

# **FIT2004**

## **Algorithms and Data Structures**

Ian Wern Han Lim  
[lim.wern.han@monash.edu](mailto:lim.wern.han@monash.edu)

Referencing materials by  
Nathan Compane, Aamir Cheema, Arun Konagurthu and Lloyd Allison



# Faculty of Information Technology, Monash University

---

## COMMONWEALTH OF AUSTRALIA

### *Copyright Regulations 1969*

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act. Do not remove this notice

Ready?

# Agenda

- Binary Search Tree (BST)

# Agenda

- Binary Search Tree (BST)
  - AVL Tree

Let us begin...

# Tree

## And Binary Search Tree (BST)

- Our lord and saviour



# Tree

## And Binary Search Tree (BST)

- What is a binary search tree?



- What is a binary search tree?
  - Empty tree is BST

- What is a binary search tree?
  - Empty tree is BST
  - All elements in left subtree  $<$  root
  - All elements in right subtree  $>$  root

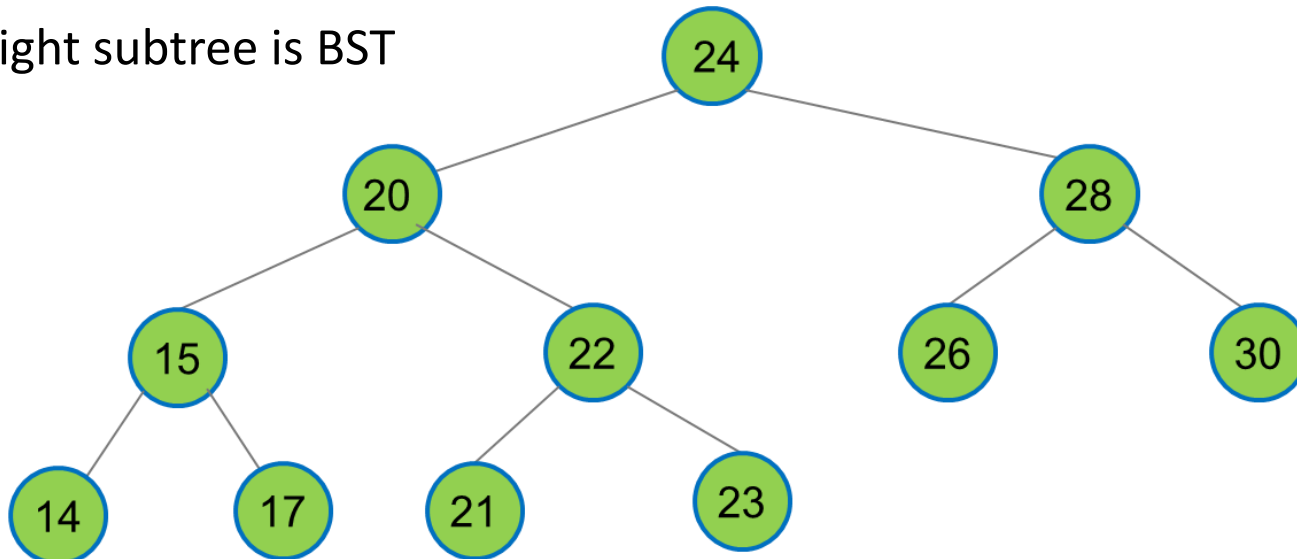
- What is a binary search tree?
  - Empty tree is BST
  - All elements in left subtree  $<$  root
  - All elements in right subtree  $>$  root
  - What about  $=$ ?

- What is a binary search tree?
  - Empty tree is BST
  - All elements in left subtree  $<$  root
  - All elements in right subtree  $>$  root
  - What about  $=$ ? Key are unique!

- What is a binary search tree?
  - Empty tree is BST
  - All elements in left subtree  $<$  root
  - All elements in right subtree  $>$  root
  - What about  $=$ ? Key are unique!
  - Left subtree is BST
  - Right subtree is BST

- What is a binary search tree?
    - Empty tree is BST
    - All elements in left subtree  $<$  root
    - All elements in right subtree  $>$  root
    - What about  $=$ ? Key are unique!
    - Left subtree is BST
    - Right subtree is BST
- } Recursive definition

- What is a binary search tree?
  - Empty tree is BST
  - All elements in left subtree  $<$  root
  - All elements in right subtree  $>$  root
  - What about  $=$ ? Key are unique!
  - Left subtree is BST
  - Right subtree is BST



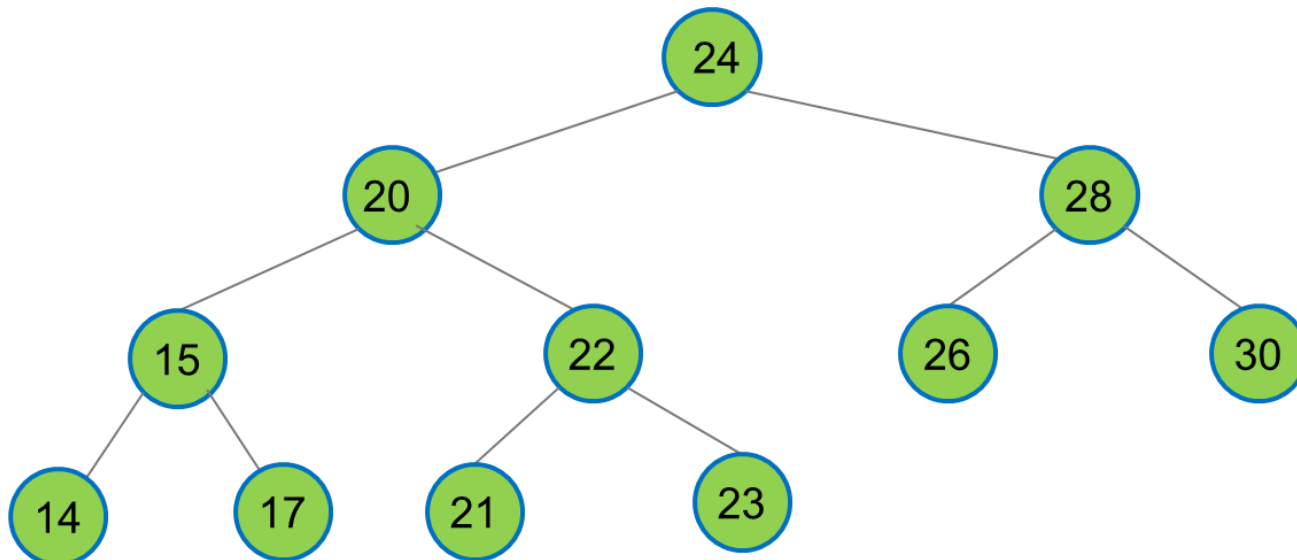
Questions?



# Tree

## And Binary Search Tree (BST)

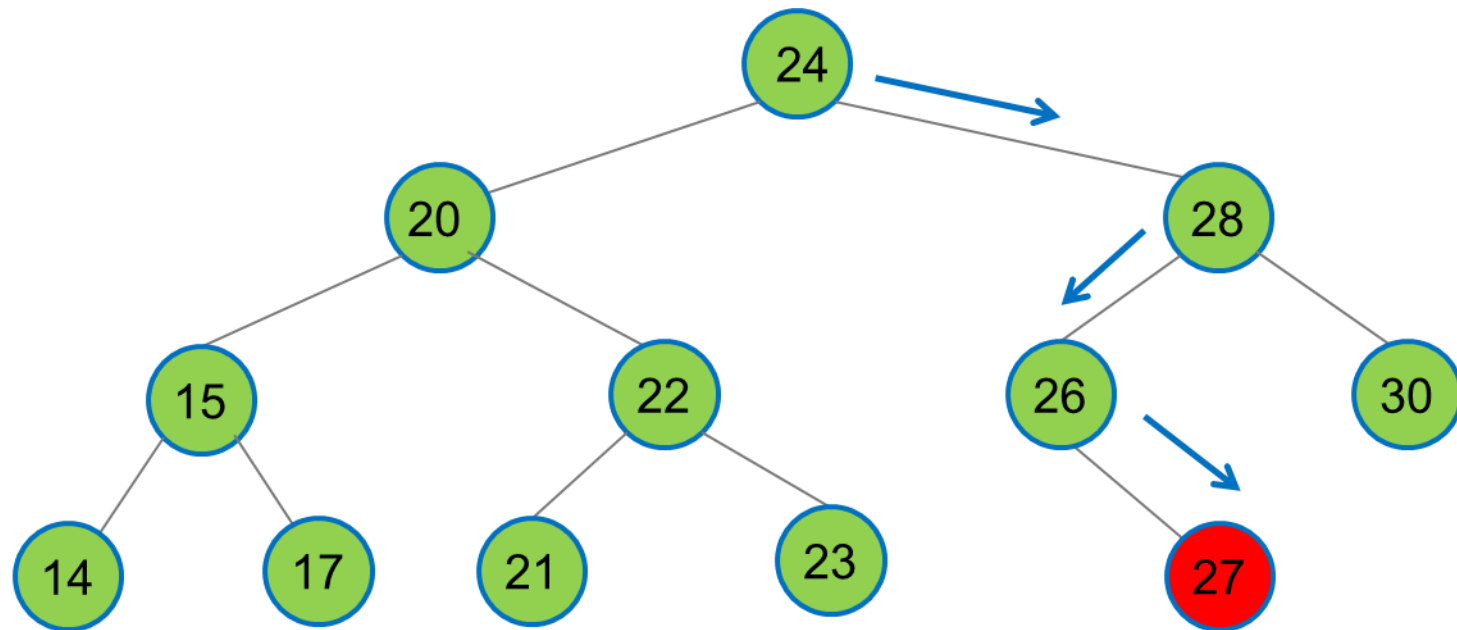
- The main functions?
  - Insert
  - Delete



# Tree

## And Binary Search Tree (BST)

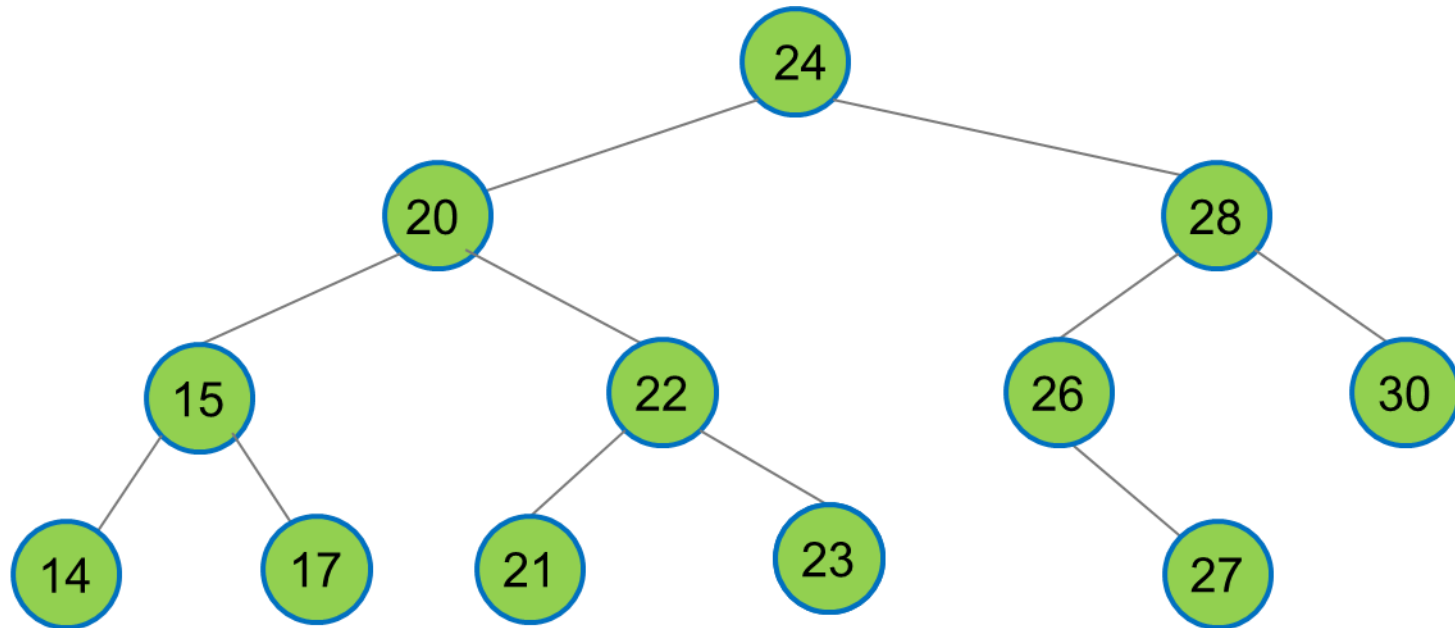
- The main functions?
  - Insert 27



# Tree

## And Binary Search Tree (BST)

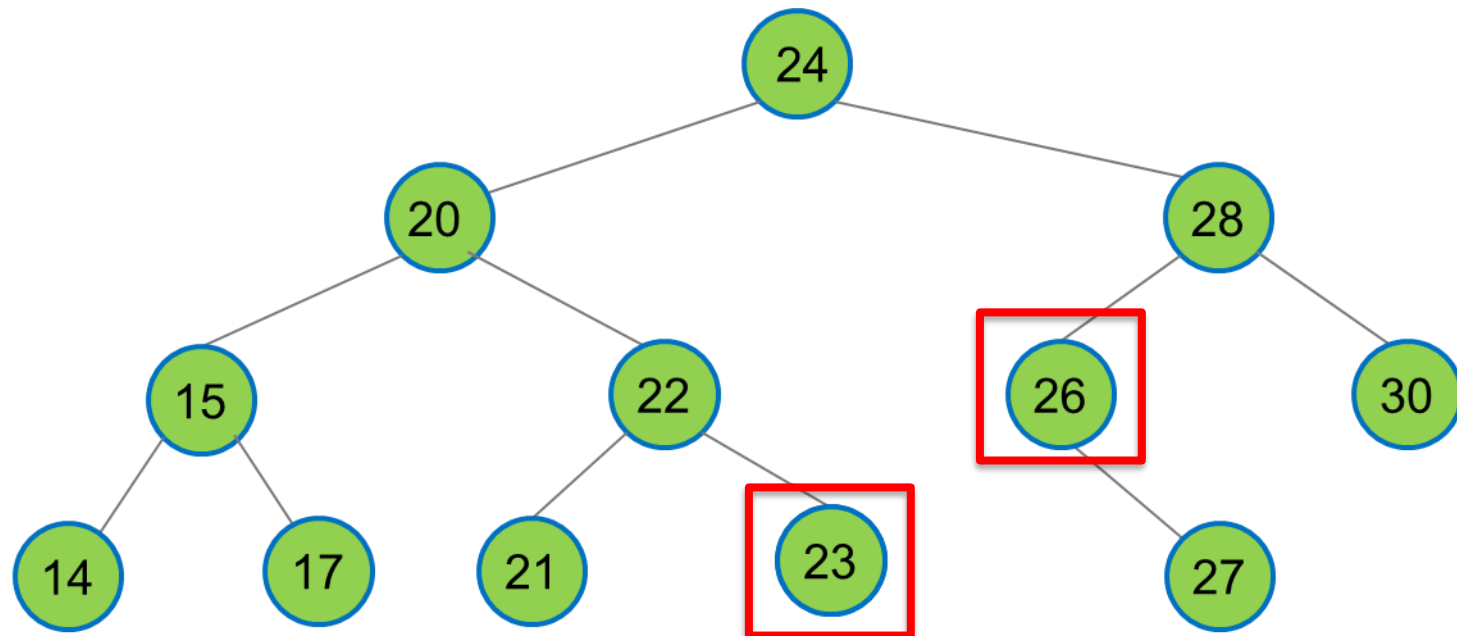
- The main functions?
  - Delete 24



# Tree

## And Binary Search Tree (BST)

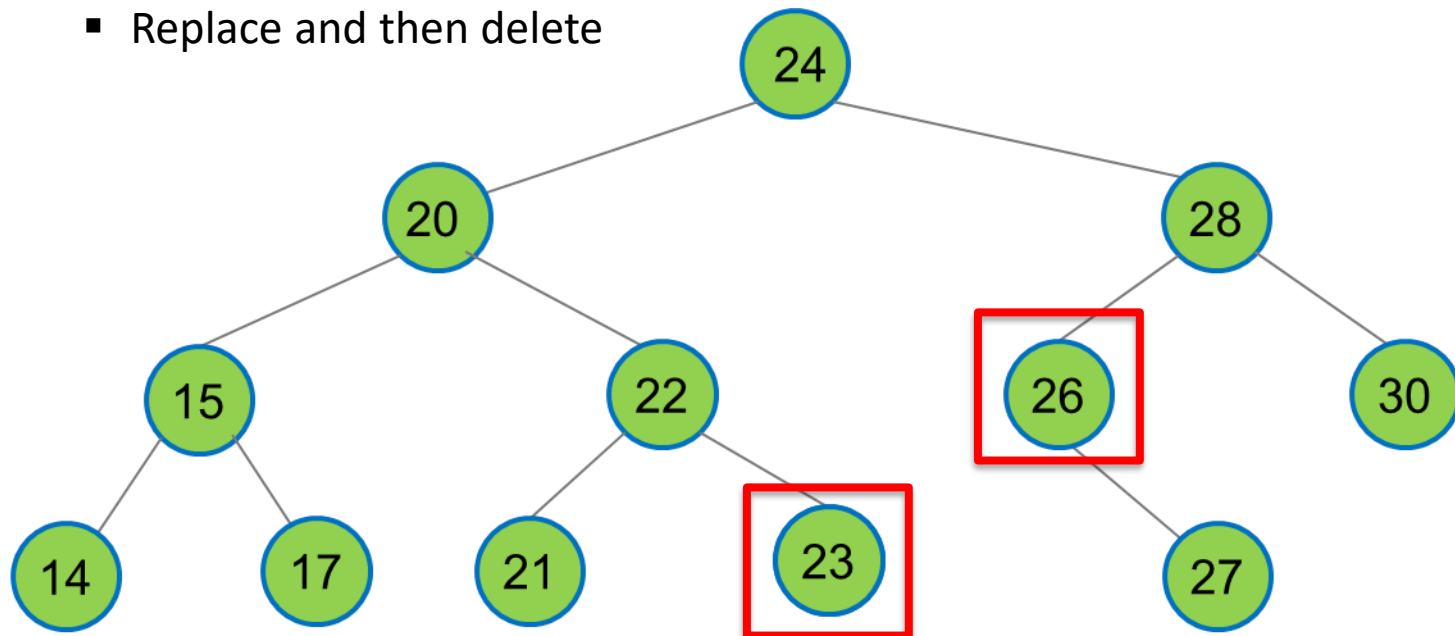
- The main functions?
  - Delete 24
    - Find the biggest left child
    - Find the smallest right child



# Tree

## And Binary Search Tree (BST)

- The main functions?
  - Delete 24
    - Find the biggest left child
    - Find the smallest right child
    - Replace and then delete

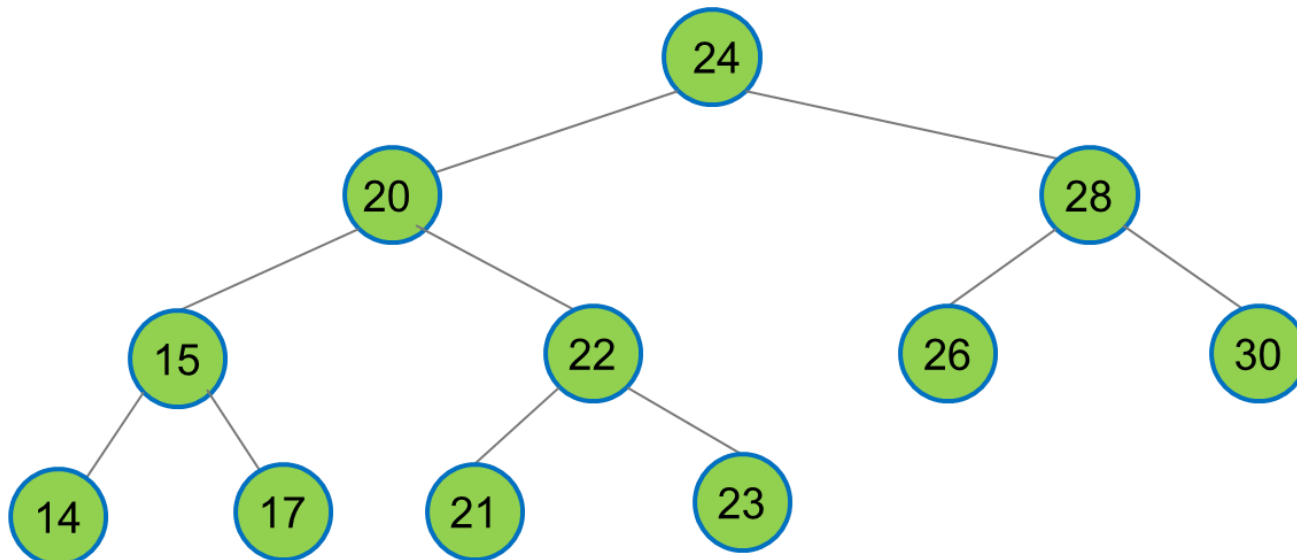


Questions?

# Tree

## And Binary Search Tree (BST)

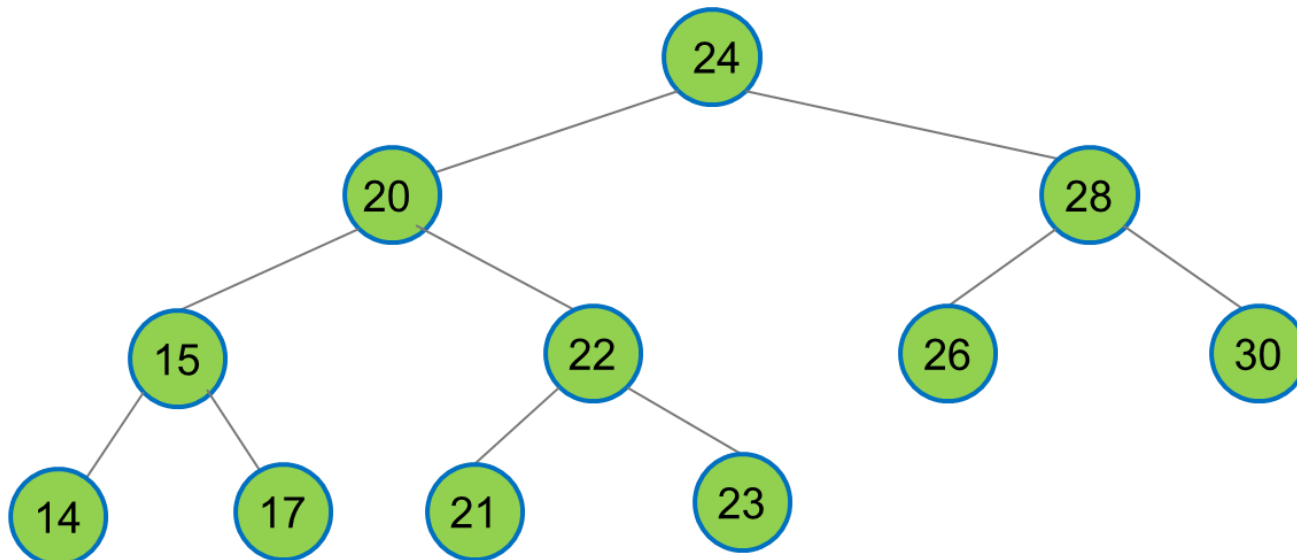
- The main functions?
  - Search
  - Traversal



# Tree

## And Binary Search Tree (BST)

- The main functions?
  - Search 24

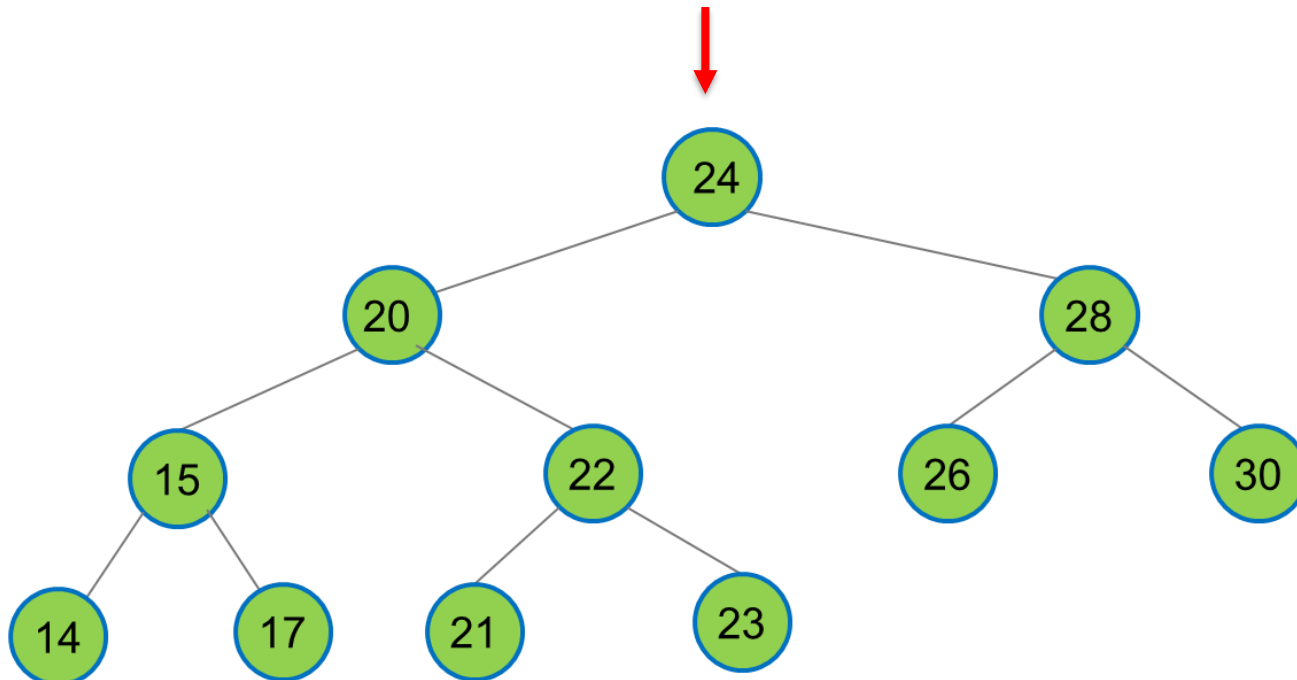




# Tree

## And Binary Search Tree (BST)

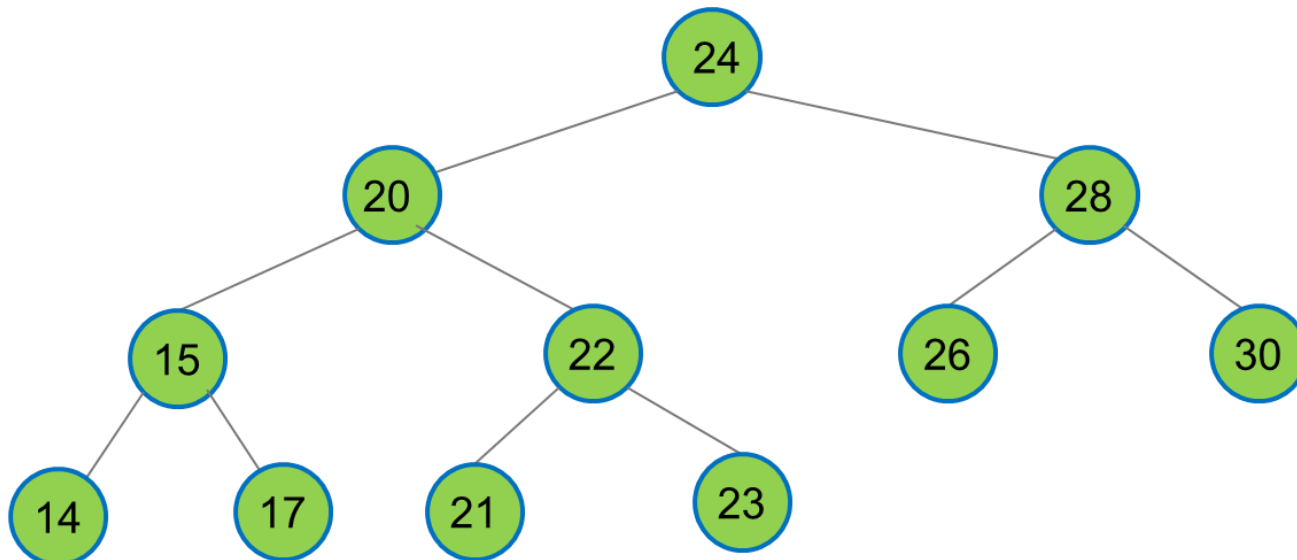
- The main functions?
  - Search 24



# Tree

## And Binary Search Tree (BST)

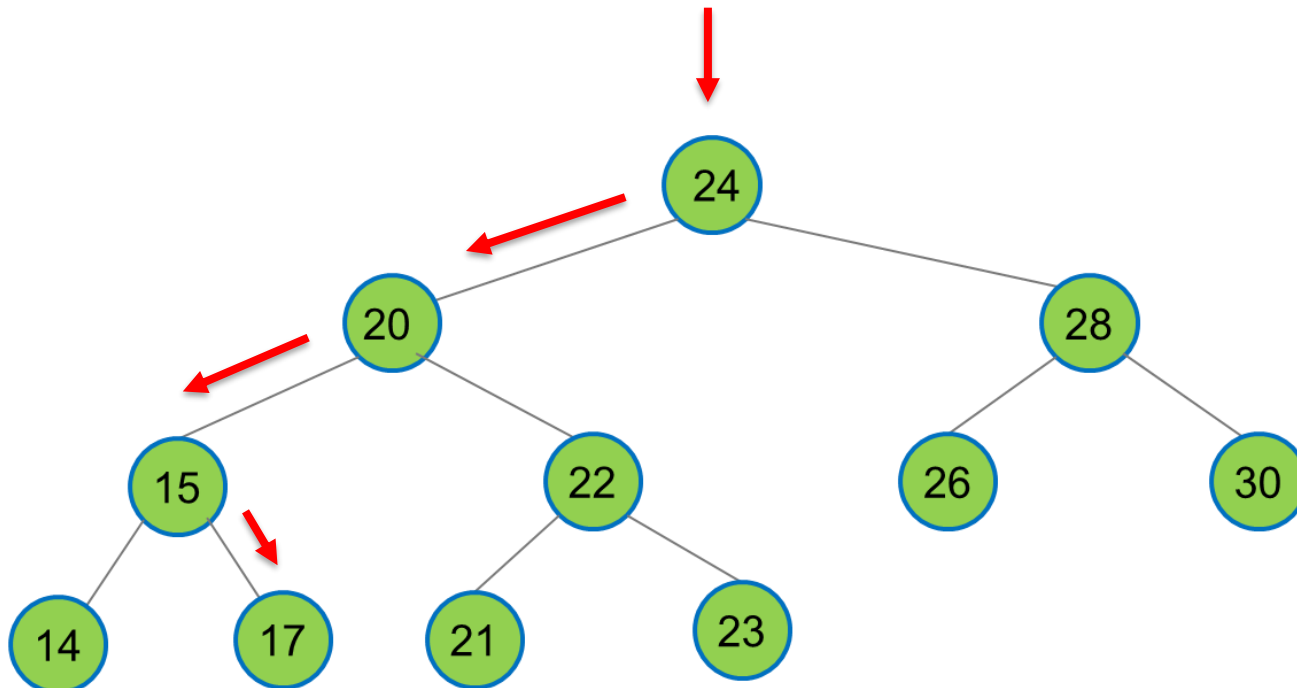
- The main functions?
  - Search 17



# Tree

## And Binary Search Tree (BST)

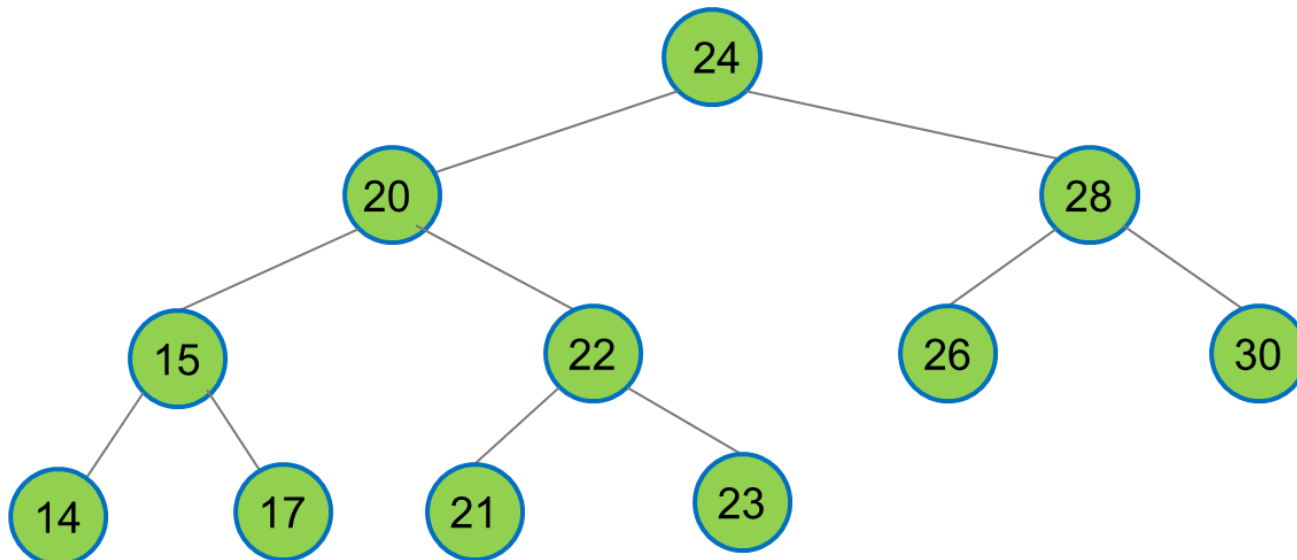
- The main functions?
  - Search 17



# Tree

## And Binary Search Tree (BST)

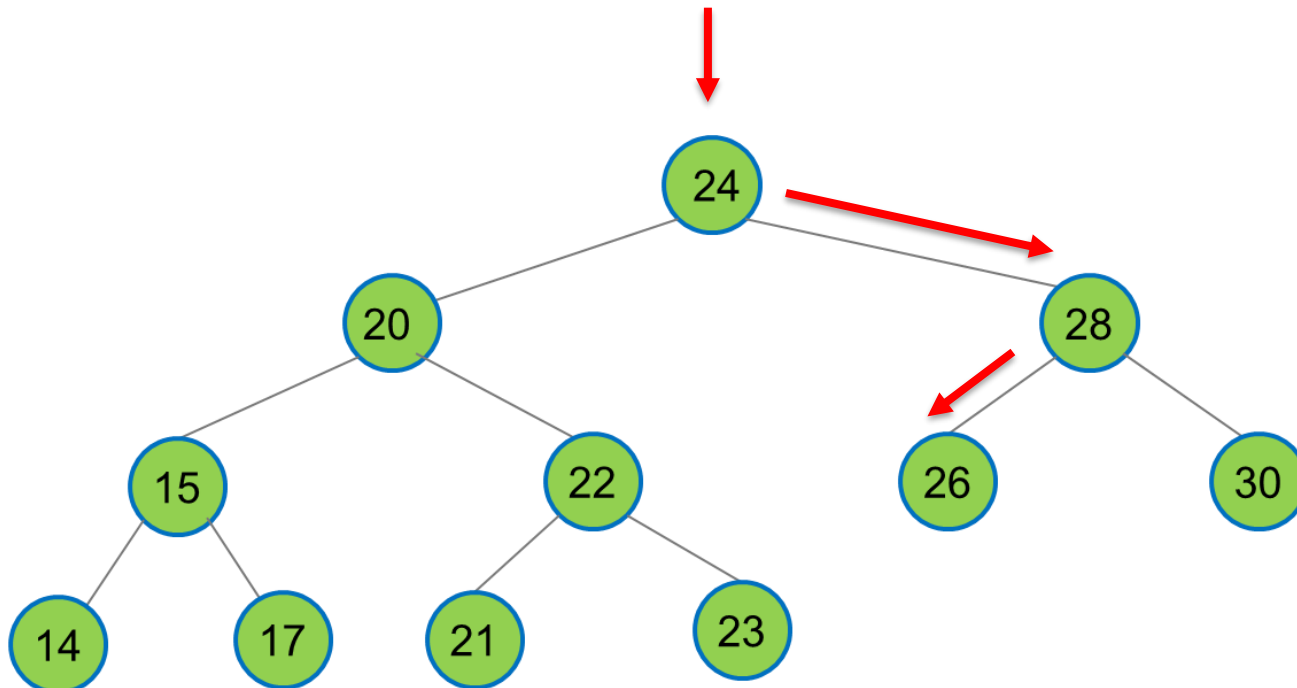
- The main functions?
  - Search 26



# Tree

## And Binary Search Tree (BST)

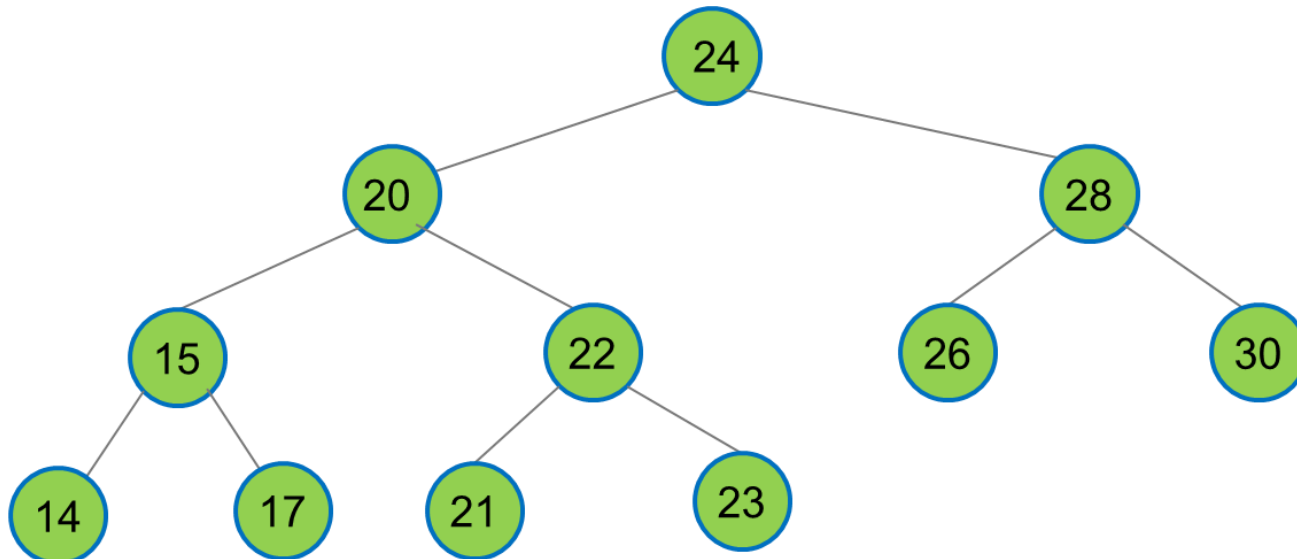
- The main functions?
  - Search 26



# Tree

## And Binary Search Tree (BST)

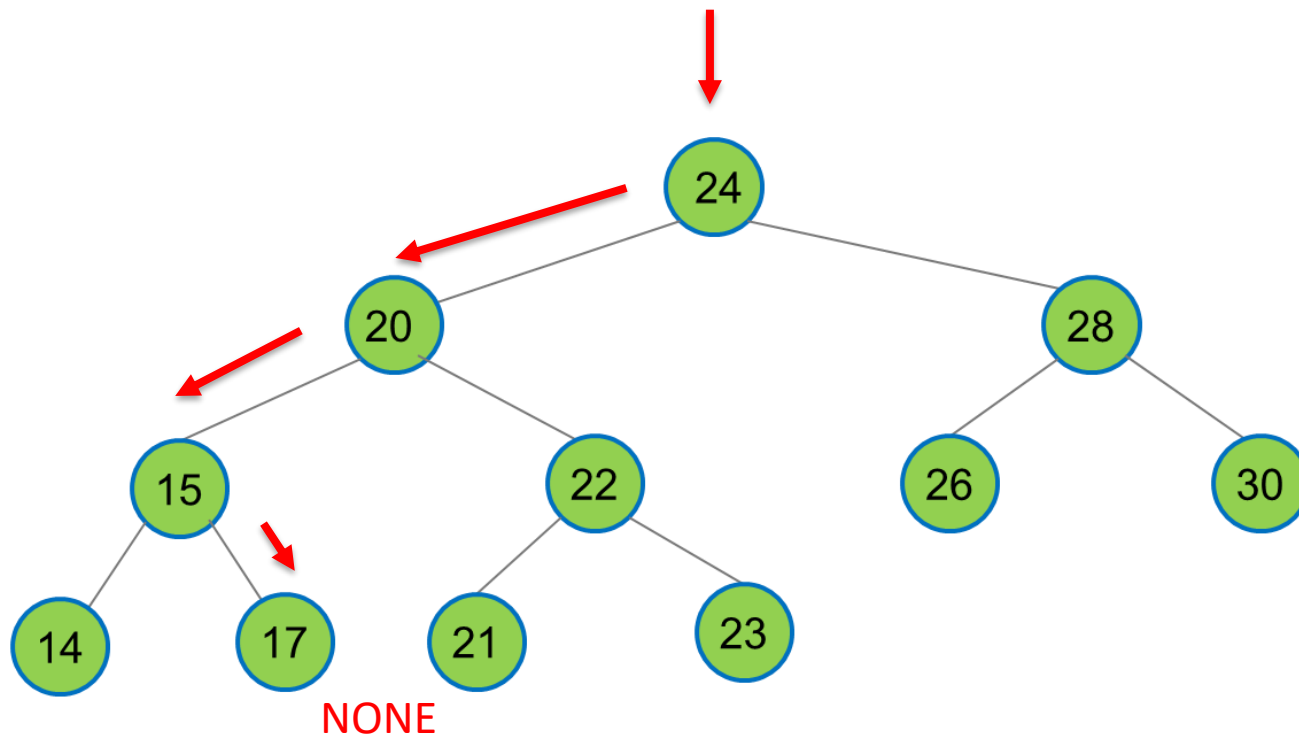
- The main functions?
  - Search 19



# Tree

## And Binary Search Tree (BST)

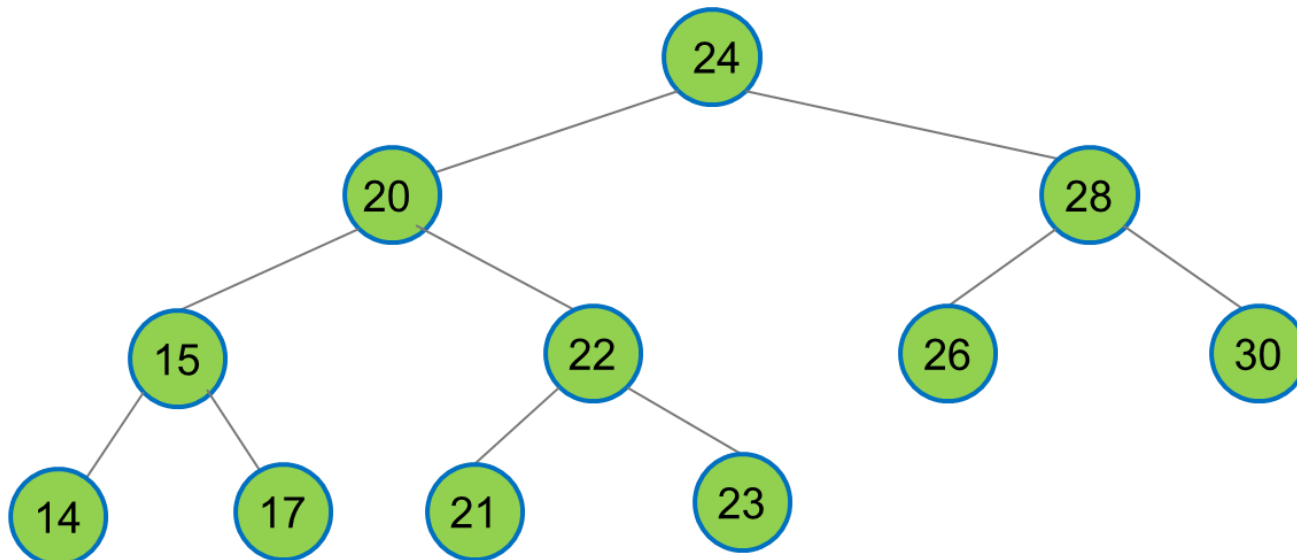
- The main functions?
  - Search 19



# Tree

## And Binary Search Tree (BST)

- The main functions?
  - Search for smallest number in BST

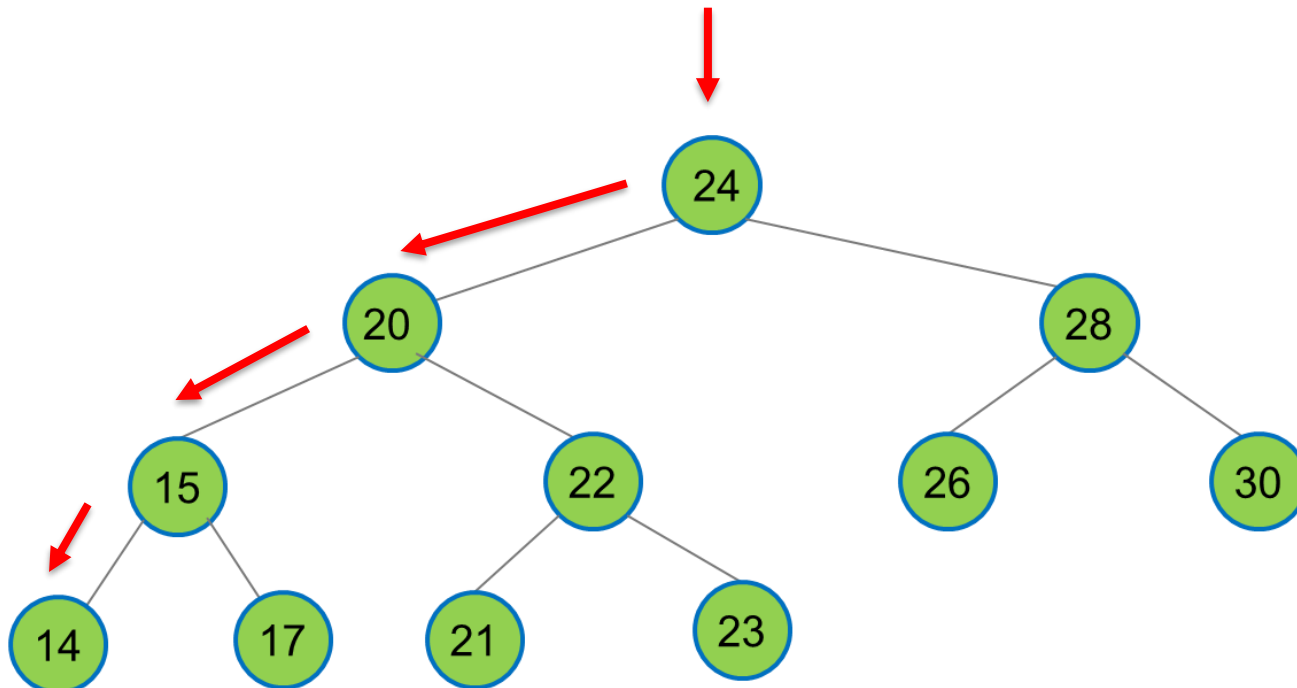




# Tree

## And Binary Search Tree (BST)

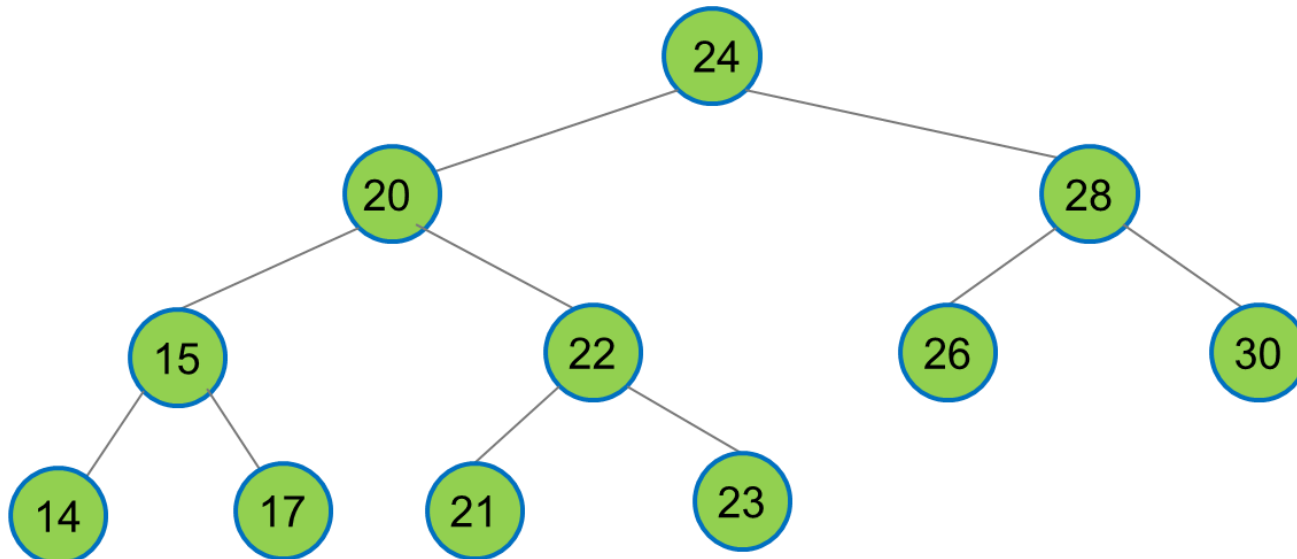
- The main functions?
  - Search for smallest number in BST



# Tree

## And Binary Search Tree (BST)

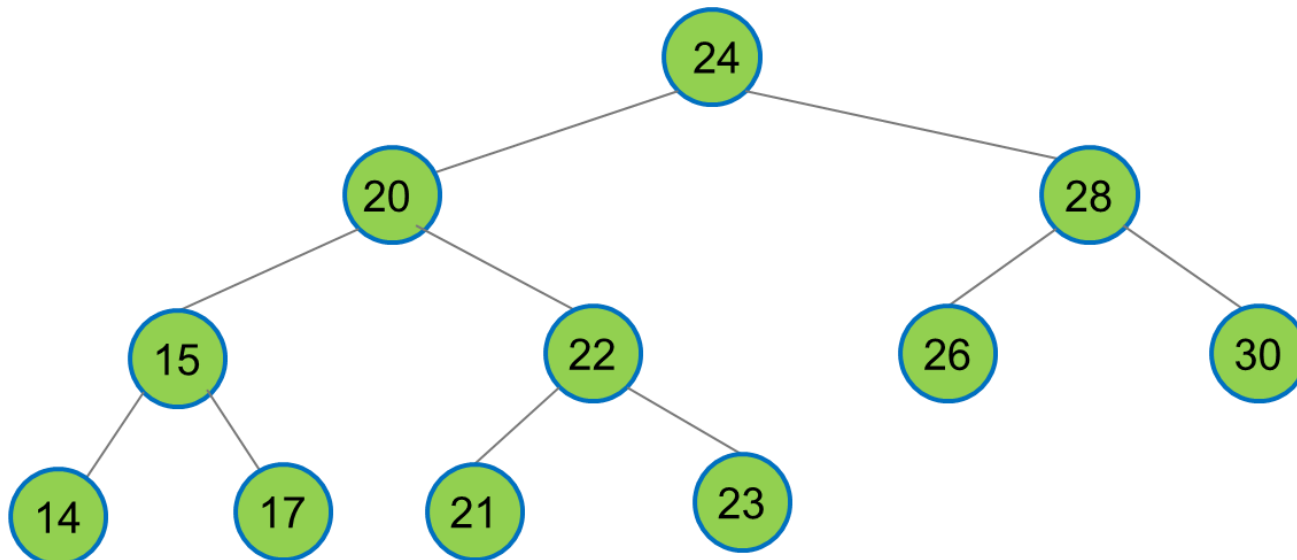
- The main functions?
  - Search complexity?



# Tree

## And Binary Search Tree (BST)

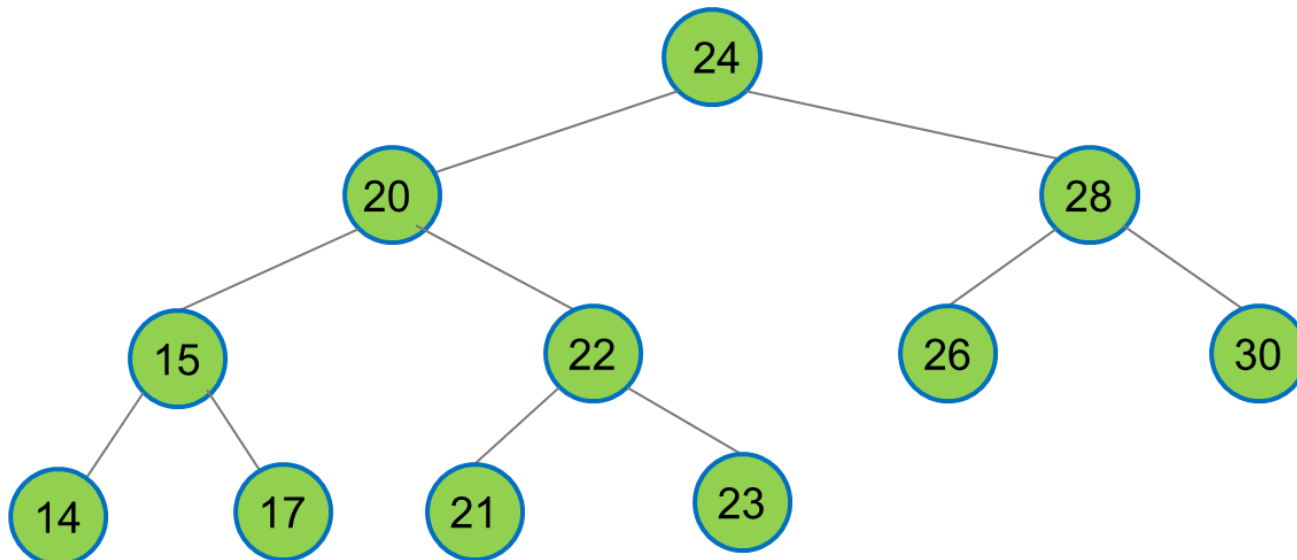
- The main functions?
  - Search complexity?  $O(\log N)$ ?



# Tree

## And Binary Search Tree (BST)

- The main functions?
  - Search complexity?  $O(N)$



# Tree

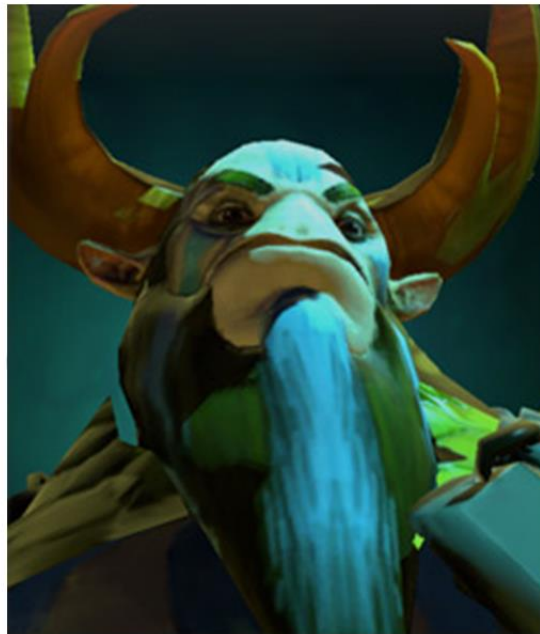
## And Binary Search Tree (BST)

- The main functions?
  - Search complexity?  $O(N)$ 
    - Complexity is high when tree is imbalanced!

Non-balanced



Balanced

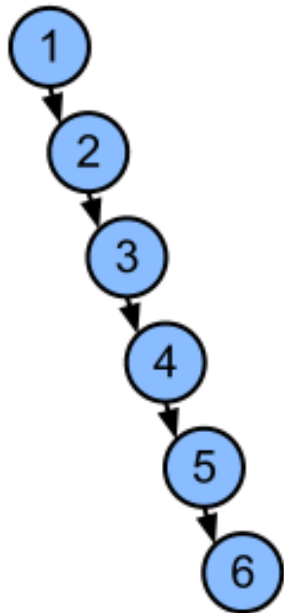


# Tree

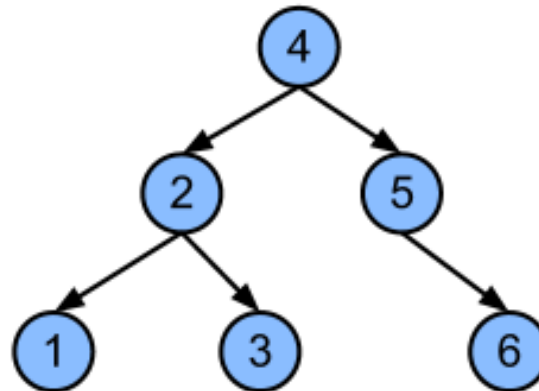
## And Binary Search Tree (BST)

- The main functions?
  - Search complexity?  $O(N)$ 
    - Complexity is high when tree is imbalanced!

Non-balanced



Balanced



Questions?

# AVL Tree

## The better BST!

- This is BST





# AVL Tree

## The better BST!

- This is BST



This is AVL



# AVL Tree

## The better BST!

- This is BST



This is Adelson-Velskii Landis (AVL)



# AVL Tree

## The better BST!

- AVL trees are self balancing...



# AVL Tree

## The better BST!

- AVL trees are self **balancing**...
- Height balanced
  - $\text{absolute}(\text{Height}(\text{left}) - \text{Height}(\text{right})) \leq 1$
  - Ensure a maximum of  $O(\log N)$  height



# AVL Tree

## The better BST!

- AVL trees are self **balancing**...
- Height balanced
  - $\text{absolute}(\text{Height}(\text{left}) - \text{Height}(\text{right})) \leq 1$
  - Ensure a maximum of  $O(\log N)$  height
  - True for each subtree as well



# AVL Tree

## The better BST!

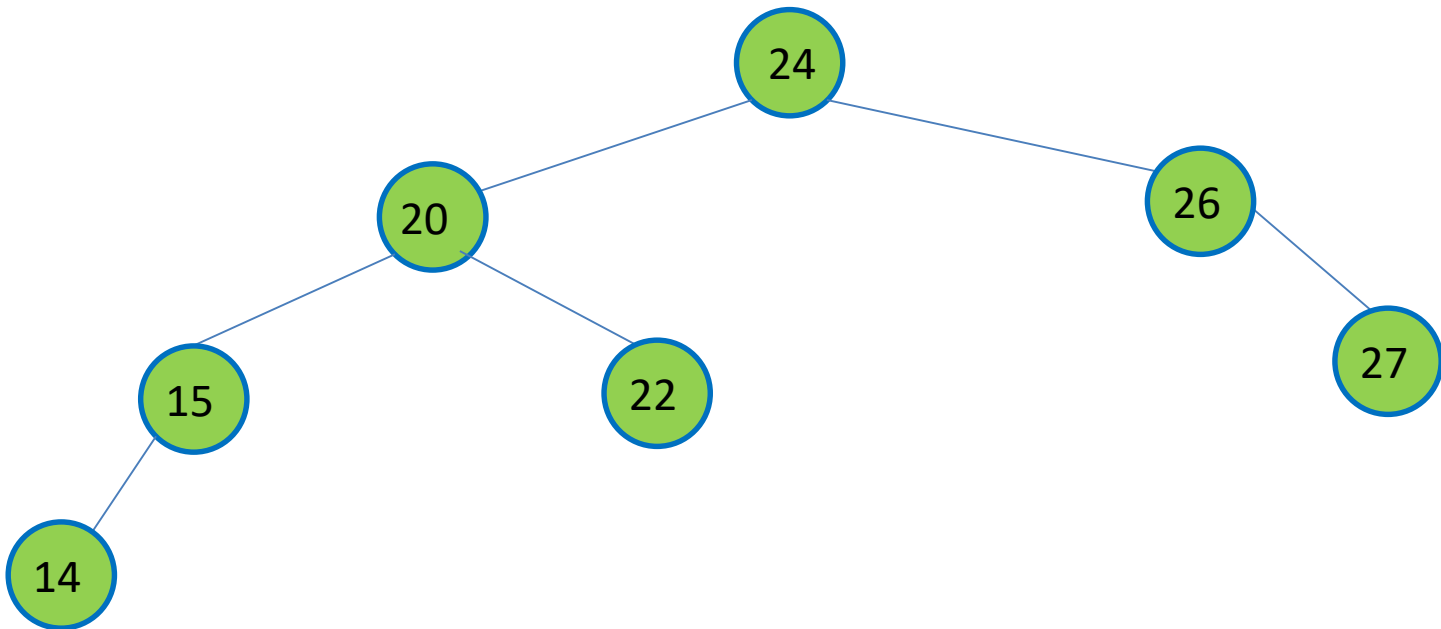
- AVL trees are self **balancing**...
- Height balanced
  - $\text{absolute}(\text{Height}(\text{left}) - \text{Height}(\text{right})) \leq 1$
  - Ensure a maximum of  $O(\log N)$  height
  - True for each subtree as well
  - Now let us see the trees



# AVL Tree

## The better BST!

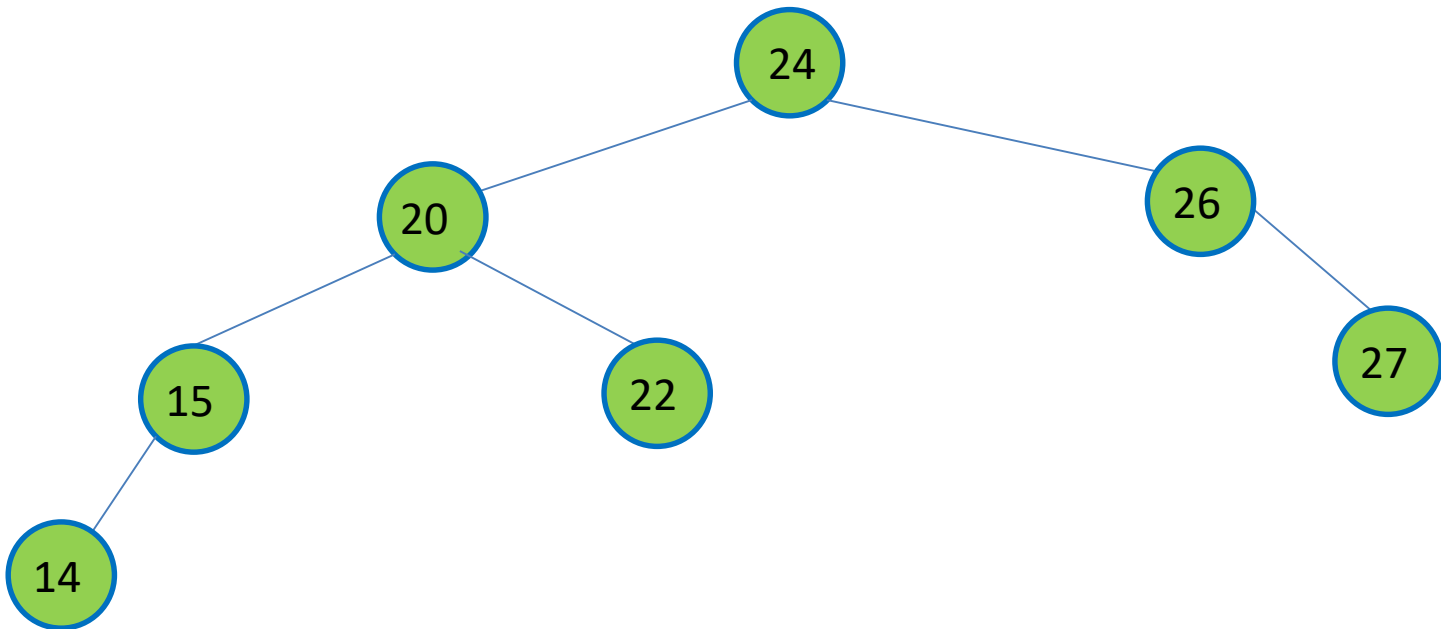
- What is the height?



# AVL Tree

## The better BST!

- What is the height?
  - Let us do it by hand

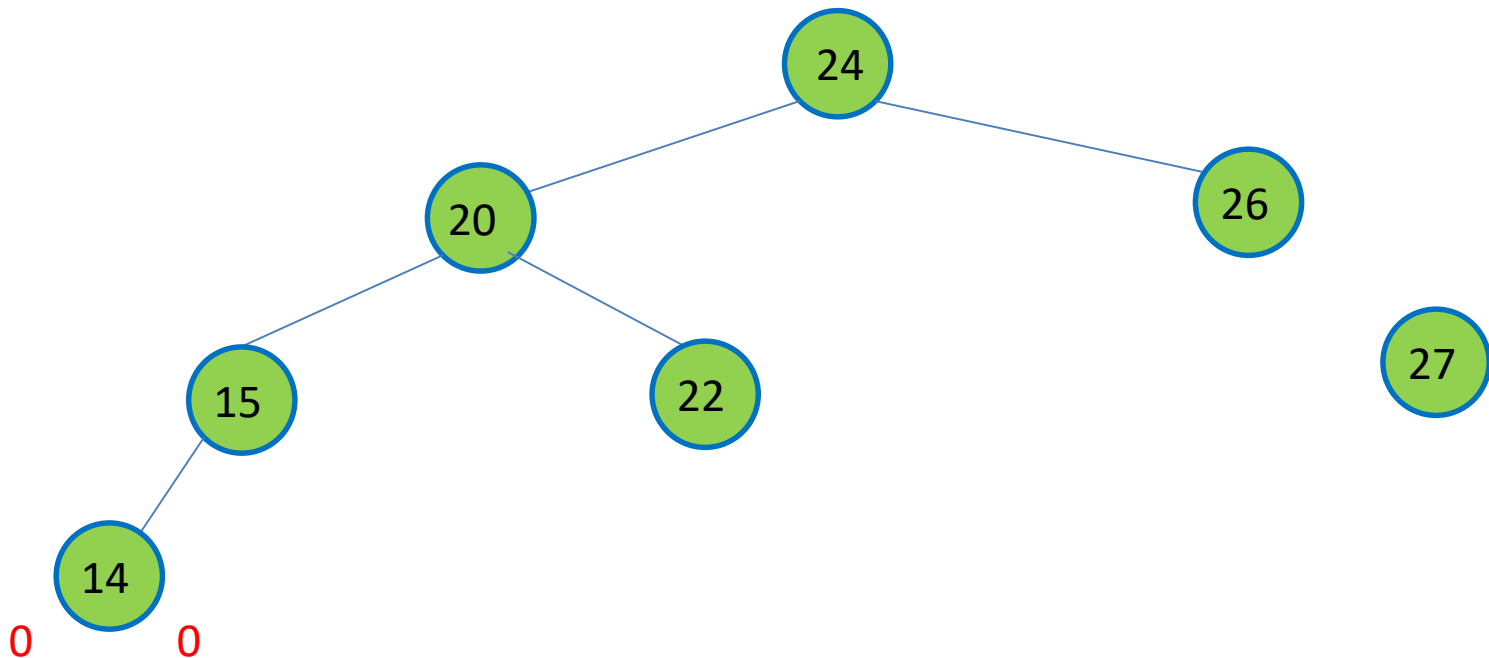




# AVL Tree

## The better BST!

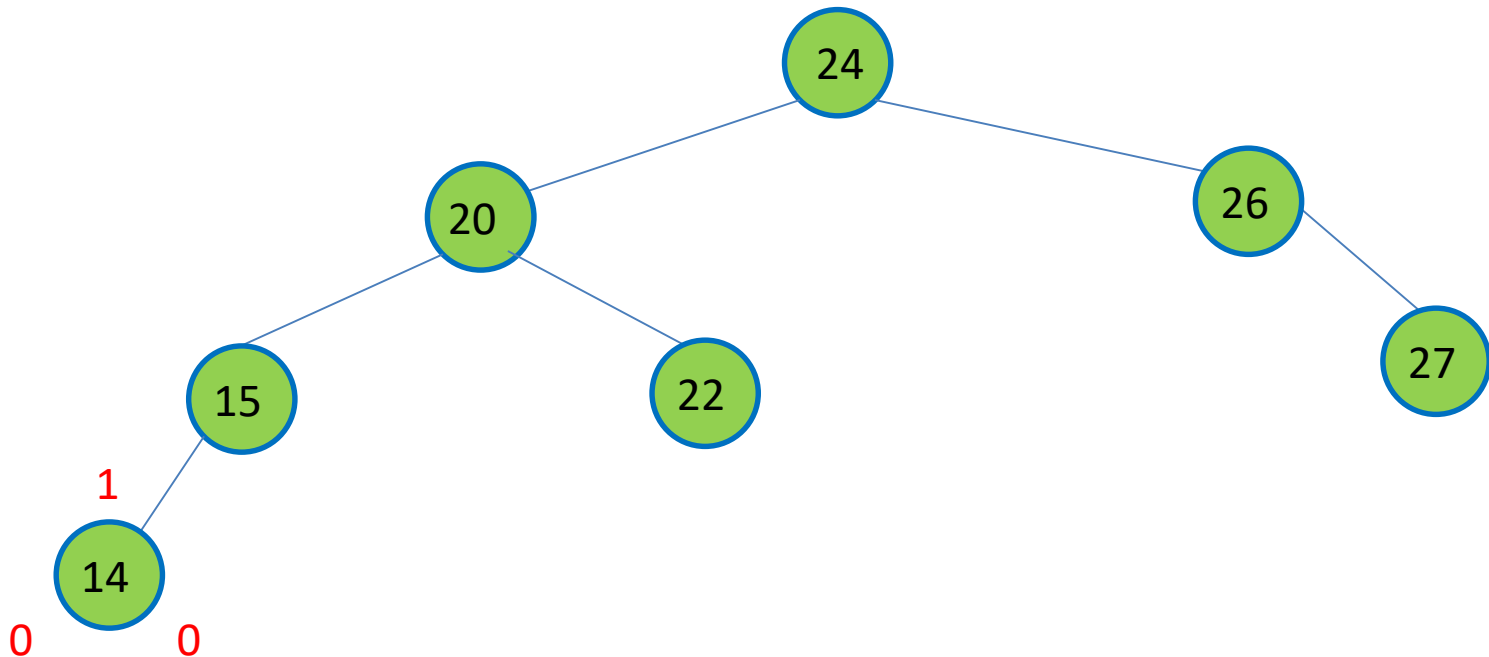
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

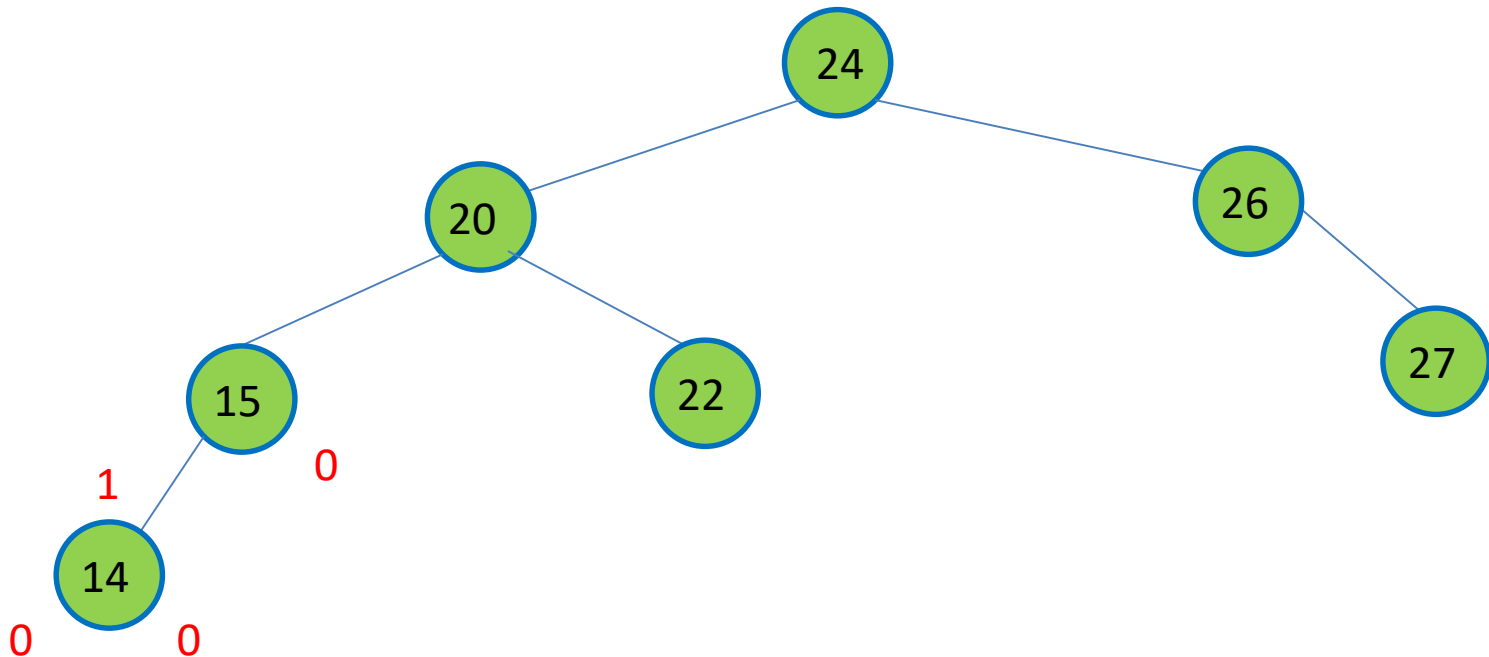
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

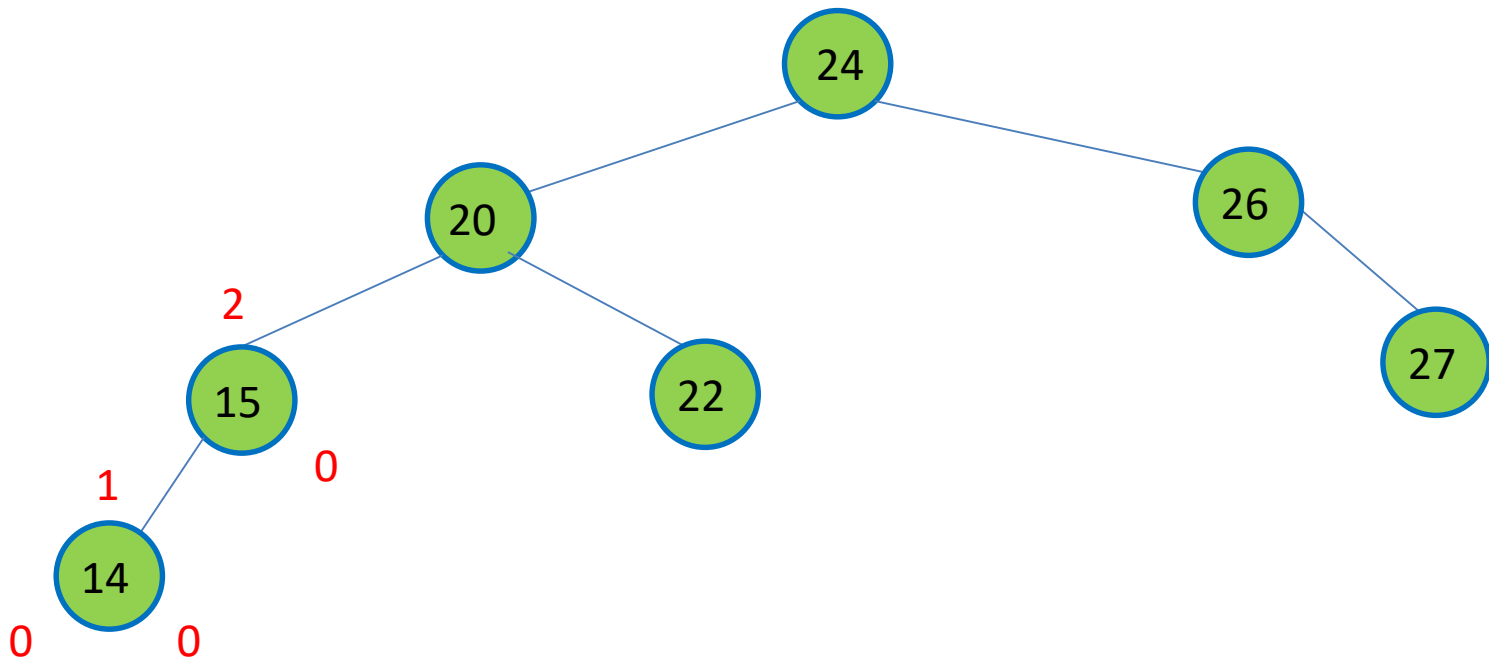
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

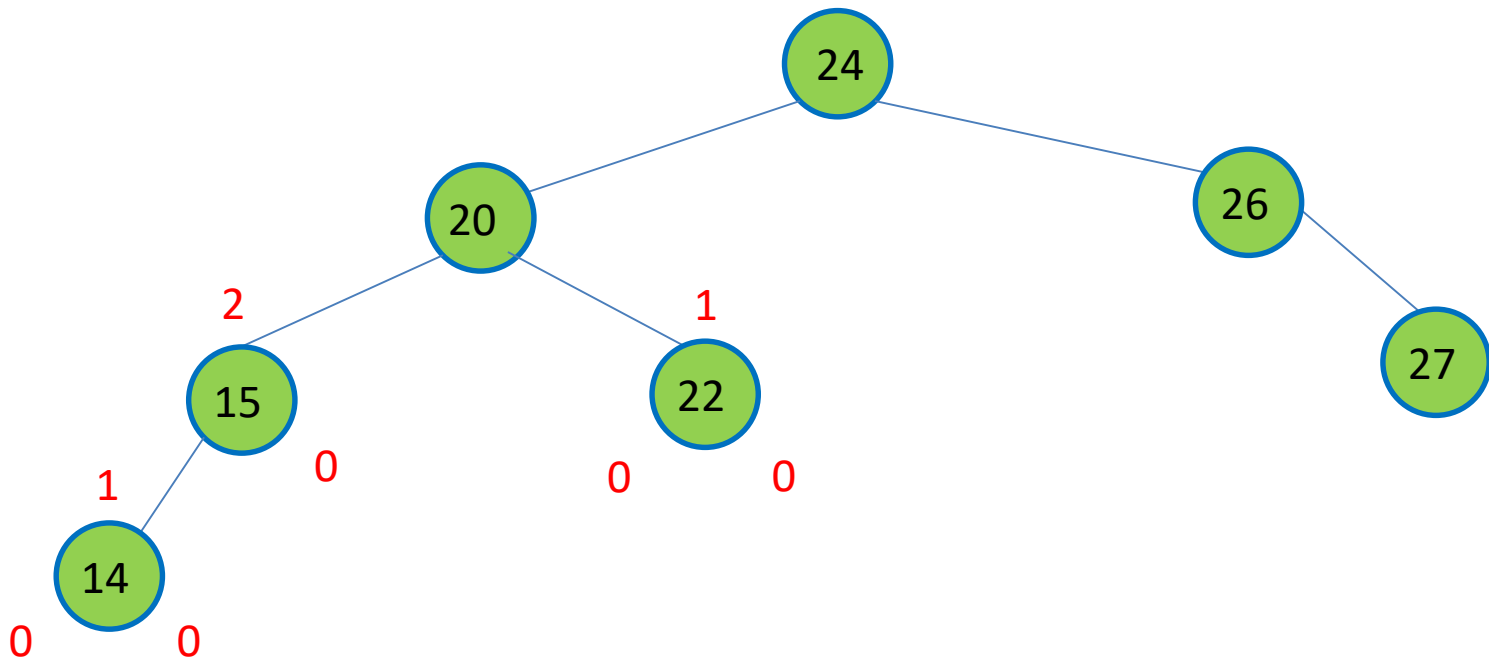
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

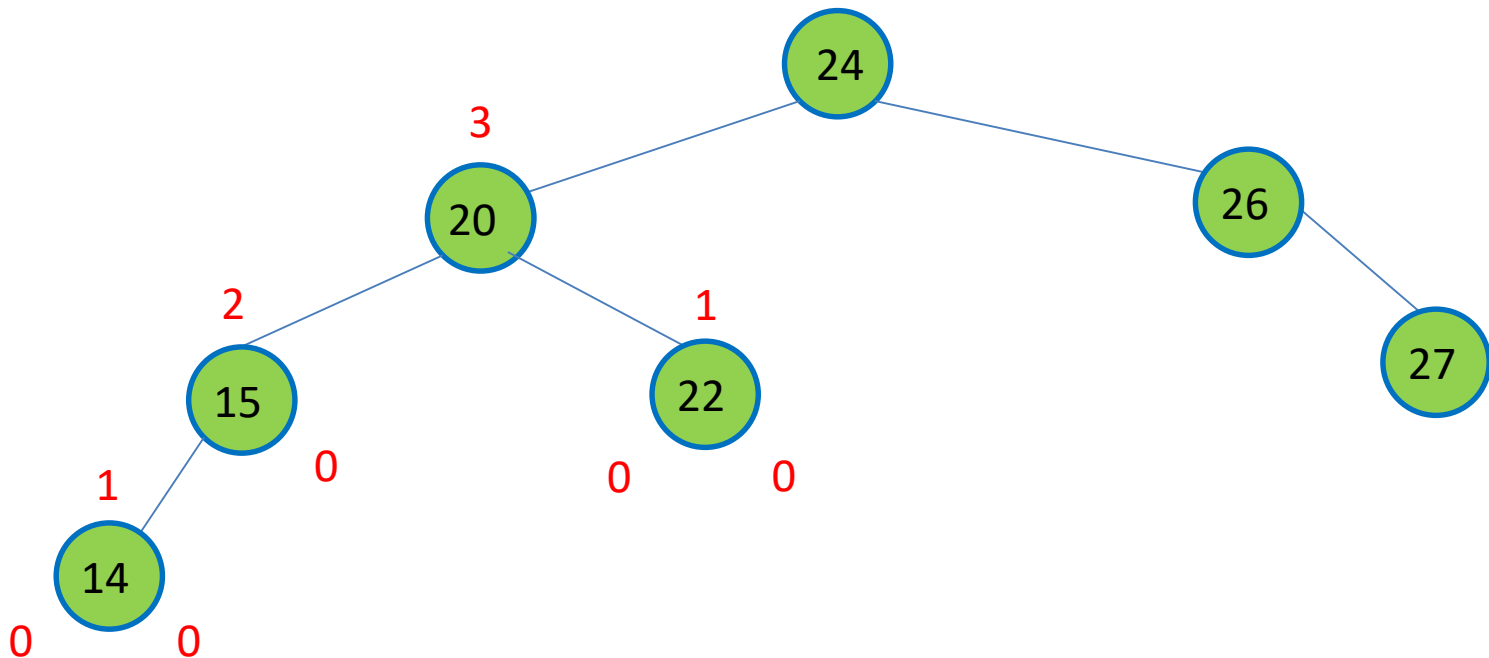
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

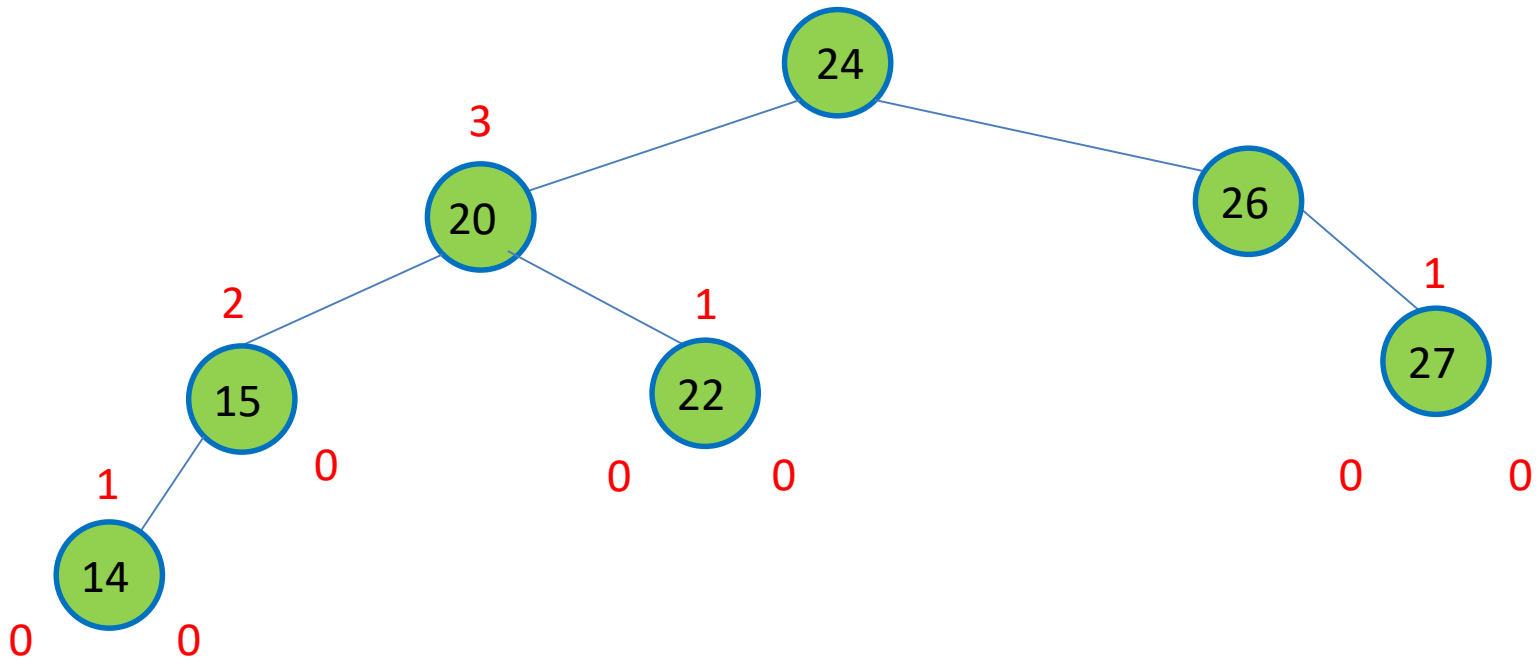
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

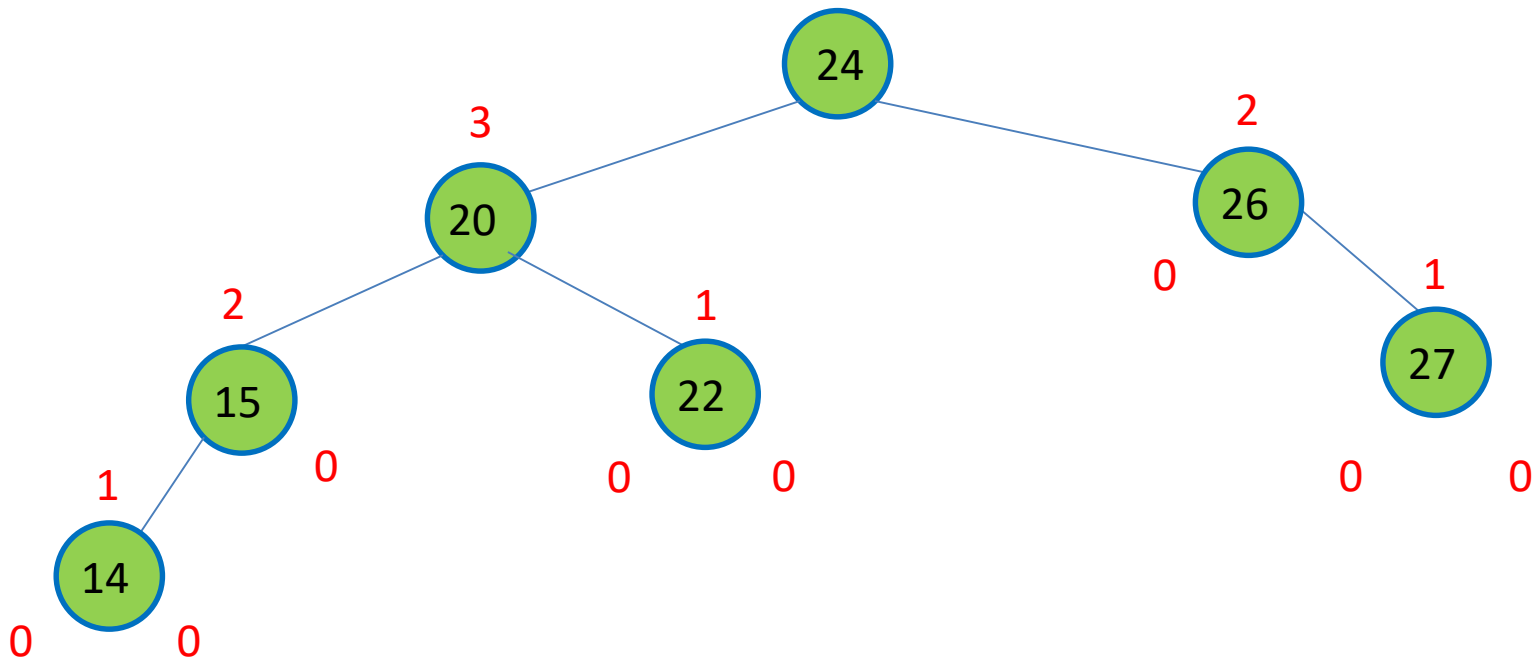
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

- What is the height?
  - Let us do it by hand

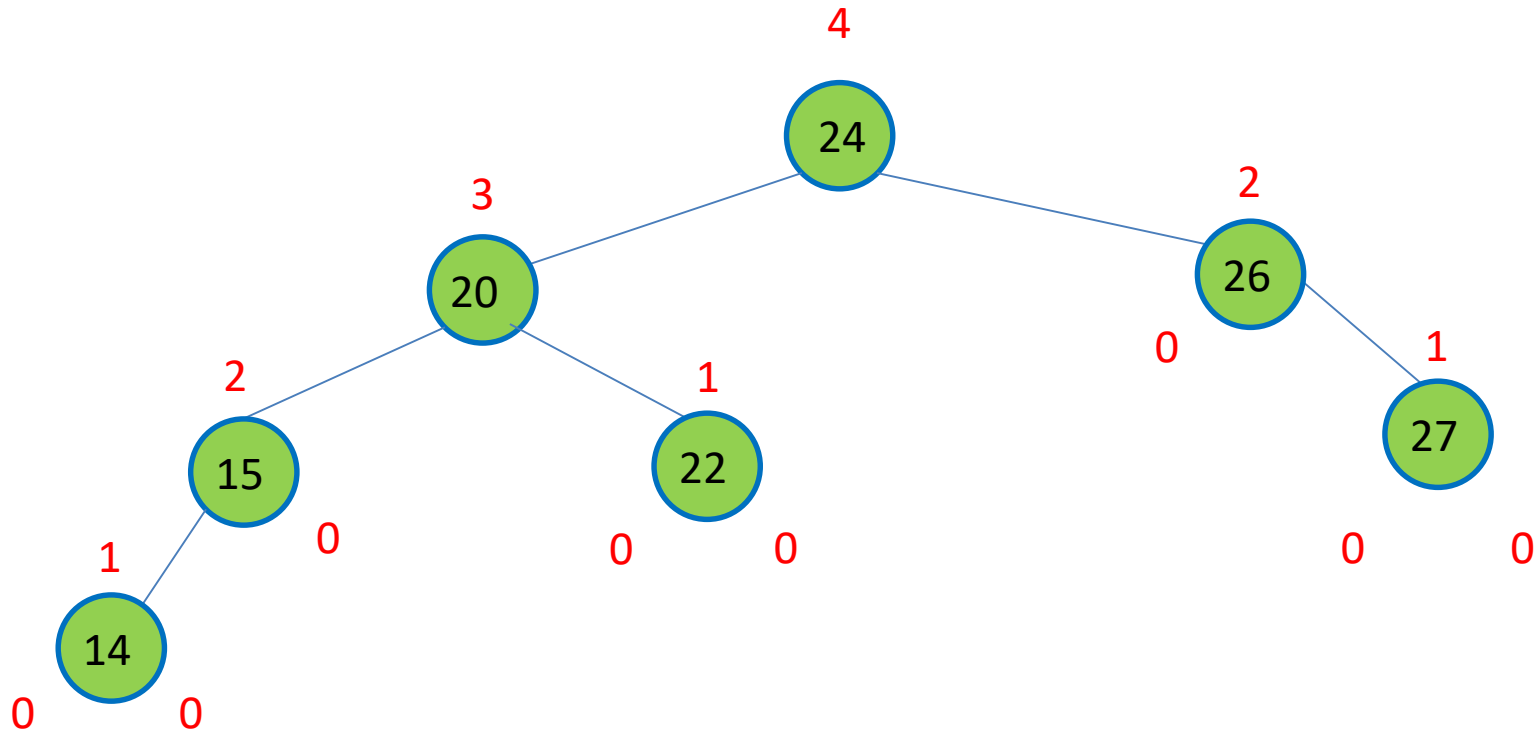




# AVL Tree

## The better BST!

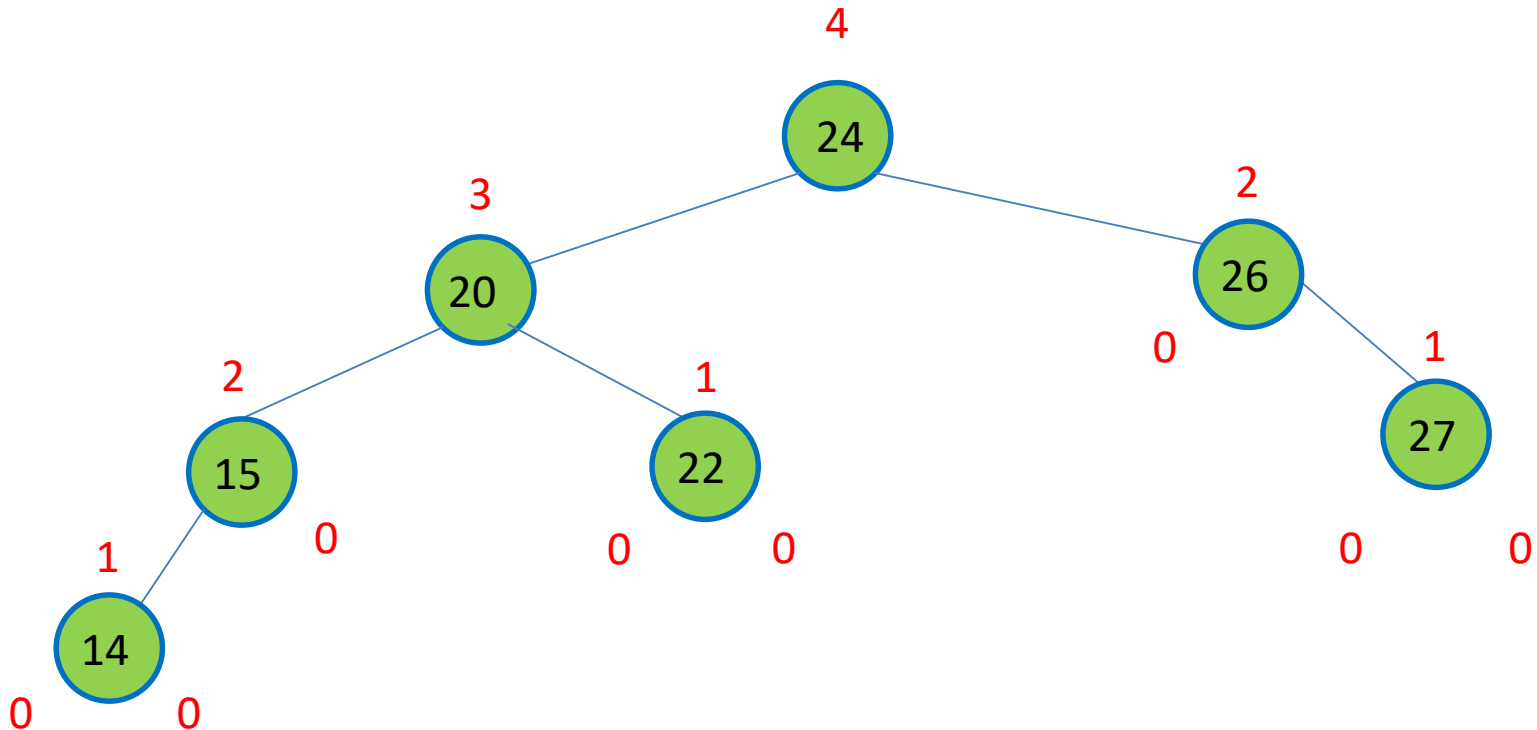
- What is the height?
  - Let us do it by hand



# AVL Tree

## The better BST!

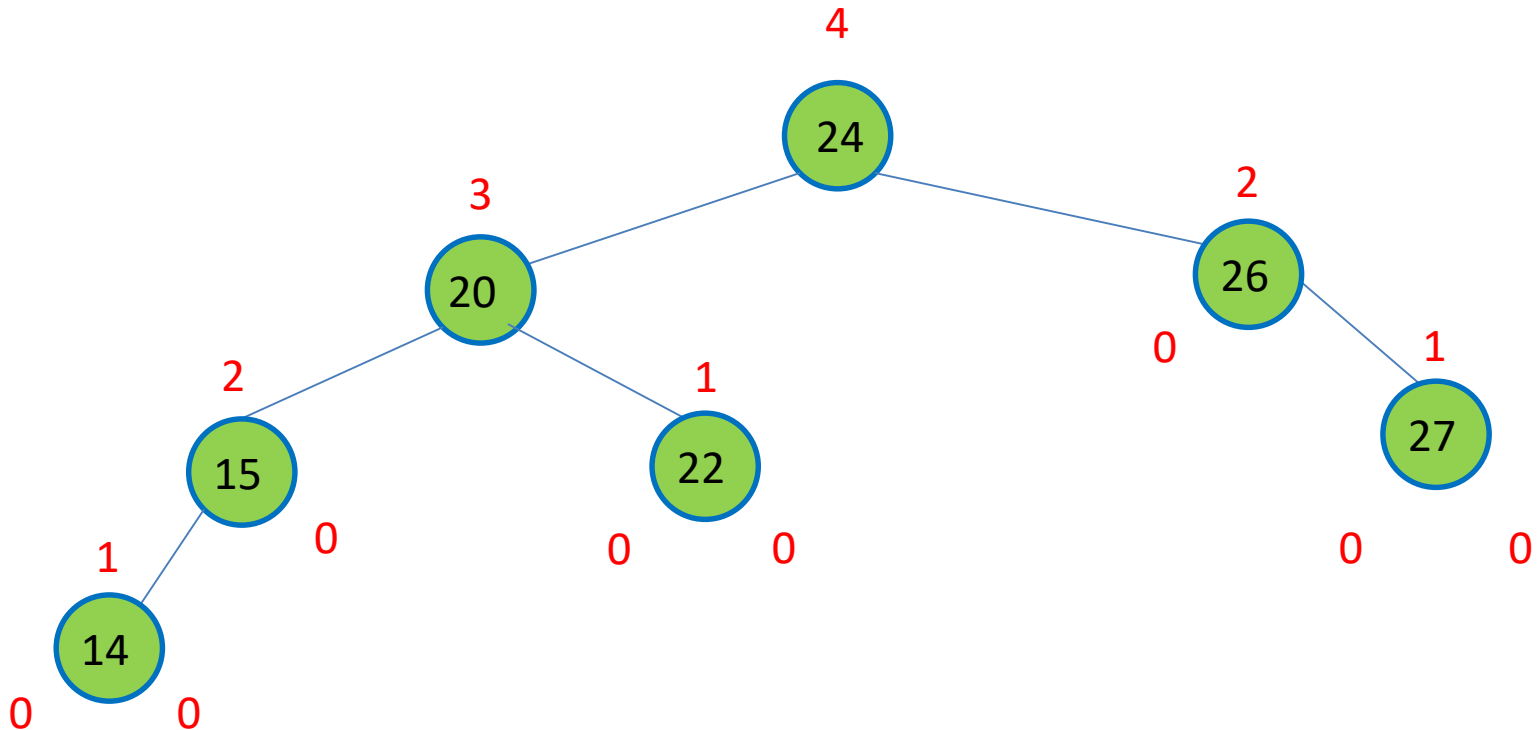
- What is the height?
- Is the tree balanced?



# AVL Tree

## The better BST!

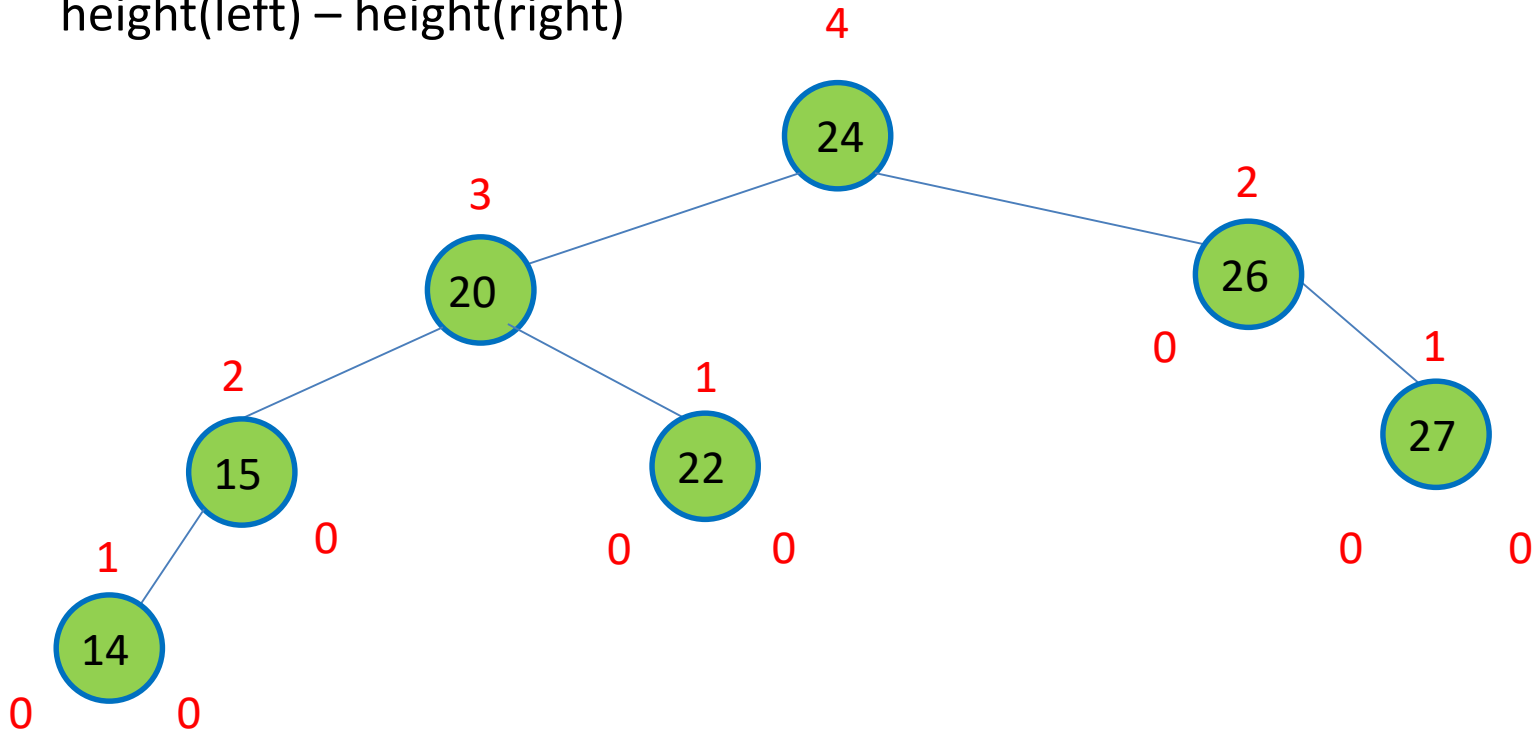
- What is the height?
- Is the tree balanced?
  - Balance factor =  $\{-1, 0, 1\}$



# AVL Tree

## The better BST!

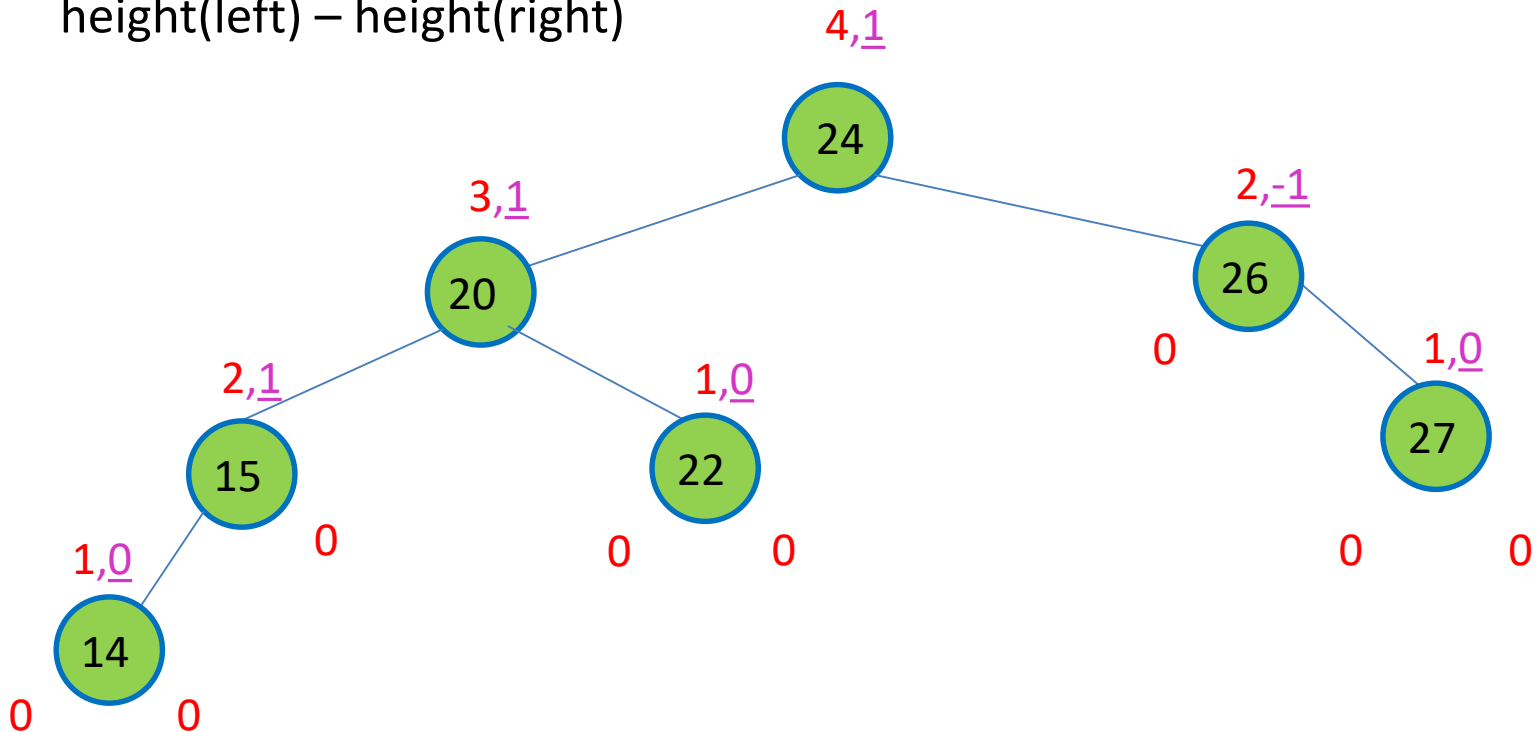
- What is the height?
- Is the tree balanced?
  - Balance factor =  $\{-1, 0, 1\}$   
 $\text{height}(\text{left}) - \text{height}(\text{right})$



# AVL Tree

## The better BST!

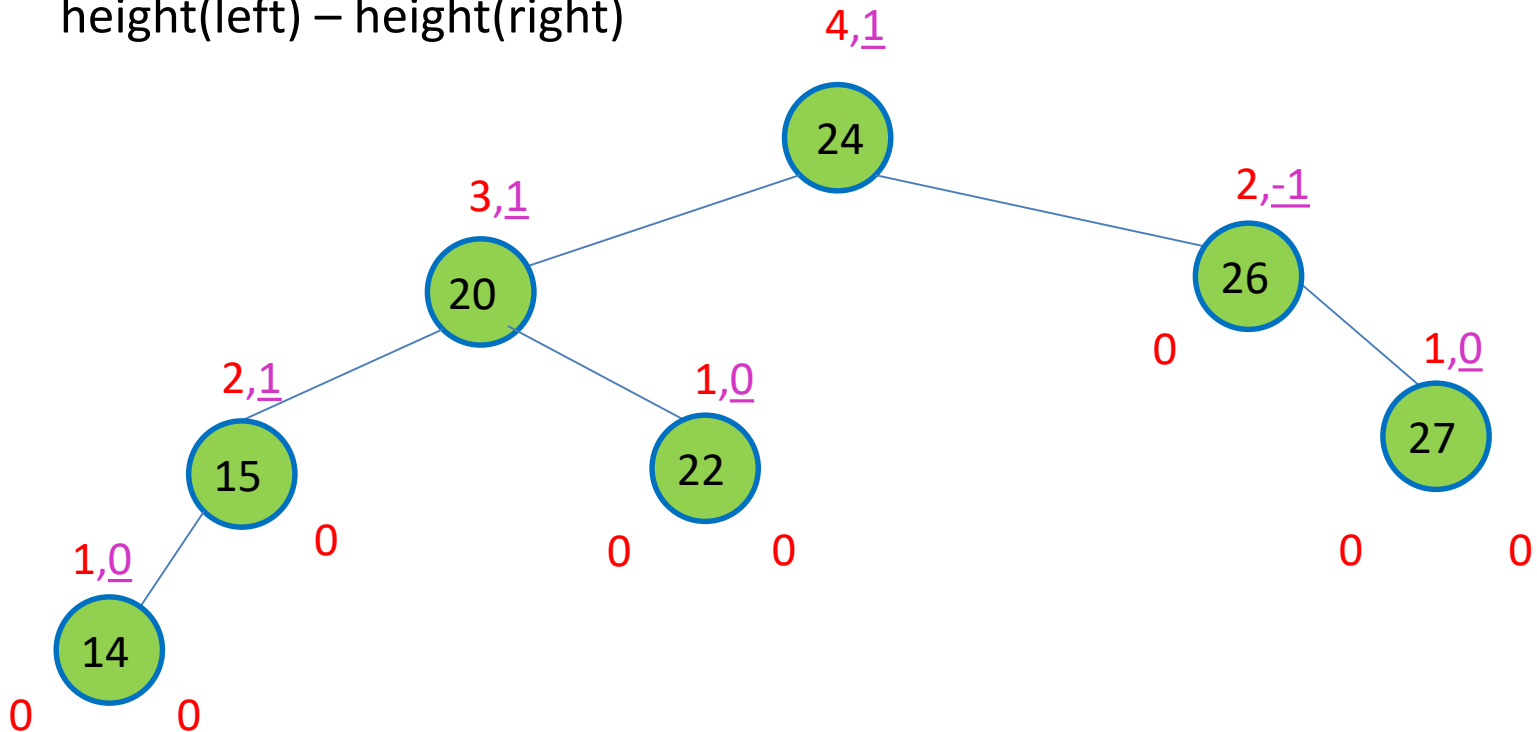
- What is the height?
- Is the tree balanced?
  - Balance factor =  $\{-1, 0, 1\}$   
height(left) – height(right)



# AVL Tree

## The better BST!

- What is the height?
- Is the tree balanced? **YES**
  - Balance factor =  $\{-1, 0, 1\}$   
 $\text{height}(\text{left}) - \text{height}(\text{right})$

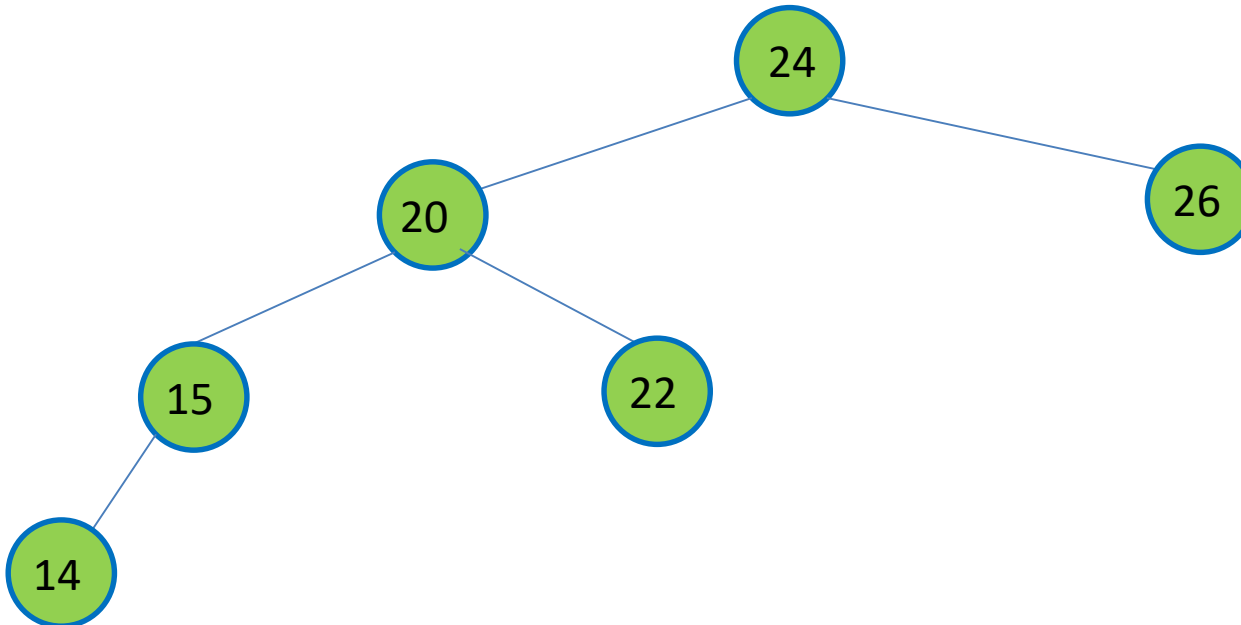


Questions?

# AVL Tree

## The better BST!

- Let's try a new tree...

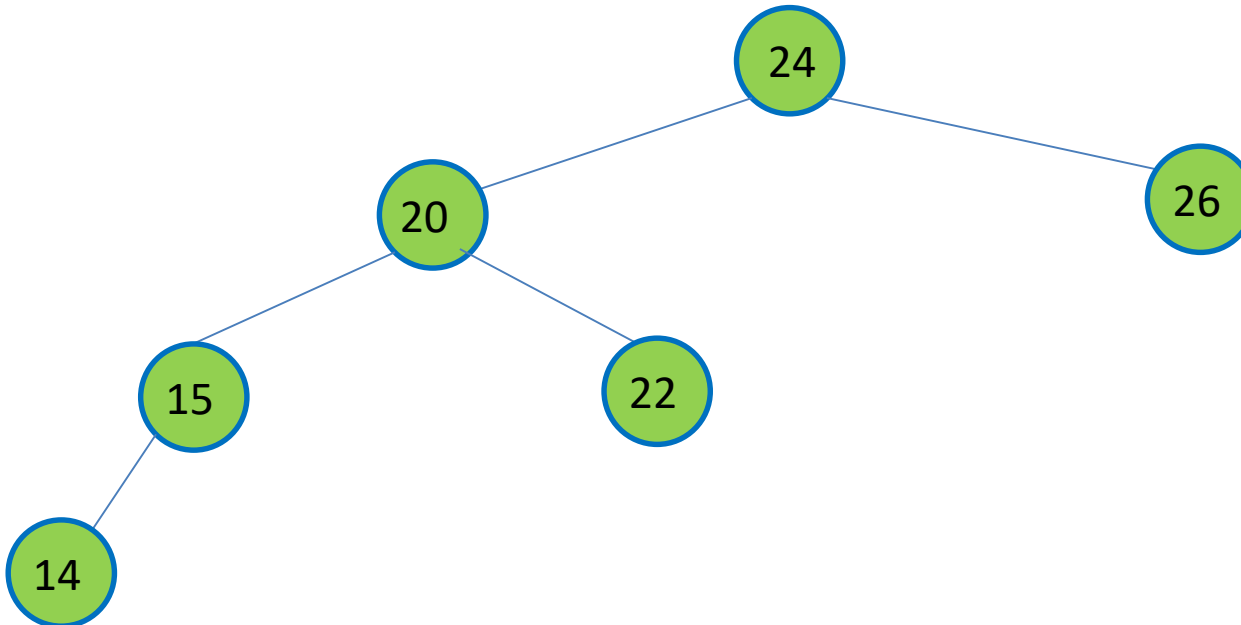




# AVL Tree

## The better BST!

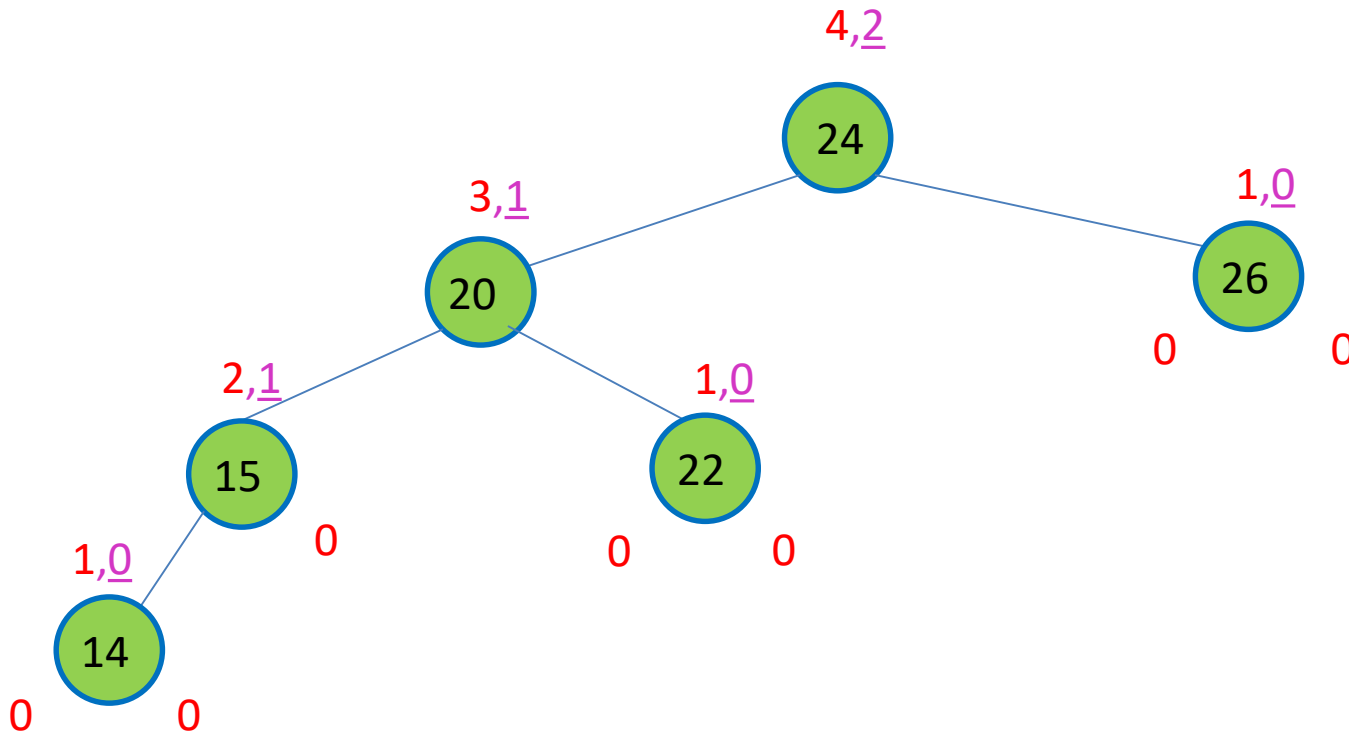
- Let's try a new tree...
  - Is this balanced?



# AVL Tree

## The better BST!

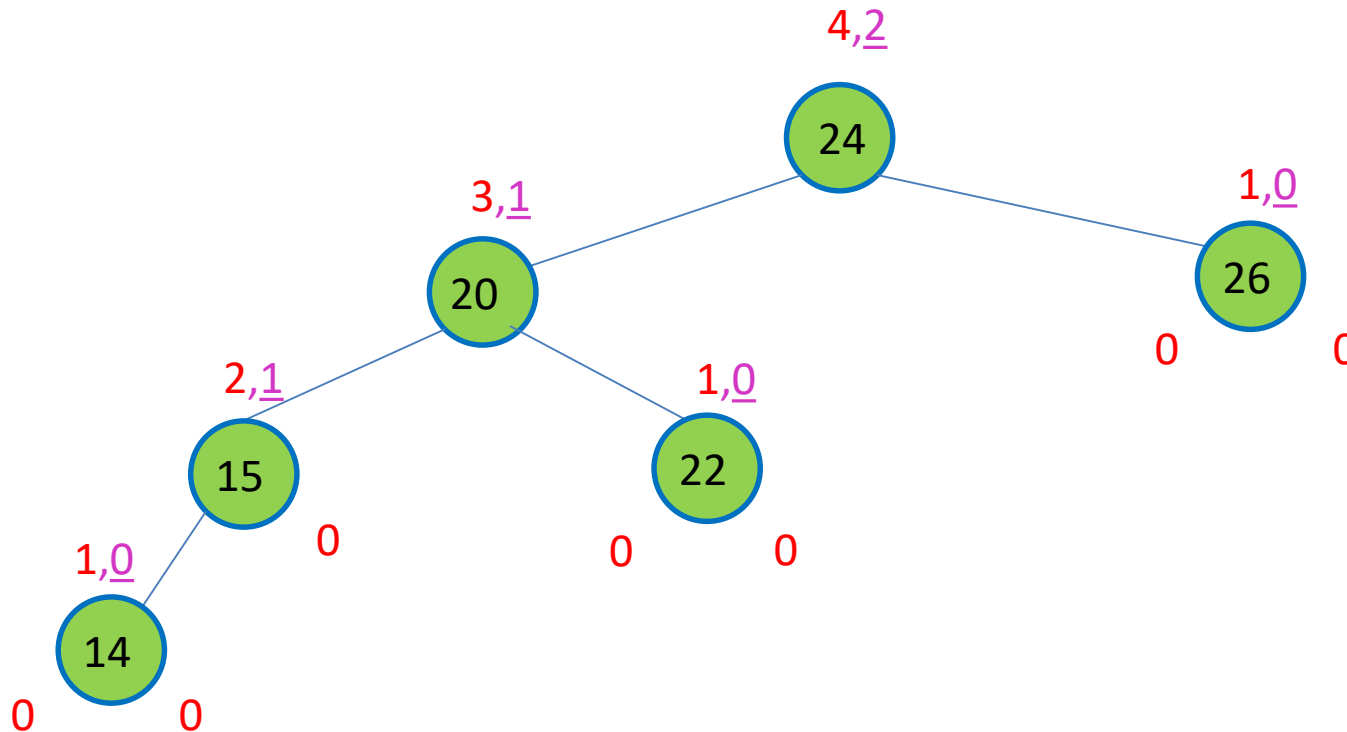
- Let's try a new tree...
  - Is this balanced?



# AVL Tree

## The better BST!

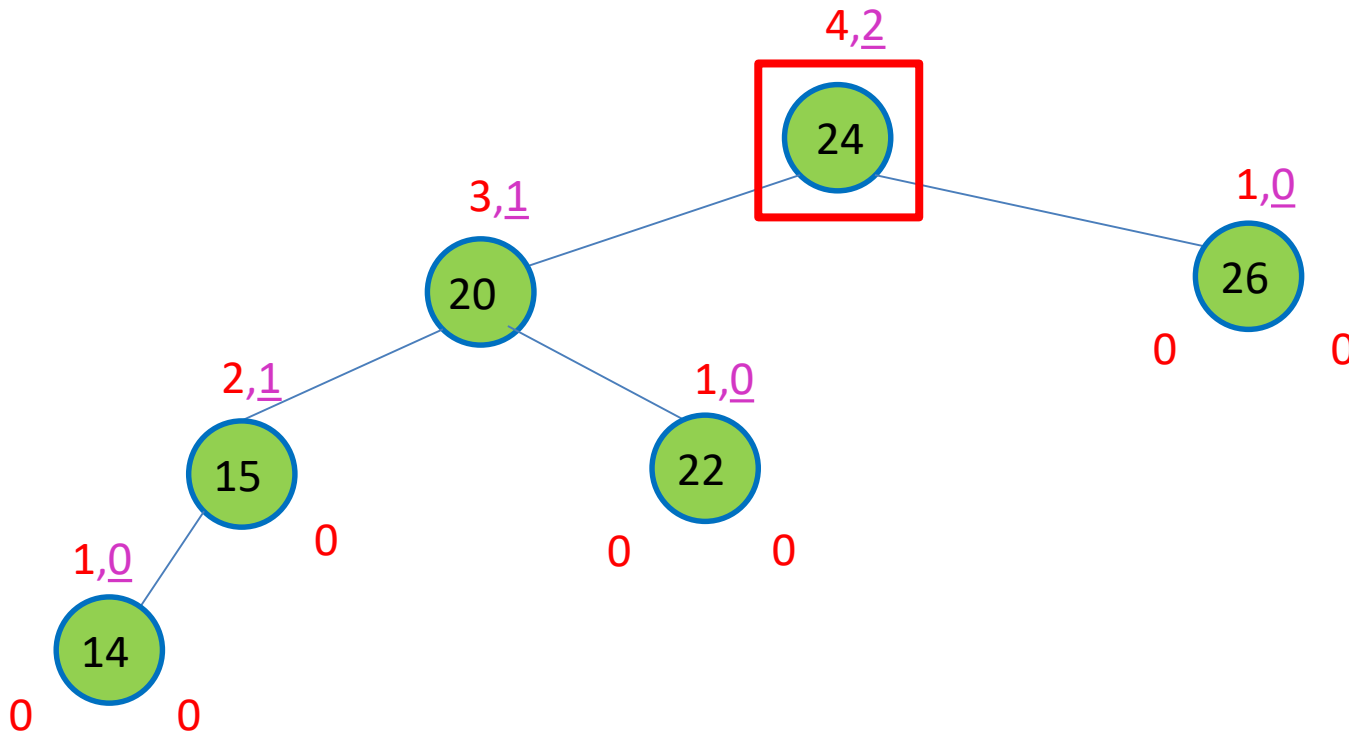
- Let's try a new tree...
  - Is this balanced?



# AVL Tree

## The better BST!

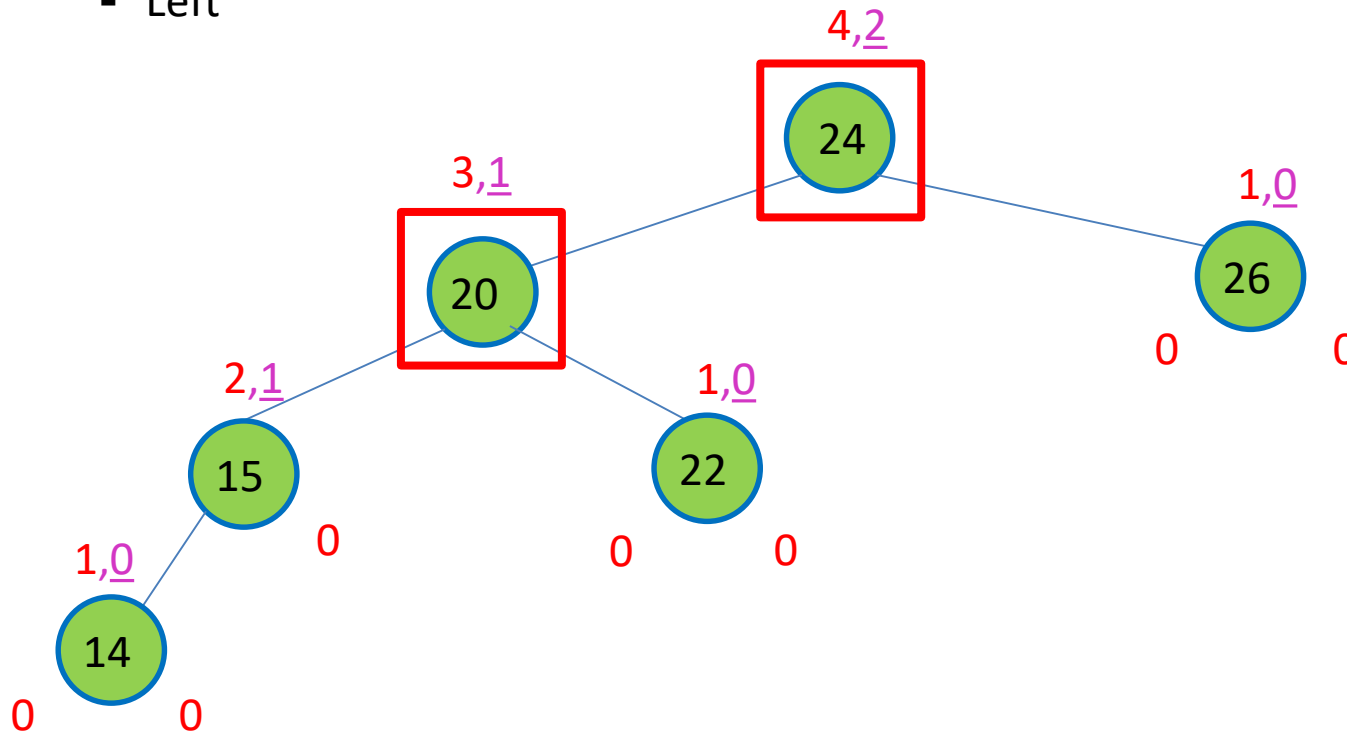
- Let's try a new tree...
  - Is this balanced?
    - No



# AVL Tree

## The better BST!

- Let's try a new tree...
  - Is this balanced?
    - No
    - Left



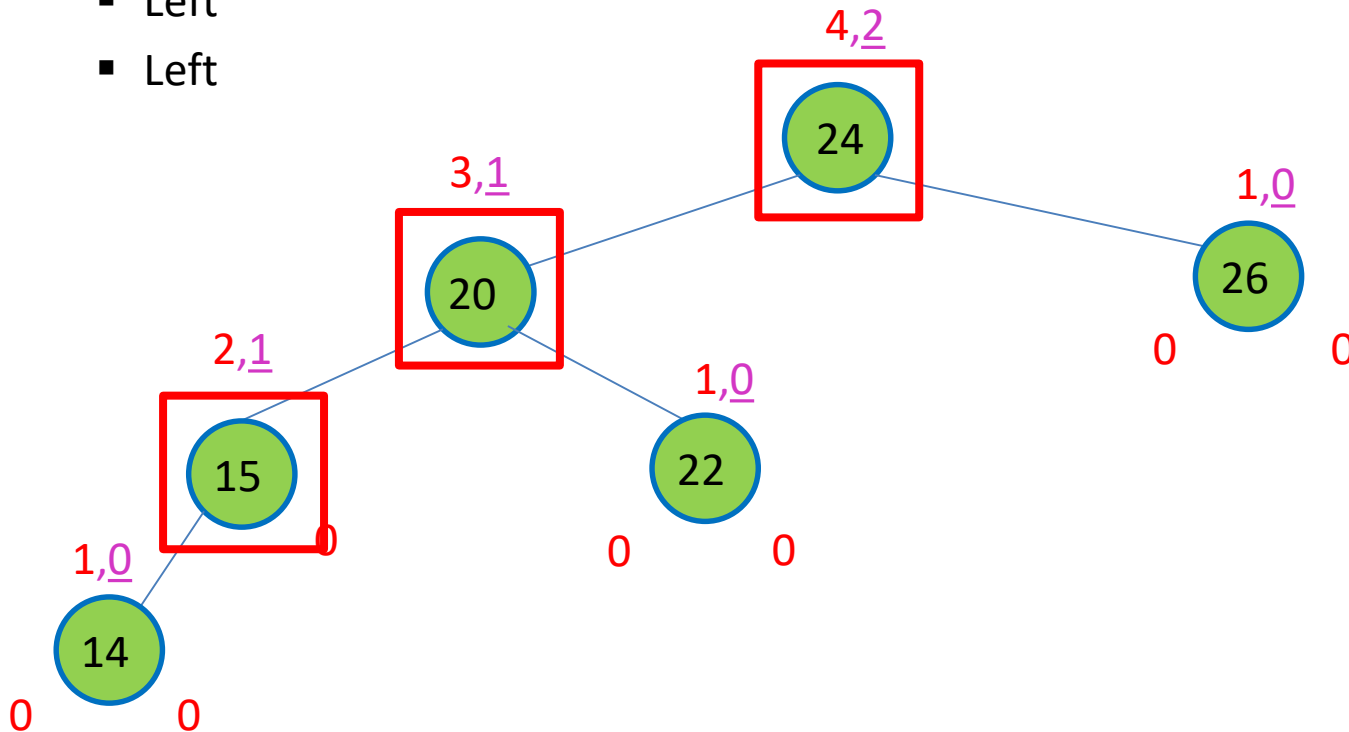
# AVL Tree

## The better BST!

- Let's try a new tree...

- Is this balanced?

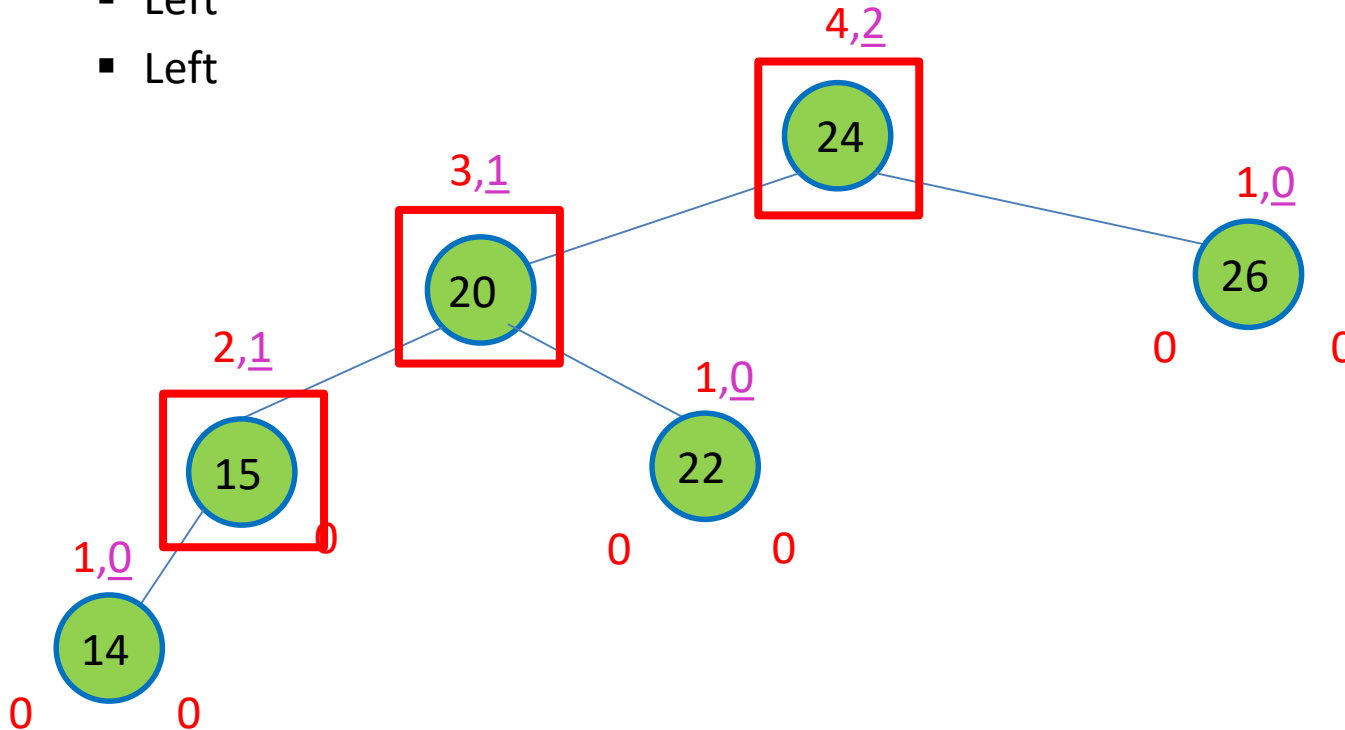
- No
    - Left
    - Left



# AVL Tree

## The better BST!

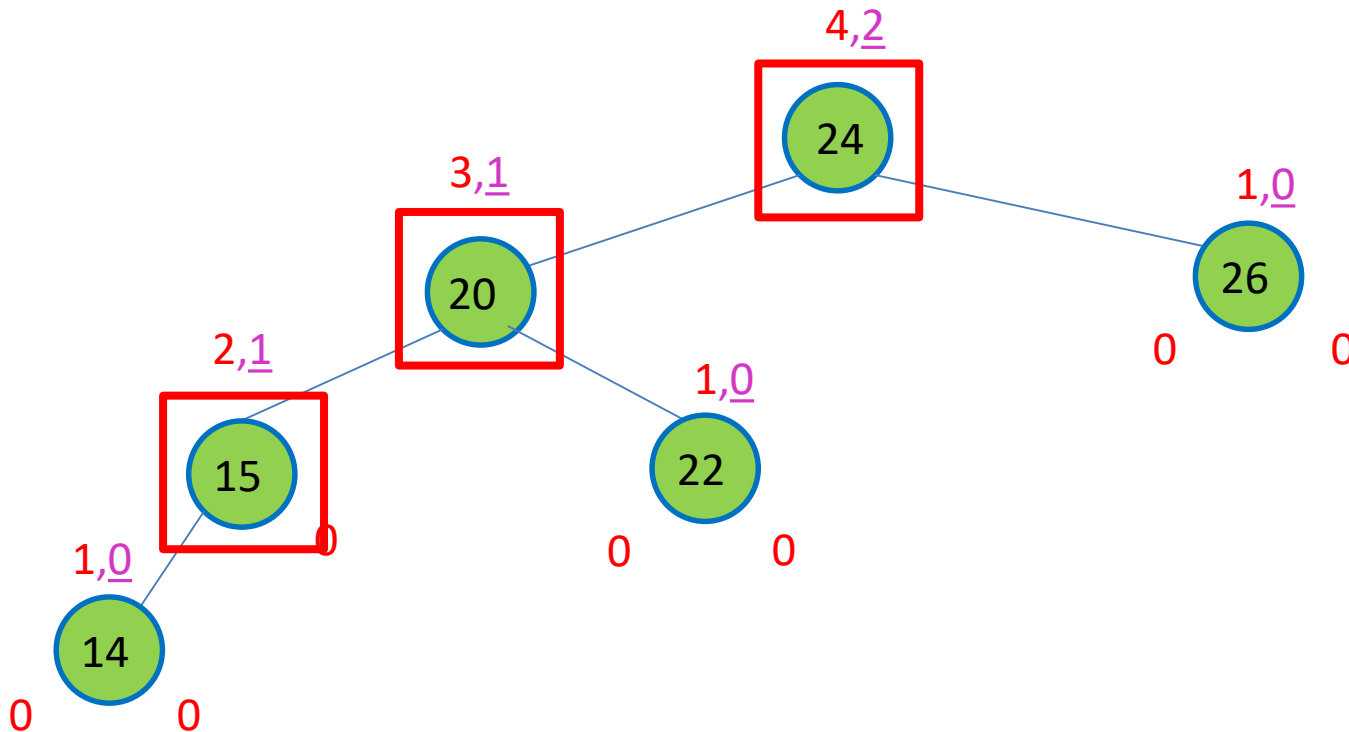
- Let's try a new tree...
  - Is this balanced? Left-Left imbalanced
    - No
    - Left
    - Left



# AVL Tree

## The better BST!

- How to balance?
  - If left-left imbalanced, rotate right

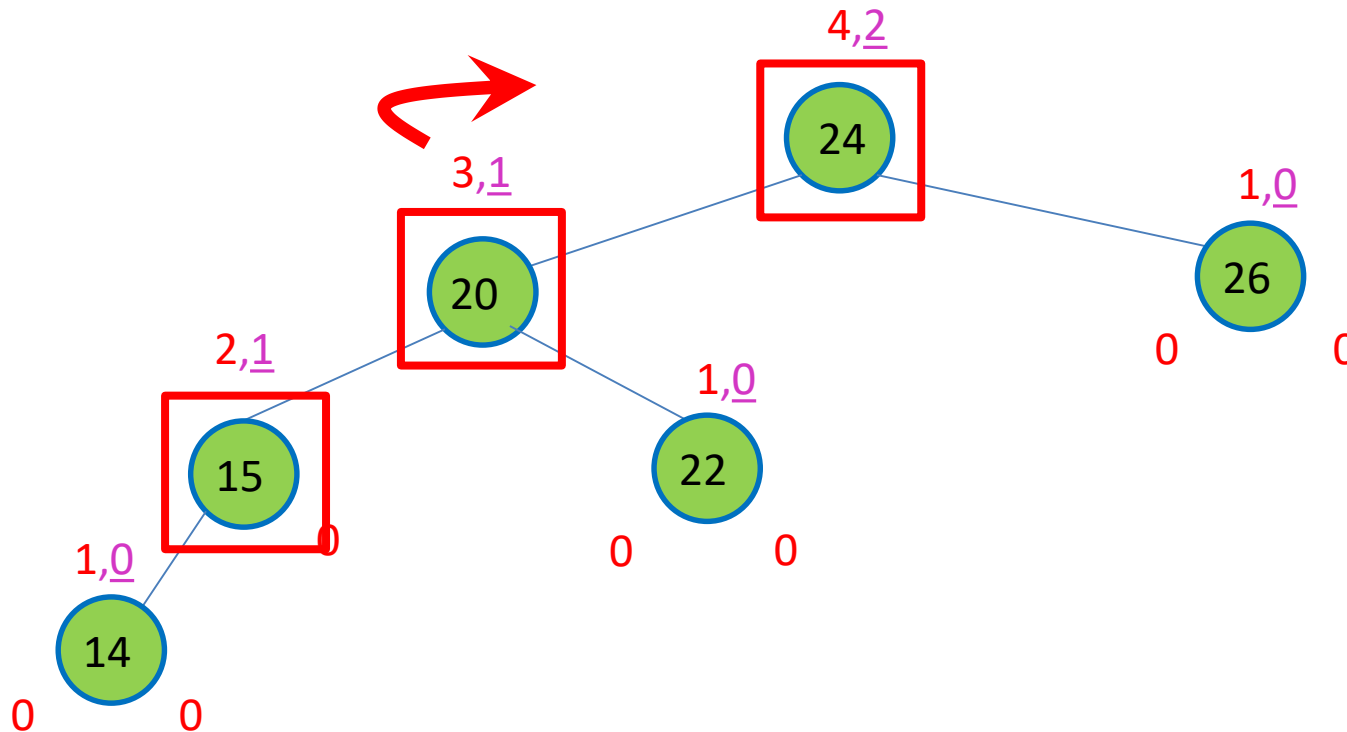




# AVL Tree

## The better BST!

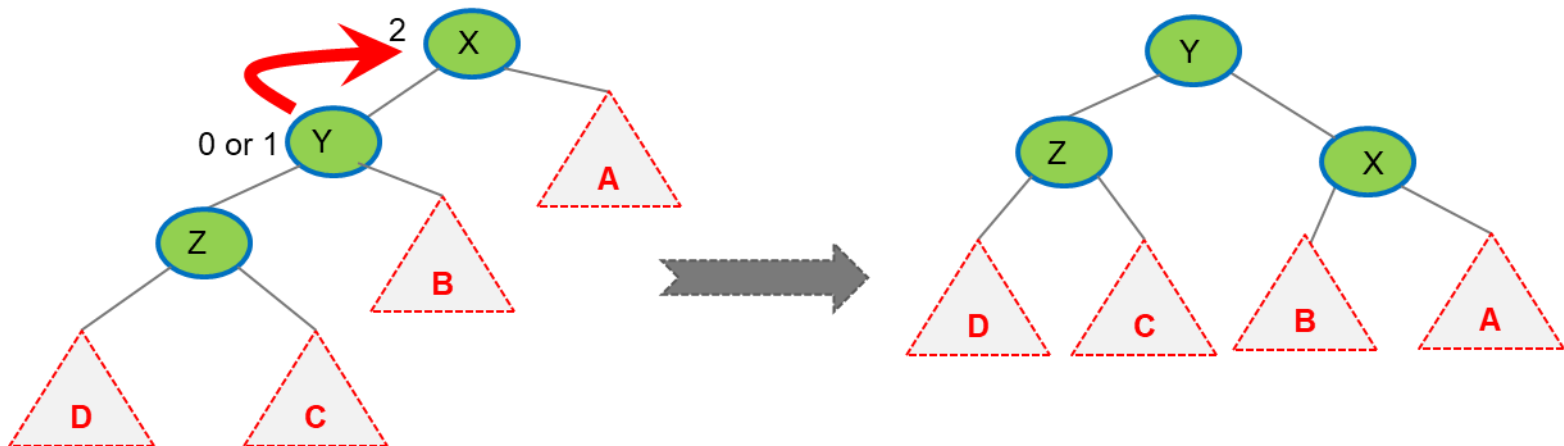
- How to balance?
  - If left-left imbalanced, rotate right



# AVL Tree

## The better BST!

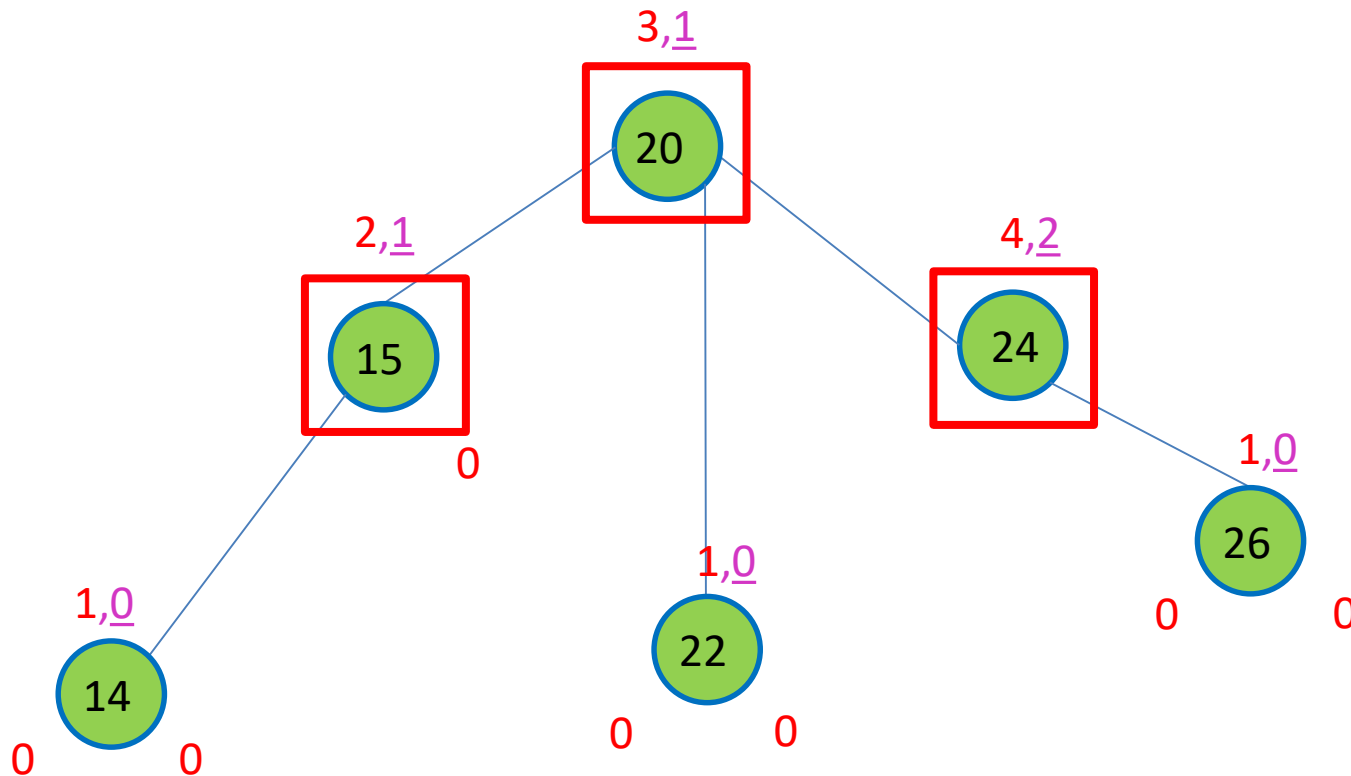
- How to balance?
  - If left-left imbalanced, rotate right
  - If left-right, we rotate left



# AVL Tree

## The better BST!

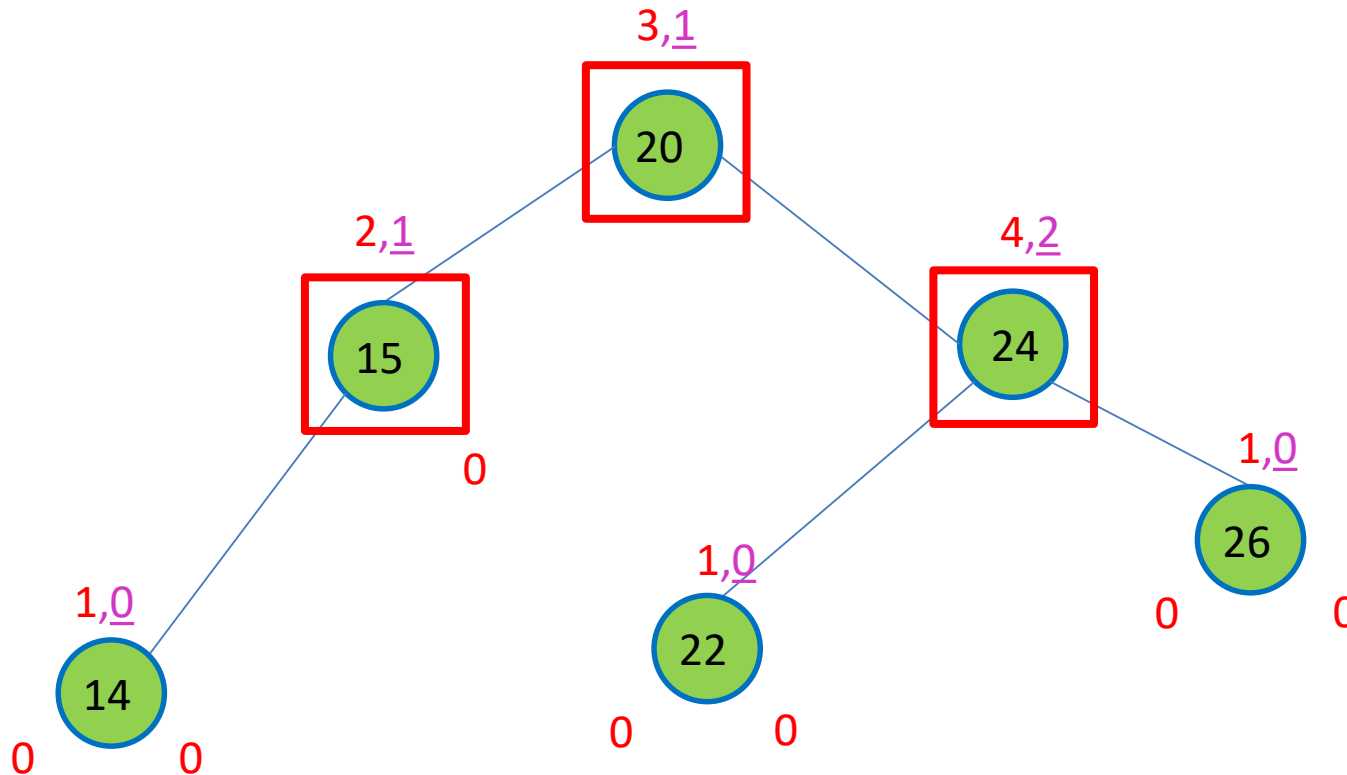
- How to balance?
  - If left-left imbalanced, rotate right



# AVL Tree

## The better BST!

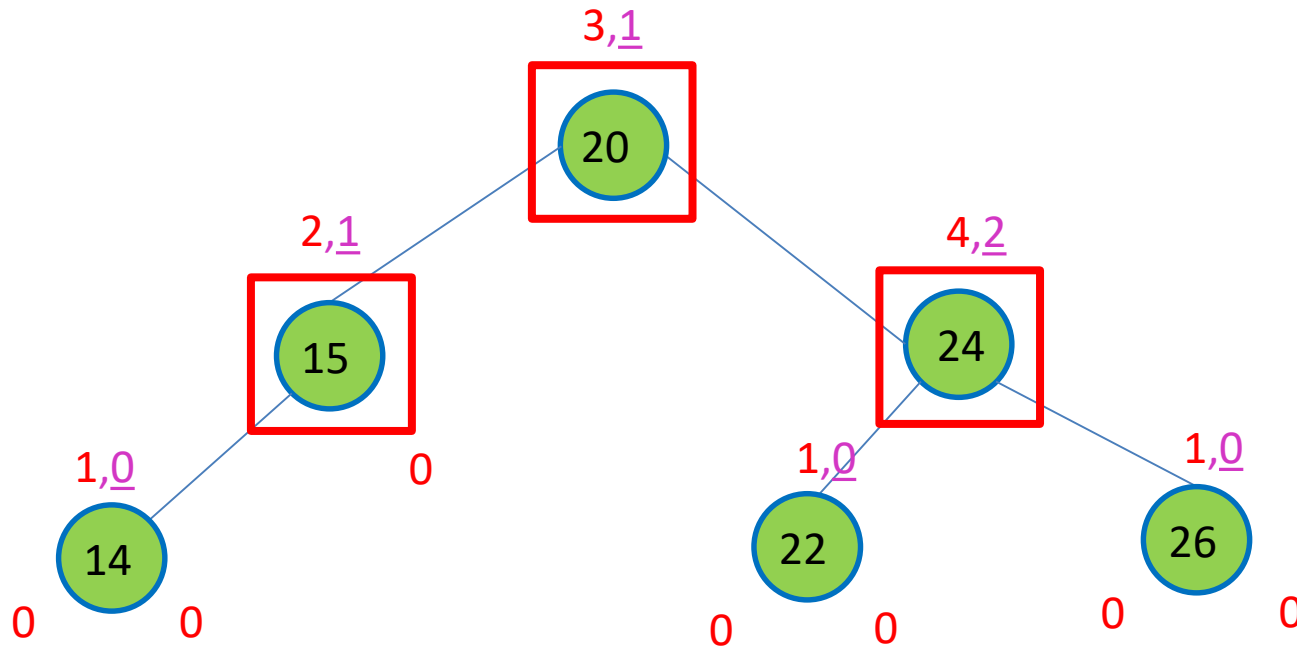
- How to balance?
  - If left-left imbalanced, rotate right



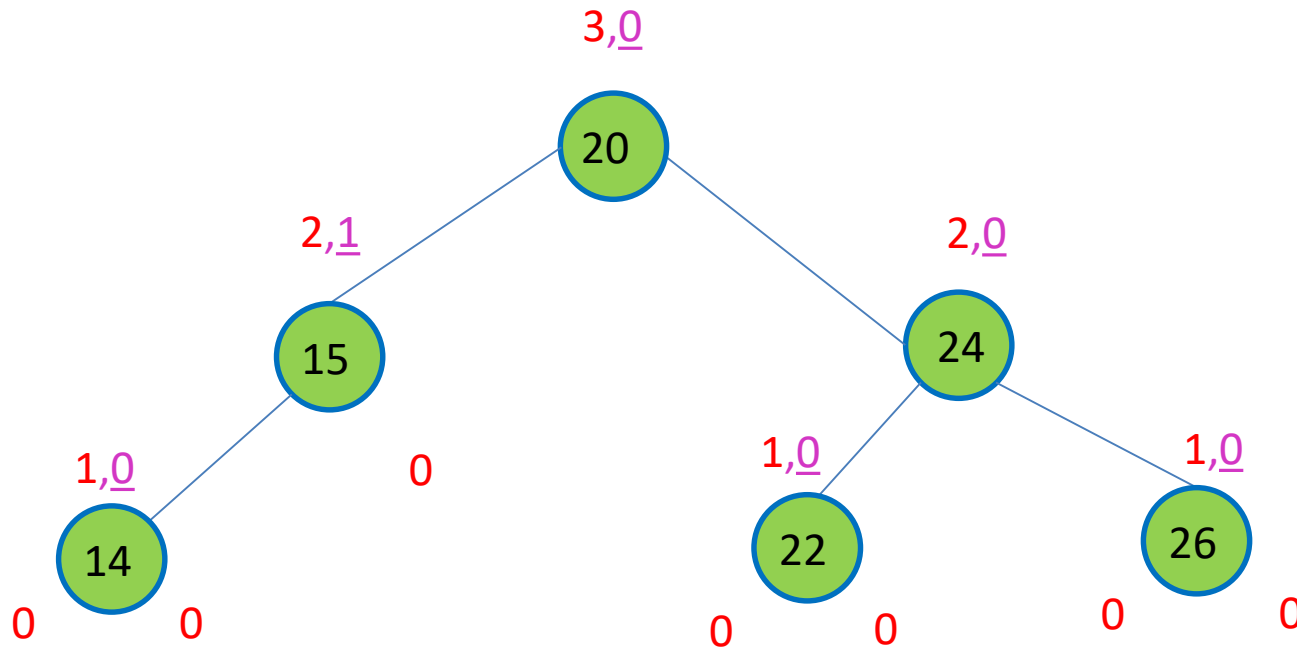
# AVL Tree

## The better BST!

- How to balance?
  - If left-left imbalanced, rotate right



- How to balance?
  - If left-left imbalanced, rotate right
  - Recalculate



# AVL Tree

## The better BST!

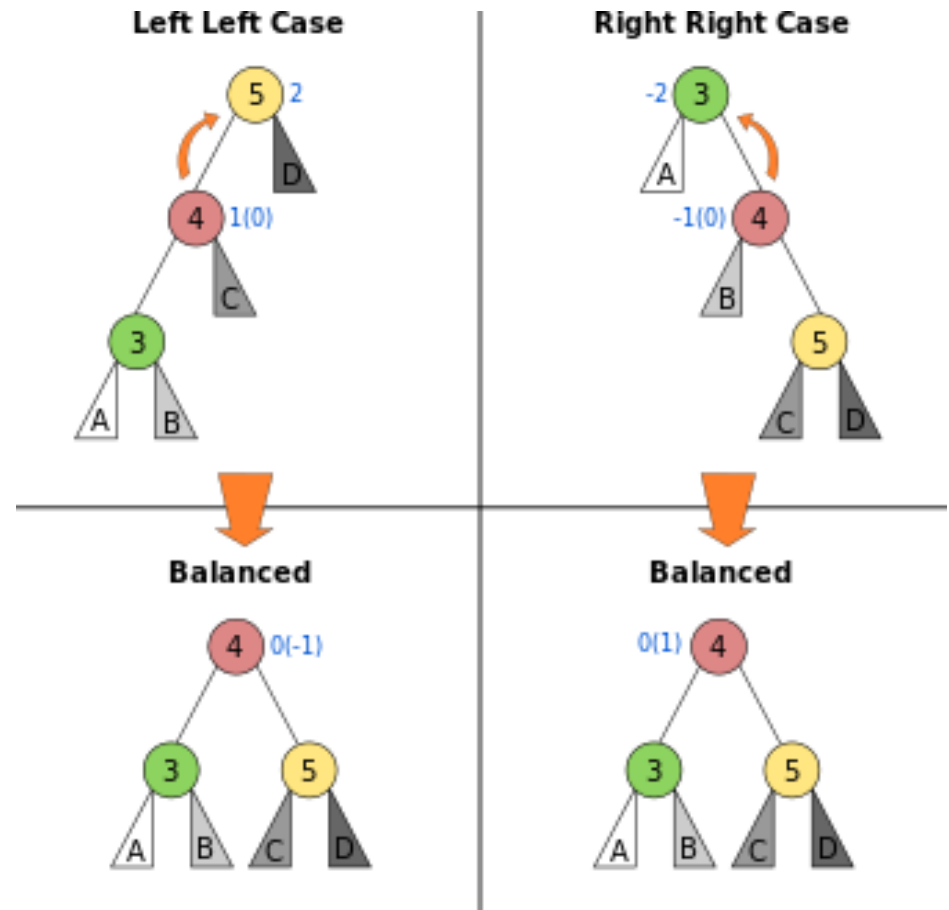
- How to balance?
  - If left-left imbalanced, rotate right
  - Recalculate



# AVL Tree

## The better BST!

- How to balance?
  - If left-left imbalanced, rotate right
  - If right-right, rotate left



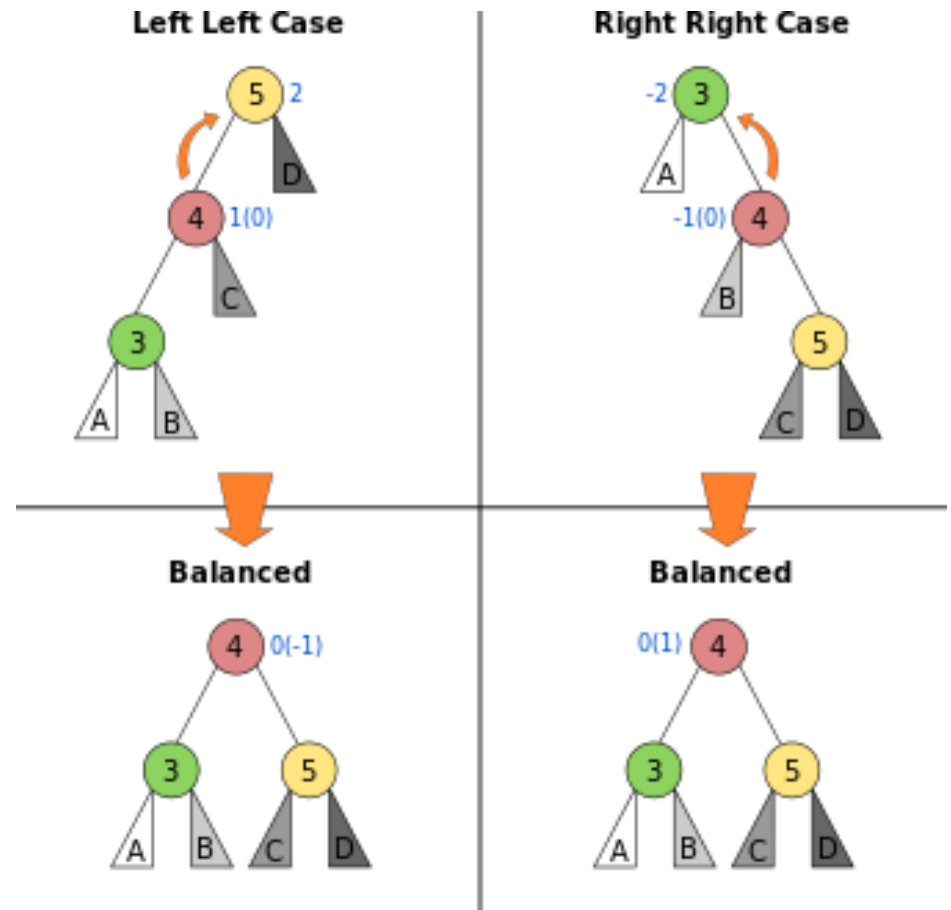


Questions?

# AVL Tree

## The better BST!

- How to balance?
  - If left-left imbalanced, rotate right
  - If right-right, rotate left
- But what if
  - Left-right?
  - Right-left?

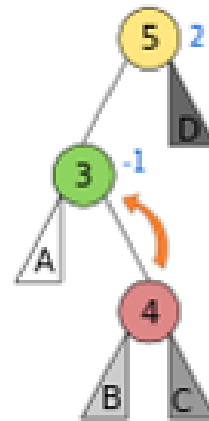


# AVL Tree

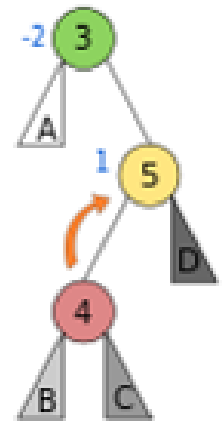
## The better BST!

- How to balance?
  - If left-left imbalanced, rotate right
  - If right-right, rotate left
- But what if
  - Left-right?
  - Right-left?

Left Right Case



Right Left Case

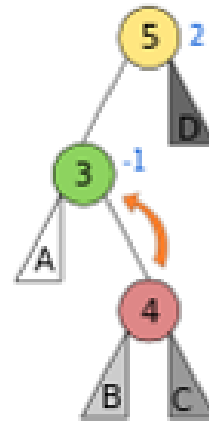


# AVL Tree

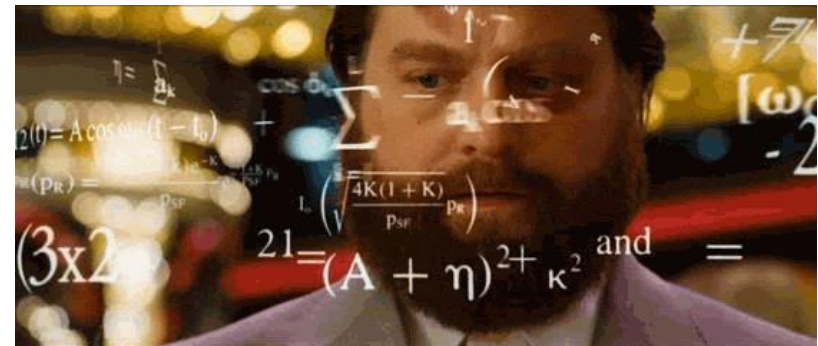
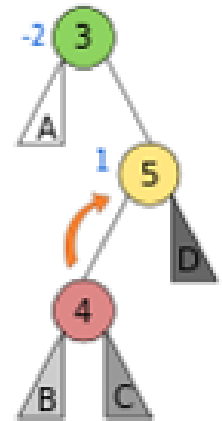
## The better BST!

- How to balance?
  - If left-left imbalanced, rotate right
  - If right-right, rotate left
- But what if
  - Left-right?
  - Right-left?

Left Right Case



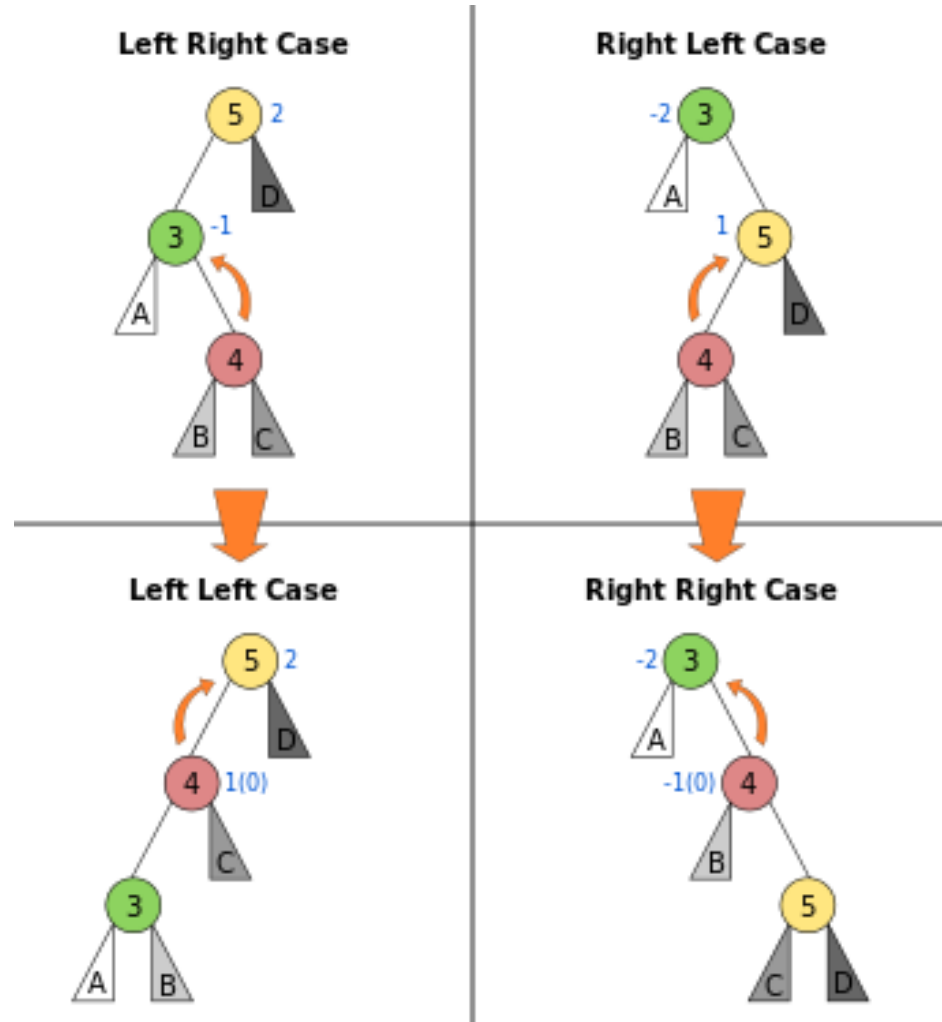
Right Left Case



# AVL Tree

## The better BST!

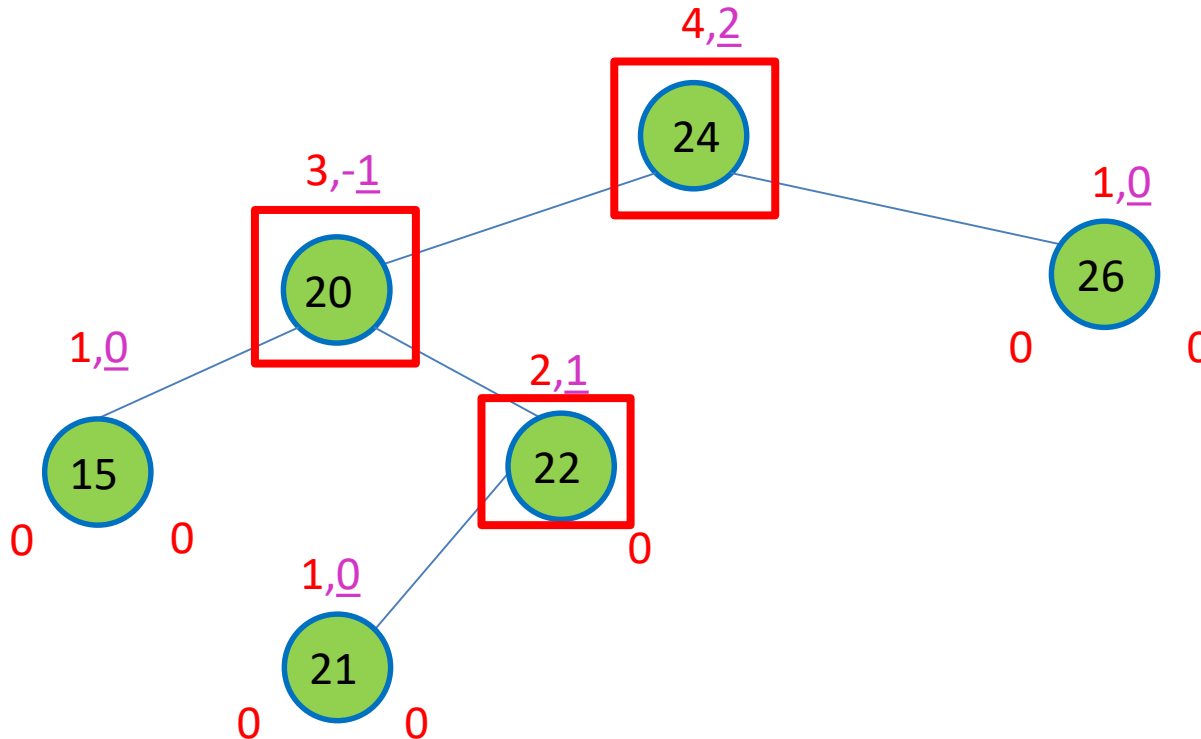
- How to balance?
  - If left-left imbalanced, rotate right
  - If right-right, rotate left
- But what if
  - Left-right?
  - Right-left?
  - We do double rotation



# AVL Tree

## The better BST!

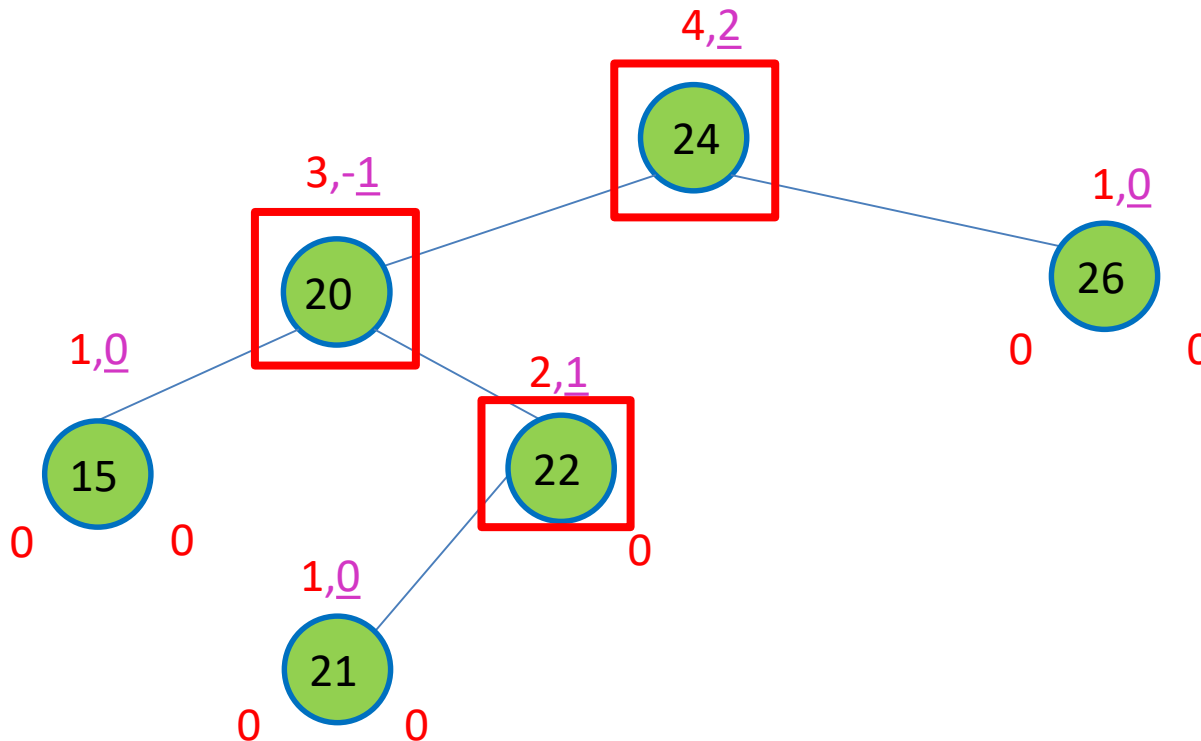
- Left-right imbalanced example



# AVL Tree

## The better BST!

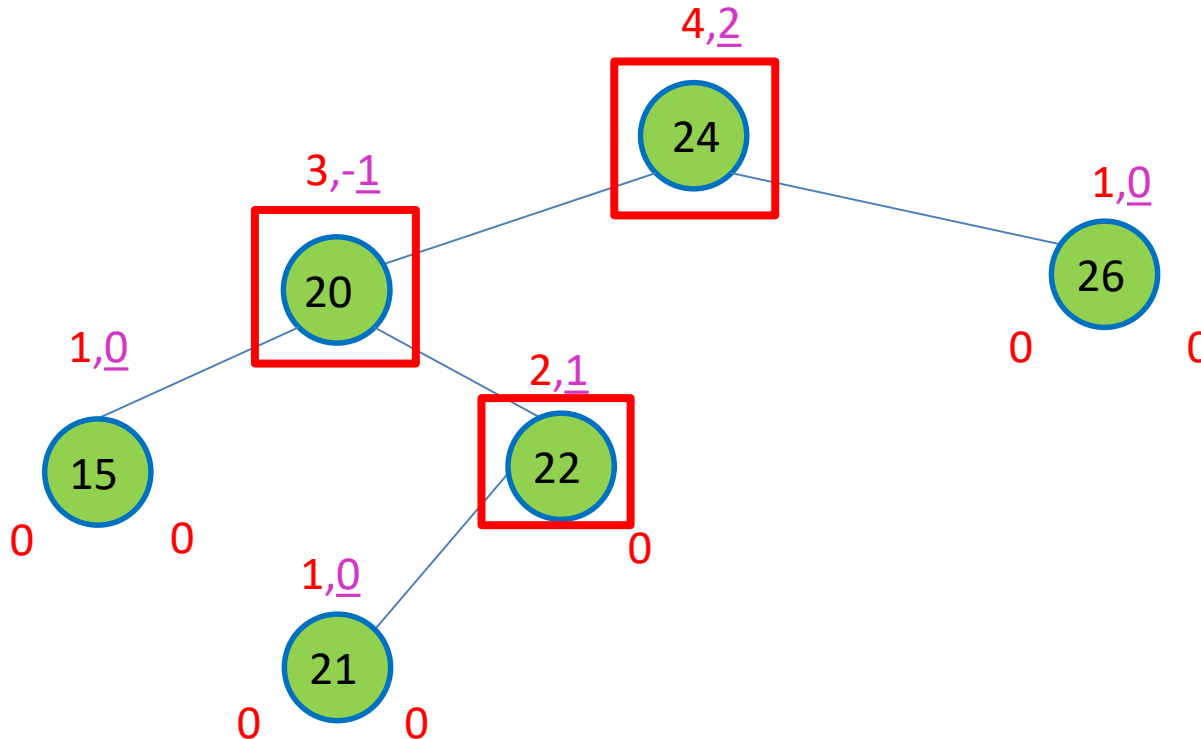
- Left-right imbalanced example
  - How do we balance?



# AVL Tree

## The better BST!

- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**

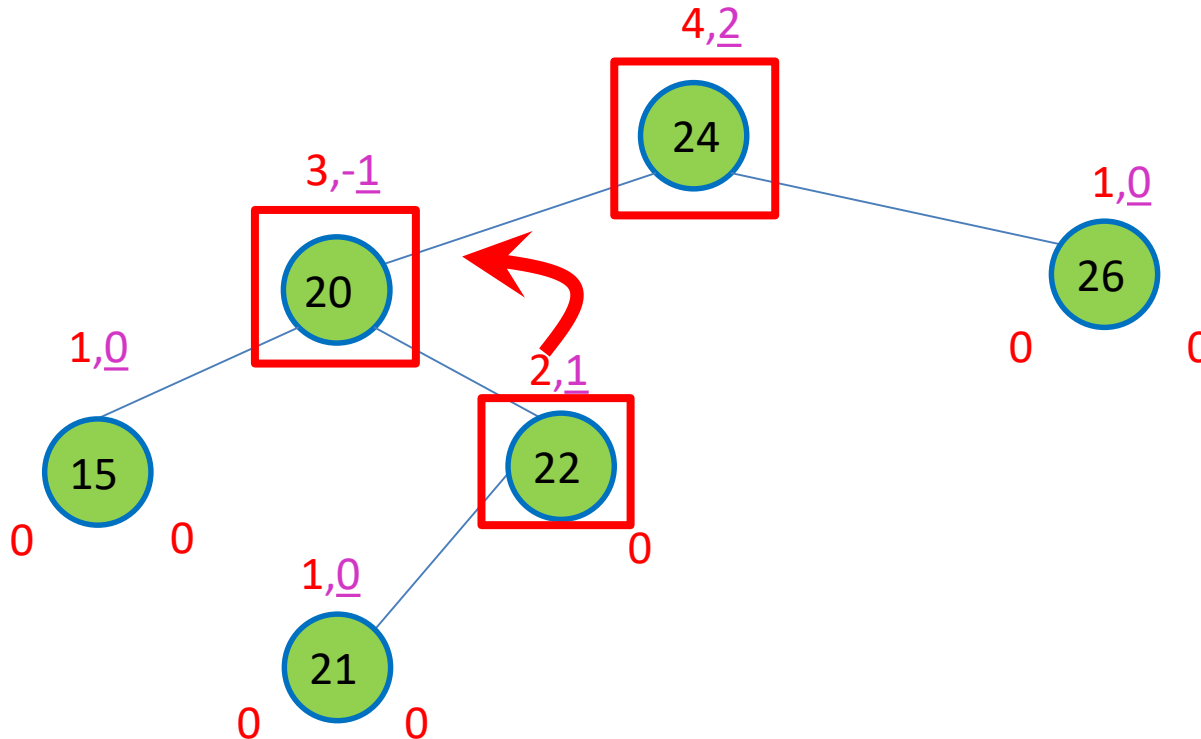




# AVL Tree

## The better BST!

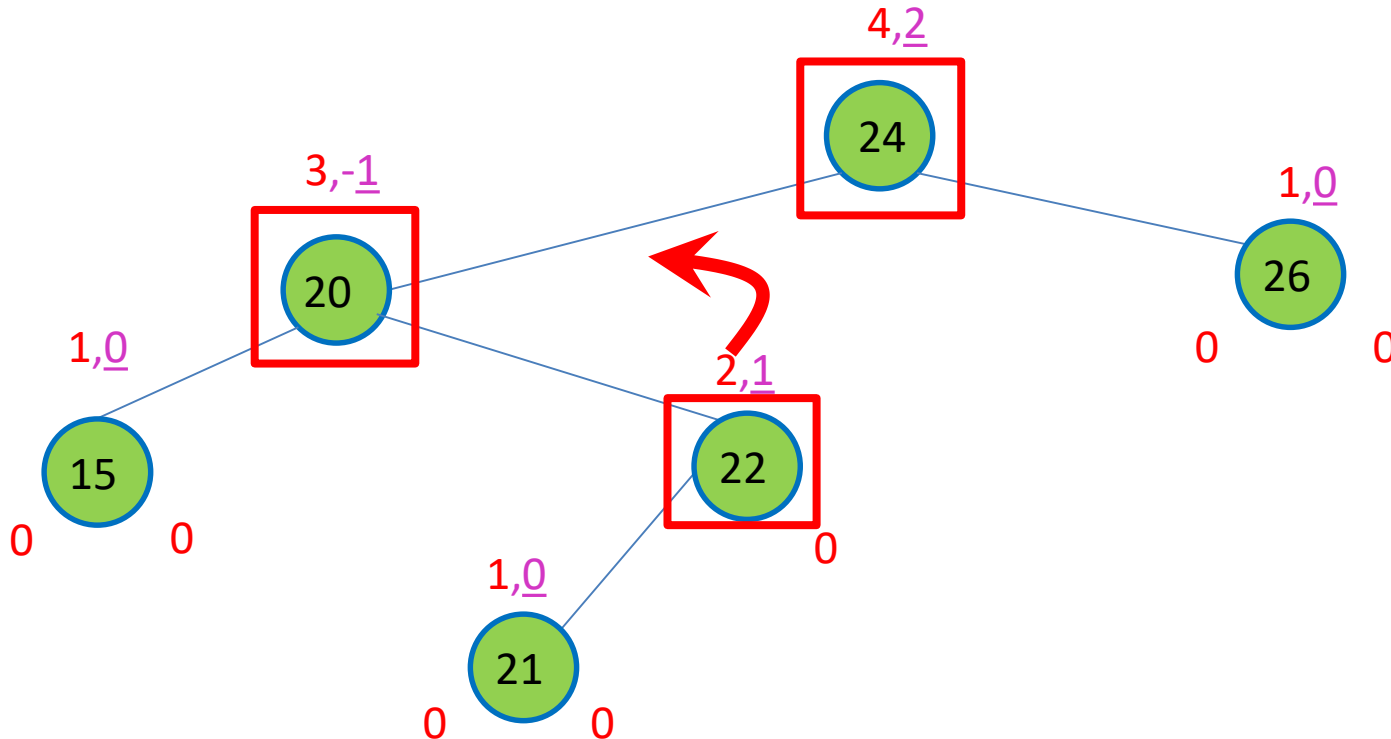
- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**



# AVL Tree

## The better BST!

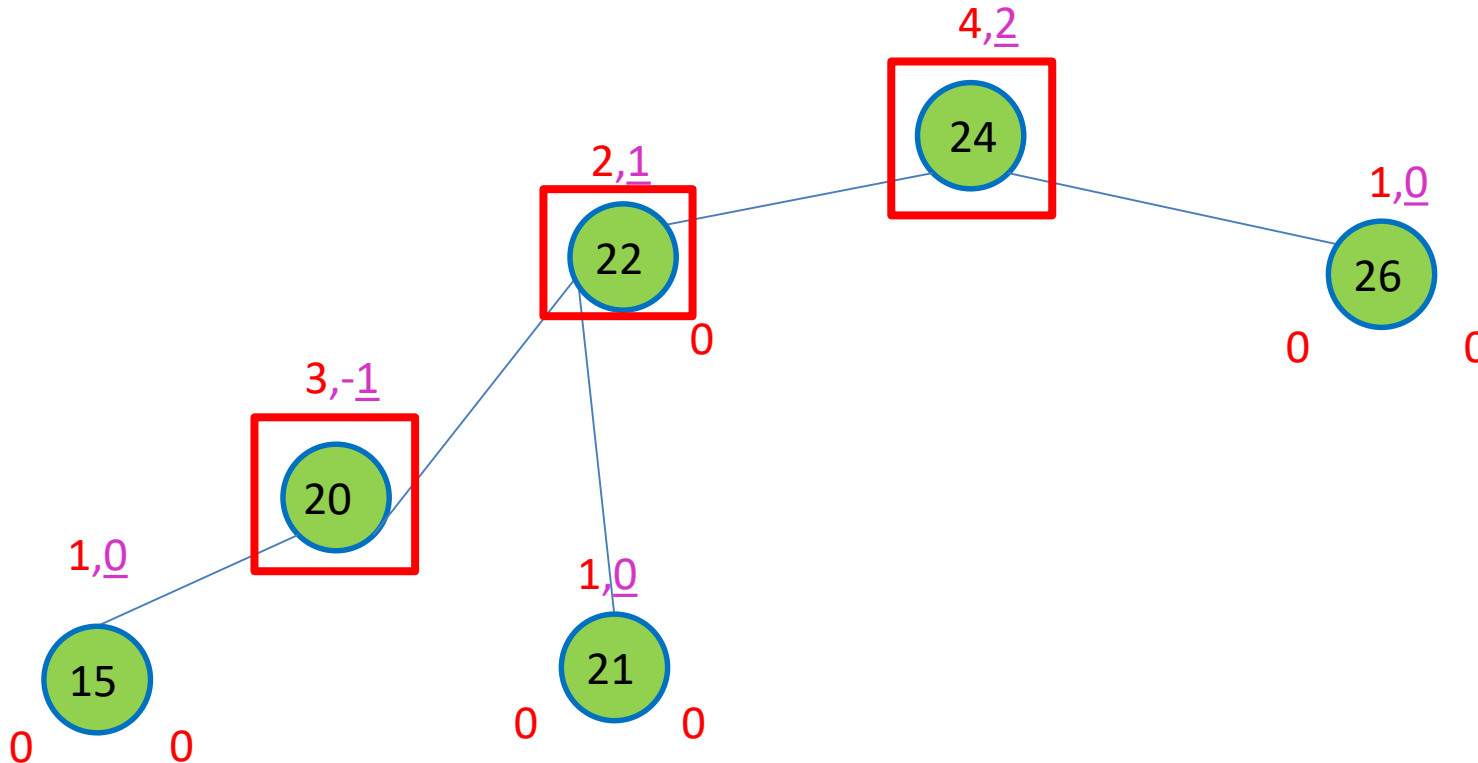
- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**



# AVL Tree

## The better BST!

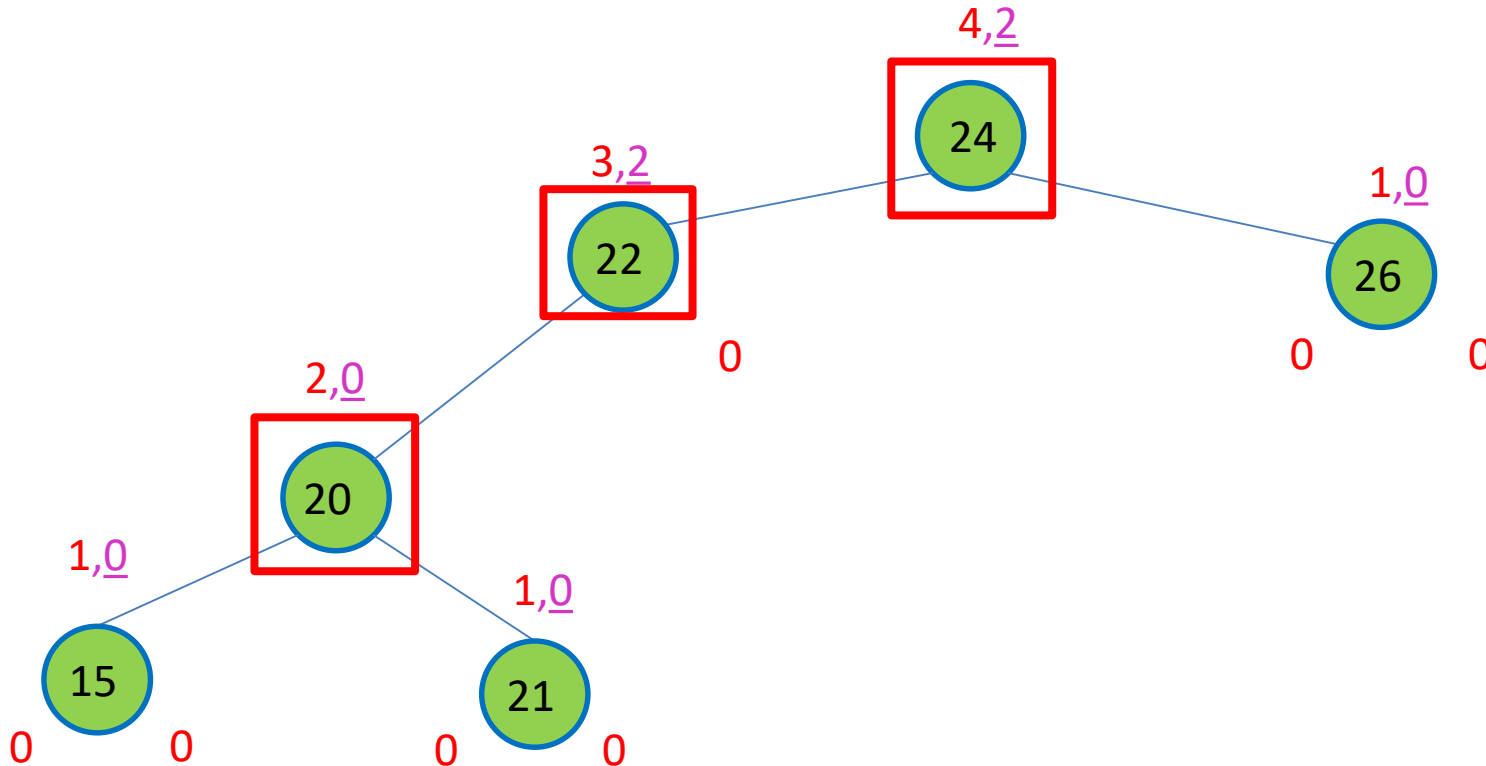
- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**



# AVL Tree

## The better BST!

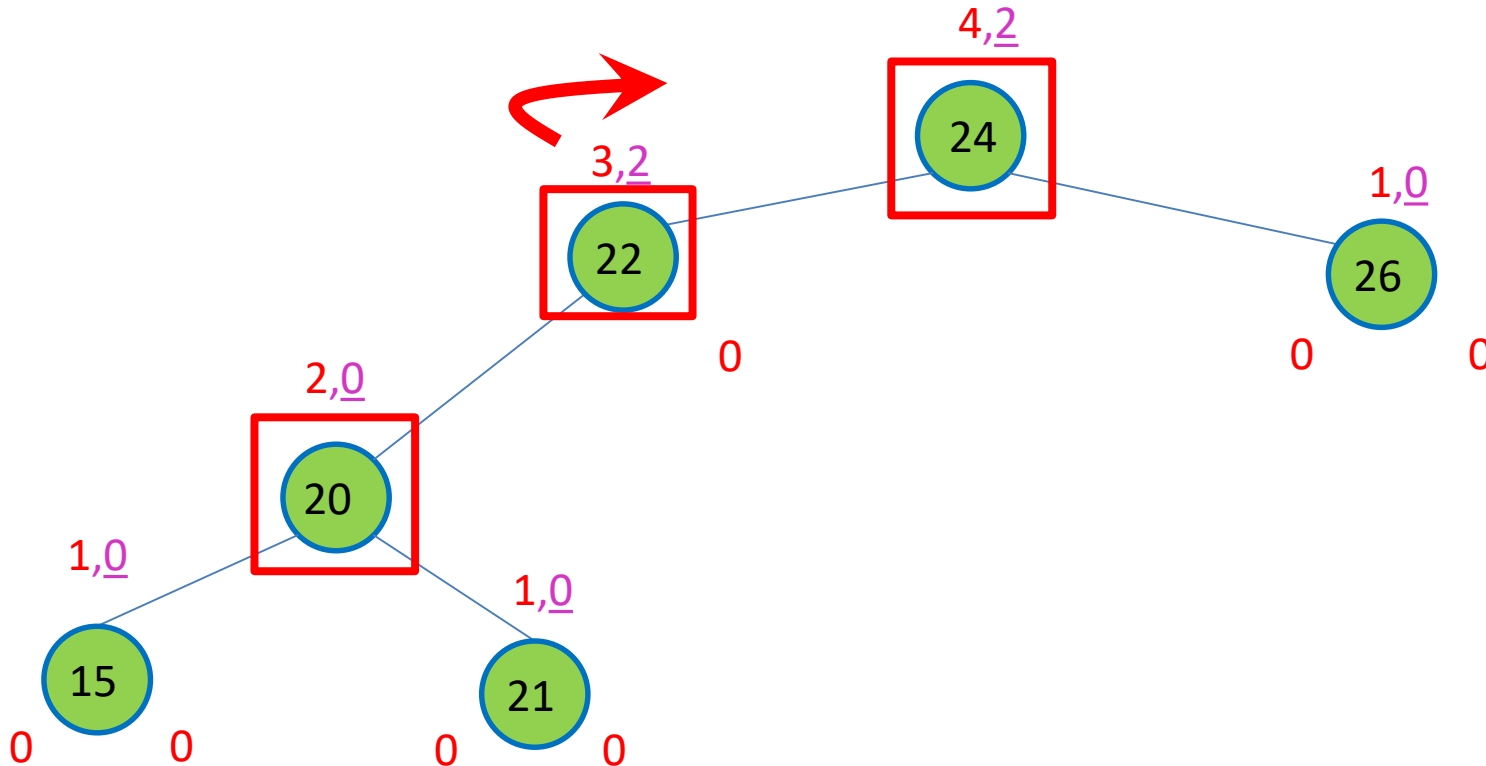
- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**



# AVL Tree

## The better BST!

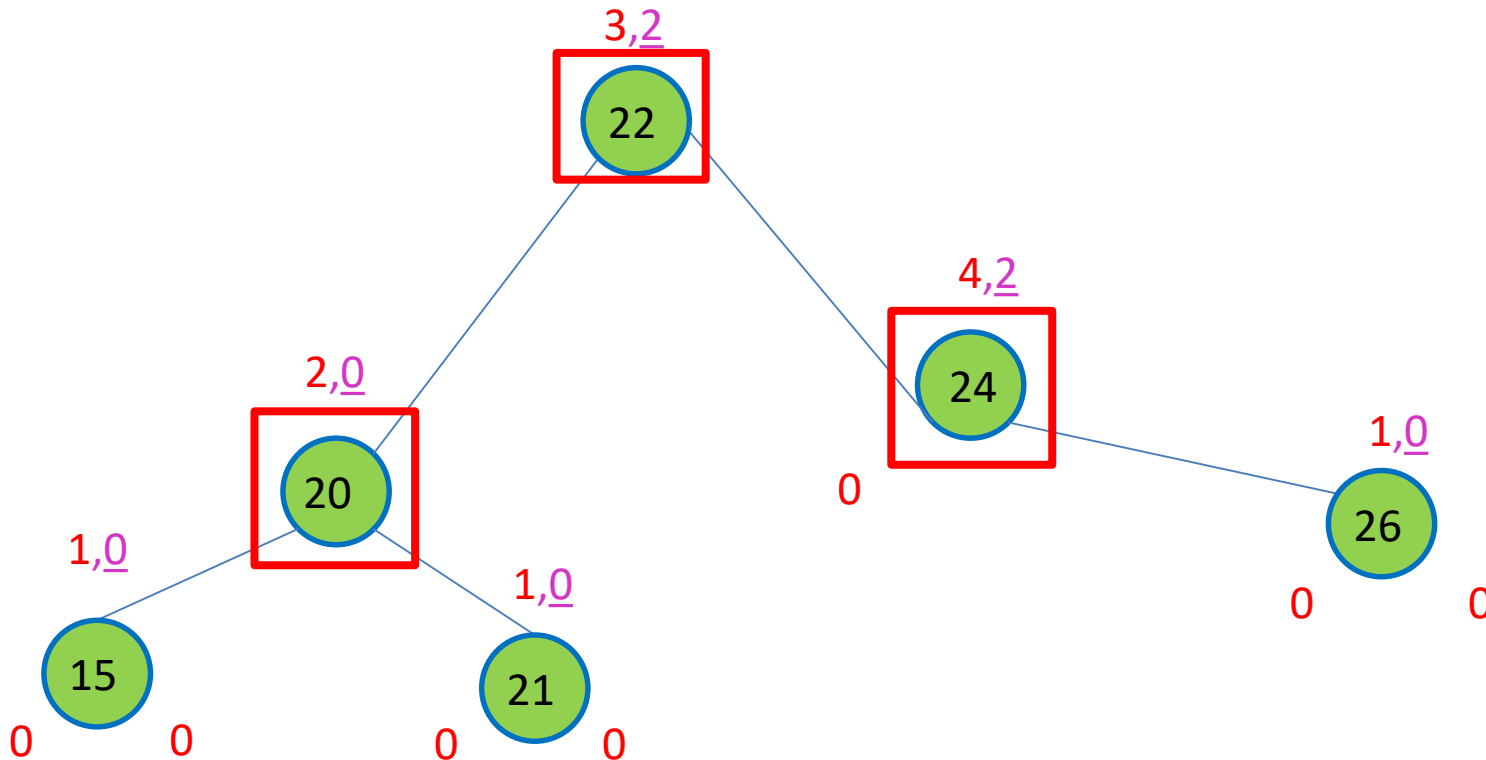
- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**
    - Then rotate right



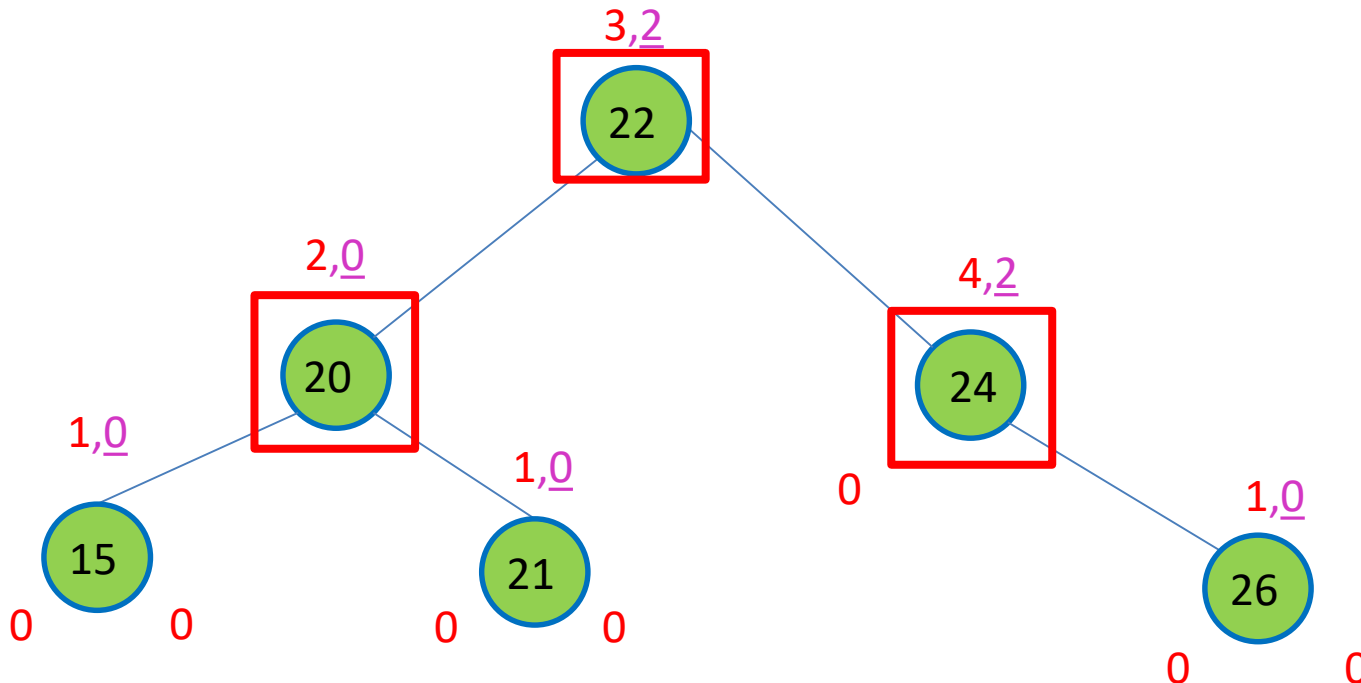
# AVL Tree

## The better BST!

- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**
    - Then rotate right



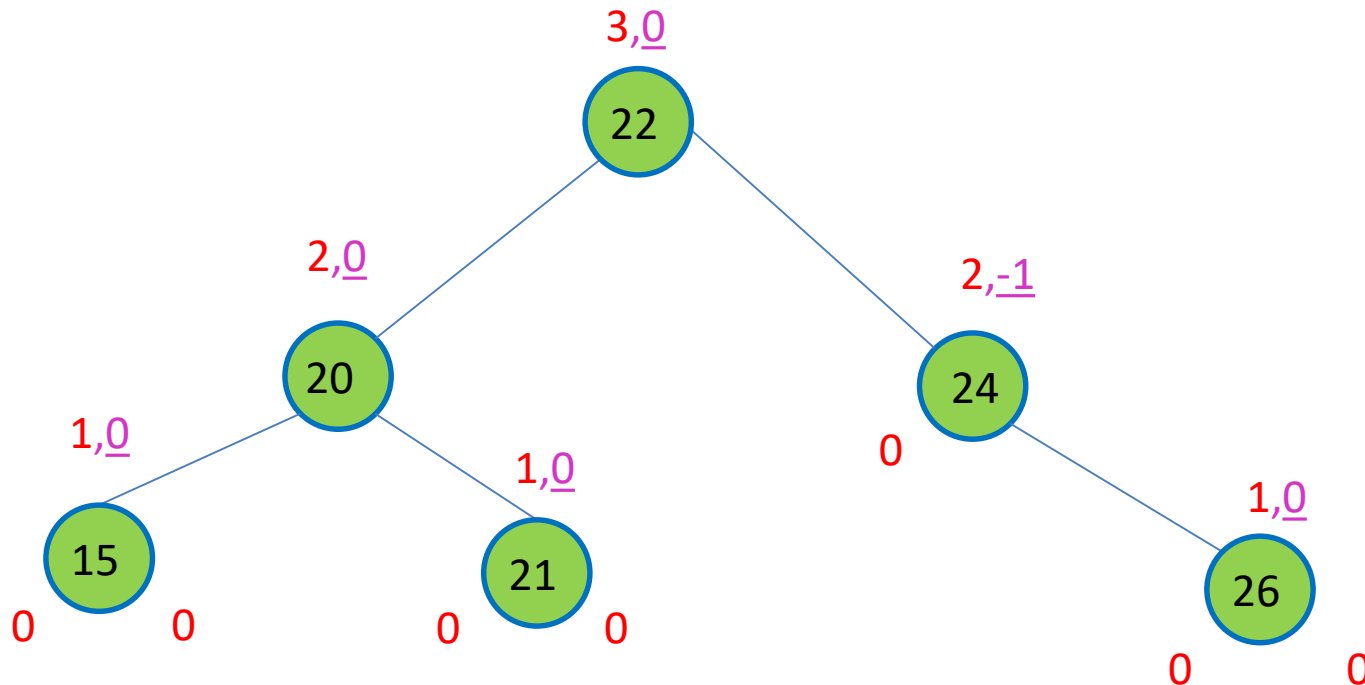
- Left-right imbalanced example
  - How do we balance? **Rotate left**, so it become **left-left imbalanced**
    - Then rotate right



# AVL Tree

## The better BST!

- Left-right imbalanced example
  - Let us check balance again

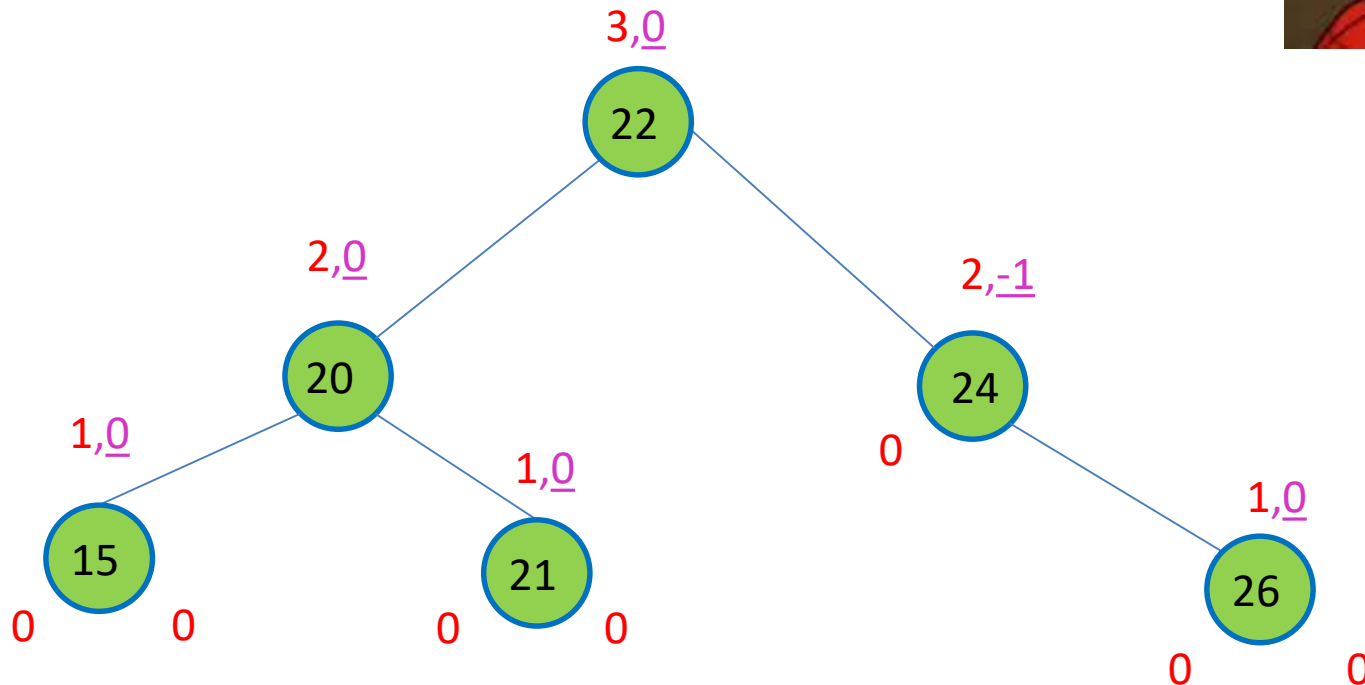




# AVL Tree

## The better BST!

- Left-right imbalanced example
  - Let us check balance again



Questions?

# AVL Tree

## Complexity?

- Everything is the same as BST
  - Insert
  - Delete
  - Search

# AVL Tree

## Complexity?

- Everything is the same as BST
    - Insert
    - Delete
    - Search
- } Check balance after that

# AVL Tree

## Complexity?

- Everything is the same as BST
    - Insert
    - Delete
    - Search
- } Check balance after that
- So what is the balancing cost?

# AVL Tree

## Complexity?

- Everything is the same as BST
  - Insert
  - Delete
  - Search

} Check balance after that
- So what is the balancing cost?
  - Check balance from the bottom
  - If we find imbalance, we balance (at most 2 rotations – a constant)

# AVL Tree

## Complexity?

- Everything is the same as BST
    - Insert
    - Delete
    - Search
- } Check balance after that
- So what is the balancing cost?
    - Check balance from the bottom
    - If we find imbalance, we balance (at most 2 rotations – a constant)
    - So this would be  $O(\log N)$

# AVL Tree

## Complexity?

- Everything is the same as BST
    - Insert
    - Delete
    - Search
- } Check balance after that
- 
- So what is the balancing cost?
    - Check balance from the bottom
      - Done via recursion...
    - If we find imbalance, we balance (at most 2 rotations – a constant)
    - So this would be  $O(\log N)$



# AVL Tree

## Complexity?

- Everything is the same as BST
    - Insert
    - Delete
    - Search
- } Check balance after that
- So what is the balancing cost?
    - Check balance from the bottom
      - Done via recursion...
        - Note: This isn't  $O(N)$  because we would just update and not calculate from scratch as how we do by hand
    - If we find imbalance, we balance (at most 2 rotations – a constant)
    - So this would be  $O(\log N)$

# AVL Tree

## Complexity?

- Everything is the same as BST
  - Insert
  - Delete
  - Search

} Check balance after that
  
- So what is the balancing cost?
  - Check balance from the bottom
    - Done via recursion...
      - Note: This isn't  $O(N)$  because we would just update and not calculate from scratch as how we do by hand
  - If we find imbalance, we balance (at most 2 rotations – a constant)
  - So this would be  $O(\log N)$
  - Note: Analyzed further in tutorial supplementary question

Questions?

- Now let us look at Ian's way...

- Now let us look at Ian's way...
  - Let us model it like how exams would ask

- Now let us look at Ian's way...
  - Let us model it like how exams would ask
  - Give me a list of numbers...

- Now let us look at Ian's way...
  - Let us model it like how exams would ask
  - Give me a list of numbers...
  - I will insert these into a AVL tree, checking balance

- Now let us look at Ian's way...
  - Let us model it like how exams would ask
  - Give me a list of numbers...
  - I will insert these into a AVL tree, checking balance
  - Then give me a list of numbers...
  - I will delete these from the AVL tree, checking balance



- Now let us look at Ian's way...
  - Let us model it like how exams would ask
  - Give me a list of numbers...
  - I will insert these into a AVL tree, checking balance
  - Then give me a list of numbers...
  - I will delete these from the AVL tree, checking balance
  
- This will be our active learning activity!
  - And I won't repeat in the tutorial >.<

# AVL Tree

## Ian's way

- Now let us look at Ian's way...
  - Let us model it like how exams would ask
  - Give me a list of numbers...
  - I will insert these into a AVL tree, checking balance
  - Then give me a list of numbers...
  - I will delete these from the AVL tree, checking balance
- This will be our active learning activity!
  - And I won't repeat in the tutorial >.<
- Since we don't have face to face, we use RNG



Questions?

Thank You