CSE 373

OCTOBER 13TH – AVL TREES

ASSORTED MINUTIAE

- **P1 scripts run on Sunday**
  - Apologies for part 1 script failures
  - You will receive the part 1 grade for your part 2 code
  - Given opportunity next week to get points back
  - **Leave team member as comment on canvas**

TODAY'S LECTURE

- **AVL Trees**
  - Balance
  - Implementation

REVIEW

- **AVL Trees**

REVIEW

- **AVL Trees**
  - BST trees with AVL property

REVIEW

- **AVL Trees**
  - BST trees with AVL property
  - Abs(height(left) – height(right)) <= 1

REVIEW

- **AVL Trees**
  - BST trees with AVL property
  - Abs(height(left) – height(right)) <= 1
  - Heights of subtrees can differ by at most one

REVIEW

- **AVL Trees**
  - BST trees with AVL property
  - Abs(height(left) – height(right)) <= 1
  - Heights of subtrees can differ by at most one
  - This property must be preserved throughout the tree

AVL OPERATIONS

- **Since AVL trees are also BST trees, they should support the same functionality**
  - Insert(key k, value v)
  - *Find(key k): Same as BST!*
  - *Delete(key k)*:
- **For insert, we should maintain AVL property as we build**

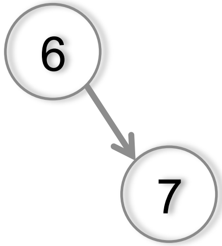
AVL OPERATIONS

- **Insert(key k, value v):**
  - Insert the key value pair into the dictionary
  - Verify that balance is maintained
  - If not, correct the tree
- **How do we correct the tree?**

AVL INSERT

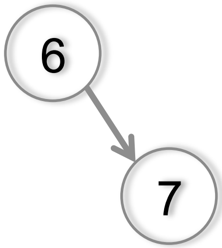
- **Start with the single root**

AVL INSERT



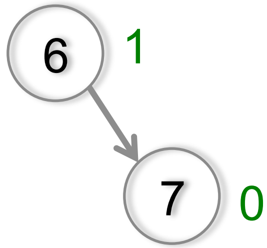
- **Add 7 to the tree**

AVL INSERT



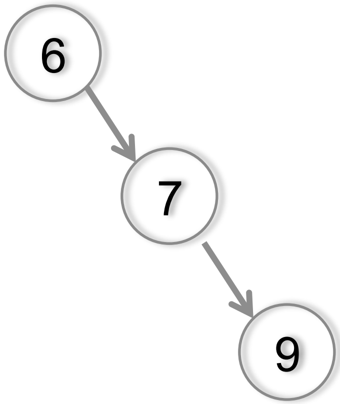
- Add 7 to the tree. Is balance preserved?

AVL INSERT



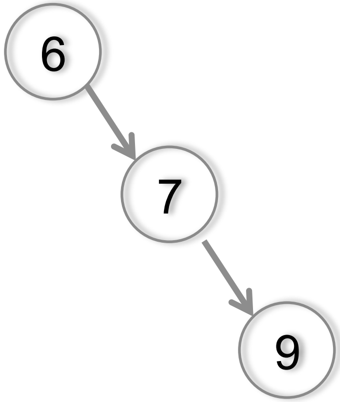
- **Add 7 to the tree. Is balance preserved?**
  - Yes

AVL INSERT



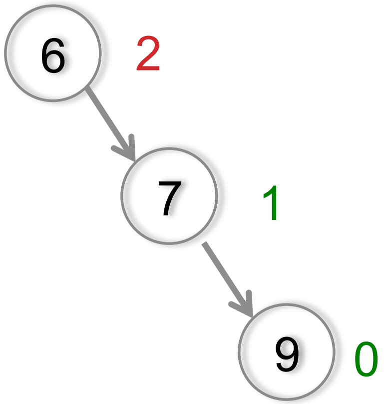
- **Add 9 to the tree**

AVL INSERT



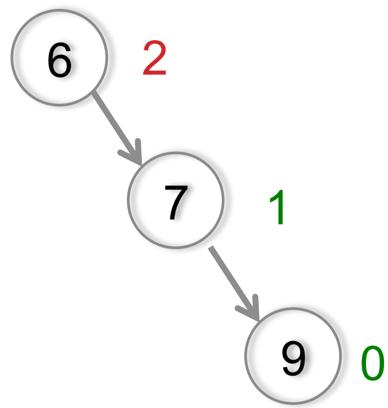
- **Add 9 to the tree. Is balance preserved?**

AVL INSERT



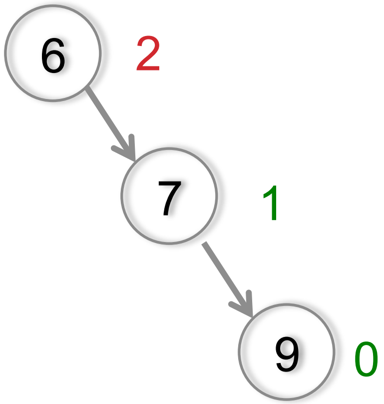
- **Add 9 to the tree. Is balance preserved?**
  - No.

AVL INSERT

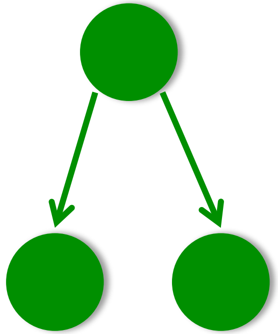


- **How do we correct this imbalance?**

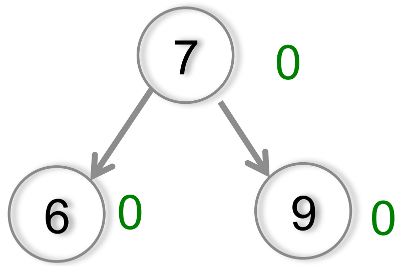
AVL INSERT



- **What shape do we want?**
  - What then do we have as the root?



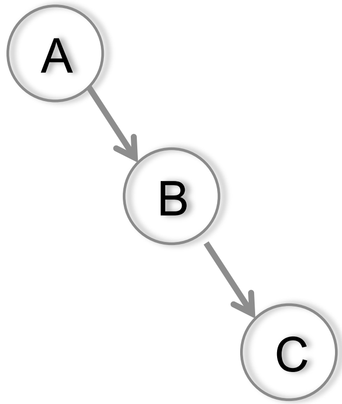
AVL INSERT



- **Since 7 must be the root, we “rotate” that node into position.**

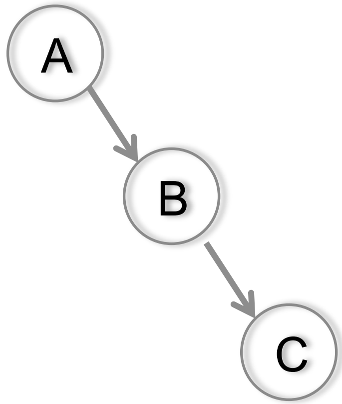
AVL "ROTATION"

- **To correct this case:**
  - B must become the root



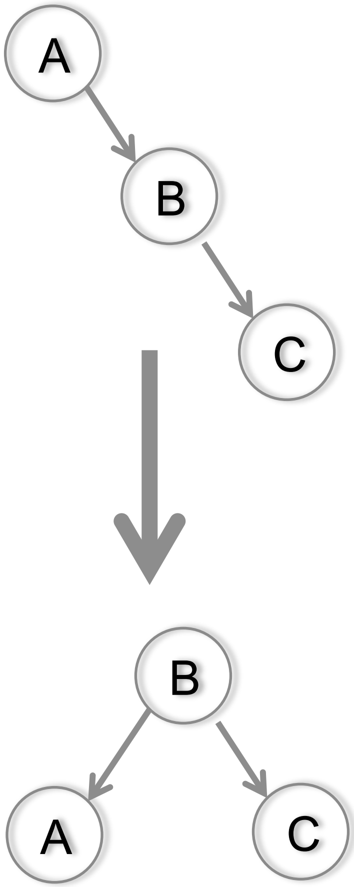
AVL "ROTATION"

- **To correct this case:**
  - B must become the root
  - We rotate B to the root position



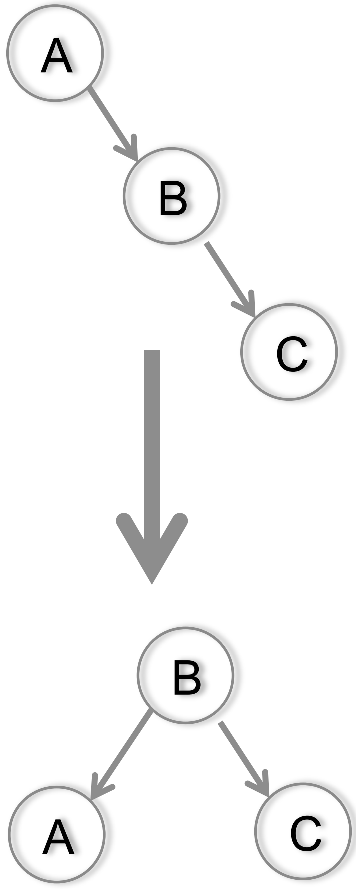
AVL “ROTATION”

- **To correct this case:**
  - B must become the root
  - We rotate B to the root position
  - A becomes the left child of B



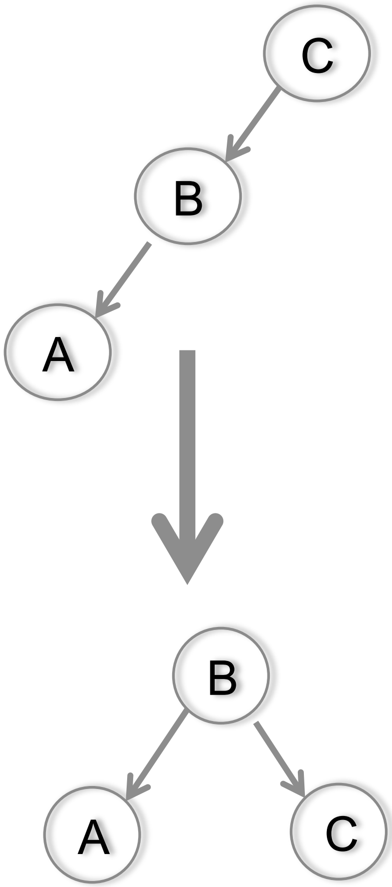
AVL “ROTATION”

- **To correct this case:**
  - B must become the root
  - We rotate B to the root position
  - A becomes the left child of B
  - This is called the “left rotation”



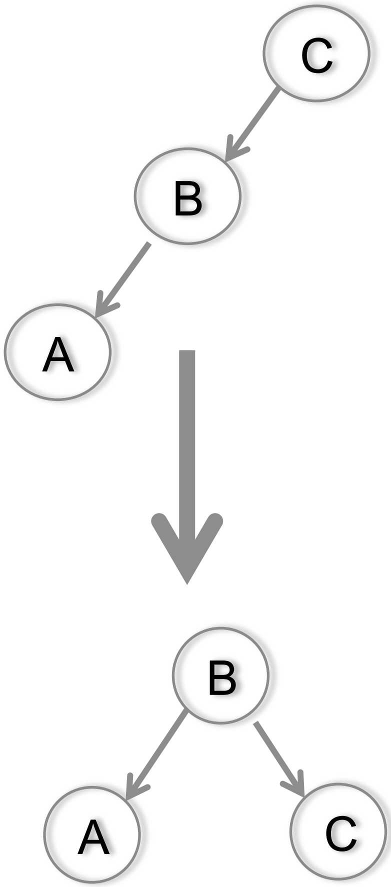
AVL "ROTATION"

- **Right rotation**



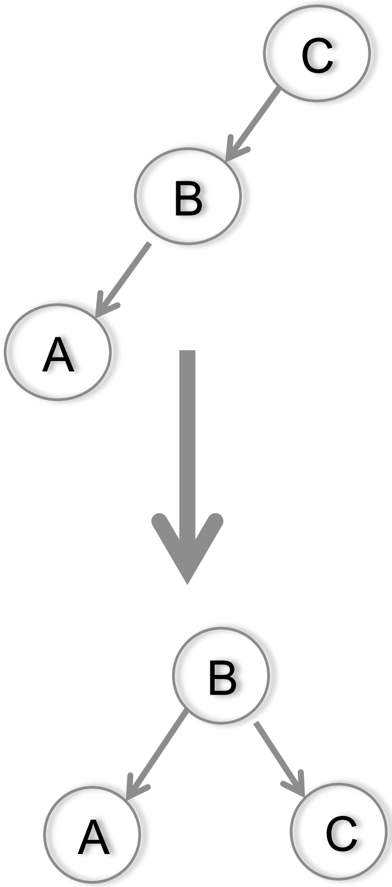
AVL "ROTATION"

- **Right rotation**
  - Symmetric concept



AVL "ROTATION"

- **Right rotation**
  - Symmetric concept
  - B must become the new root



AVL "ROTATION"

- **These are the "single" rotations**

AVL "ROTATION"

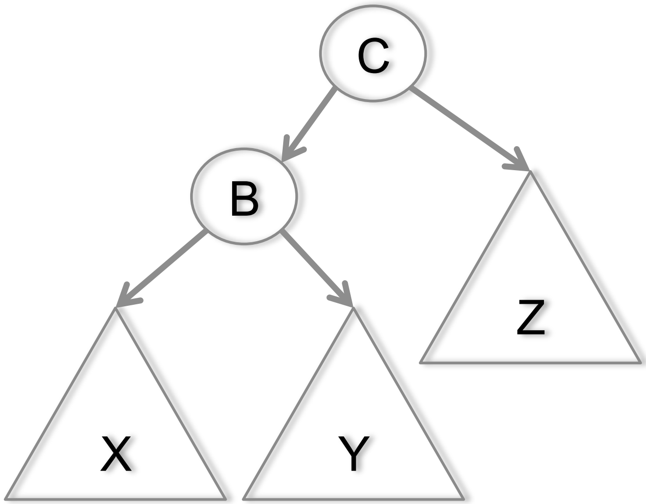
- **These are the "single" rotations**
  - In general, this rotation occurs when an addition is made to the right-right or left-left grandchild

AVL "ROTATION"

- **These are the "single" rotations**
  - In general, this rotation occurs when an addition is made to the right-right or left-left grandchild
  - **The balance might not be off on the parent! An insert might upset balance up the tree**

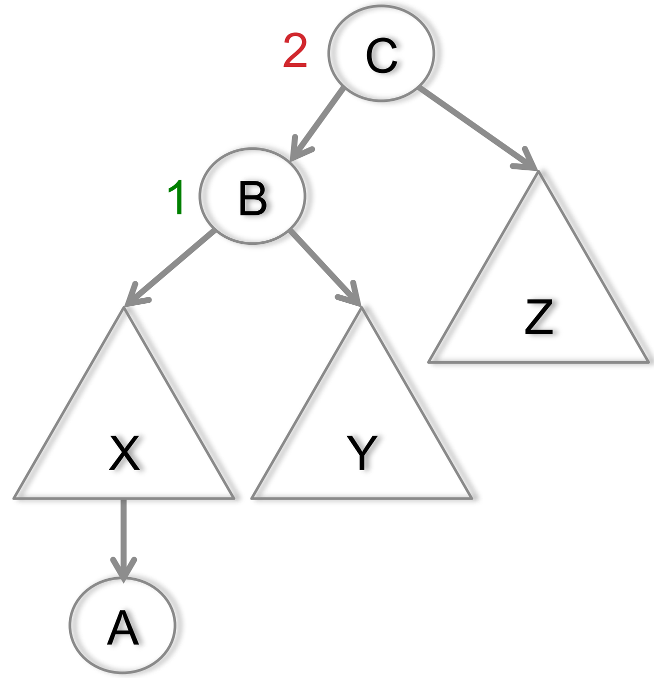
AVL "ROTATION"

- **General case**
  - Suppose this tree is balanced, {X,Y,Z} all have the same height



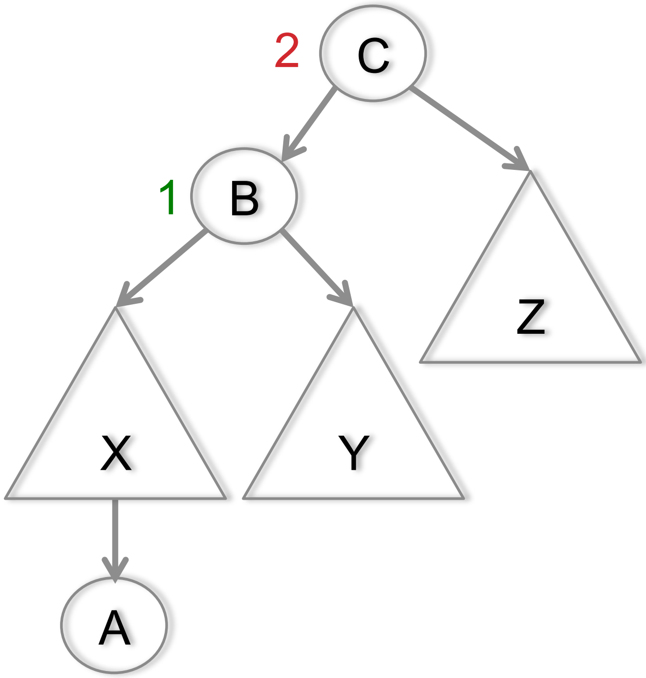
AVL "ROTATION"

- **General case**
  - Suppose this tree is balanced, {X,Y,Z} all have the same height
  - Adding A, puts C out of balance



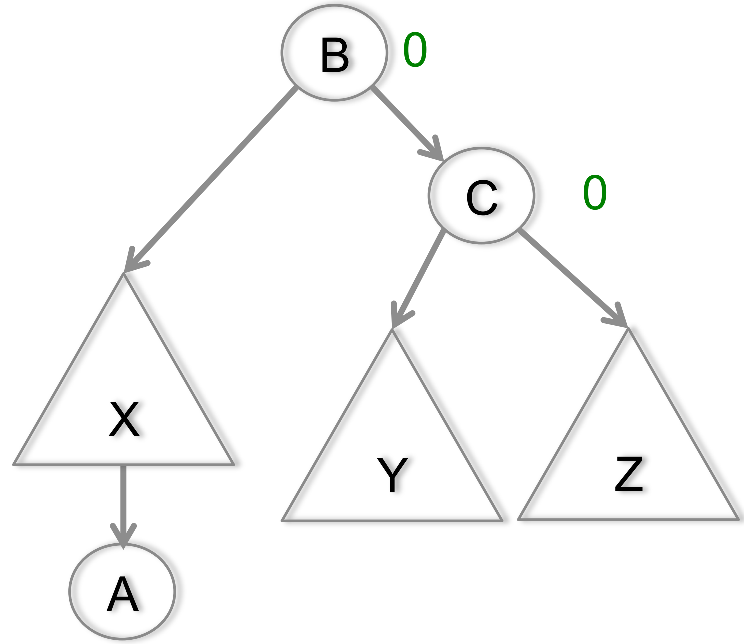
AVL "ROTATION"

- **General case**
  - Suppose this tree is balanced, {X,Y,Z} all have the same height
  - Adding A, puts C out of balance
  - Rotate B up and pass the Y subtree to C



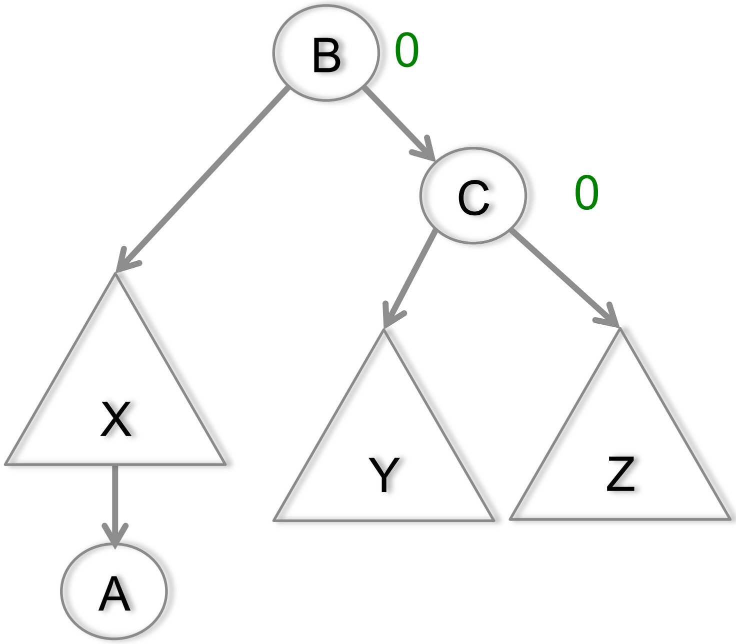
AVL "ROTATION"

- **General case**
  - Suppose this tree is balanced, {X,Y,Z} all have the same height
  - Adding A, puts C out of balance
  - Rotate B up and pass the Y subtree to C



AVL "ROTATION"

- **General case**
  - Suppose this tree is balanced, {X,Y,Z} all have the same height
  - Adding A, puts C out of balance
  - Rotate B up and pass the Y subtree to C
  - **Perform this rotation at the lowest point of imbalance**



AVL "ROTATION"

- These two rotations (right-right and left-left) are symmetric and can be solved the same way

AVL “ROTATION”

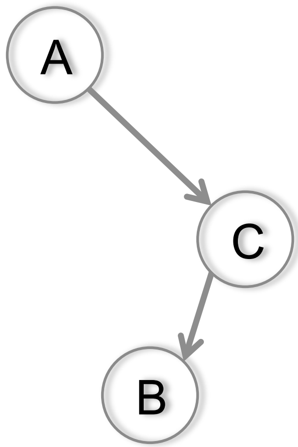
- **These two rotations (right-right and left-left) are symmetric and can be solved the same way**
  - Named by the location of the added node relative to the unbalanced node

AVL “ROTATION”

- **These two rotations (right-right and left-left) are symmetric and can be solved the same way**
  - Named by the location of the added node relative to the unbalanced node
  - What are the other two cases?

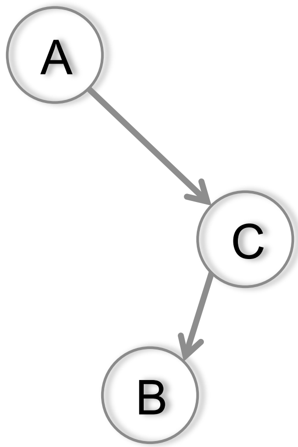
AVL "ROTATION"

- Right left case



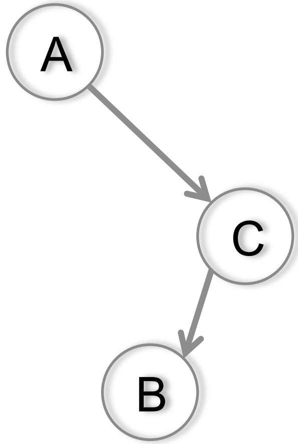
AVL "ROTATION"

- **Right left case**
  - Again, A is out of balance



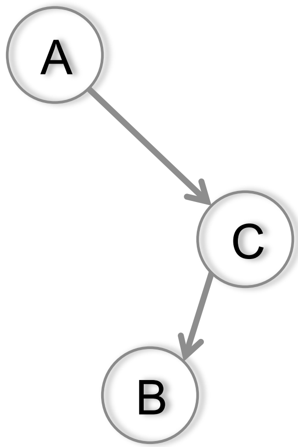
AVL "ROTATION"

- **Right left case**
  - Again, A is out of balance
  - This time, the addition (B) comes between A and C



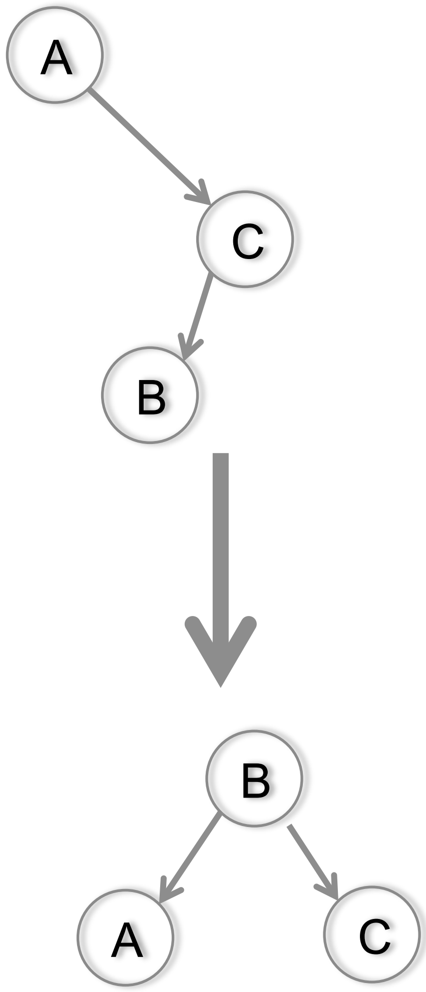
AVL "ROTATION"

- **Right left case**
  - Again, A is out of balance
  - This time, the addition (B) comes between A and C
  - In this case, the grandchild must become the root.



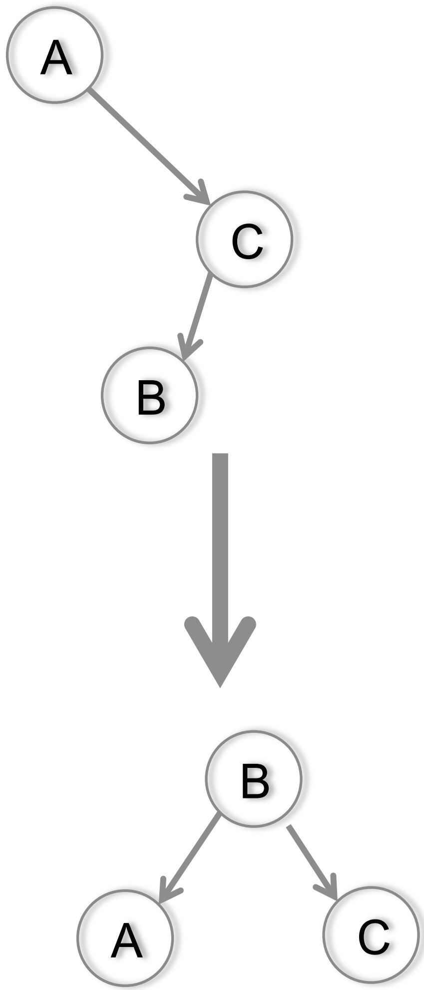
AVL "ROTATION"

- **Right left case**
  - Again, A is out of balance
  - This time, the addition (B) comes between A and C
  - In this case, the grandchild must become the root.



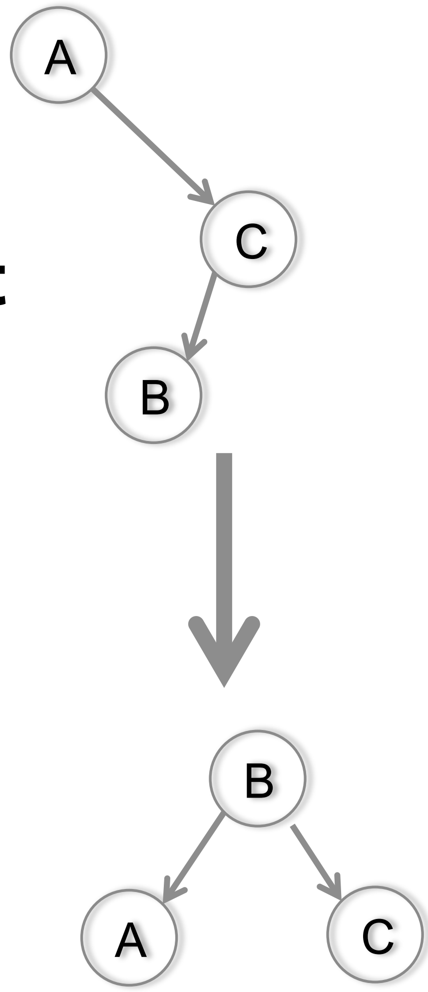
AVL "ROTATION"

- Identifying what should be the new root is key



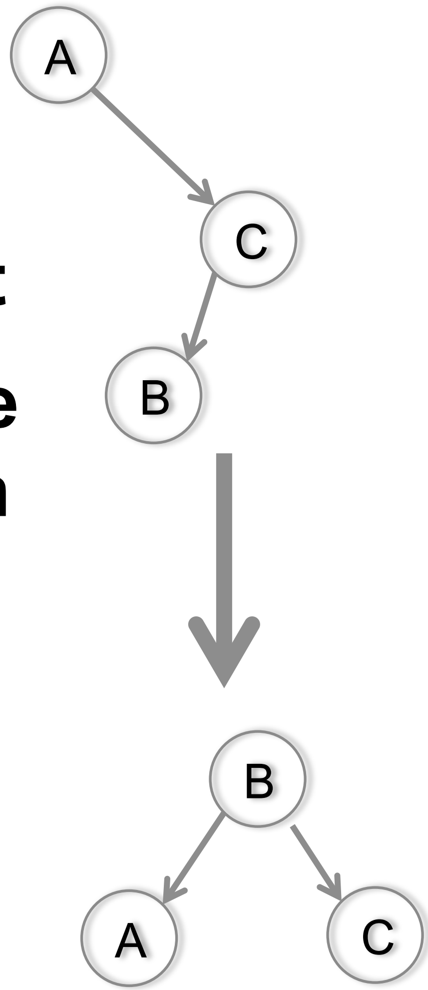
AVL "ROTATION"

- Identifying what should be the new root is key
- Imagine "lifting" up the root



AVL "ROTATION"

- **Identifying what should be the new root is key**
- **Imagine "lifting" up the root**
- **Where will the children have to go to maintain the search property?**

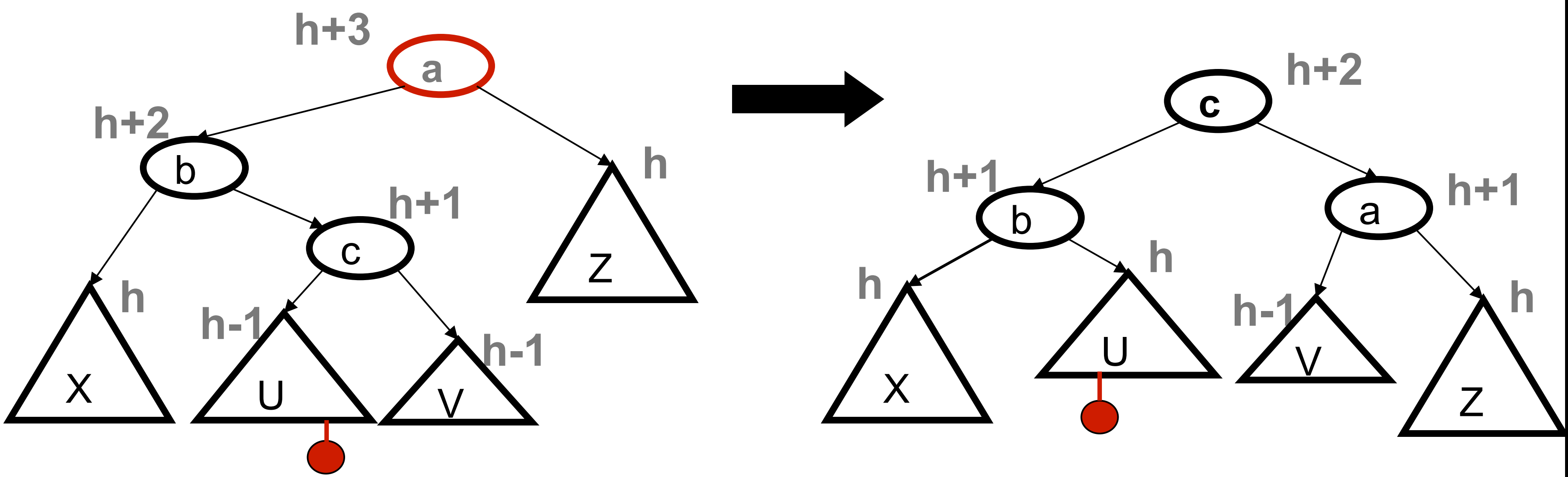


AVL "ROTATION"

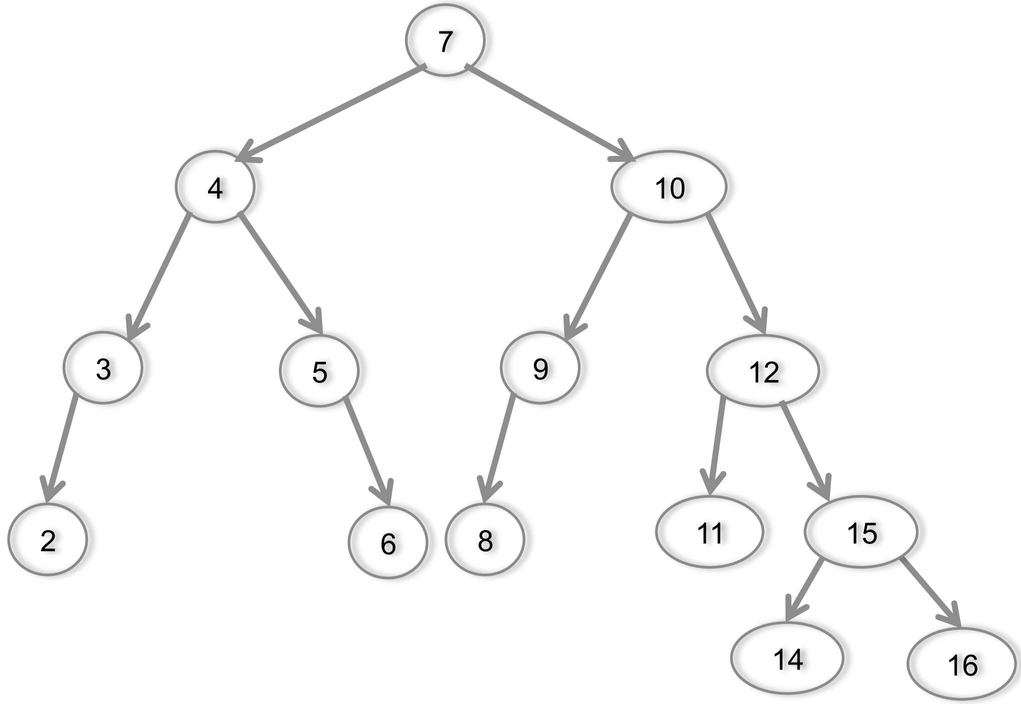
- I apologize for what you are about to see…

AVL "ROTATION"

- This is for your reference later.

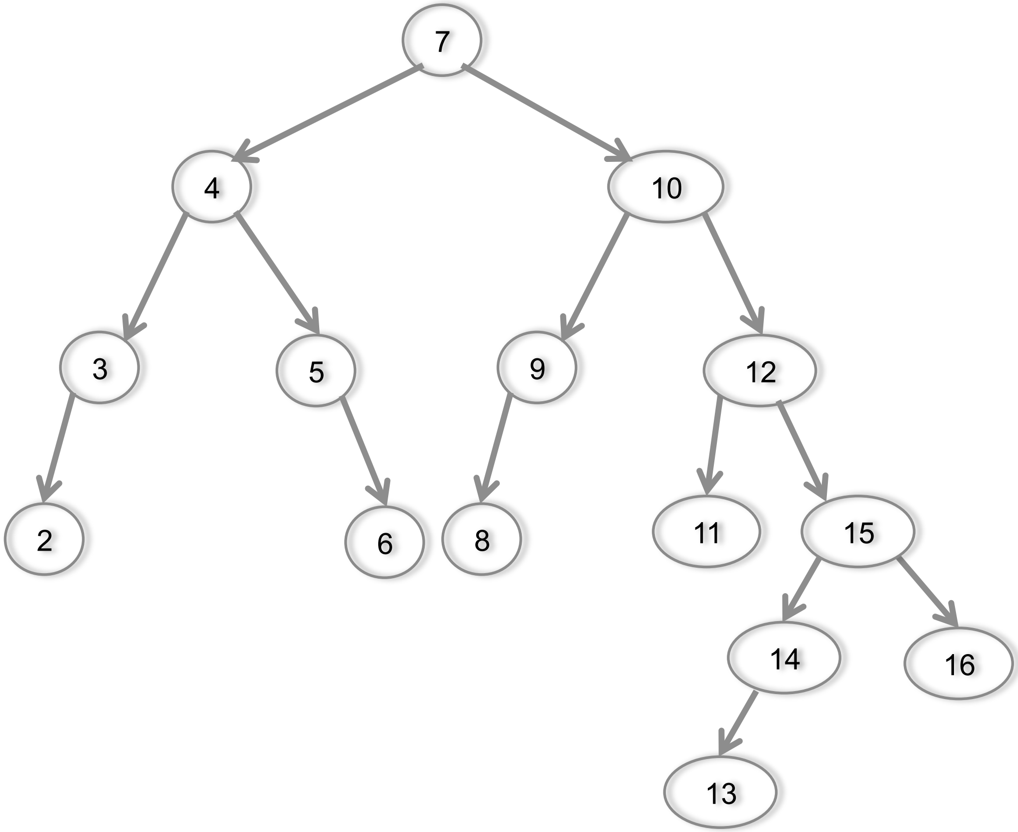


AVL "ROTATION"



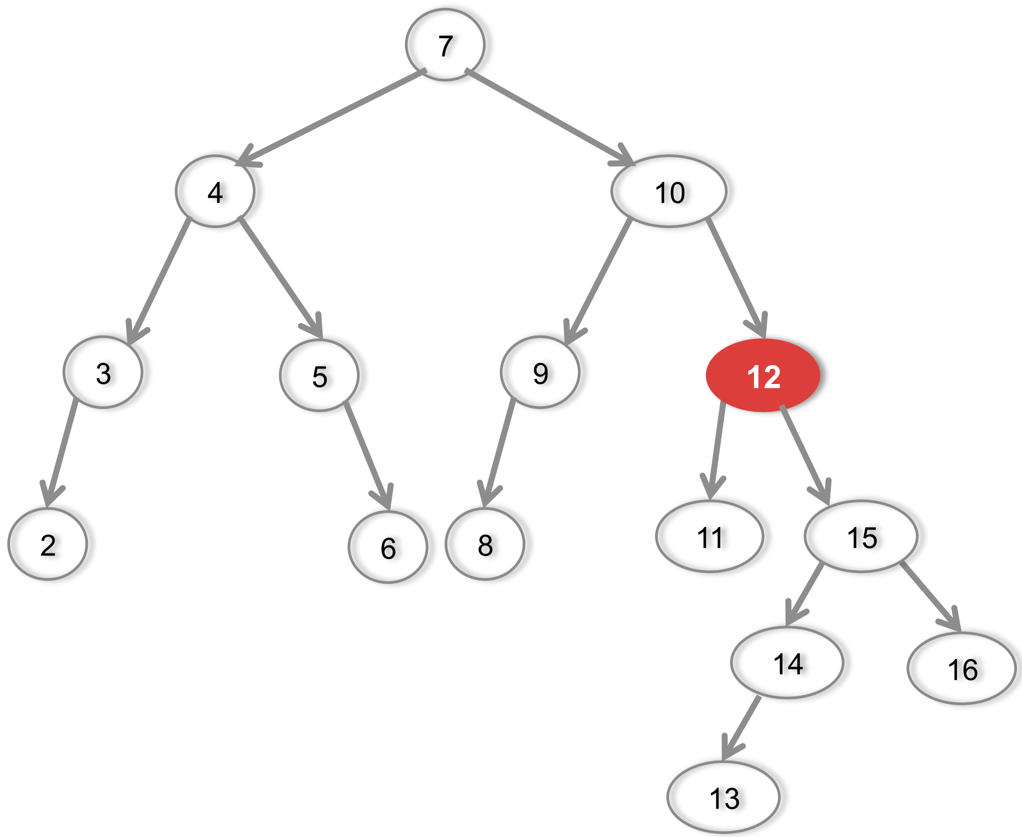
- **Let's do an example. Insert(13)**

AVL "ROTATION"



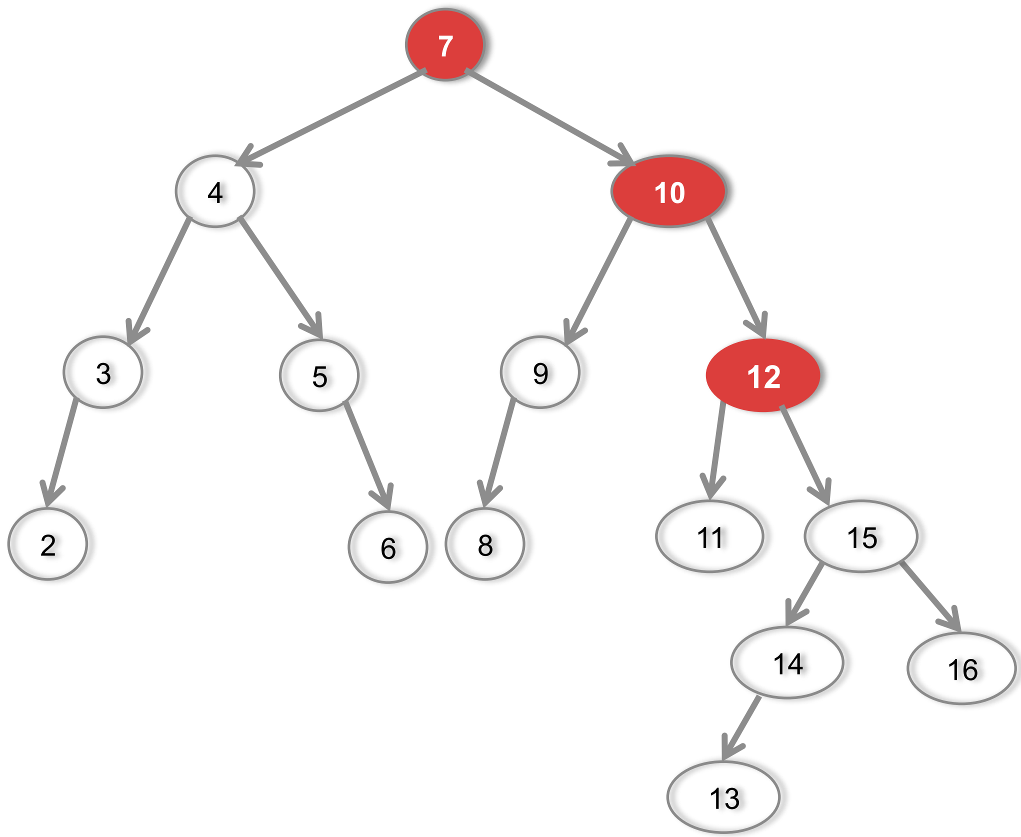
- **Where is the imbalance?**

AVL "ROTATION"



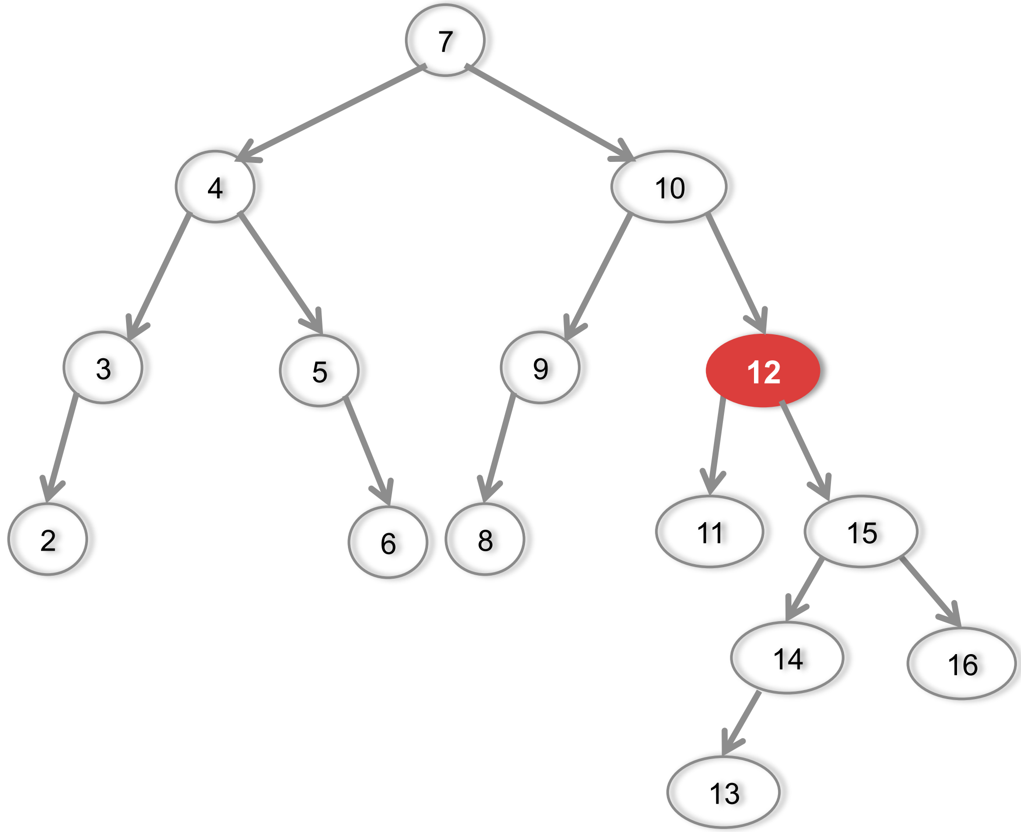
- **Where is the imbalance?**

AVL "ROTATION"



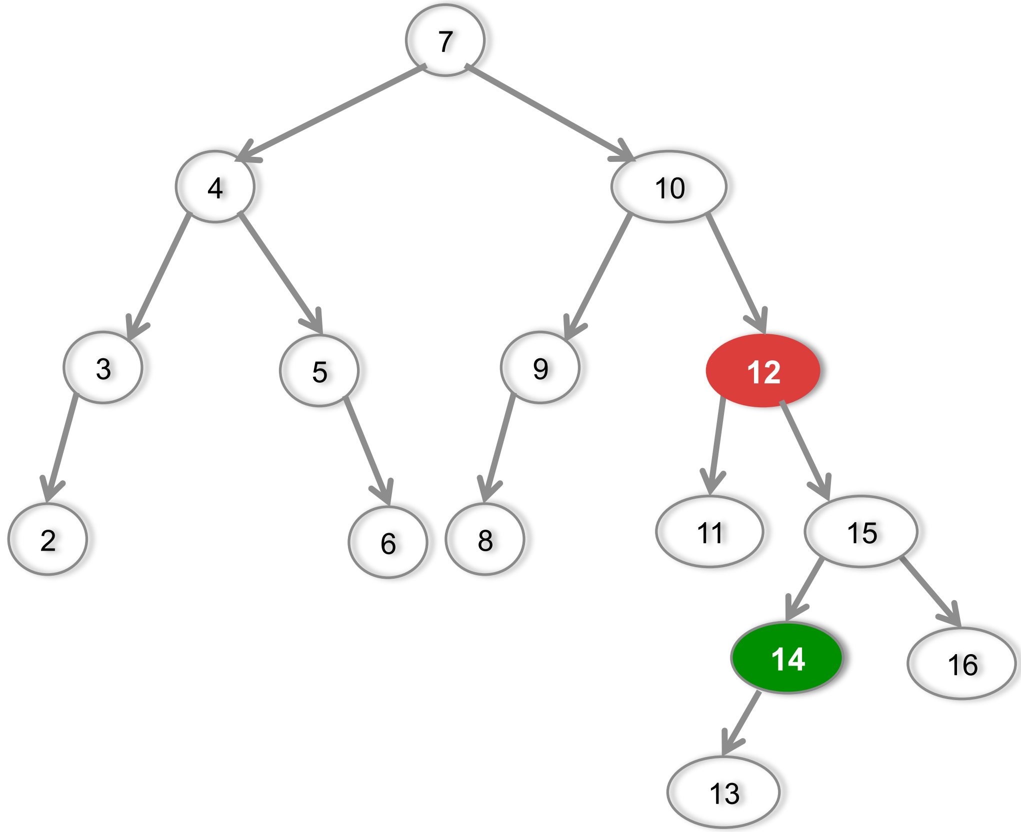
- **Where is the imbalance?** (also 7 and 10)

AVL "ROTATION"



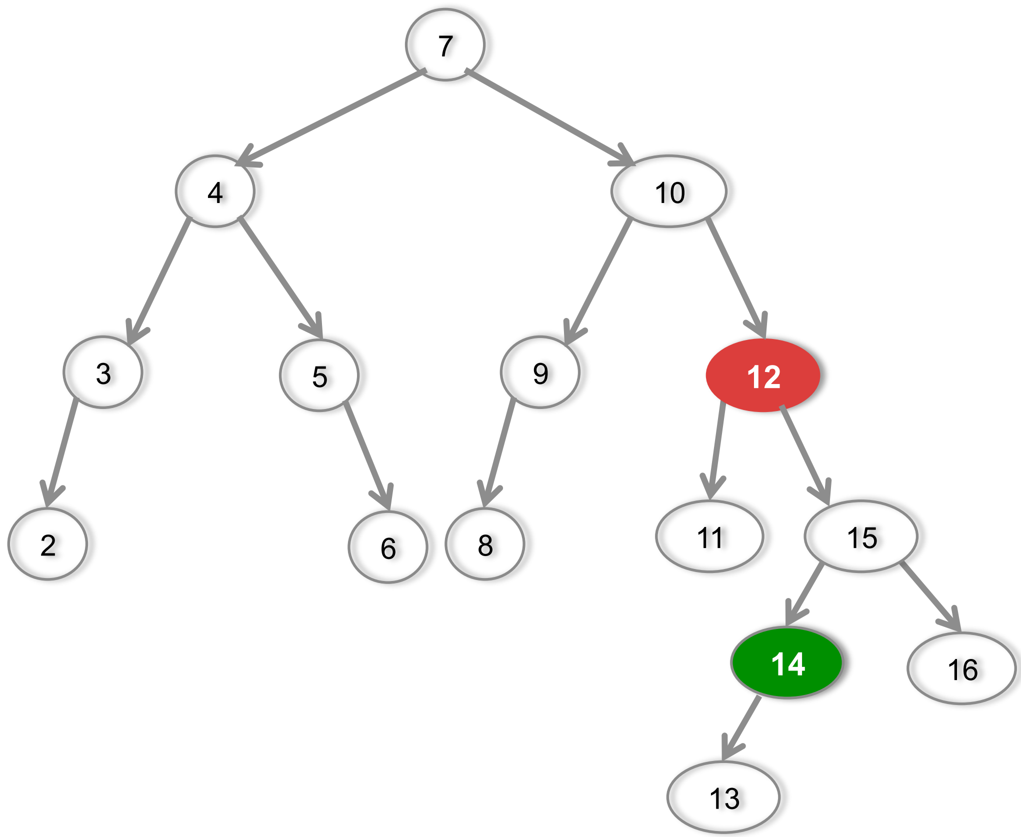
- **What will be the new root?**

AVL "ROTATION"



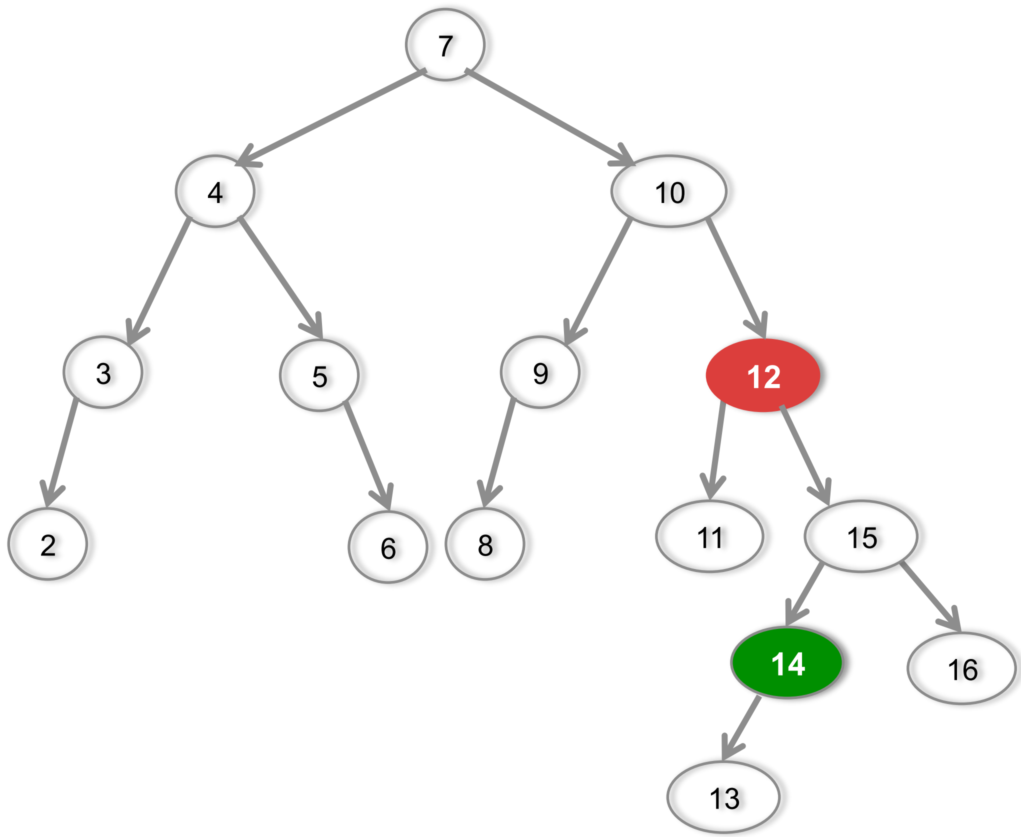
- **What will be the new root?**

AVL "ROTATION"



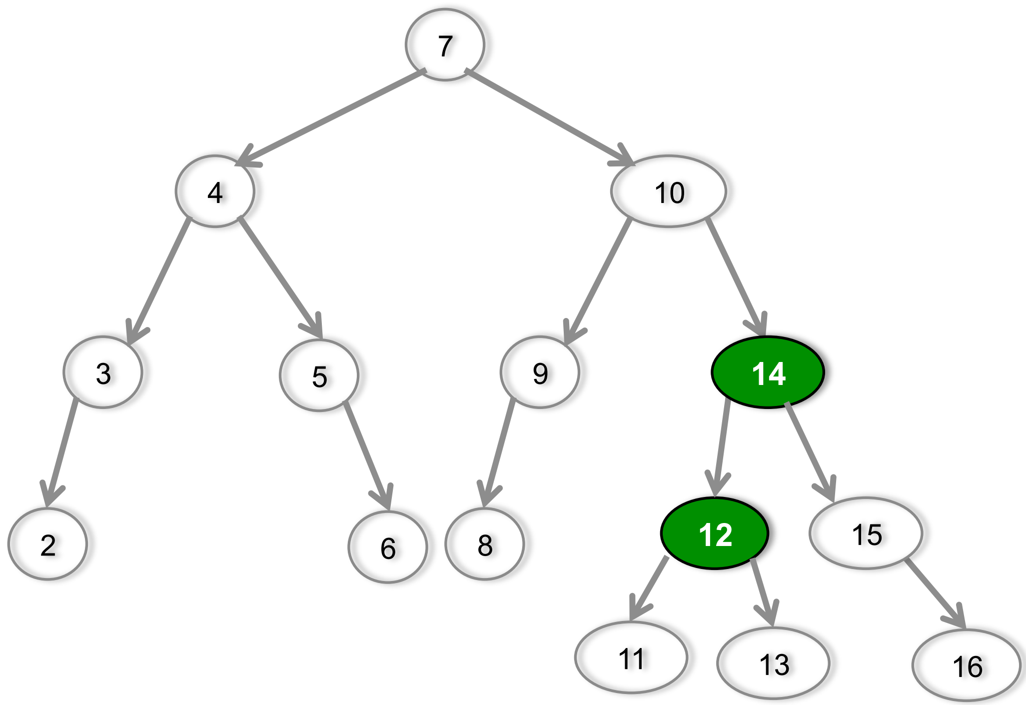
- **What will be the new root? Why?**

AVL "ROTATION"



- What does the new tree look like?

AVL "ROTATION"



- **The replaced root is always a child of the new root! Whether single or double**

AVL VISUALIZER

- https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
- **Note that this tool uses a different definition for height than we do.**

AVL HEIGHT (PROOF)

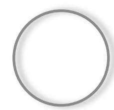
- **You do not need to memorize this proof, you need to understand it**

AVL HEIGHT (PROOF)

- **You do not need to memorize this proof, you need to understand it**
  - Let’s consider the most “unbalanced” AVL tree, that is: the tree for each height that has the fewest nodes

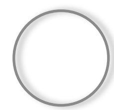
AVL HEIGHT (PROOF)

- For height 0, there is only one possible tree.



AVL HEIGHT (PROOF)

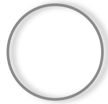
- For height 0, there is only one possible tree.



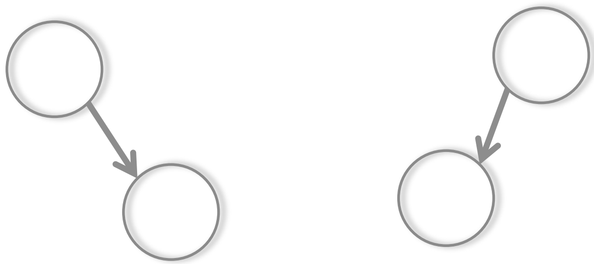
- For height 1, there are two possible trees, each with two nodes.

AVL HEIGHT (PROOF)

- For height 0, there is only one possible tree.



- For height 1, there are two possible trees, each with two nodes.



AVL HEIGHT (PROOF)

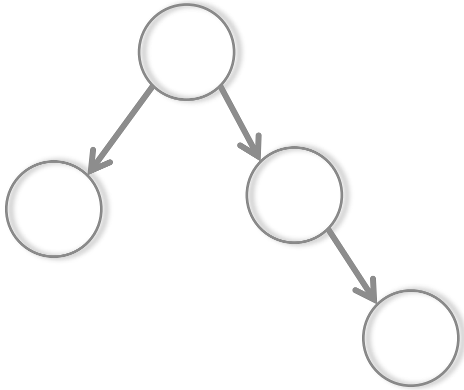
- What about for height two? What tree has the fewest number of nodes?

AVL HEIGHT (PROOF)

- **What about for height two? What tree has the fewest number of nodes?**
  - *Hint: balance will probably not be zero*

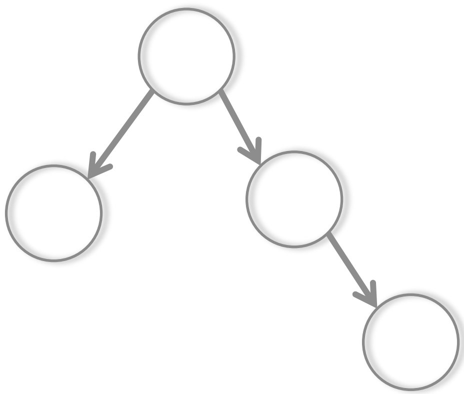
AVL HEIGHT (PROOF)

- **What about for height two? What tree has the fewest number of nodes?**
  - *Hint: balance will probably not be zero*



AVL HEIGHT (PROOF)

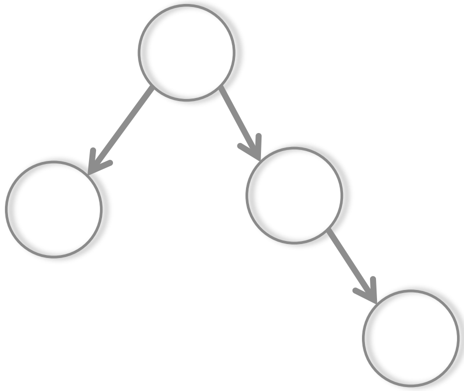
- **What about for height two? What tree has the fewest number of nodes?**
  - *Hint: balance will probably not be zero*



There are multiple of these trees, but what's special about it?

AVL HEIGHT (PROOF)

- The smallest tree of height two is a node where one child is the smallest tree of heigh one and the other one is the smallest tree of height zero.



AVL HEIGHT (PROOF)

- In general then, if $N_0 = 1$ and $N_1 = 2$ and $N_2 = 4$, what is N_k?

AVL HEIGHT (PROOF)

- **In general then, if $N_0 = 1$ and $N_1 = 2$ and $N_2 = 4$, what is N_k?**
  - Powers of two seems intuitive, but this is a good case of why 3 doesn't always make the pattern.

AVL HEIGHT (PROOF)

- **In general then, if $N_0 = 1$ and $N_1 = 2$ and $N_2 = 4$, what is N_k?**
  - Powers of two seems intuitive, but this is a good case of why 3 doesn't always make the pattern.
  - **$N_4 = 7$, how do I know?**

AVL HEIGHT (PROOF)

- **In general then, if $N_0 = 1$ and $N_1 = 2$ and $N_2 = 4$, what is N_k?**
  - **$N_k = 1 + N_{k-1} + N_{k-2}$**
    Because the smallest AVL tree is a node (1) with a child that is the smallest AVL tree of height k-1 (N_{k-1}) and the other child is the smallest AVL tree of height k-2 (N_{k-2}).

AVL HEIGHT (PROOF)

- **In general then, if $N_0 = 1$ and $N_1 = 2$ and $N_2 = 4$, what is N_k?**
  - **$N_k = 1 + N_{k-1} + N_{k-2}$**
    Because the smallest AVL tree is a node (1) with a child that is the smallest AVL tree of height k-1 (N_{k-1}) and the other child is the smallest AVL tree of height k-2 (N_{k-2}).
  - **This means every non-leaf has balance 1**

AVL HEIGHT (PROOF)

- **In general then, if $N_0 = 1$ and $N_1 = 2$ and $N_2 = 4$, what is N_k?**
  - **$N_k = 1 + N_{k-1} + N_{k-2}$**
    Because the smallest AVL tree is a node (1) with a child that is the smallest AVL tree of height k-1 (N_{k-1}) and the other child is the smallest AVL tree of height k-2 (N_{k-2}).
  - **This means every non-leaf has balance 1**
  - **Nothing in the tree is perfectly balanced.**

AVL HEIGHT (PROOF)

$$N_k = 1 + N_{k-1} + N_{k-2}$$
$$N_{k-1} = 1 + N_{k-2} + N_{k-3}$$

AVL HEIGHT (PROOF)

$$N_k = 1 + N_{k-1} + N_{k-2}$$
$$N_{k-1} = 1 + N_{k-2} + N_{k-3}$$

AVL HEIGHT (PROOF)

Substitute the k-1 into the original equation

$$N_k = 1 + N_{k-1} + N_{k-2}$$
$$N_{k-1} = 1 + N_{k-2} + N_{k-3}$$

AVL HEIGHT (PROOF)

1 + N_{k-3} must be greater than zero

$N_k = 1 + N_{k-1} + N_{k-2}$
$N_{k-1} = 1 + N_{k-2} + N_{k-3}$

$N_k = 1 + (1 + N_{k-2} + N_{k-3}) + N_{k-2}$

$N_k = 2 + 2N_{k-2} + N_{k-3}$

$N_k > 2N_{k-2}$

AVL HEIGHT (PROOF)

1 + N_{k-3} must be greater than zero

$$N_k = 1 + N_{k-1} + N_{k-2}$$
$$N_{k-1} = 1 + N_{k-2} + N_{k-3}$$

$$N_k = 1 + (1 + N_{k-2} + N_{k-3}) + N_{k-2}$$

$$N_k = 2 + 2N_{k-2} + N_{k-3}$$

$$N_k > 2N_{k-2}$$

This means the tree doubles in size after every two height (compared to a perfect tree which doubles with every added height)

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**
  - Insert(key k, value v)
  - Find(key k)

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**
  - Insert(key k, value v)
  - Find(key k)
  - Delete(key k)

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**
  - Insert(key k, value v)
  - Find(key k) : O(height) = O(log n)
  - Delete(key k)

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**
  - Insert(key k, value v) = O(log n) + balancing
  - Find(key k) : O(height) = O(log n)
  - Delete(key k)

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**
  - Insert(key k, value v) = O(log n) + balancing
  - Find(key k) : O(height) = O(log n)
  - Delete(key k): *O(log n)* + *balancing(?)*
- **How long does it take to perform a balance?**

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**
  - Insert(key k, value v) = O(log n) + balancing
  - Find(key k) : O(height) = O(log n)
  - Delete(key k): *O(log n)* + *balancing(?)*
- **How long does it take to perform a balance?**
  - There are at most three nodes and four subtrees to move around.

AVL CONCLUSION

- **If AVL rotation can enforce O(log n) height, what are the asymptotic runtimes for our functions?**
  - Insert(key k, value v) = O(log n) + balancing
  - Find(key k) : O(height) = O(log n)
  - Delete(key k): *O(log n)* + *balancing(?)*
- **How long does it take to perform a balance?**
  - There are at most three nodes and four subtrees to move around. **O(1)**

AVL CONCLUSION

- By using AVL rotations, we can keep the tree balanced

AVL CONCLUSION

- By using AVL rotations, we can keep the tree balanced
- An AVL tree has O(log n) height

AVL CONCLUSION

- By using AVL rotations, we can keep the tree balanced
- An AVL tree has O(log n) height
- This does not come at an increased asymptotic runtime for insert.

AVL CONCLUSION

- By using AVL rotations, we can keep the tree balanced
- An AVL tree has O(log n) height
- This does not come at an increased asymptotic runtime for insert.
- Rotations take a constant time.