AVL Trees 1

AVL Tree Structure

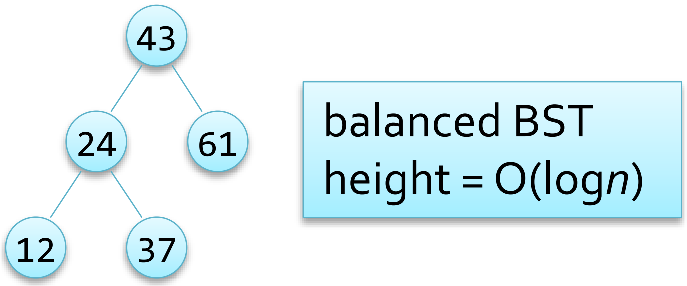
Objectives

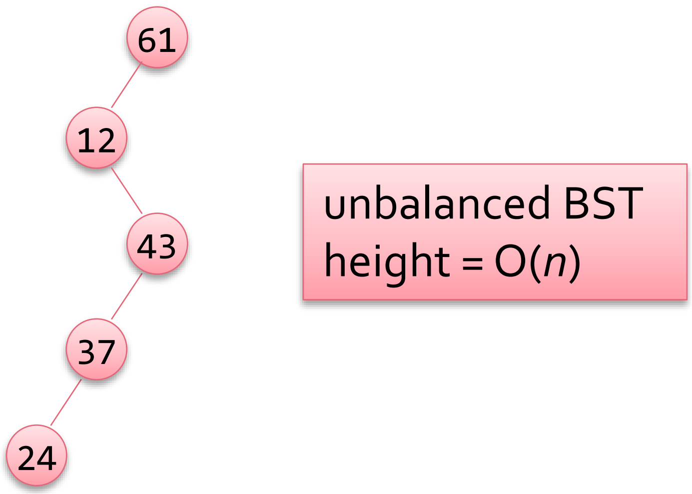
- Describe types of balanced BSTs
- Describe AVL trees
- Show that AVL trees are O(log *n*) height
- Describe and implement rotations
- Implement AVL tree insertion
- Implement AVL tree removal

AVL material with thanks to Brad Bart

Binary Search Trees – Performance

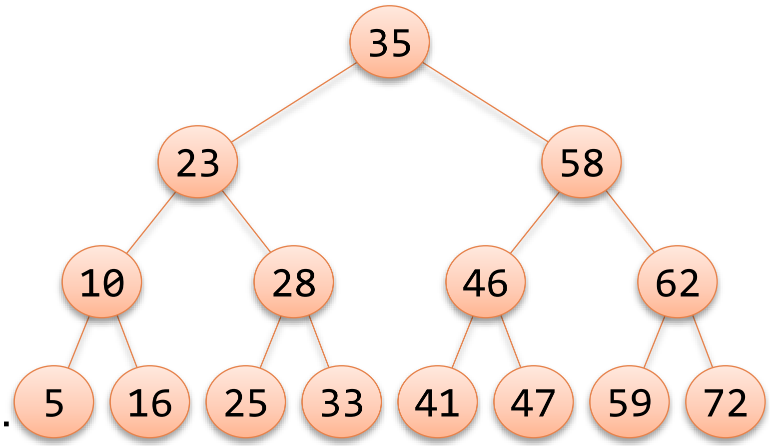
- Insertion and removal from BSTs is O(*height*)
- What is the height of a BST?
  - If the tree is perfect or complete: O(log*n*)
    - Or *balanced*
  - If the tree is very unbalanced: O(*n*)



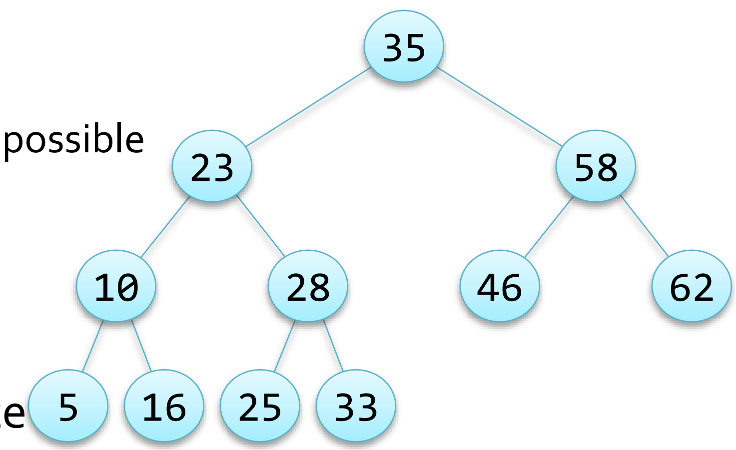


Types of Balanced Binary Trees

- Perfect trees
  - Full trees with all leaves at the same level
  - How many leaves in a tree of height *h*?
    - Level *i* has 2^i nodes
    - $n = 2^{h+1} - 1$
  - Ideal, but only works for n = 0, 1, 3, 7, 15, …



- Complete trees
  - Every level except the lowest is full
    - Leaves on lowest level are as far to the left as possible
  - How many nodes in a tree of height *h*?
    - From 2^h up to $2^{h+1} - 1$



- But
  - Unlikely that trees are perfect or complete

Balanced Trees

- Self balancing trees
  - Create invariants to guarantee a minimum tree density
    - That results in a height of O(log *n*)
  - On insert, if the tree is imbalanced - re-balance it
  - Either the structure or algorithms (or both) need to be more complex than BST insert and remove
- Splay trees
  - Operations on node *x* moves *x* to the root
  - Adjust the tree using rotations
  - Takes advantage of the *locality principle*

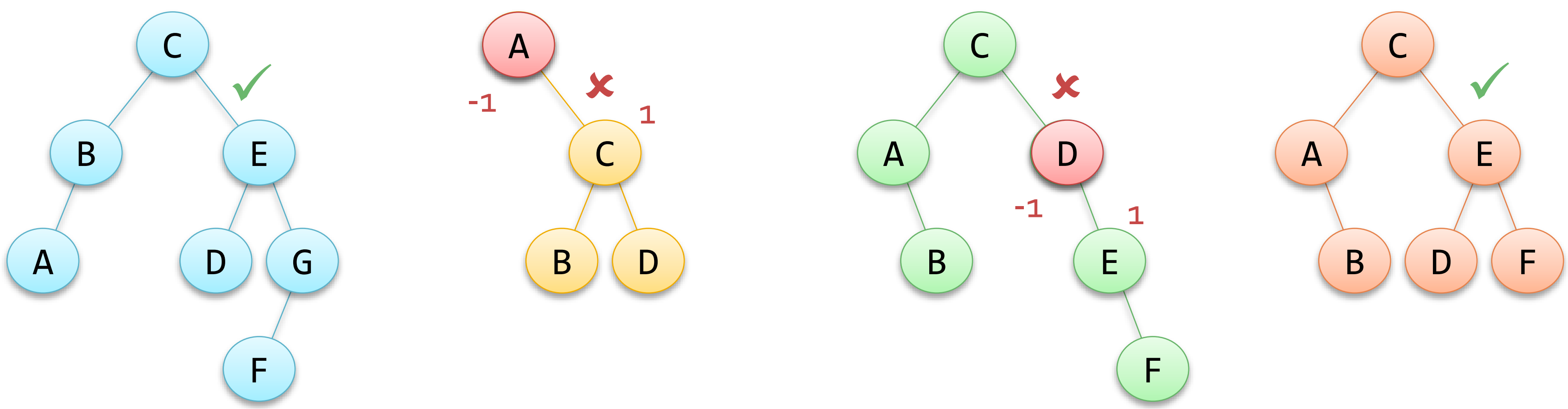
Drawing by CMU Professor Jorge Stolfi

Balanced Trees

- Height balanced trees
  - AVL trees
    - For every node, the heights of the left and right subtrees differ by at most one
  - Uses rotations
- Depth balanced trees
  - Red-Black trees
    - The depths of any two leaves differ by a factor of two or less
    - Or the height of the longest path from the root to a leaf is at most twice the shortest path from the root to a leaf
  - Also uses rotations
- All leaves at the same level
  - B-trees – branching factor is great than 2, i.e. not *binary* trees

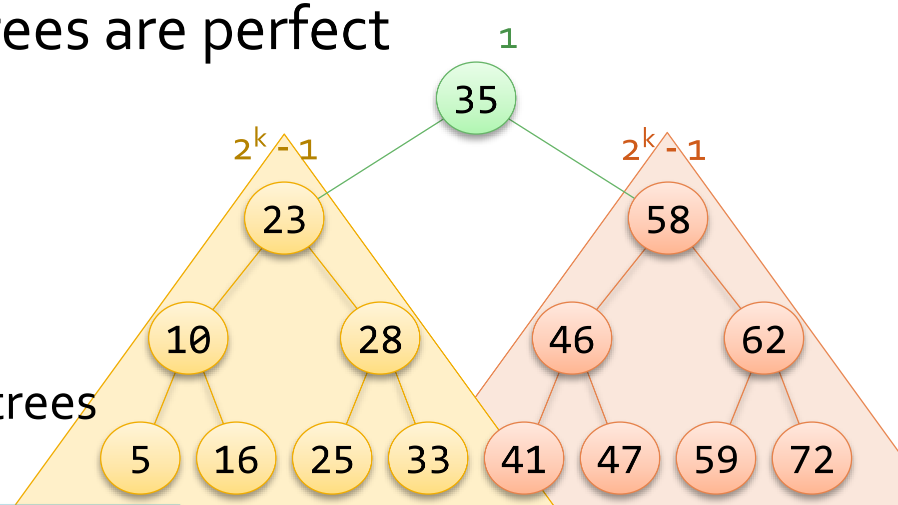
AVL Trees

- Invented by **A**delson-**V**elsky and **L**andis
- Height invariant
  - The heights of the left and right subtrees of each node differ by at most one
- According to this invariant are these trees balanced?



Minimum Tree Density

- Goal: $h = O(\log n)$
- We need: $h \le \log_a n$, i.e., $n \ge a^h$ for some $a > 1$
- Claim: a *perfect* binary tree has $n(h) \ge 2^{h+1}-1$ nodes
- Proof (by induction on h)
  - L and R subtrees of perfect trees are perfect
  - Base case
    - Empty tree ($h = -1$) has 0 nodes
  - Inductive case
    - Tree of height k has L and R subtrees of height $k - 1$

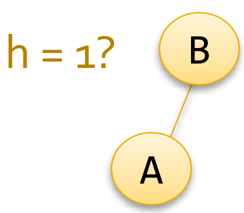


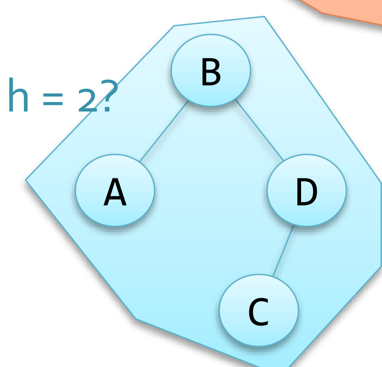
$n(k) \ge 2 * 2^k - 1$

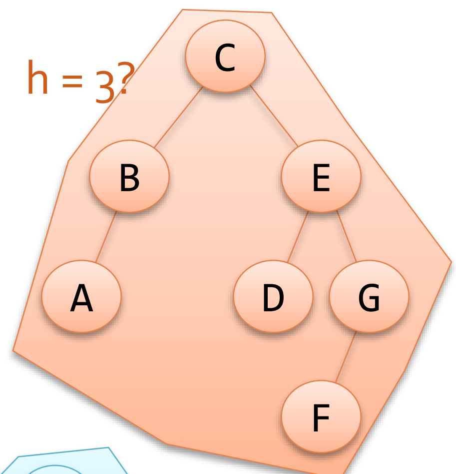
$= 2^{k+1} - 1$

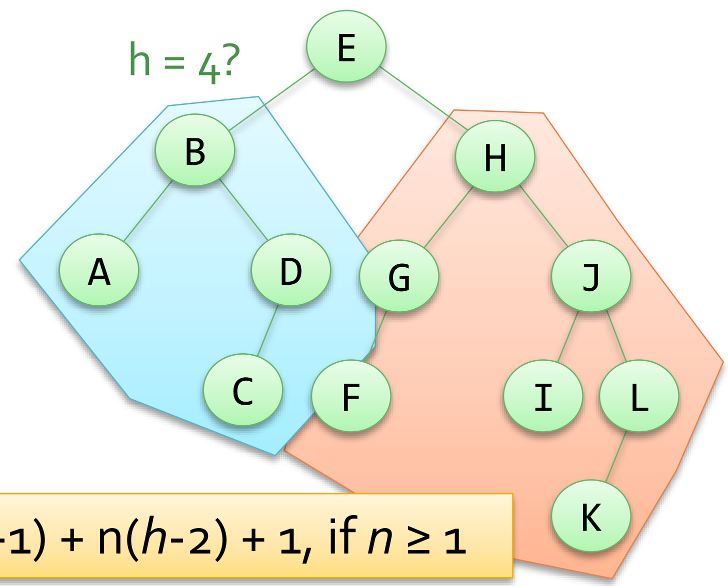
Smallest AVL Trees

- What are the smallest five AVL trees by height
  - The trees with the fewest nodes for their height









$n(h) \geq n(h-1) + n(h-2) + 1$, if $n \geq 1$

right subtree left subtree root

Minimum AVL Tree Density

- Let $n(h)$ represent the number of nodes in an AVL tree of height h
  - What is the minimum value of $n(h)$?
- $n(h) \geq n(h-1) + n(h-2) + 1$, for all $n \geq 1$
  - $n(0) = 1$, $n(-1) = 0$
- This pattern should look familiar
  - It's the Fibonacci sequence (+1)
- Claim
  - $n(k) \geq F_{h+3} - 1$

h	min *n*(h)	
-1	0	1 (F_2)
0	1	2 (F_3)
1	2	3 (F_4)
2	4	5 (F_5)
3	7	8 (F_6)
4	12	13 (F_7)

Minimum AVL Tree Density

- Claim: An AVL tree holds at least $n(h) \geq F_{h+3} - 1$
- Proof by induction on h
- Strategy: use the recursive definition
  - Base case?
    - Both $h = -1$ and $h = 0$ satisfy the claim
  - Inductive case? Consider an AVL tree of height $k \geq 1$
    - $n(k) \geq n(k-1) + n(k-2) + 1$
    - $\geq (F_{k+2}-1) + (F_{k+1}-1) + 1$
    - $= (F_{k+2} + F_{k+1}) - 1$
    - $= F_{k+3}-1$

Note that $F_{h+3} \approx \phi^{h+3} / \sqrt{5}$ and $n \geq F_{h+3} - 1$

$n(h) \geq n(h-1) + n(h-2) + 1$

$n(-1) = 0$, $n(0) = 1$

ϕ (phi) is the Golden Ratio

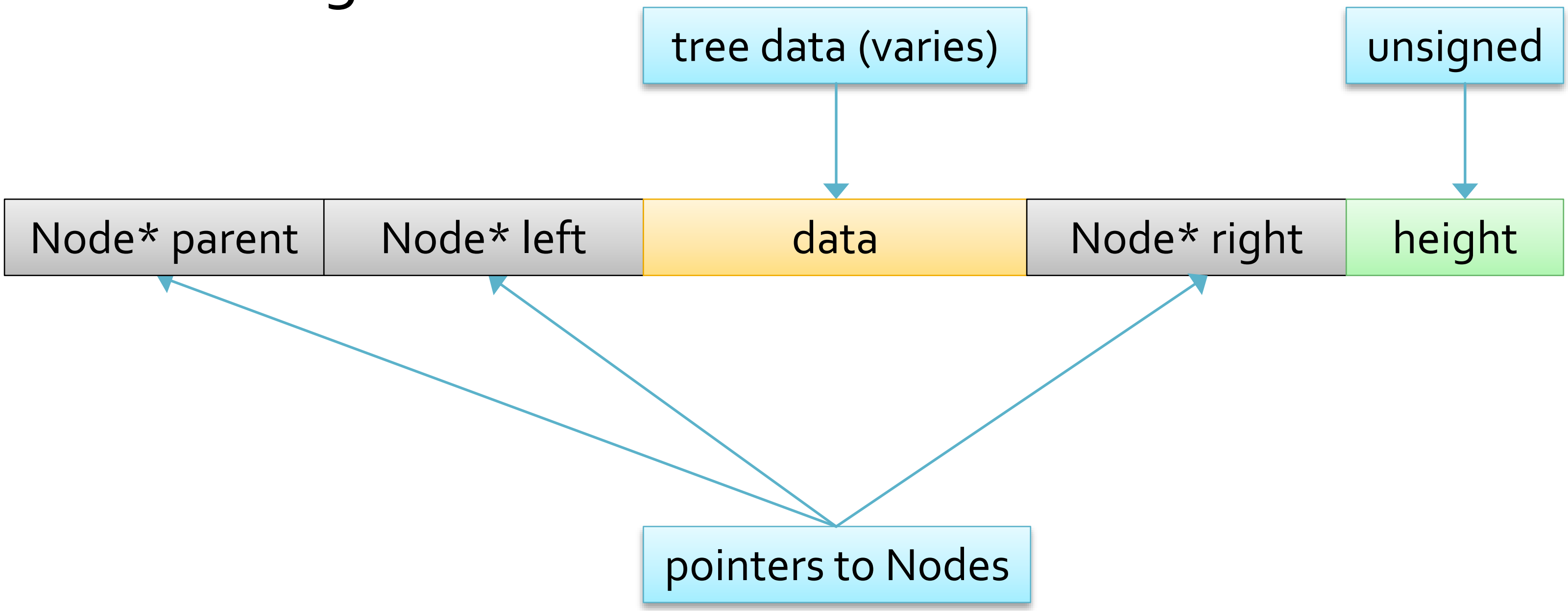
$\phi = (1+ \sqrt{5}) / 2$

$\phi = 1.6180339887$

F_n grows exponentially
$F_n \approx \phi^n / \sqrt{5}$
$\Rightarrow n \geq \phi^{h+3} / \sqrt{5} - 1$
$\Rightarrow h \leq c \times \log_{\phi} n$
$h = O(\log n)$

AVL Tree Nodes

- AVL trees are reference structures made up of nodes and pointer to nodes
- Nodes contain data, three pointers to nodes, and the node's height

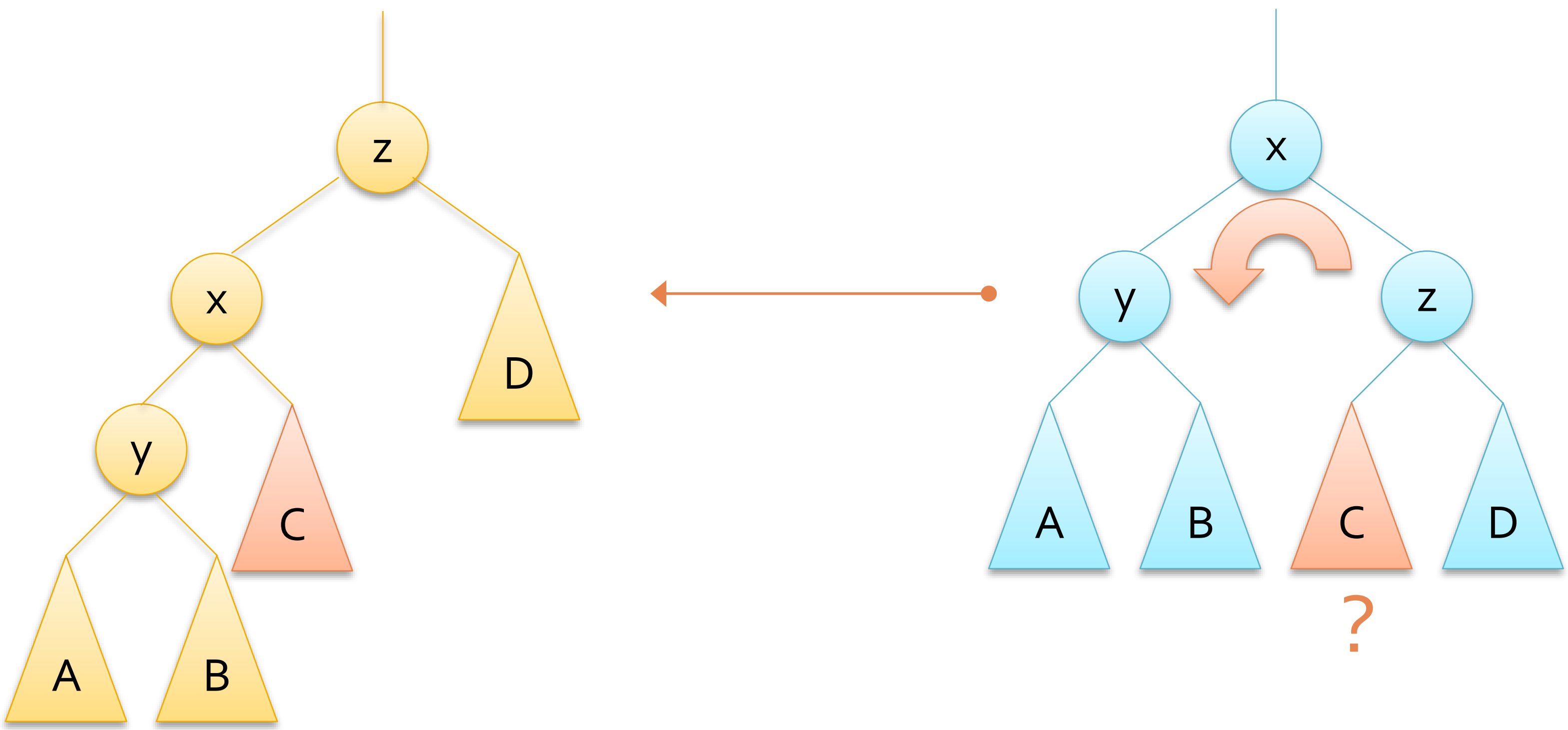


Rotations

- An item must be inserted into an AVL tree into the position given by the BST insert algorithm
- The shape of a tree is determined by
  - The values of the items inserted into the tree
  - The order in which those values are inserted
- This suggests that there is more than one tree (shape) that can contain the same values
- A tree's shape can be altered by *rotation* while still preserving the *bst* property

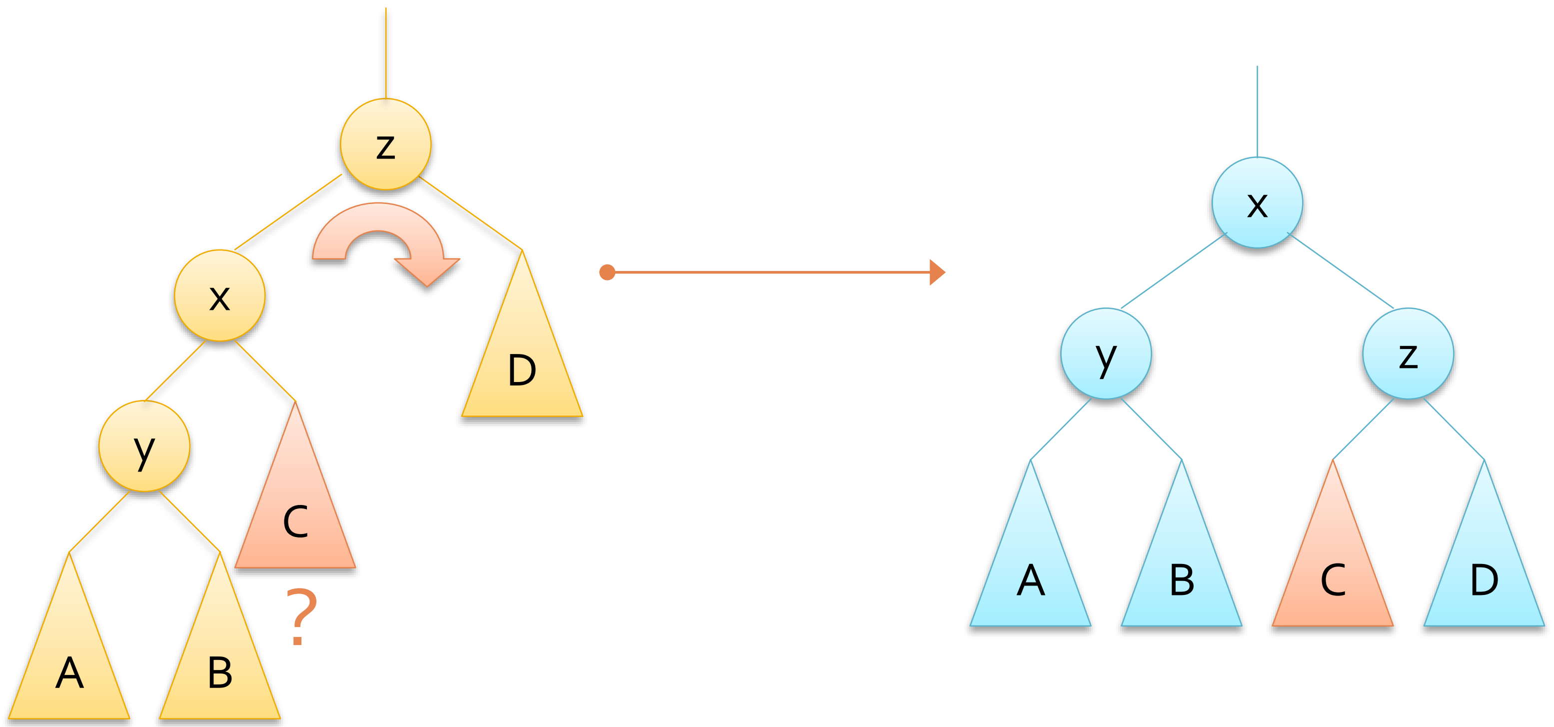
Left Rotation

Left rotate (x)



Right Rotation

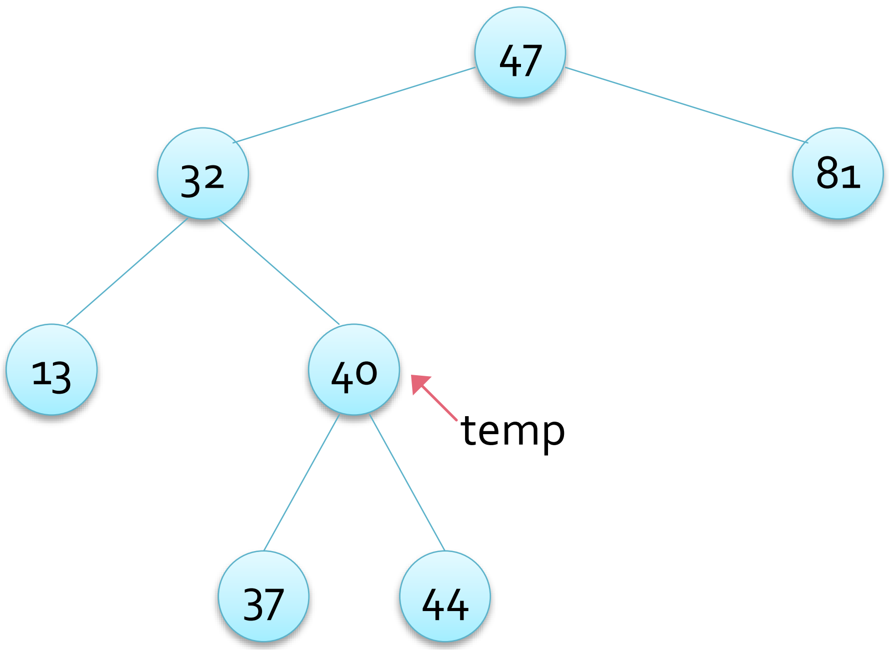
Right rotate (z)



Left Rotation Example

Left rotation of 32 (referred to as x)

Create a pointer to *x*'s right child



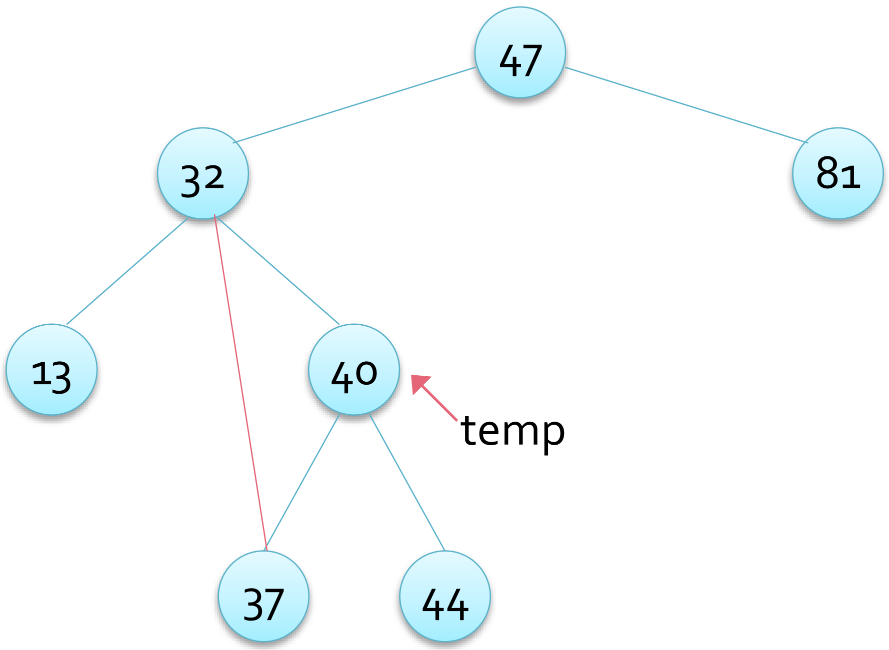
Left Rotation Example

Left rotation of 32 (referred to as x)

Create a pointer to *x*'s right child

Make *temp*'s left child, *x*'s right child

Detach *temp*'s left child



Left Rotation Example

Left rotation of 32 (referred to as x)

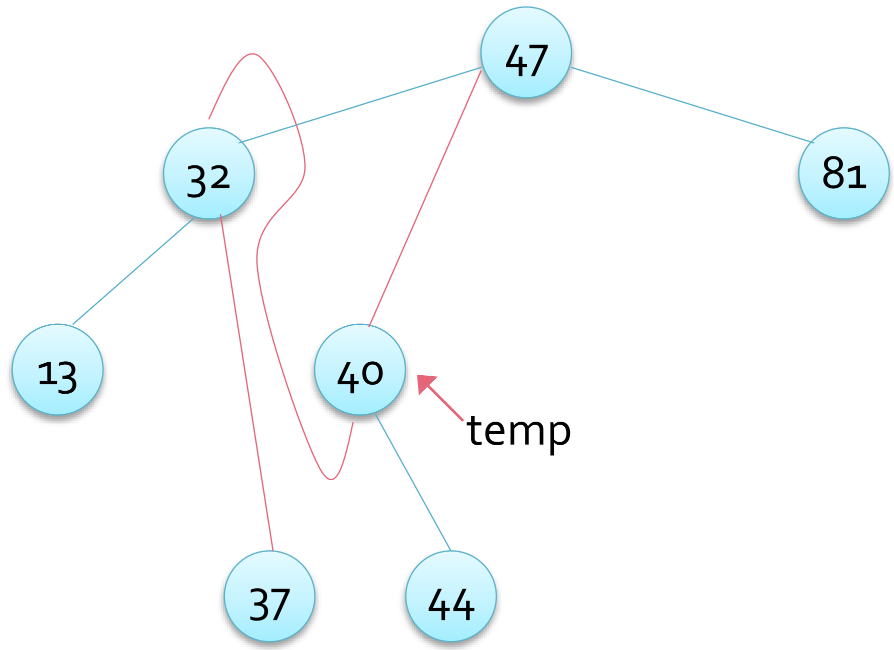
Create a pointer to *x*'s right child

Make *temp*'s left child, *x*'s right child

Detach *temp*'s left child

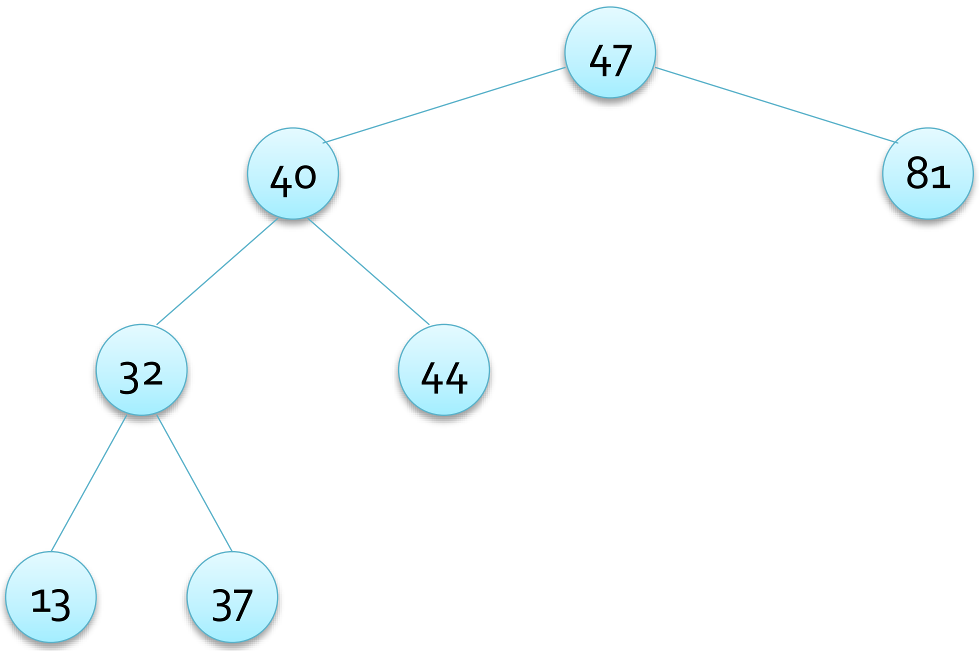
Make *x* the left child of *temp*

Make *temp* the child of *x*'s parent



Left Rotation Example

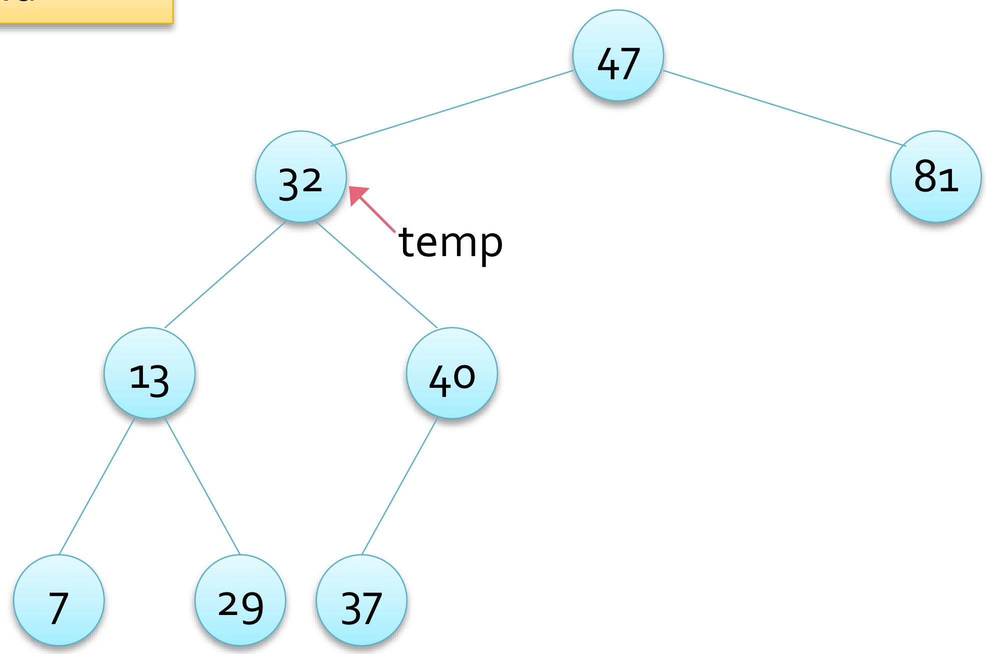
Left rotation of 32 (complete)



Right Rotation Example

Right rotation of 47 (referred to as x)

Create a pointer to *x*'s left child



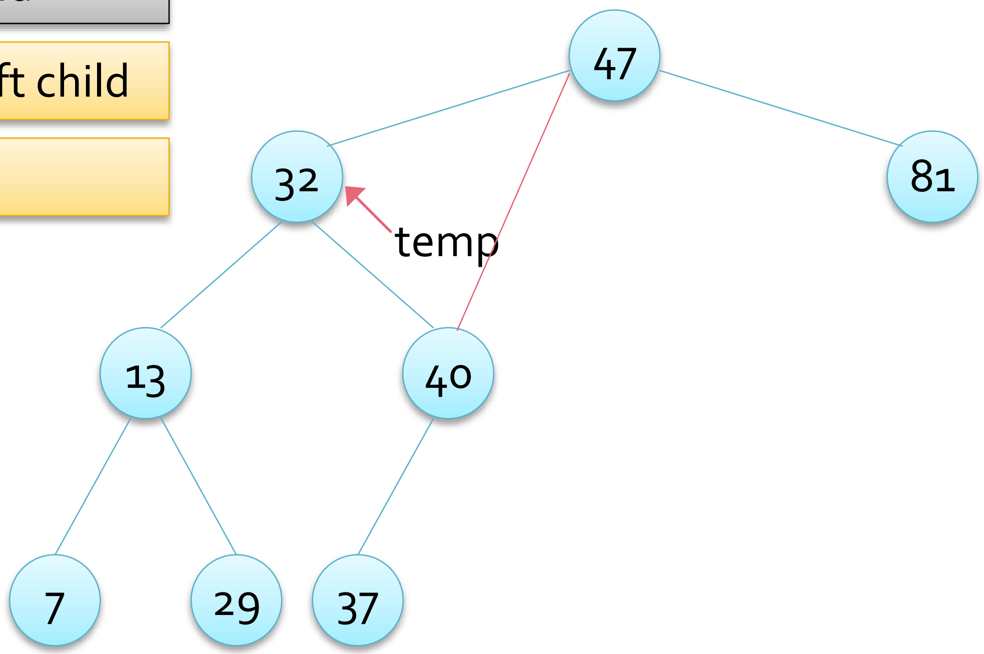
Right Rotation Example

Right rotation of 47 (referred to as x)

Create a pointer to *x*'s left child

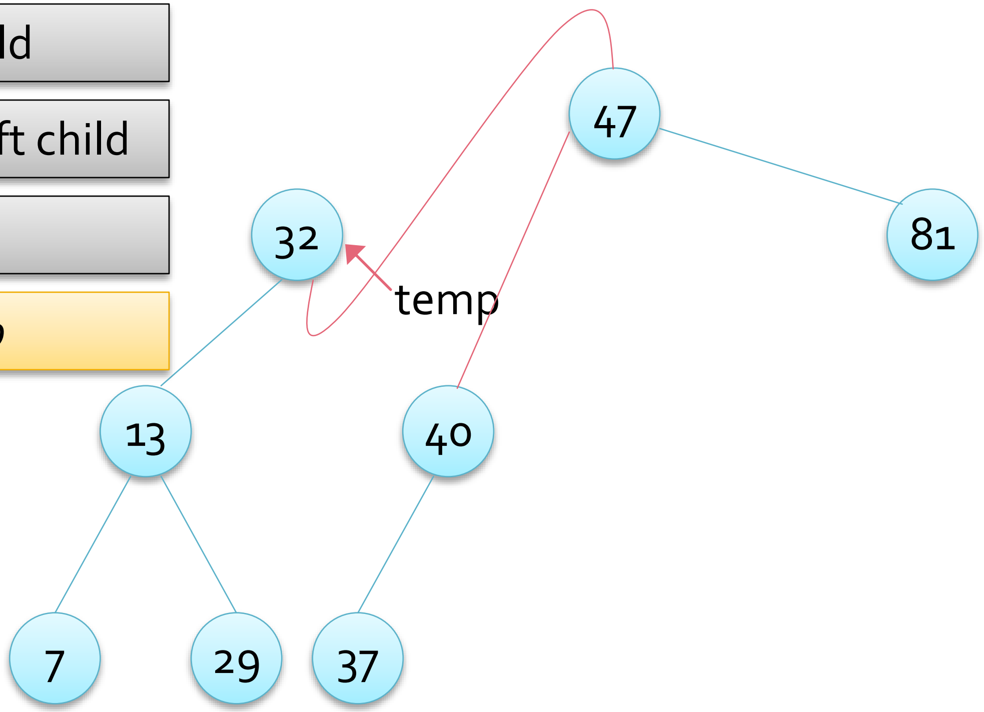
Make *temp*'s right child, *x*'s left child

Detach *temp*'s right child



Right Rotation Example

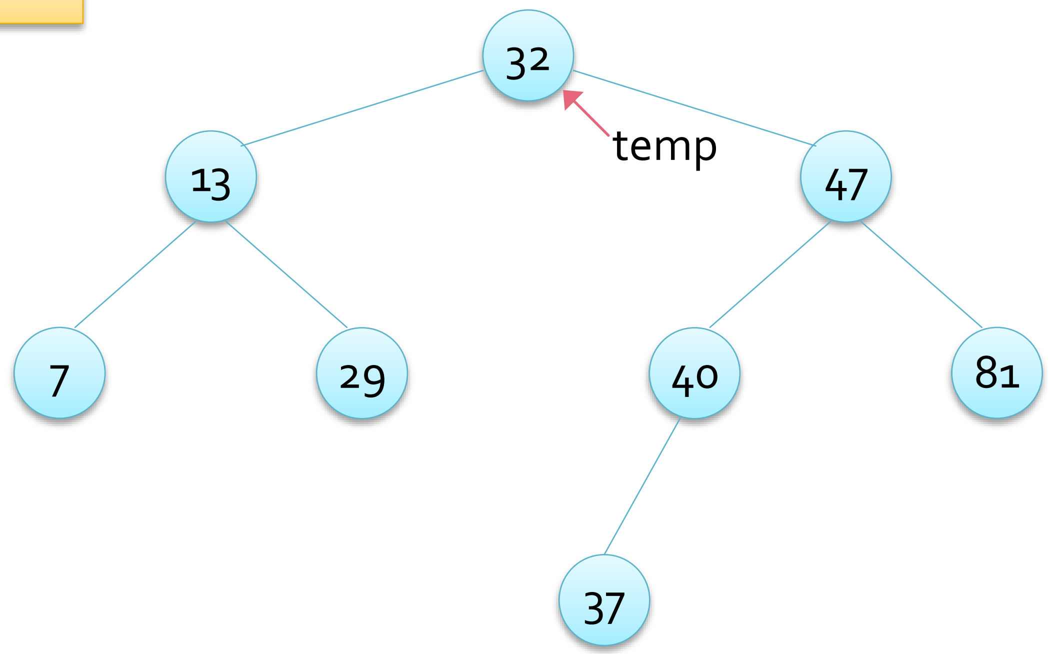
- Right rotation of 47 (referred to as x)
- Create a pointer to *x*'s left child
- Make *temp*'s right child, *x*'s left child
- Detach *temp*'s right child
- Make *x* the right child of *temp*



Right Rotation Example

Right rotation of 47

Make temp the new root



Left Rotation Code

Notation

.left is left child, *.right* is right child, *.p* is parent

```
leftRotate(x) // x is the node to be rotated
    y = x.right
    x.right = y.left
    // Set nodes' parent references
    // y's left child
    if (y.left != null)
            y.left.p = x
    // y
    y.p = x.p

    // Set child reference of x's parent
    if (x.p == null) //x was root
        root = y
    else if (x == x.p.left) //left child
        x.p.left = y
    else
        x.p.right = y
    // Make x y's left child
    y.left = x
    x.p = y
```