if h = O(log n)

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 (~ 50 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 (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**:

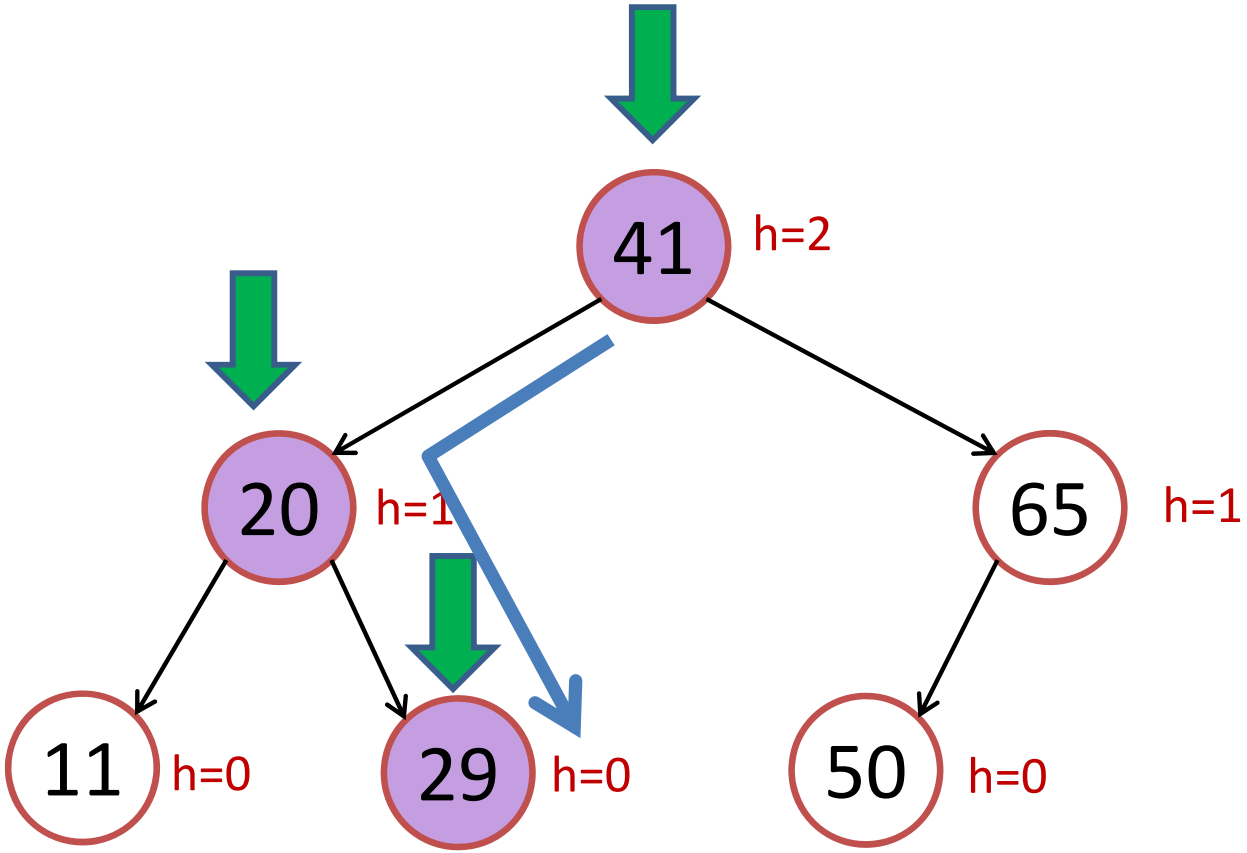
```
insert(x, v)
  // ... same as before ...
  x.height = max(x.left.height, x.right.height) + 1

// update height during deletion too (not shown)
```

Binary Search Trees

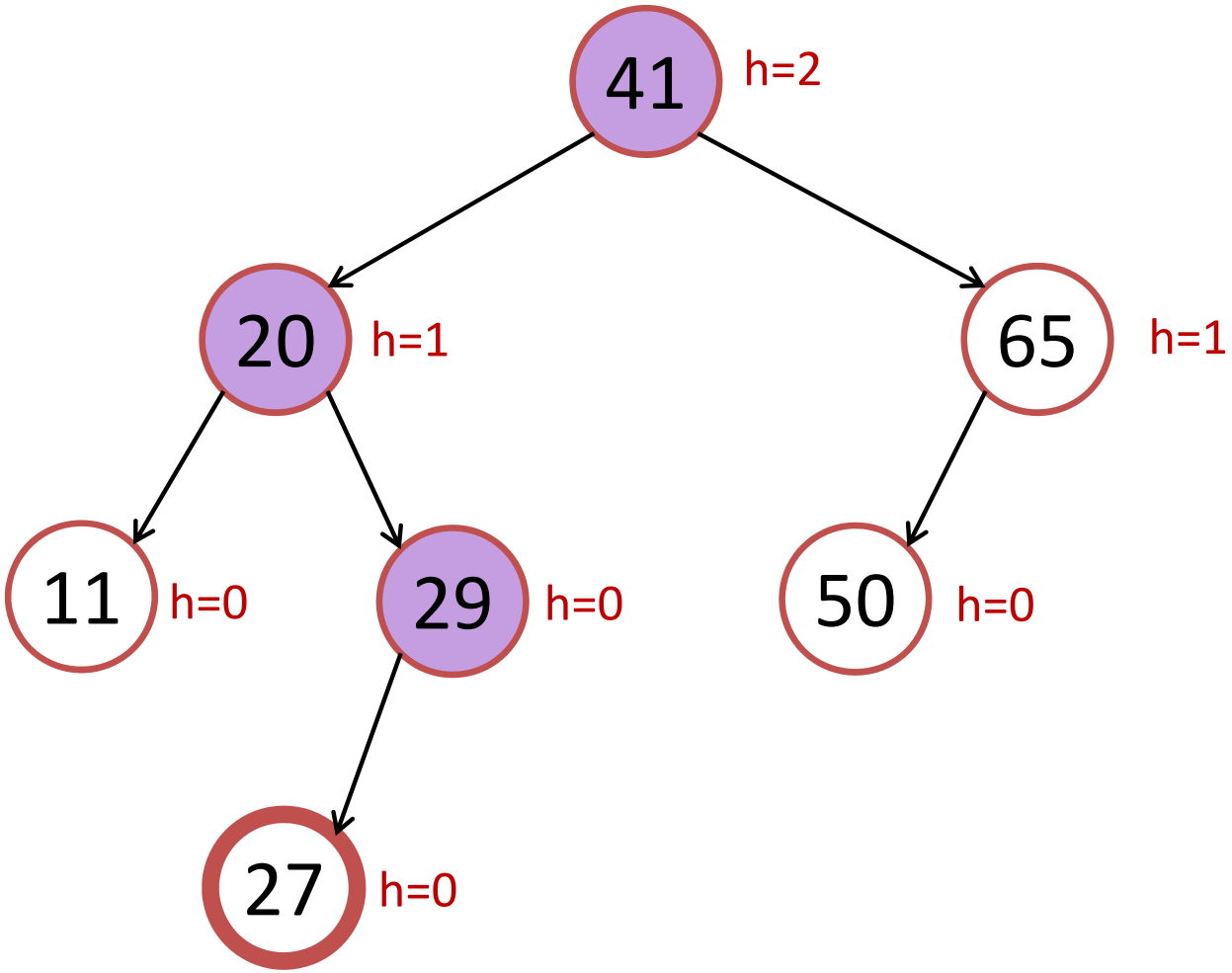
Height of empty trees are ignored in this illustration (all -1)

insert(27)



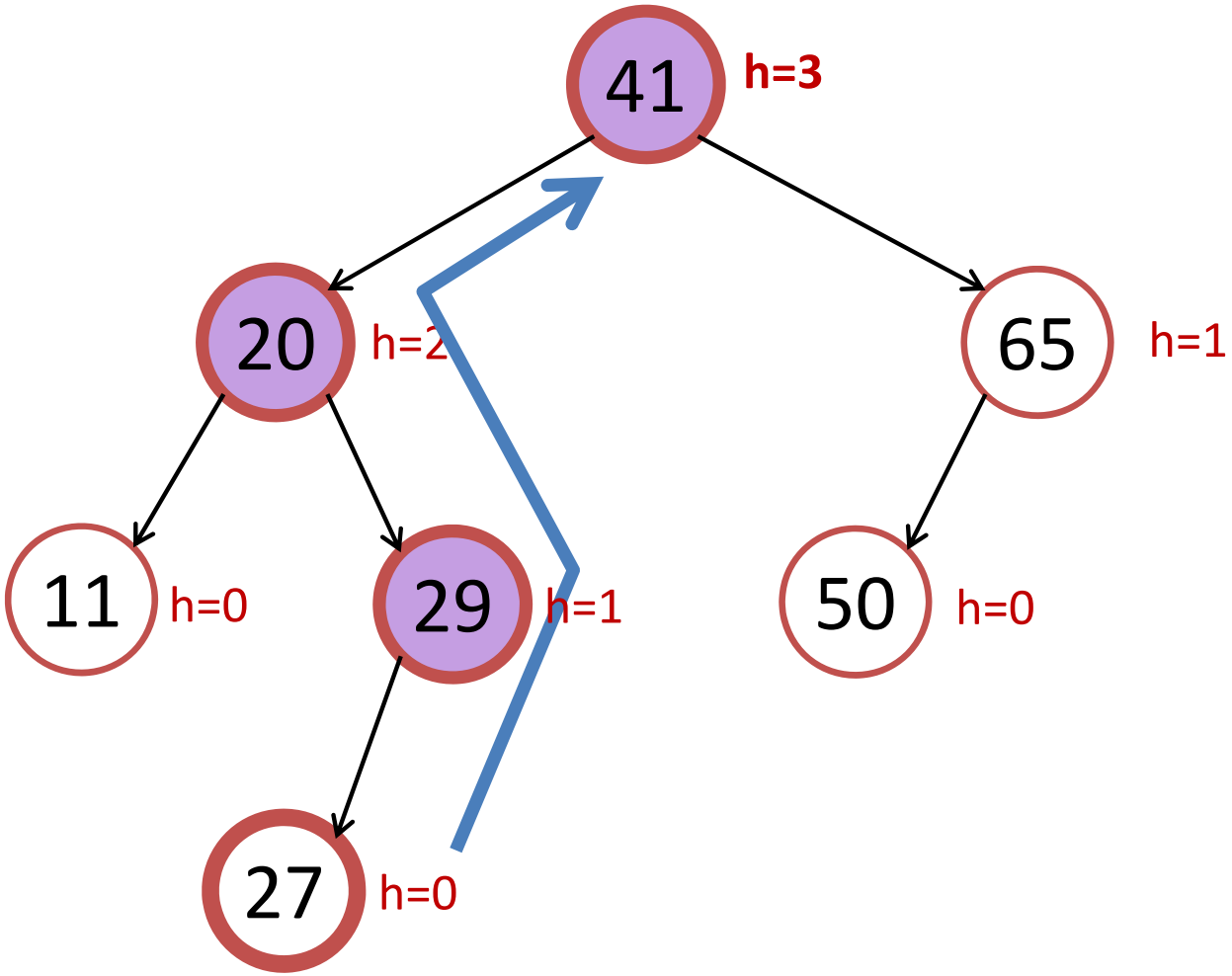
Binary Search Trees

insert(27)



Binary Search Trees

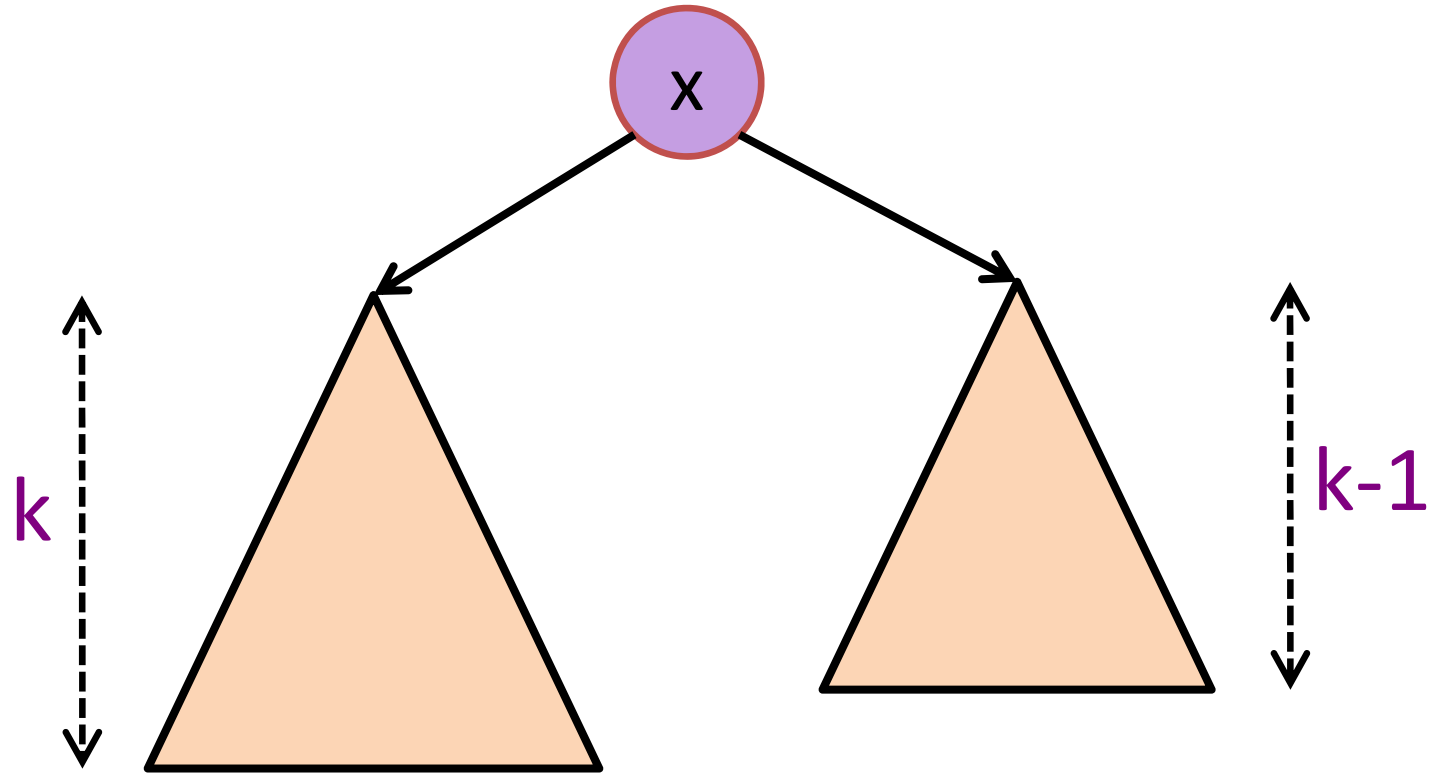
insert(27)



AVL Trees [Adelson-Velskii & Landis 1962]

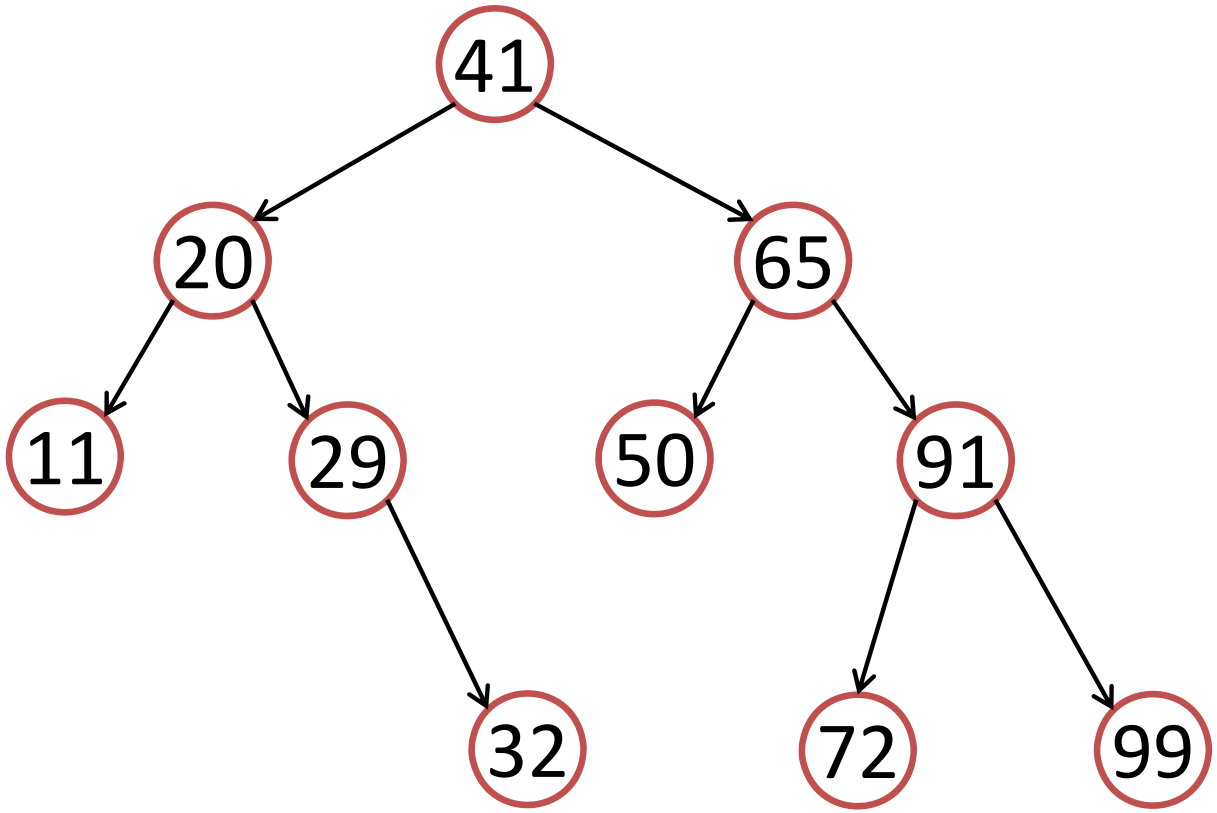
Step 2: Define Invariant (something that will not change)

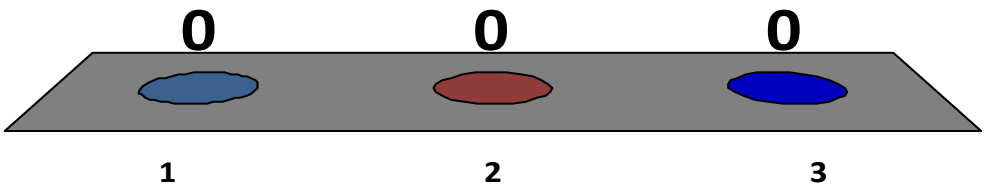
- A vertex x is said to be height-balanced if:
  $|x.left.height - x.right.height| \leq 1$
- An binary search tree is said to be height balanced if:
  every vertex in the tree is height-balanced



Is this tree height-balanced?

1. Yes
2. No
3. I am confused… ☹

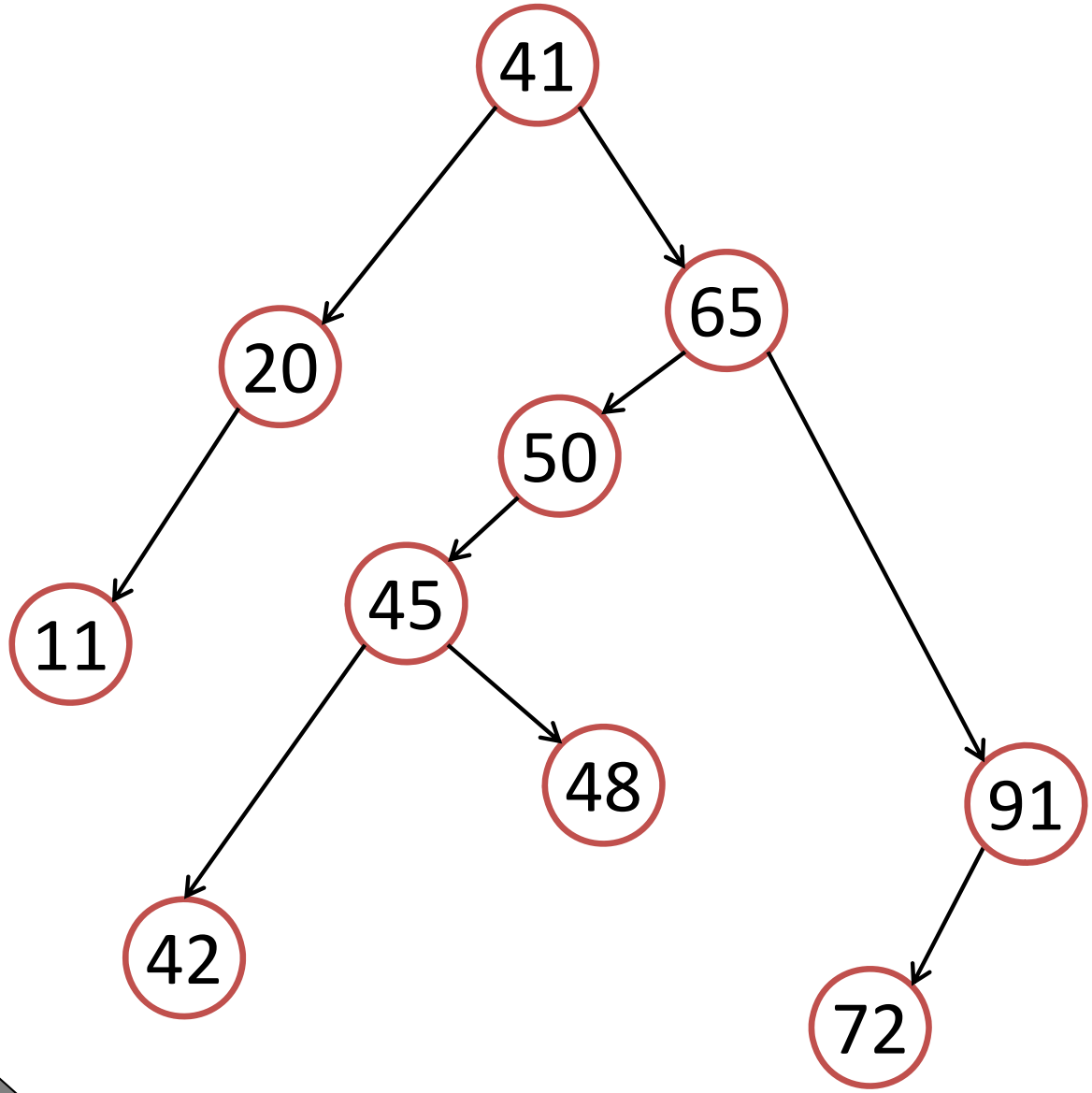


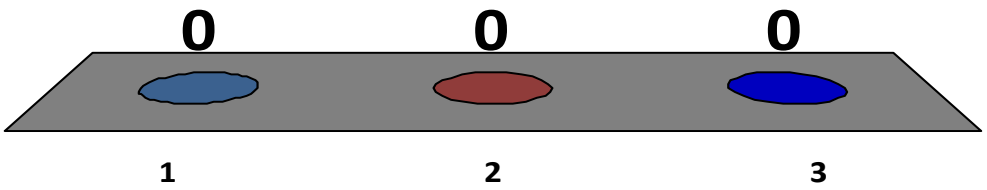


0 of 120

Is this tree height-balanced?

1. Yes
2. No
3. I am confused… ☹





0 of 120

Height-Balanced Trees

Claim:

A height-balanced tree with n vertices has height $h \leq 2 * \log_2(n)$

Proof:

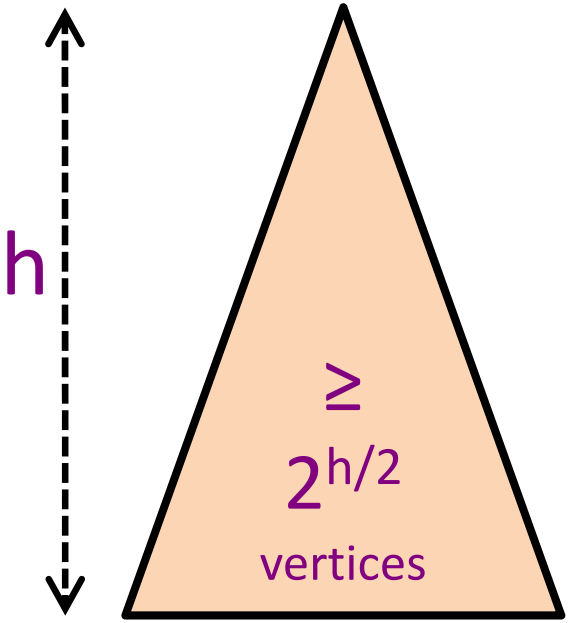
Let n_h be the minimum number of vertices in a height-balanced tree of height h

We will see later that:

$n_h \geq 2^{h/2}$

$\Rightarrow$ (put $\log_2$ on both side)

$2 * \log_2(n_h) \geq h$



Height-Balanced Trees

Proof:

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

$n_h \geq 1 + n_{h-1} + n_{h-2}$

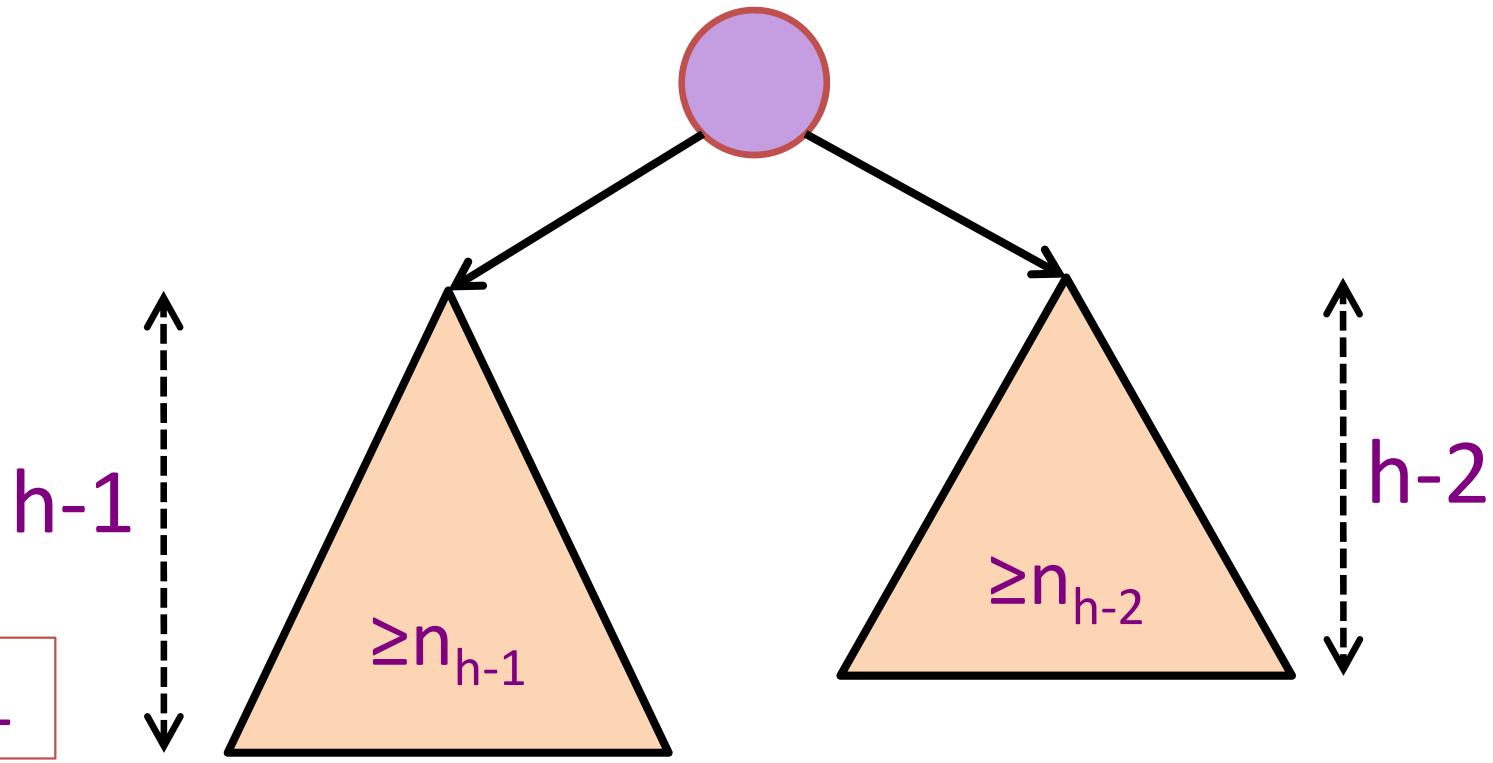
$n_h \geq 2n_{h-2}$

$n_h \geq 4n_{h-4}$

$n_h \geq 8n_{h-6}$

$\geq \ldots$

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

$n_h \geq 1 + n_{h-1} + n_{h-2}$

$n_h \geq 2n_{h-2}$

$n_h \geq 4n_{h-4}$

$n_h \geq 8n_{h-6}$

$\geq \ldots$

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

$n_h \geq 2^{h/2} n_0$

$n_h \geq 2^{h/2}$

Base case: $n_0 = 1$

Height-Balanced Trees

Claim:

A height-balanced tree is balanced,
i.e. has height $h = O(\log(n))$

We can show that:

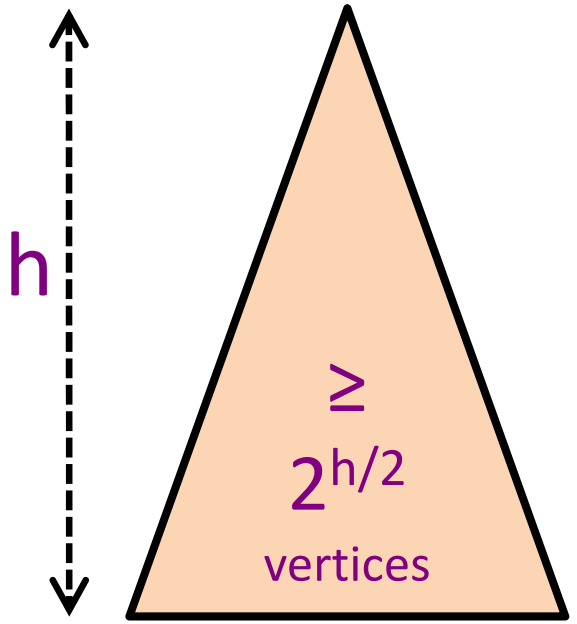
$n_h \geq 2^{h/2}$

$\Rightarrow$ (put $\log_2$ on both side)

$2 * \log_2(n_h) \geq h$

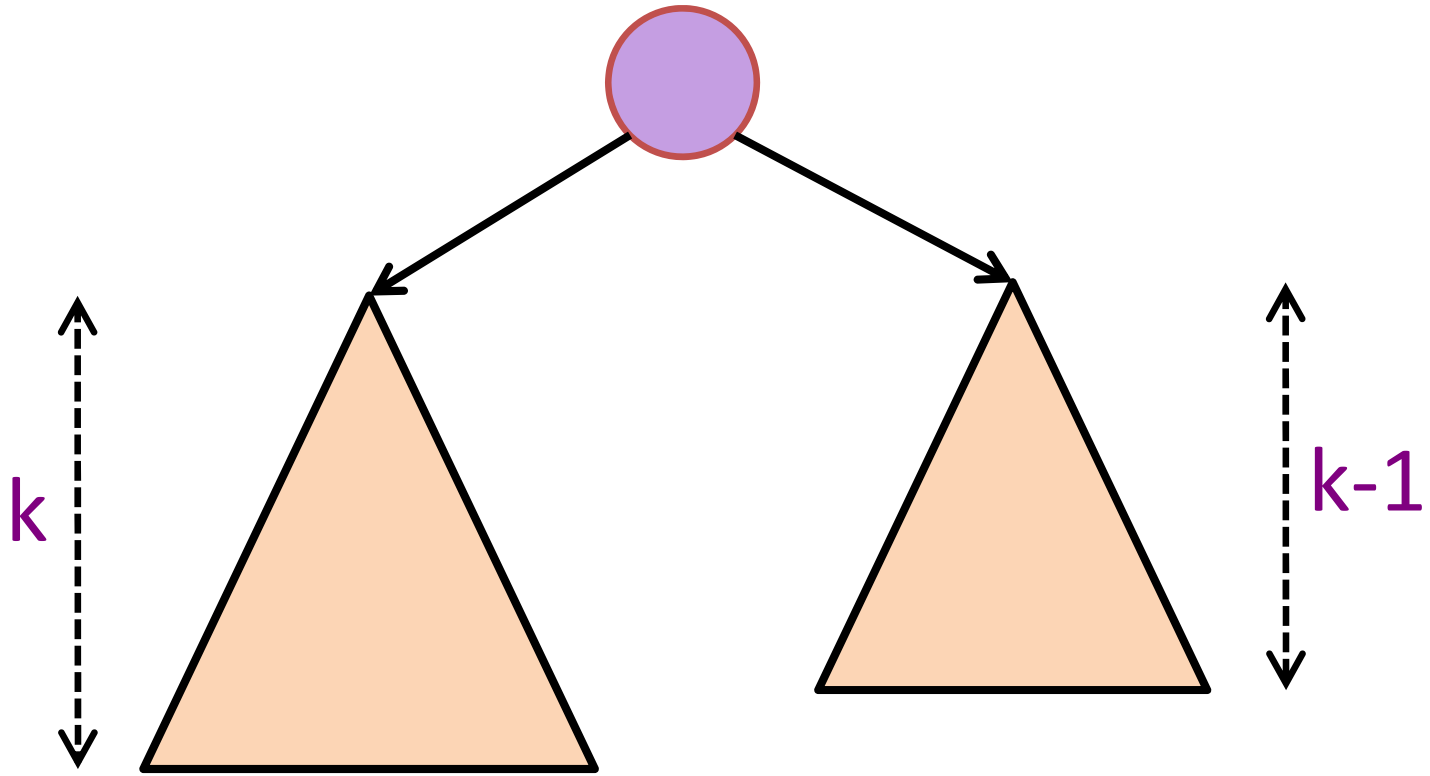
$h \leq 2 * \log_2(n_h)$

$h = O(\log(n))$



AVL Trees [Adelson-Velskii & Landis 1962]

Step 3: Show how to maintain height-balance



Insertion to an AVL Tree

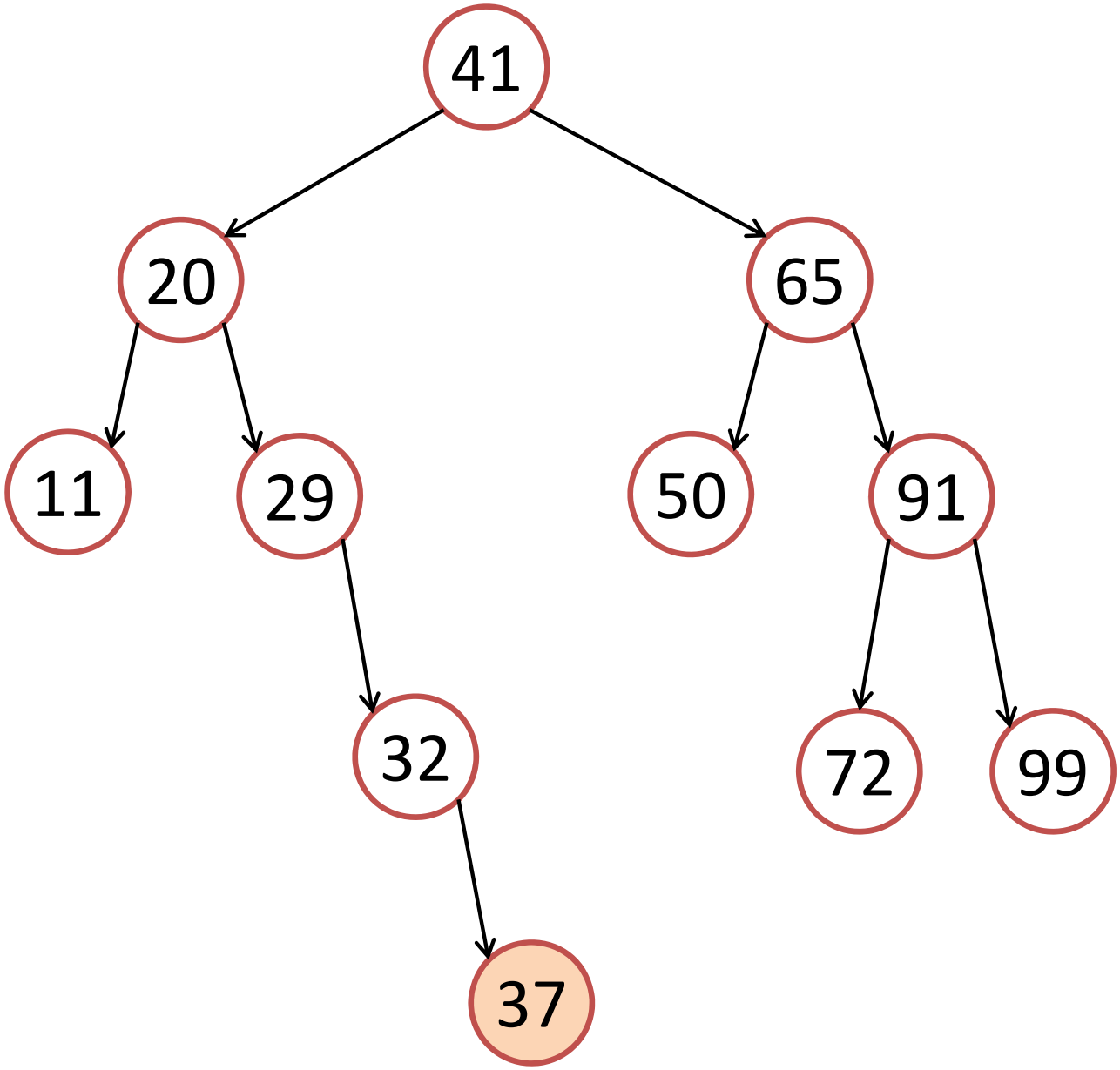
insert(37)

Initially balanced

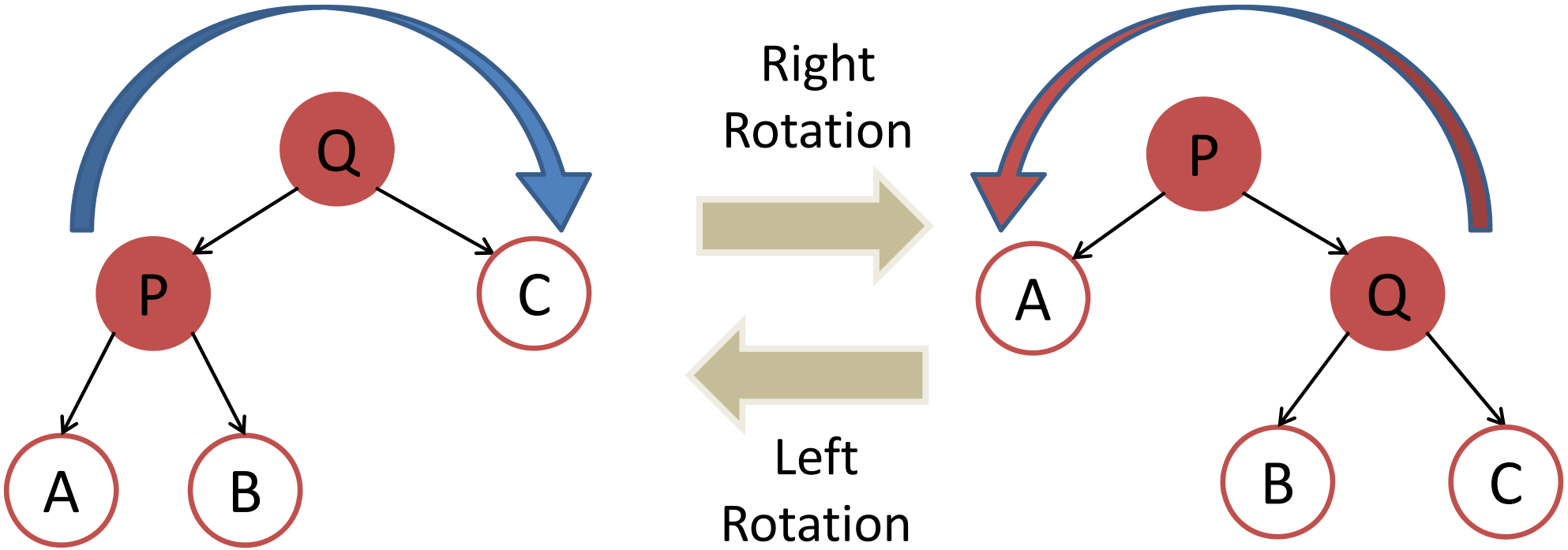
But no longer balanced after Inserting 37

Need to rebalance!

But how?



Tree Rotations



Rotations maintain ordering of keys

⇒Maintains BST property

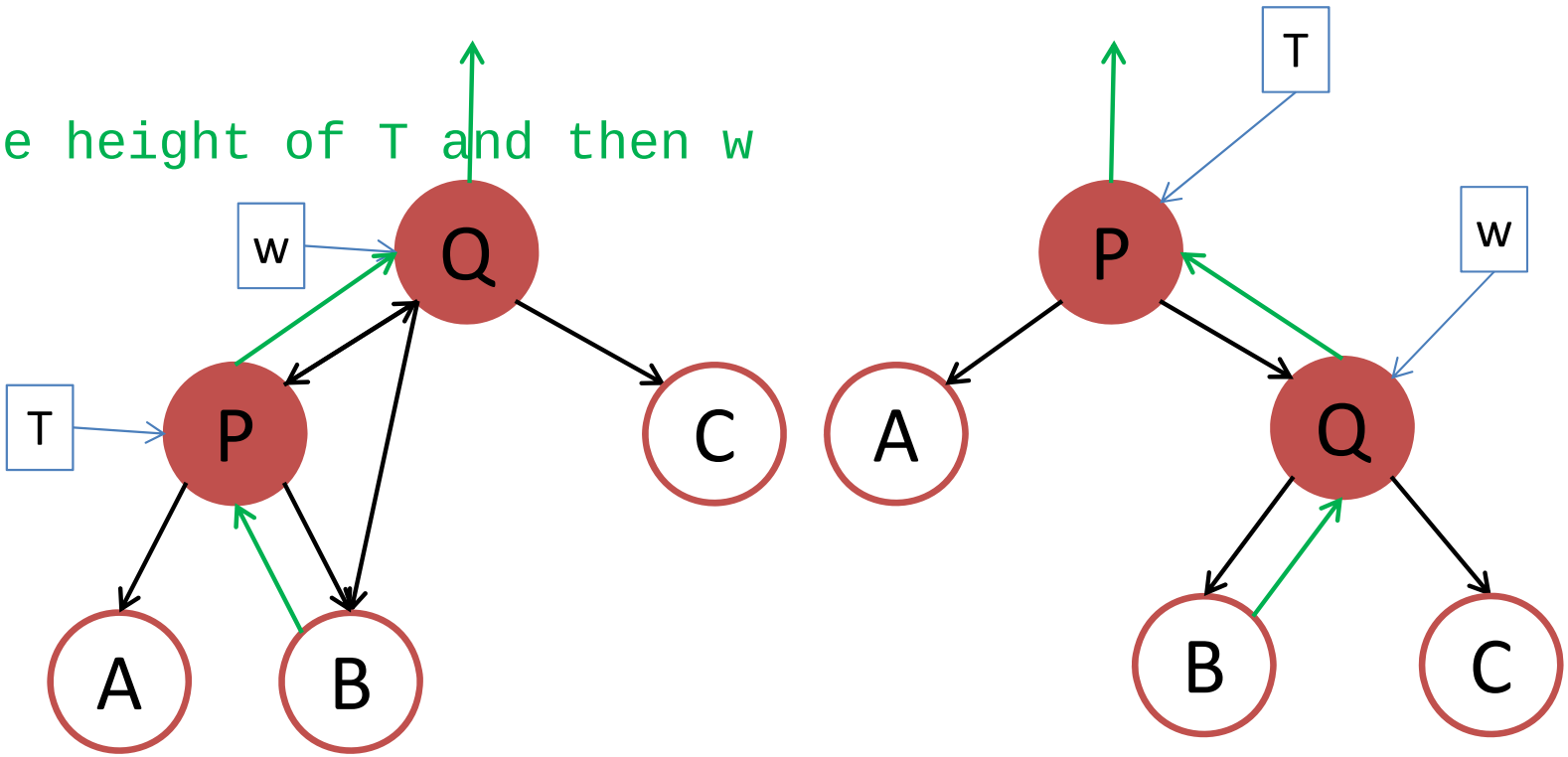
rotateRight requires a left child

rotateLeft requires a right child

Tree Rotations Pseudo Code → O(1)

```
BSTVertex rotateLeft(BSTVertex T) // T.left != 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



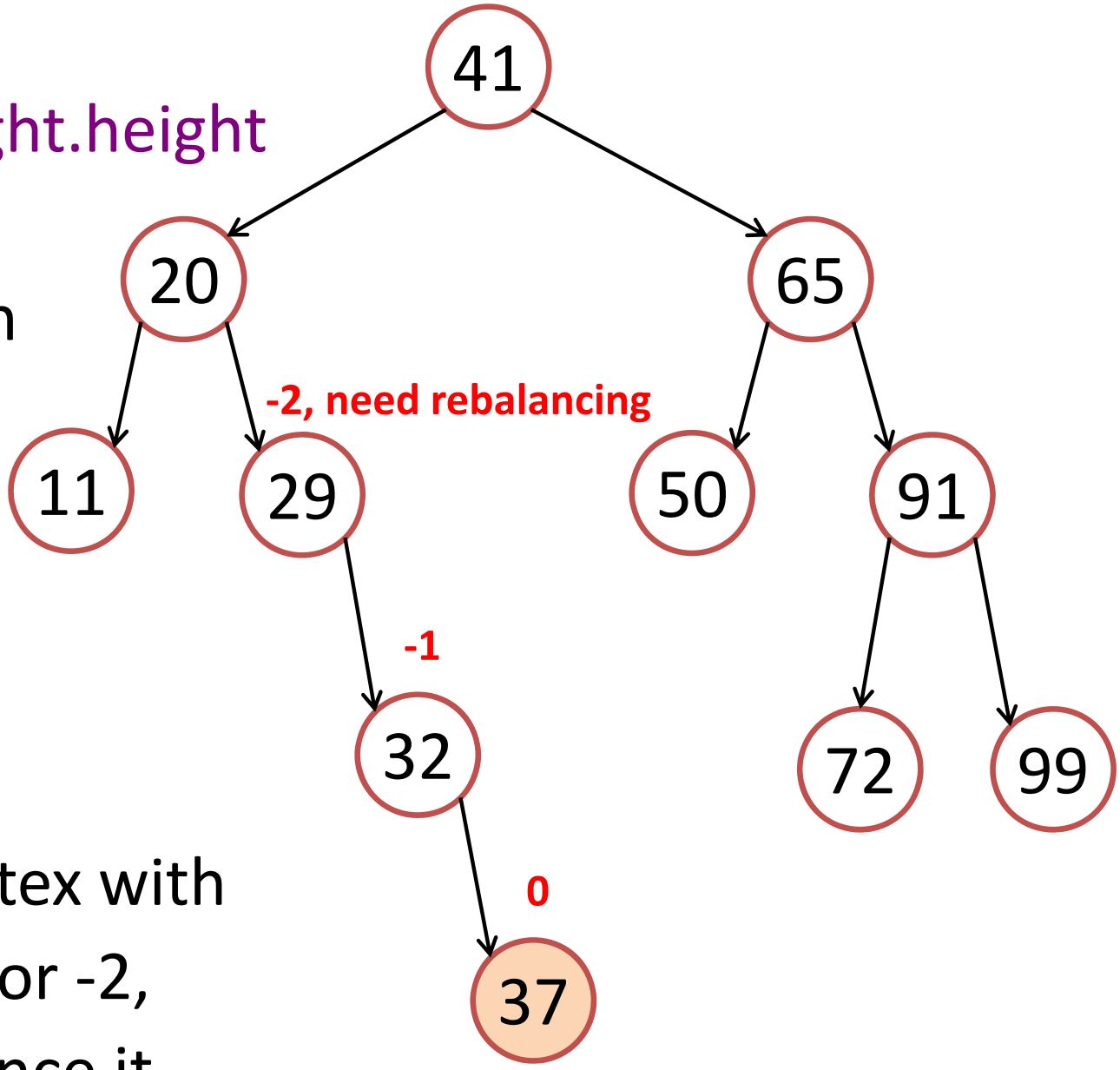
This slide is can be confusing without the animation

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)

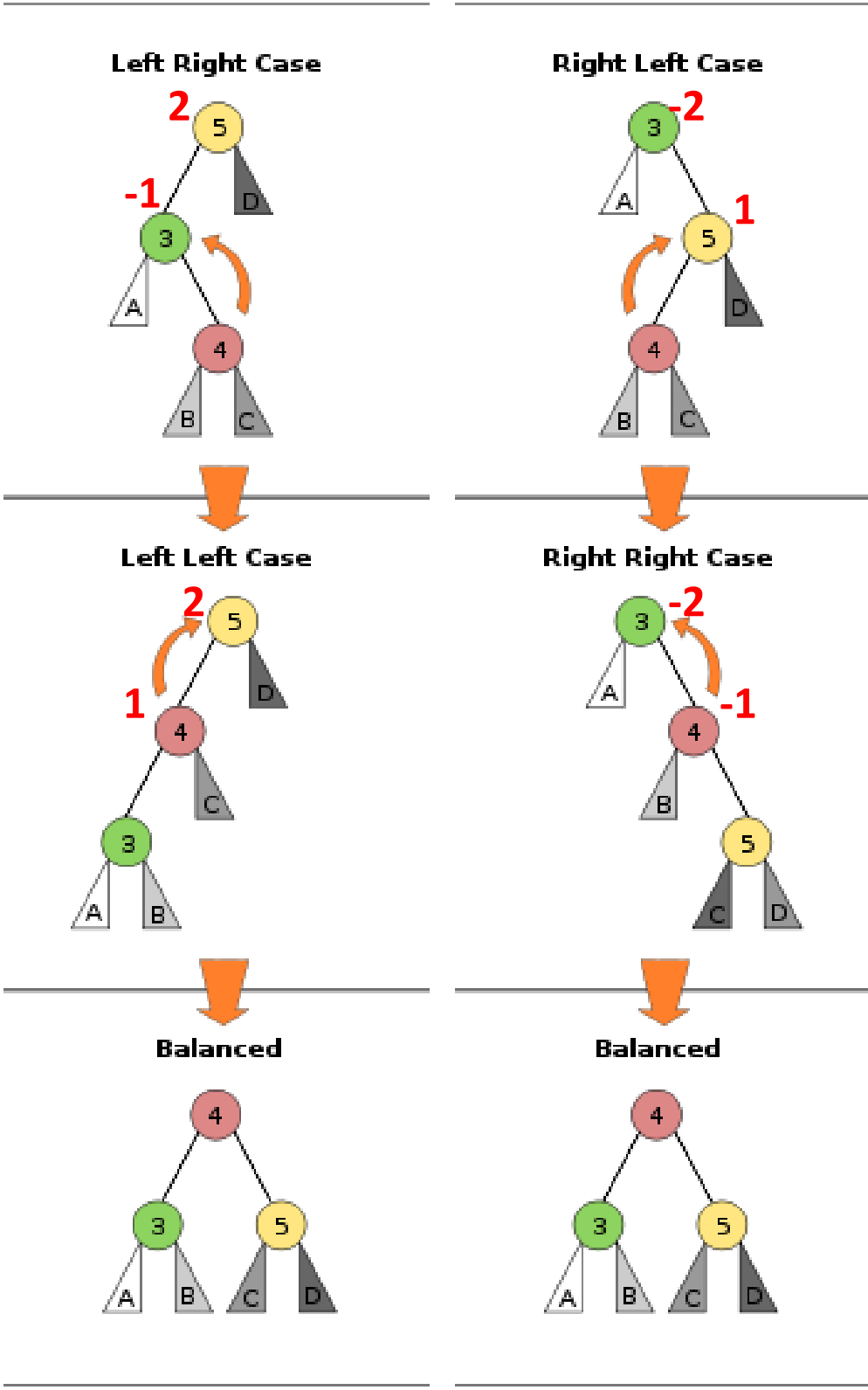
bf(x) = -2 and bf(x.right) = -1

leftRotate(x)

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

rightRotate(x.right)

leftRotate(x)

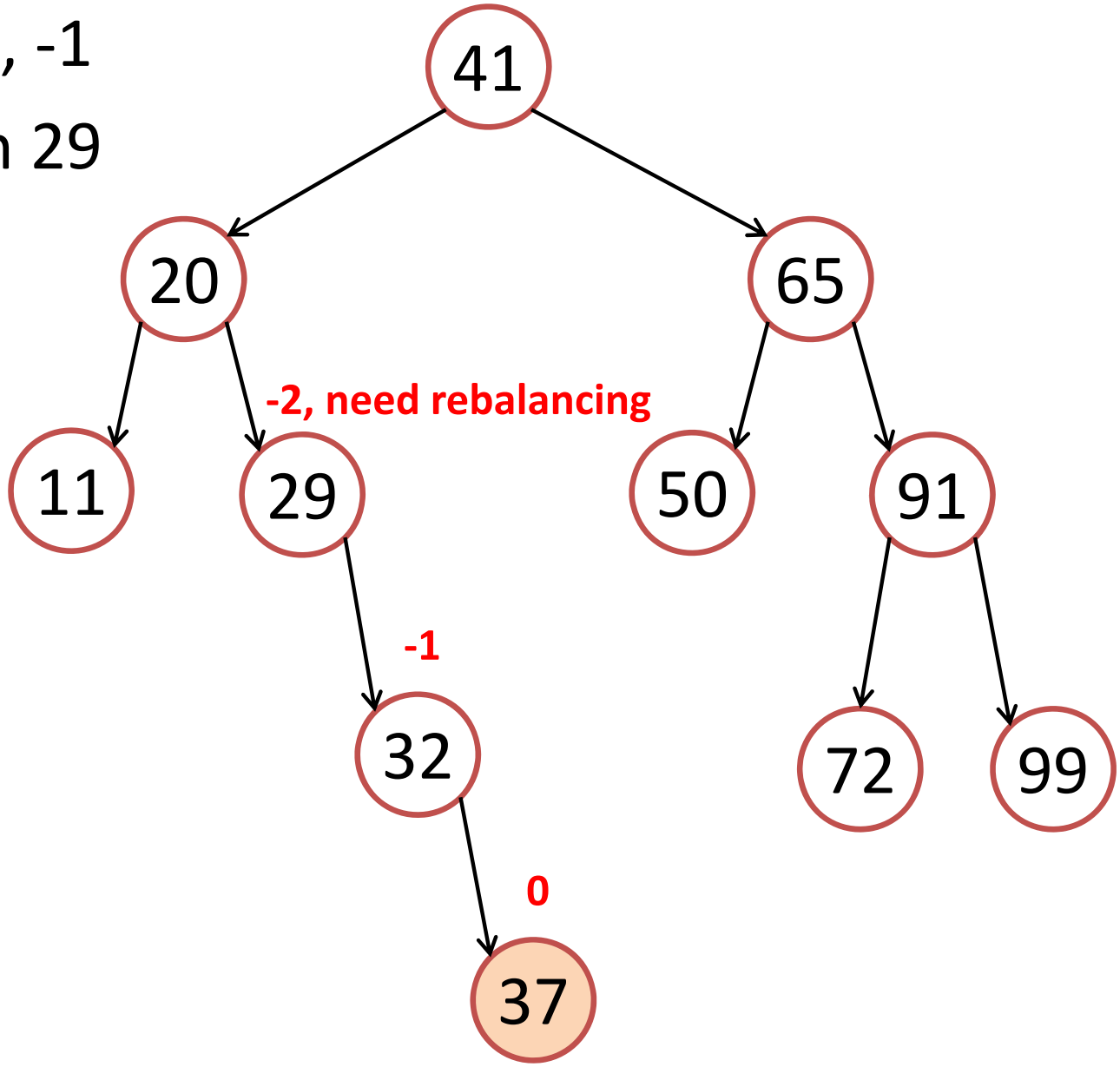


Pictures from Wikipedia

Rebalancing (1)

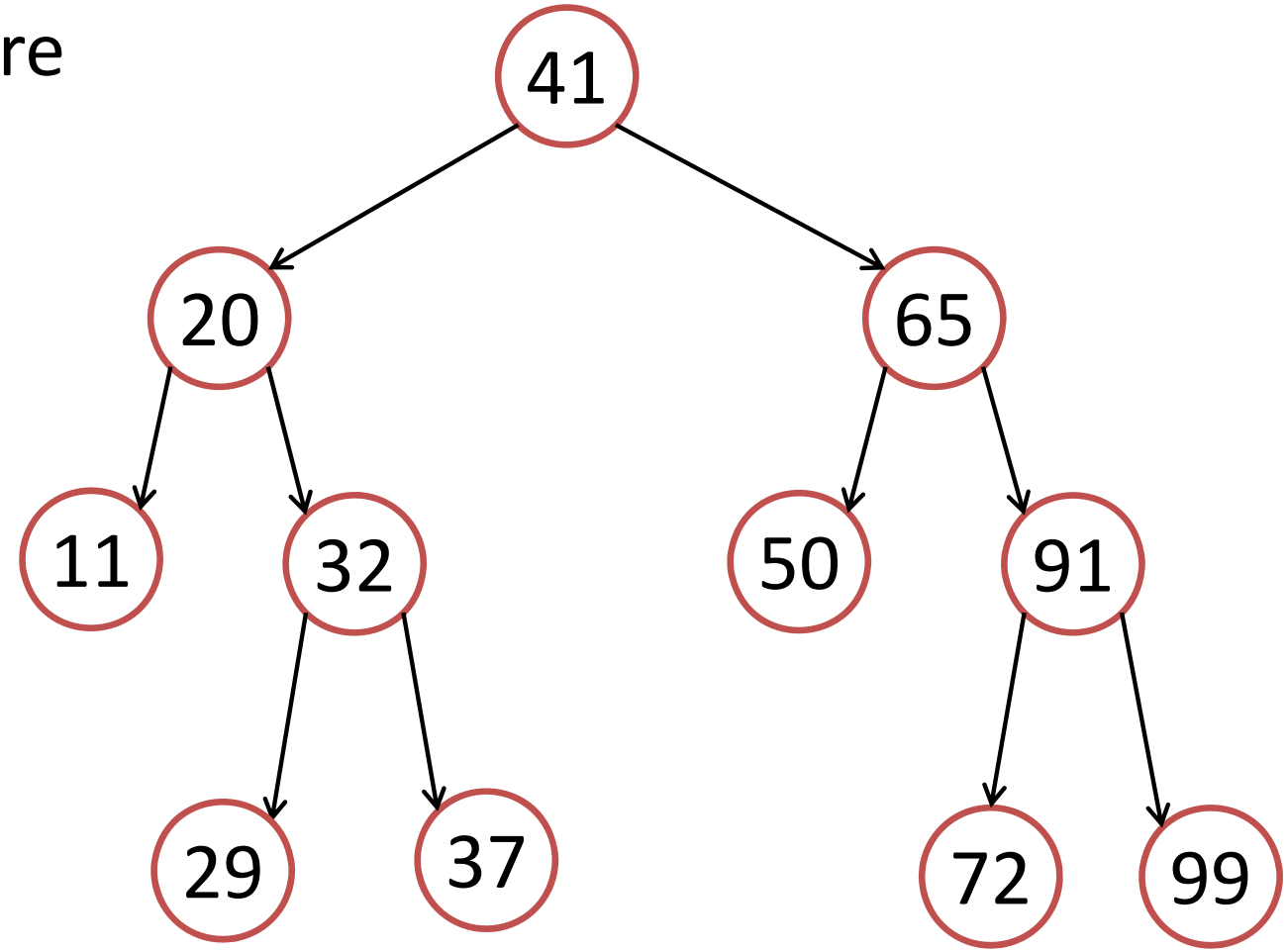
This is a case of -2, -1

Do left rotate on 29



Rebalancing (2)

Now all vertices are balanced again



Insertion to an AVL Tree

Summary:

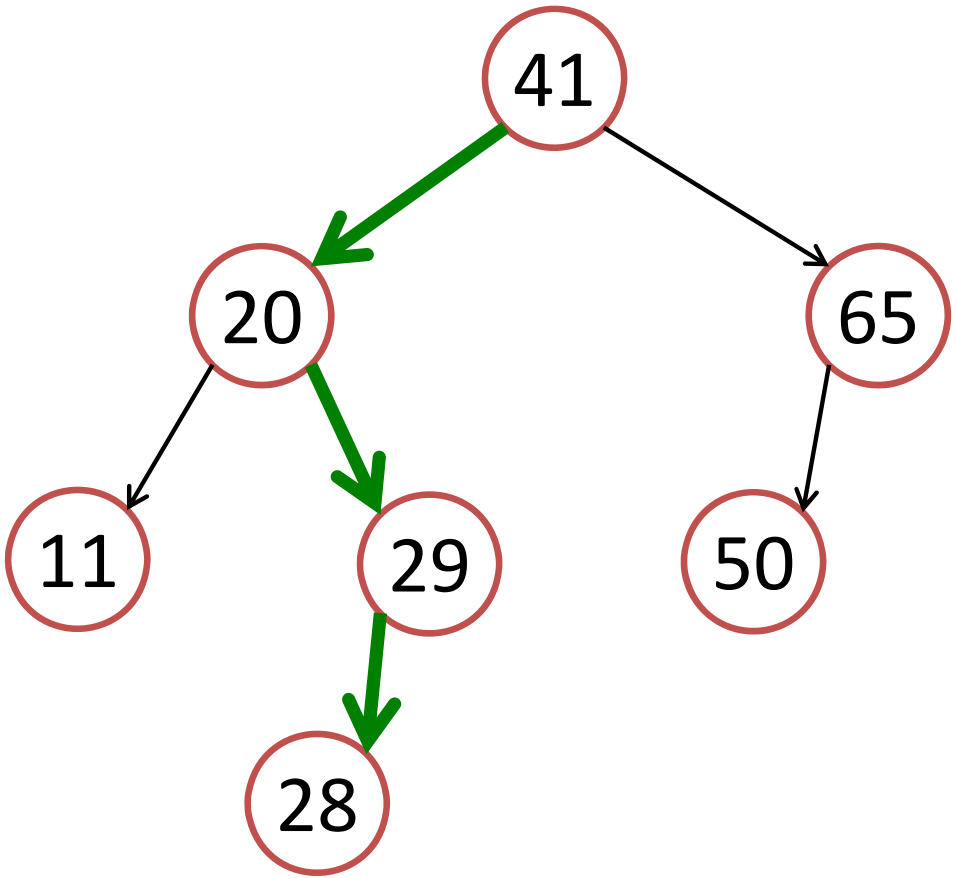
- Just insert the key as in normal BST
- Then, walk up the AVL tree from the insertion point to root:
  - At every step, check for balance factor
  - If a certain vertex is out-of-balance (+2 or -2), use rotations to rebalance
    - Four possible cases

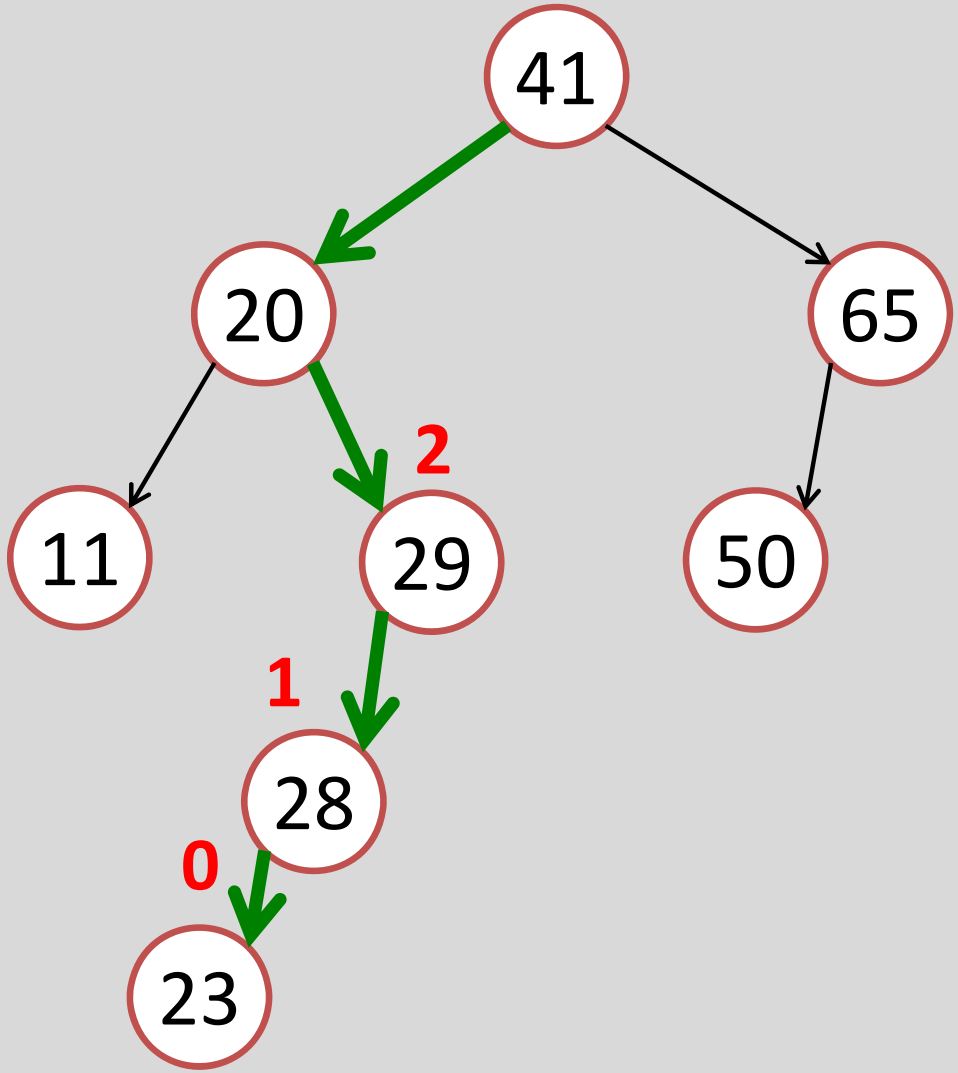
Note: Deletion is a little bit more complicated and we will skip this for CS2010…

i.e. our tests will not involve deletion from a balanced BST

More Examples (1)

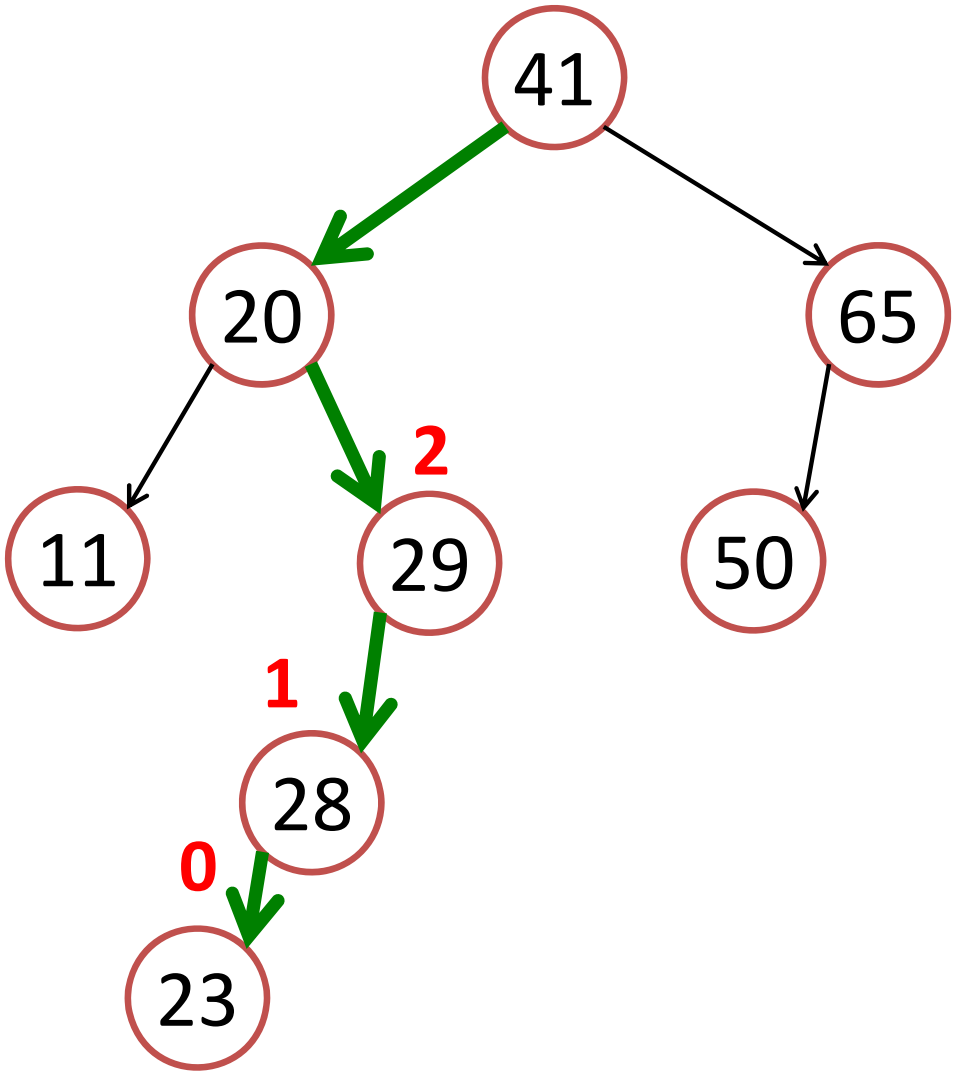
insert(23)

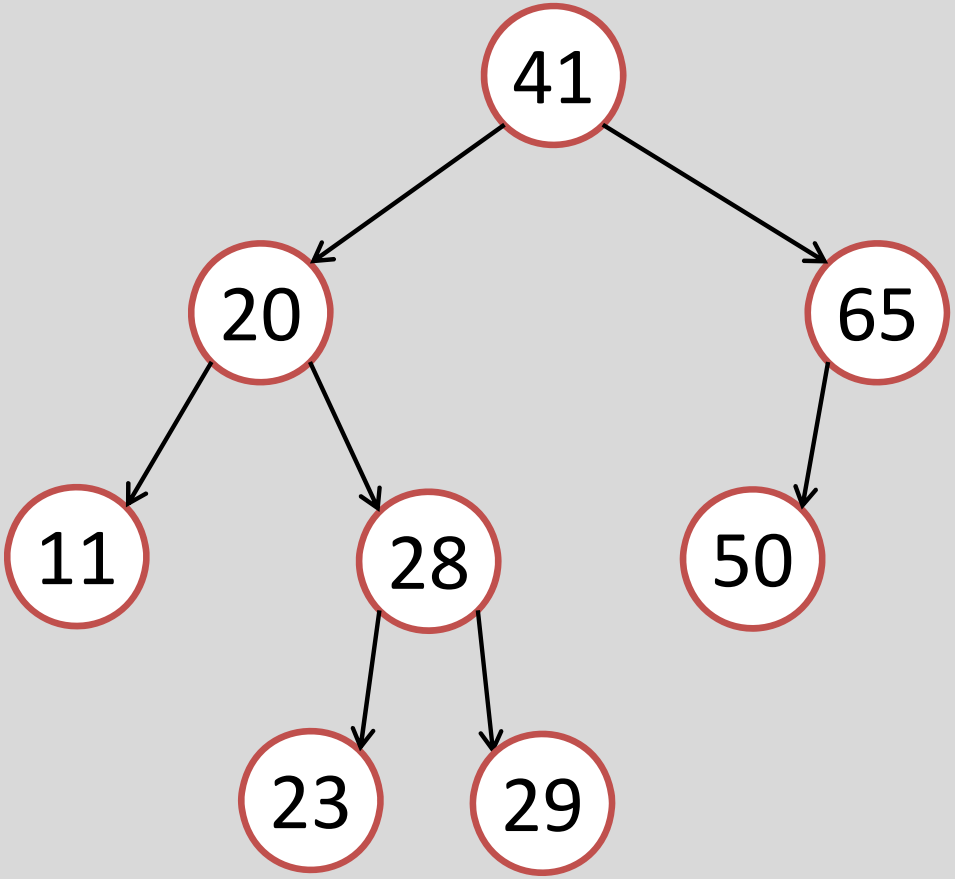




More Examples (2)

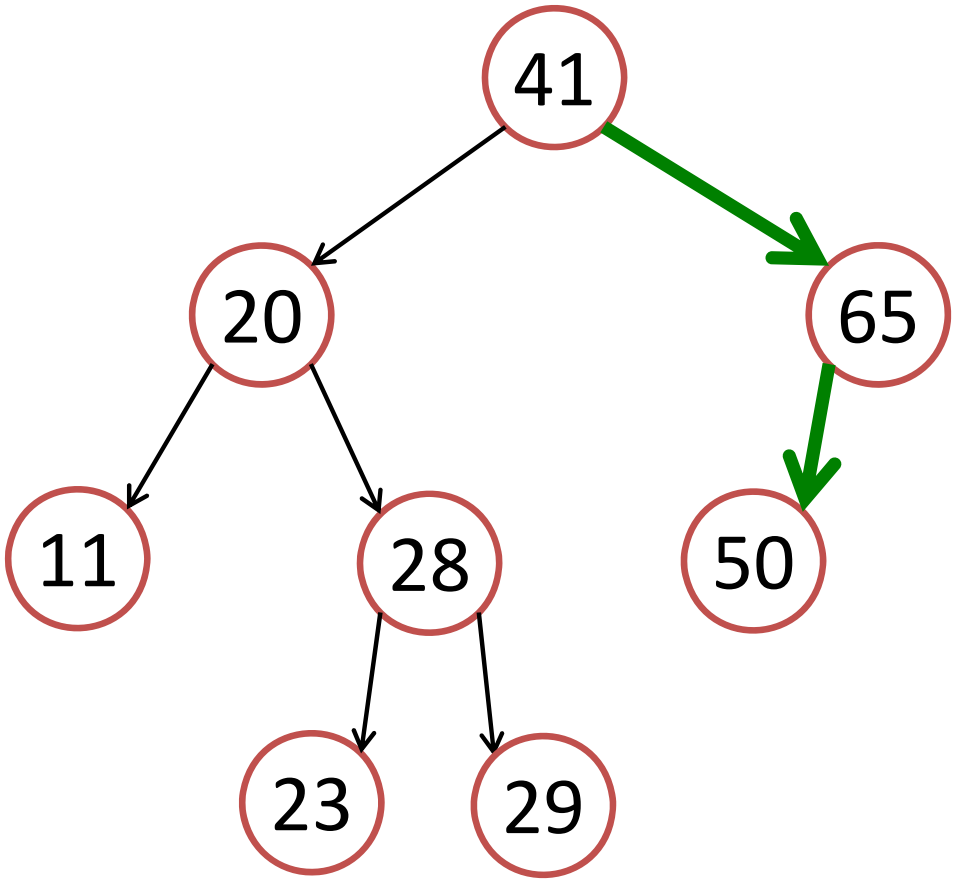
rotateRight(29)

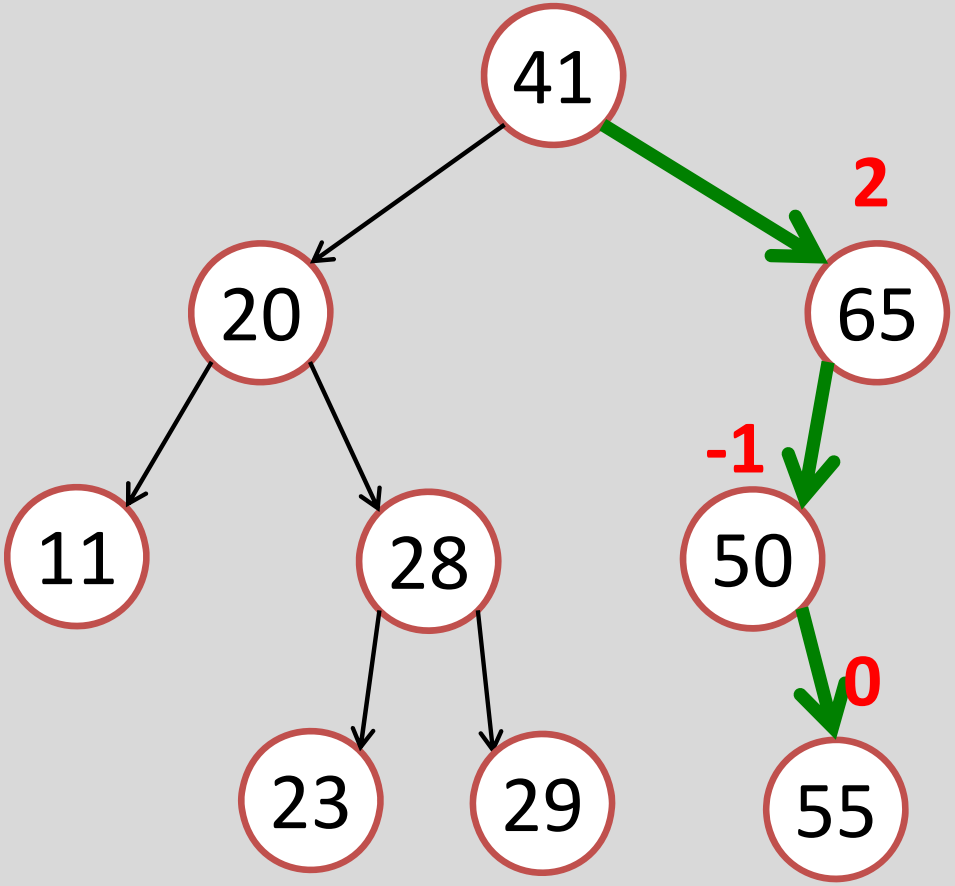




More Examples (3)

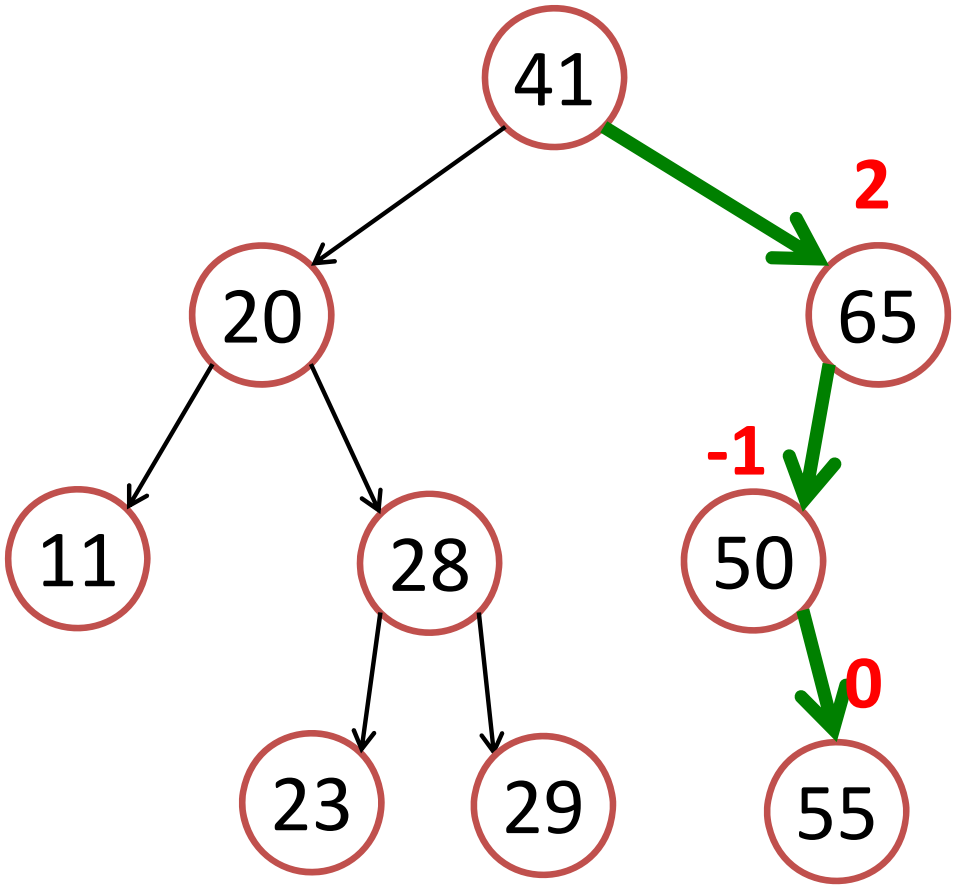
insert(55)

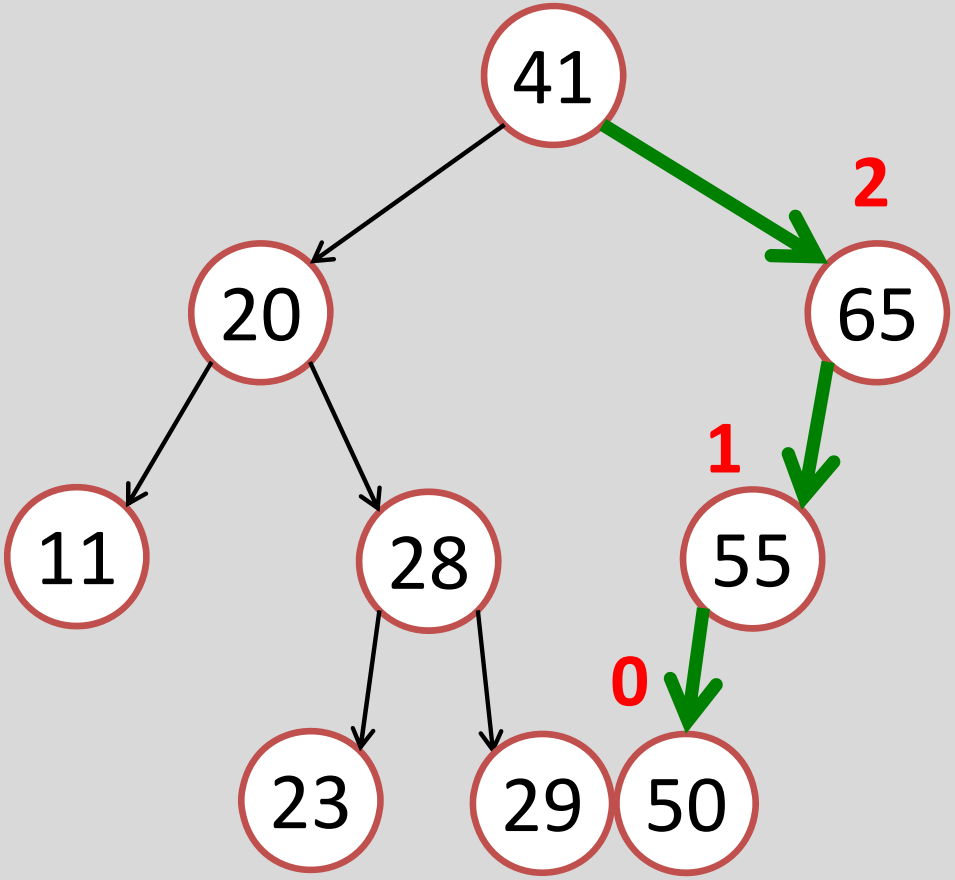




More Examples (4)

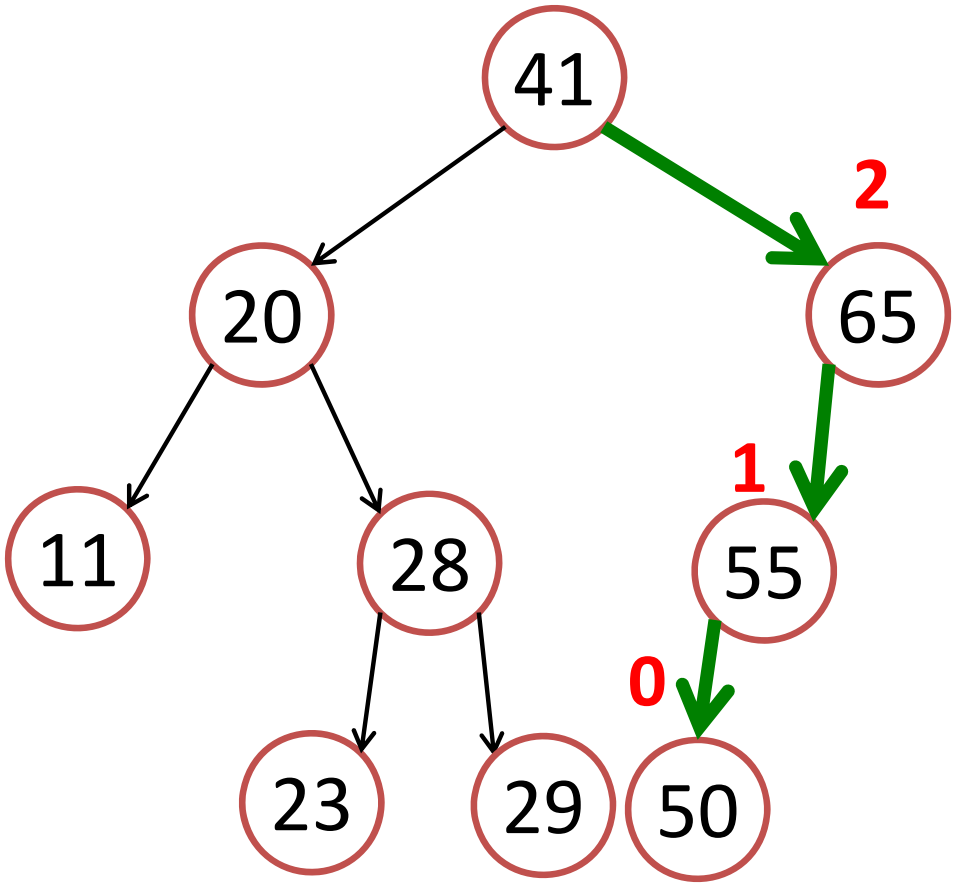
rotateLeft(50)

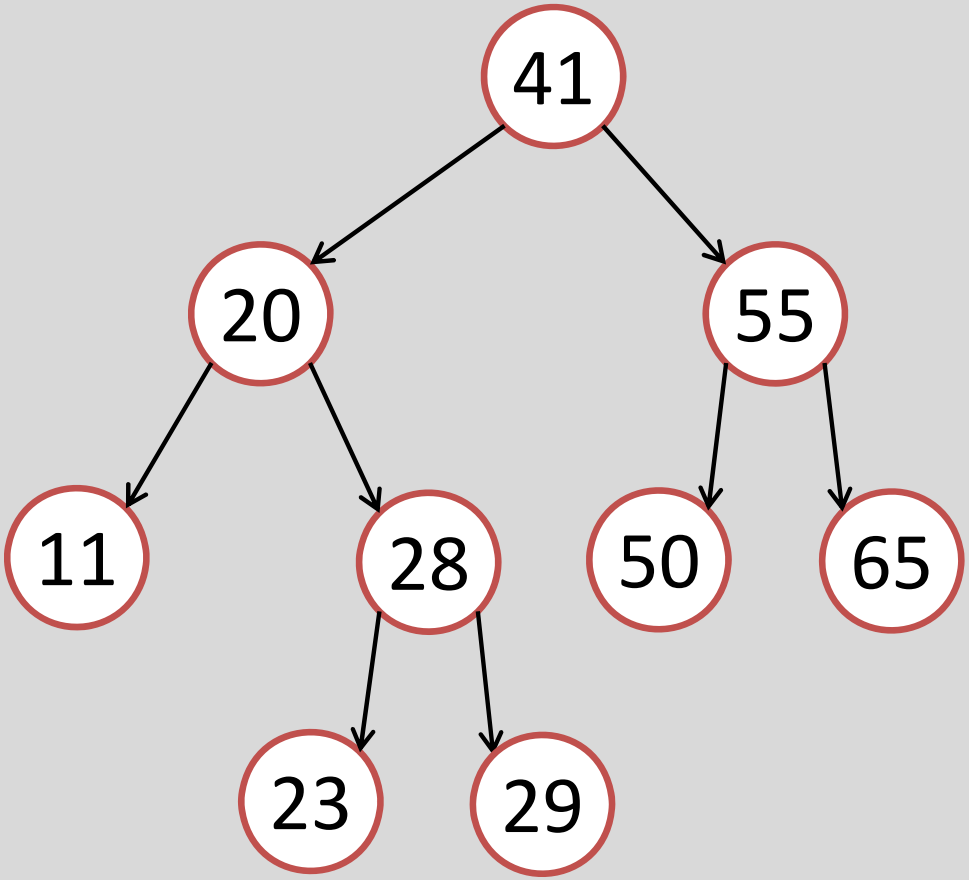




More Examples (5)

rotateRight(65)





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

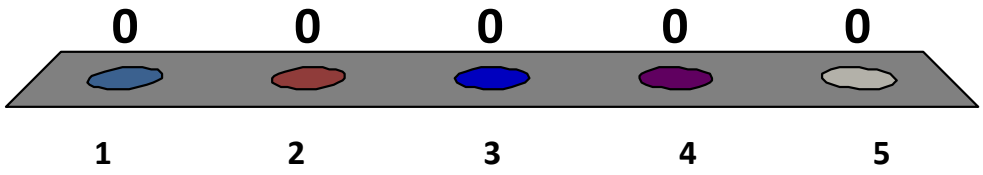
- On access (search or insert),
  move vertex to root (via rotations)
- Height can be linear!
- On average, O(log n) per operation (amortized)

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)?

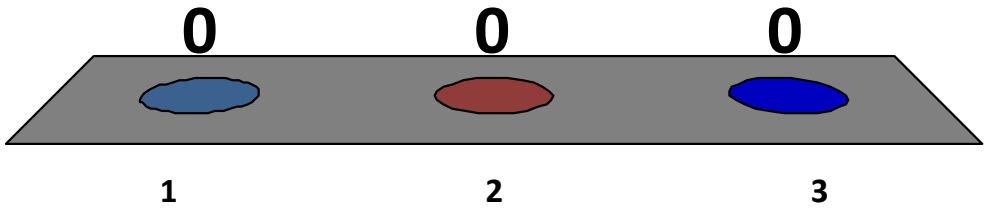
Quick review: A rotation costs:

1. O(1)
2. O(log n)
3. O(n)
4. $O(n^2)$
5. $O(2^n)$



Every insertion requires at least 1 rotation?

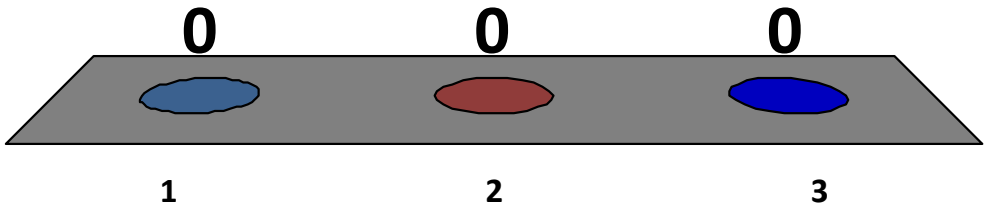
1. Yes
2. No
3. I do not know ☹



0 of 120

A tree is balanced if every vertex's children differ in height by at most 1?

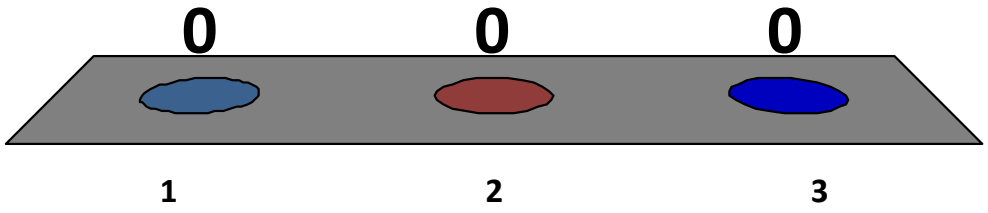
1. Yes
2. No
3. I do not know ☹



0 of 120

A tree is balanced if every vertex either has two children or zero children?

1. Yes
2. No
3. I do not know ☹



0 of 120

Comparison (Continued)

- After we spend two lectures learning balanced BST

Operation	Unsorted Array	Sorted Array	bBST
1. Search(age)	**O(n)**	**O(log n)**	**O(log n)** – search
2. Insert(age)	O(1)	**O(n)**	**O(log n)** – insert
3. Delete(age)	**O(n)**	**O(n)**	**O(log n)** – delete Not shown, but it is log n
4. Find older(age)	**O(n)**	**O(log n)**	**O(log n)** – successor
5. List ages in sorted order	**O(n log n)**	**O(n)**	**O(n)** – inorder traversal
6. Compute median age	**O(n log n)** or **O(n)**	**O(1)**	O(log n) – with a trick Not shown, but it is log n
7. Your PS1	**O(slow)**	**O(slow too)**	**O(faster)**

  - We now have an efficient data structure ☺

The Implementation

Let’s look at AVLDemo.java, AVL.java

These codes are not given, they will be used in PS1

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 Week04

Balanced BST

Summary:

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

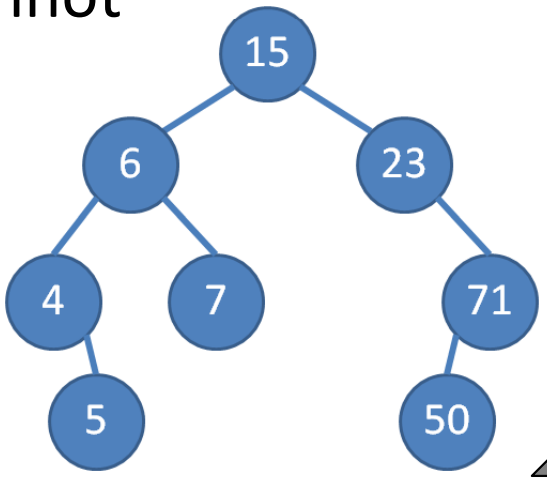
Next Lecture:

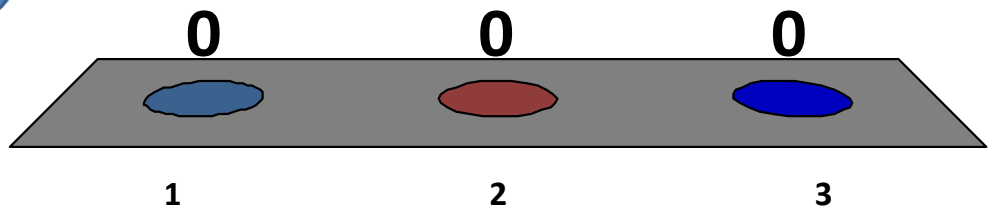
- Heaps
- Priority Queues

Teaser: Quiz 1 last year: Now that you have seen “inorder”, “findMin”, and “successor”, can we implement “inorder” as “findMin” one time, and then repeatedly call “successor” until we run out of successor?

1. Yes, we can, and the time complexity will be the same, O(___)

2. Yes, we can, but the time complexity increases to O(____)
3. No, we cannot





0 of 120

Help Session

- Current plan so far:
  - Saturday, 8 September 2012, 12.30-2pm
    Topic: BST + Balanced BST + Heap
  - Venue: NUS Business canteen
  - Who can attend: Preferably those who are struggling with this module so far