AVL Trees

Saikrishna Arcot
(edits by M. Hudachek-Buswell)

May 30, 2017

Problems with Binary Search Trees

- A binary search tree has, on average, a big-O of $O(\log n)$ for adding, searching, and removing.

Problems with Binary Search Trees

- A binary search tree has, on average, a big-O of $O(\log n)$ for adding, searching, and removing.
- However, in the worst case, all three operations can become $O(n)$ when the tree is unbalanced. This can easily be done by adding the items in sorted order.

Problems with Binary Search Trees

- A binary search tree has, on average, a big-O of $O(\log n)$ for adding, searching, and removing.
- However, in the worst case, all three operations can become $O(n)$ when the tree is unbalanced. This can easily be done by adding the items in sorted order.
- There needs to be a way to keep the tree balanced while adding or removing, and make sure that all three operations have a worst case of $O(n)$.

AVL Trees

- AVL trees were desgined to solve this problem.

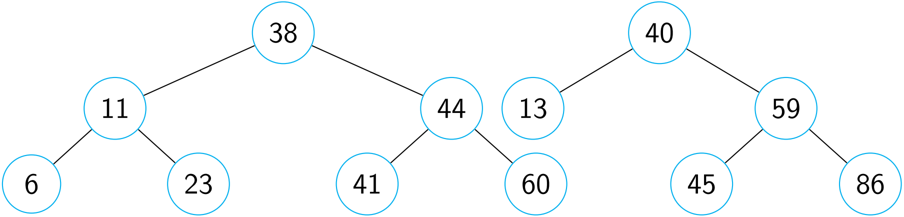
AVL Trees

- AVL trees were desgined to solve this problem.
- The idea is that whenever you're adding or removing a data item, you check to see if the other nodes are balanced; if they're not balanced, then you make rotations to balance the tree.

AVL Trees

- AVL trees were desgined to solve this problem.
- The idea is that whenever you're adding or removing a data item, you check to see if the other nodes are balanced; if they're not balanced, then you make rotations to balance the tree.
- AVL trees are just balanced BSTs; in other words, all AVL trees are also BSTs, but not all BSTs are AVL trees.

Example AVL Trees



Properties

- AVL trees have many of the same properties that are present in BSTs (maximum of two children, left child is smaller than the parent node, which is smaller than the right child, height calculation).

Properties

- AVL trees have many of the same properties that are present in BSTs (maximum of two children, left child is smaller than the parent node, which is smaller than the right child, height calculation).
- Searching in an AVL tree is the same as searching in a BST.

Properties

- AVL trees have many of the same properties that are present in BSTs (maximum of two children, left child is smaller than the parent node, which is smaller than the right child, height calculation).
- Searching in an AVL tree is the same as searching in a BST.
- Adding to and removing from an AVL tree has the same basic steps as adding to and removing from a BST, but there is more work you need to do for an AVL tree.

Properties

- AVL trees have many of the same properties that are present in BSTs (maximum of two children, left child is smaller than the parent node, which is smaller than the right child, height calculation).
- Searching in an AVL tree is the same as searching in a BST.
- Adding to and removing from an AVL tree has the same basic steps as adding to and removing from a BST, but there is more work you need to do for an AVL tree.
- In addition to each node having a certain height, each node in an AVL tree also has a balance factor.

Balance Factor

- The balance factor of a node describes how balanced the subtree of that node is and, if it is not perfectly balanced, which side of the subtree (left side or right side) is "heavier".

Balance Factor

- The balance factor of a node describes how balanced the subtree of that node is and, if it is not perfectly balanced, which side of the subtree (left side or right side) is “heavier”.
- The balace factor of a node is calculated by subtracting the height of the left subtree by the height of the right subtree.

Balance Factor

- The balance factor of a node describes how balanced the subtree of that node is and, if it is not perfectly balanced, which side of the subtree (left side or right side) is "heavier".
- The balace factor of a node is calculated by subtracting the height of the left subtree by the height of the right subtree.
- A balance factor of 0 means that the subtree is perfectly balanced and that no rotations should be performed.

Balance Factor

- A balance factor between -1 and 1 (inclusive) means that the subtree is considered balanced, but is slighly heavier on the left side (if the balance factor is positive) or the right side (if the balance factor is negative), and that no rotations needs to be done.

Balance Factor

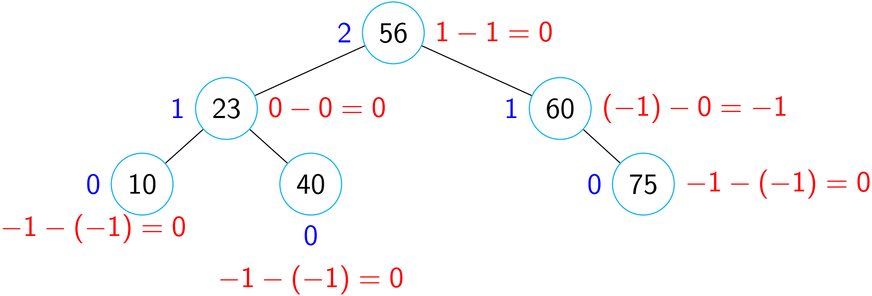
- A balance factor between -1 and 1 (inclusive) means that the subtree is considered balanced, but is slighly heavier on the left side (if the balance factor is positive) or the right side (if the balance factor is negative), and that no rotations needs to be done.
  - This is because doing any combination of rotations will not make the subtree perfectly balanced.

Balance Factor

- A balance factor between -1 and 1 (inclusive) means that the subtree is considered balanced, but is slighly heavier on the left side (if the balance factor is positive) or the right side (if the balance factor is negative), and that no rotations needs to be done.
  - This is because doing any combination of rotations will not make the subtree perfectly balanced.
- Any other balance factor means that the subtree is not balanced and that one or more rotations need to be done.

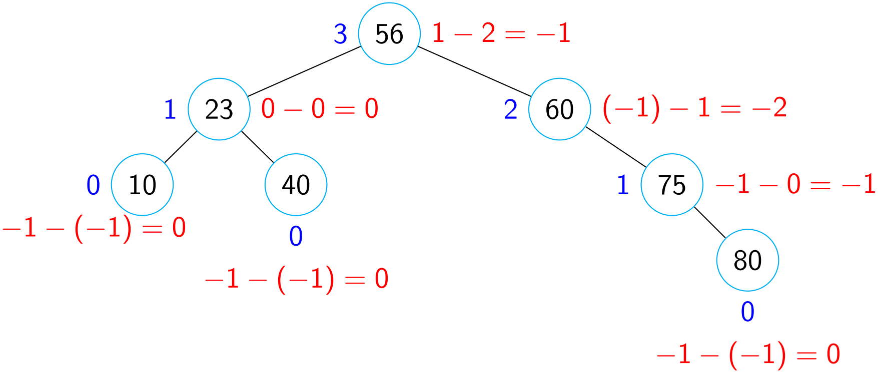
Balance Factor

In the following AVL tree, the height of each node is in blue, and the balance factor of each node is in red.



Balance Factor

In the following BST, the height of each node is in blue, and the balance factor of each node is in red. (Why is this not an AVL tree?)



Rotations

- In AVL trees, to balance a tree after adding or removing something, one or more rotations are done.

Rotations

- In AVL trees, to balance a tree after adding or removing something, one or more rotations are done.
- There are two single rotations and two double rotations to choose from. (Double rotations are just two single rotations put together.)

Rotations

- In AVL trees, to balance a tree after adding or removing something, one or more rotations are done.
- There are two single rotations and two double rotations to choose from. (Double rotations are just two single rotations put together.)
- Which rotations are done (and which rotations *can* be done) depend on the balance factor of each node.

Rotations

- When performing a rotation, the child subtrees of the node(s) being rotated might be moved around to maintain the BST property.

Rotations

- When performing a rotation, the child subtrees of the node(s) being rotated might be moved around to maintain the BST property.
- In the examples that follow, A, B, C, and D represent subtrees that may (or may not) exist. This is so that you can see where the subtrees move to after the rotation.

Left Rotation

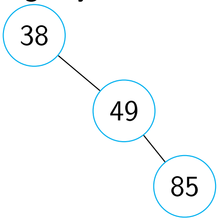
- A left rotation happens when you have a node whose balance factor is -2, and its right child has a balance factor of 0 or -1.

Left Rotation

- A left rotation happens when you have a node whose balance factor is -2, and its right child has a balance factor of 0 or -1.
- When a left rotation happens, the top node becomes the left child of the middle node.

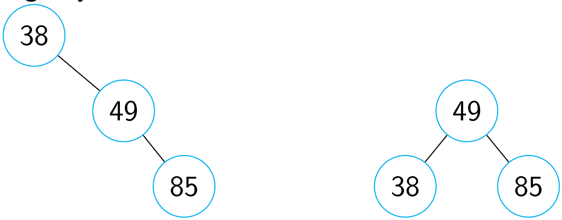
Left Rotation

Ignoring any child subtrees:



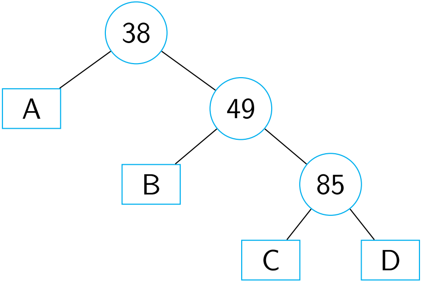
Left Rotation

Ignoring any child subtrees:



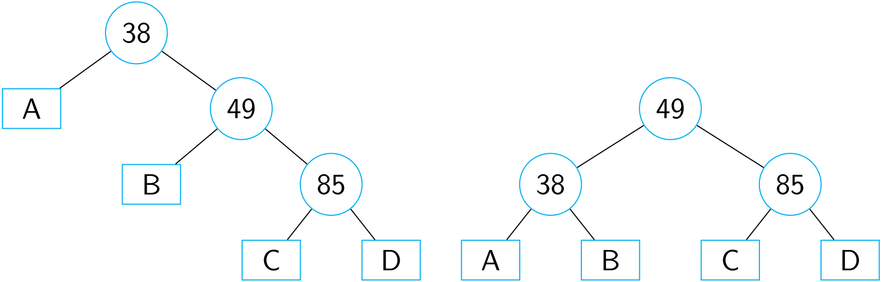
Left Rotation

With child subtrees:



Left Rotation

With child subtrees:



Right Rotation

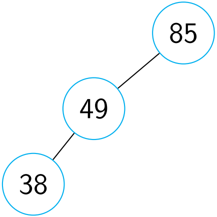
- A right rotation happens when you have a node whose balance factor is 2, and its left child has a balance factor of 0 or 1.

Right Rotation

- A right rotation happens when you have a node whose balance factor is 2, and its left child has a balance factor of 0 or 1.
- When a right rotation happens, the top node becomes the right child of the middle node.

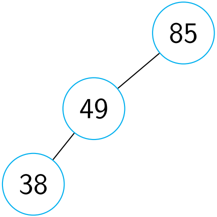
Right Rotation

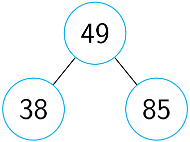
Ignoring any child subtrees:



Right Rotation

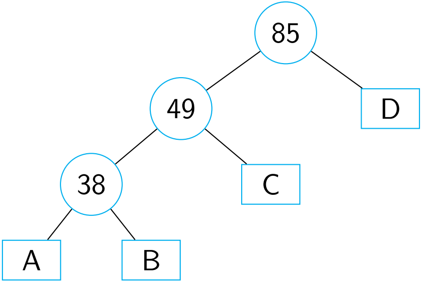
Ignoring any child subtrees:





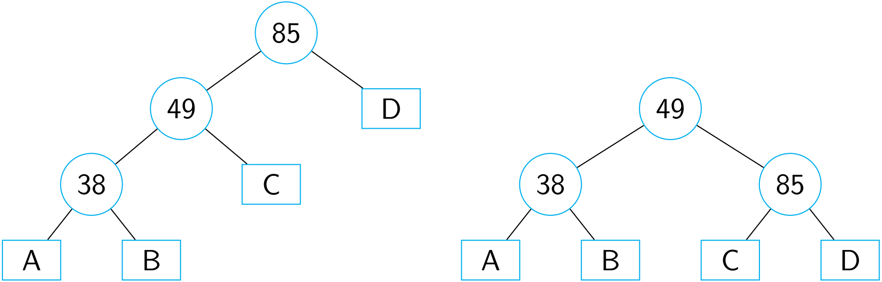
Right Rotation

With child subtrees:



Right Rotation

With child subtrees:



Left-Right Rotation

- A left-right rotation is a double rotation consisting of a left rotation followed by a right rotation.

Left-Right Rotation

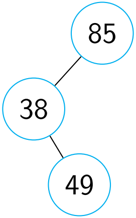
- A left-right rotation is a double rotation consisting of a left rotation followed by a right rotation.
- It happens when you have a node whose balance factor is 2, and its left child has a balance factor of -1.

Left-Right Rotation

- A left-right rotation is a double rotation consisting of a left rotation followed by a right rotation.
- It happens when you have a node whose balance factor is 2, and its left child has a balance factor of -1.
- When a left-right rotation happens, the top node becomes the right child of the bottom node, and the middle node becomes the left child of the bottom node.

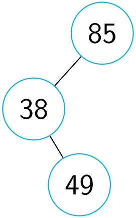
Left-Right Rotation

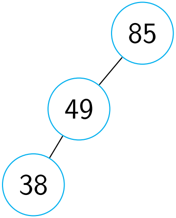
Ignoring any child subtrees:



Left-Right Rotation

Ignoring any child subtrees:

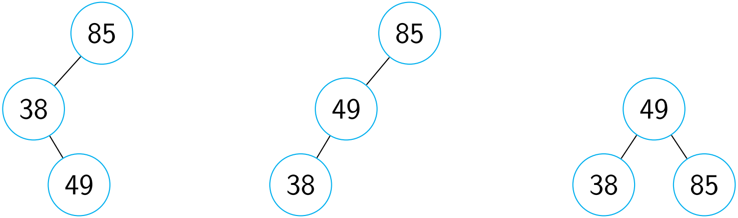




(Look familiar?)

Left-Right Rotation

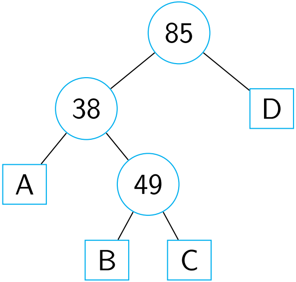
Ignoring any child subtrees:



(Look familiar?)

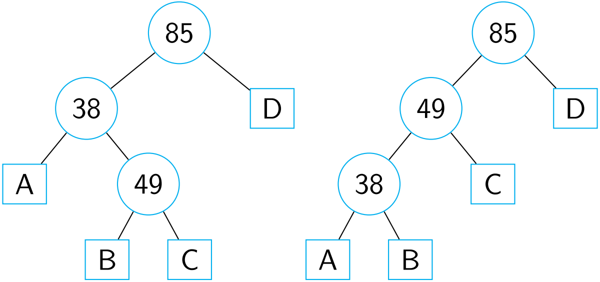
Left-Right Rotation

With child subtrees:



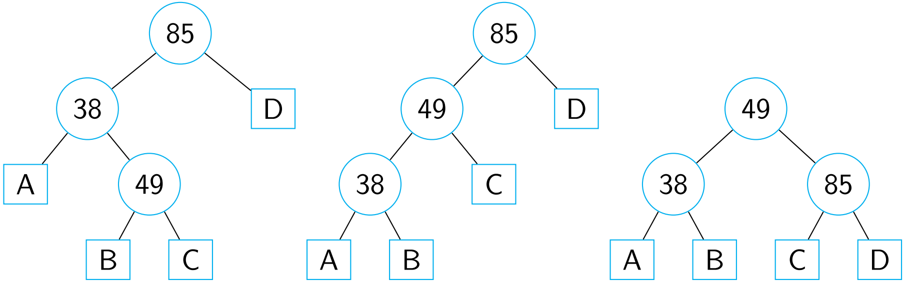
Left-Right Rotation

With child subtrees:



Left-Right Rotation

With child subtrees:



Right-Left Rotation

- A left-right rotation is a double rotation consisting of a right rotation followed by a left rotation.

Right-Left Rotation

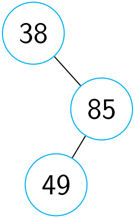
- A left-right rotation is a double rotation consisting of a right rotation followed by a left rotation.
- It happens when you have a node whose balance factor is -2, and its right child has a balance factor of 1.

Right-Left Rotation

- A left-right rotation is a double rotation consisting of a right rotation followed by a left rotation.
- It happens when you have a node whose balance factor is -2, and its right child has a balance factor of 1.
- When a right-left rotation happens, the top node becomes the left child of the bottom node, and the middle node becomes the right child of the bottom node.

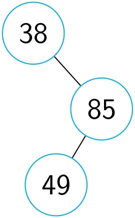
Right-Left Rotation

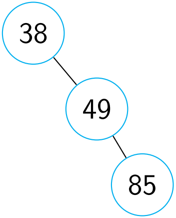
Ignoring any child subtrees:



Right-Left Rotation

Ignoring any child subtrees:

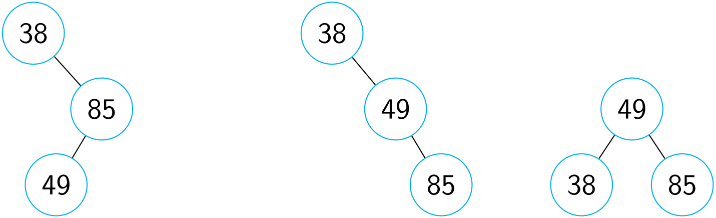




(Look familiar?)

Right-Left Rotation

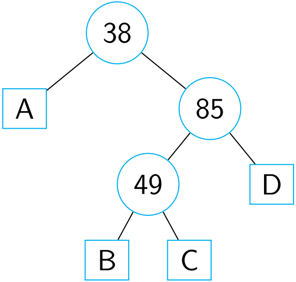
Ignoring any child subtrees:



(Look familiar?)

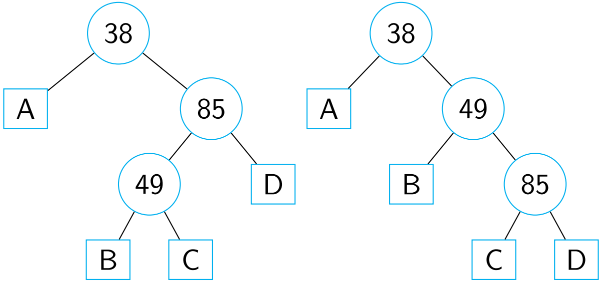
Right-Left Rotation

With child subtrees:



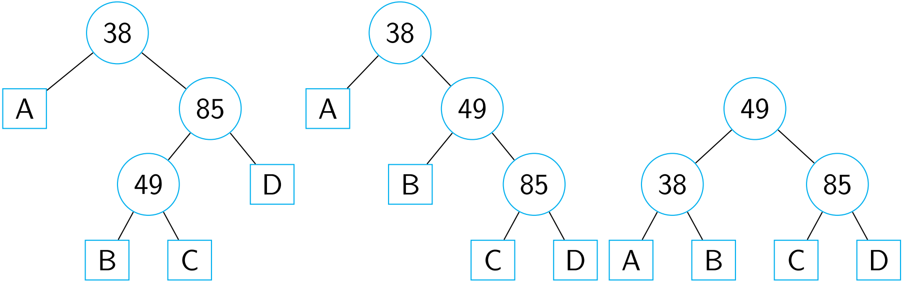
Right-Left Rotation

With child subtrees:



Right-Left Rotation

With child subtrees:



Adding and Removing

- The basic steps for adding and removing to an AVL tree are the same as adding and removing to a BST.

Adding and Removing

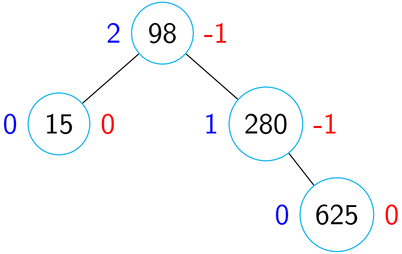
- The basic steps for adding and removing to an AVL tree are the same as adding and removing to a BST.
- After a node is added/removed, the heights and balance factor of the ancestors of the added/removed node are updated, and, if necessary, rotations are done. This step would be done recursively, which means that you would update the heights and balance factors of the parent node and do any necessary rotations, then the grandparent node, then the great-grandparent node, . . . all the way up to the root node.

Adding and Removing

- The basic steps for adding and removing to an AVL tree are the same as adding and removing to a BST.
- After a node is added/removed, the heights and balance factor of the ancestors of the added/removed node are updated, and, if necessary, rotations are done. This step would be done recursively, which means that you would update the heights and balance factors of the parent node and do any necessary rotations, then the grandparent node, then the great-grandparent node, . . . all the way up to the root node.
- If a rotation is done, the heights and balance factors of the nodes involved are updated again.

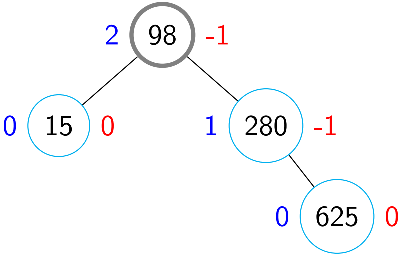
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



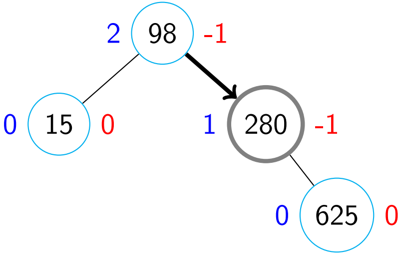
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



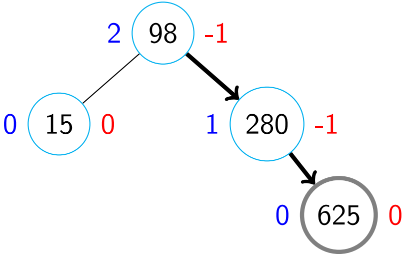
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



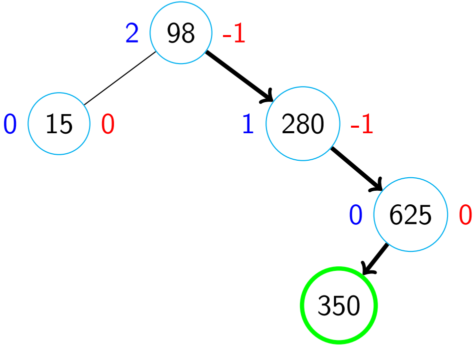
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



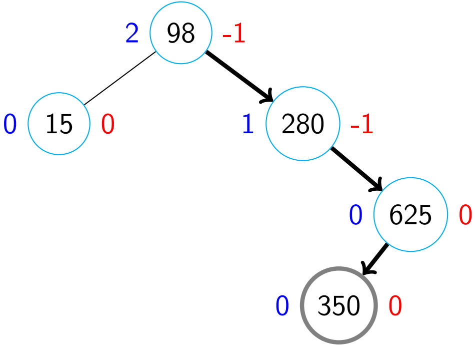
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



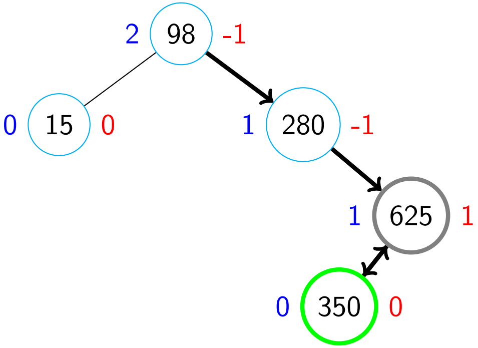
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



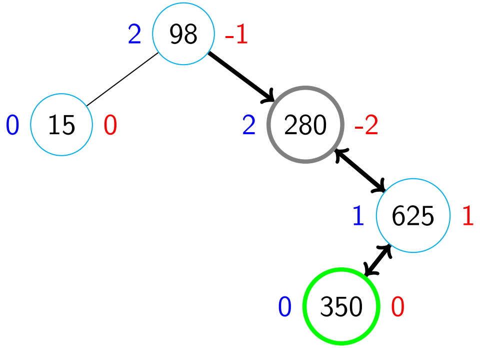
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



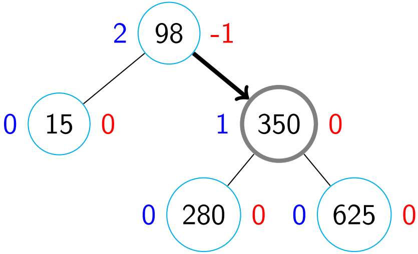
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



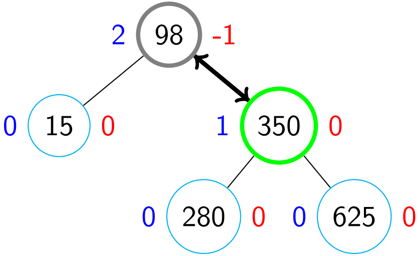
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



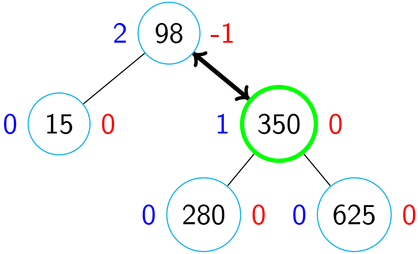
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



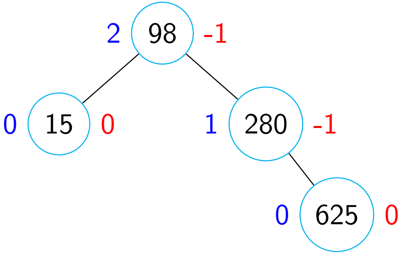
Adding

For example, if you were adding 350 (heights are to the left and in blue, and balance factors are to the right and in red):



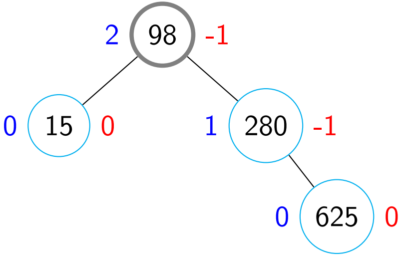
Removing

For example, if you were removing 15 (heights are to the left and in blue, and balance factors are to the right and in red):



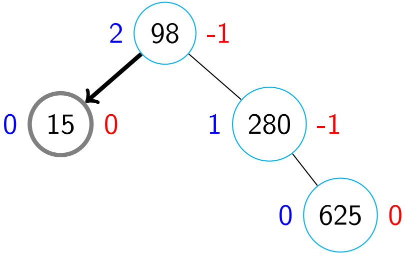
Removing

For example, if you were removing 15 (heights are to the left and in blue, and balance factors are to the right and in red):



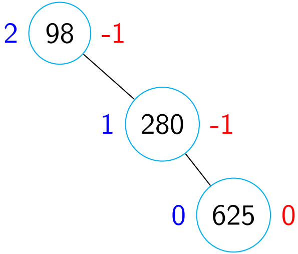
Removing

For example, if you were removing 15 (heights are to the left and in blue, and balance factors are to the right and in red):



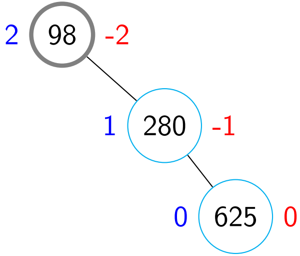
Removing

For example, if you were removing 15 (heights are to the left and in blue, and balance factors are to the right and in red):



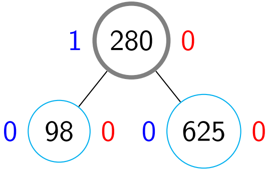
Removing

For example, if you were removing 15 (heights are to the left and in blue, and balance factors are to the right and in red):



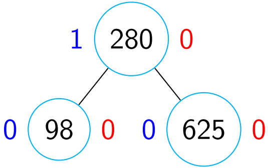
Removing

For example, if you were removing 15 (heights are to the left and in blue, and balance factors are to the right and in red):



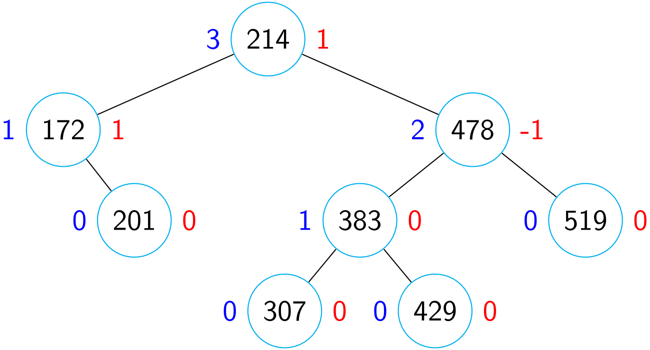
Removing

For example, if you were removing 15 (heights are to the left and in blue, and balance factors are to the right and in red):



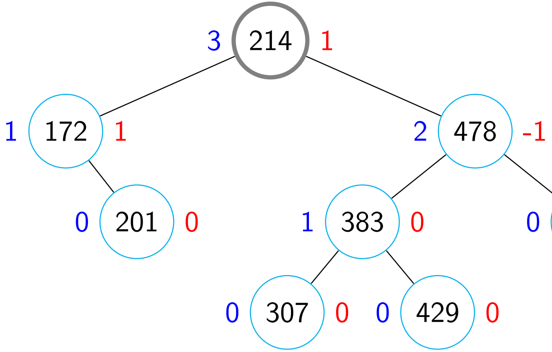
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



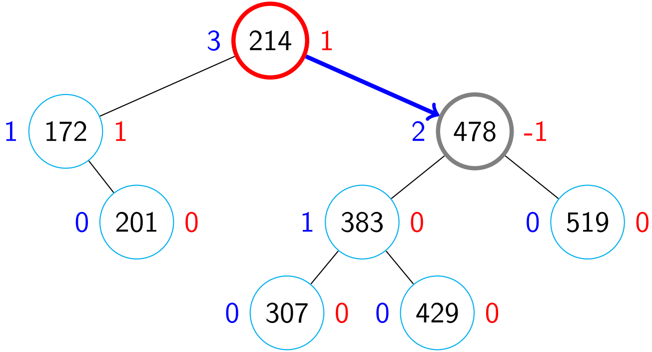
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



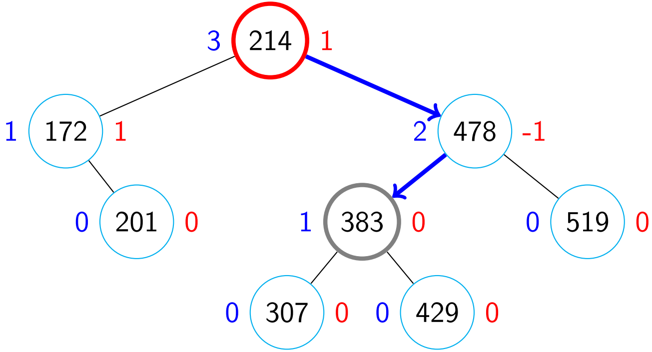
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



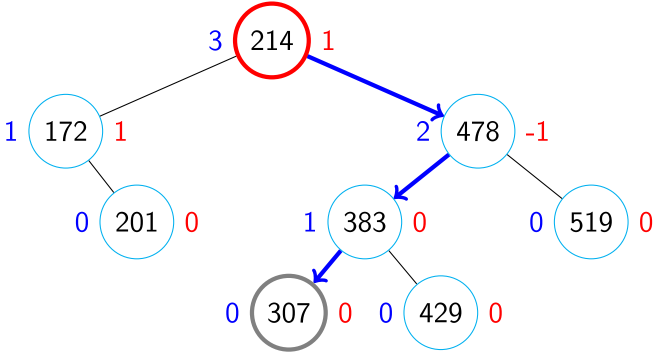
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



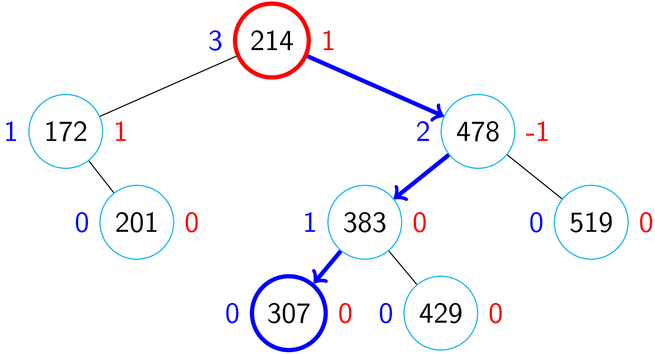
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



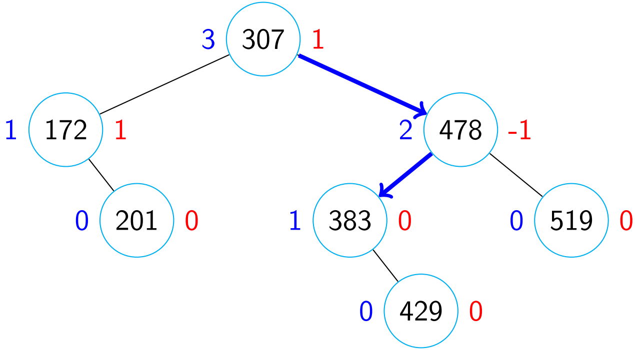
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



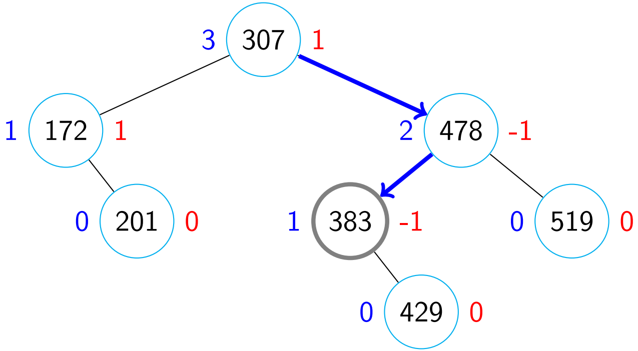
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



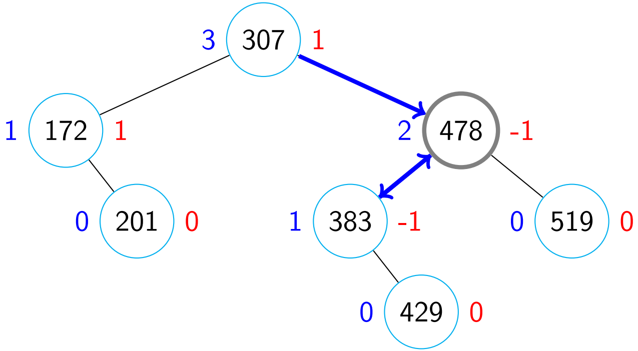
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



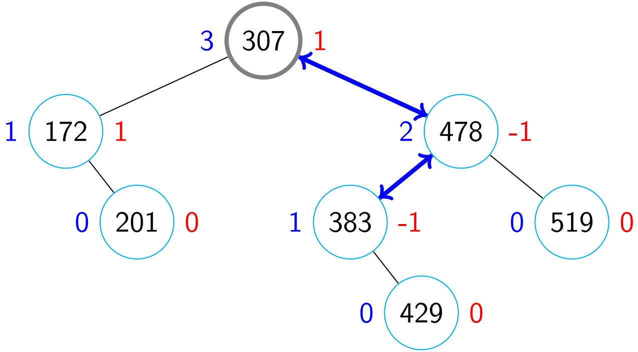
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



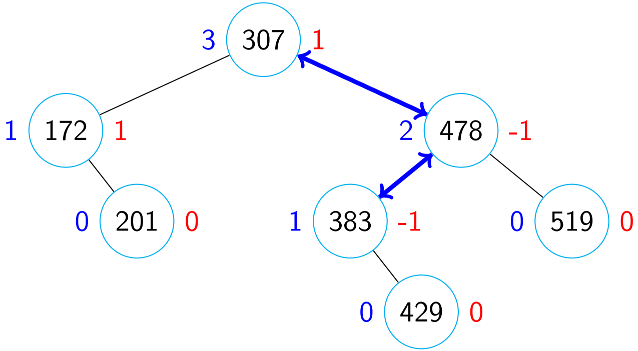
Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



Removing

For example, if you were removing 214 (heights are to the left and in blue, and balance factors are to the right and in red; use the successor):



Adding

procedure $\textsc{Add}(\mathit{data}, \mathit{node})$
  if *node* is not valid **then**
    return new node containing *data*
  else
    if $\mathit{data} = \mathit{node.data}$ **then**
      Implementation-defined behavior
    else if $\mathit{data} < \mathit{node.data}$ **then**
      $\mathit{node.left} = \textsc{Add}(\mathit{data}, \mathit{node.left})$
    else
      $\mathit{node.right} = \textsc{Add}(\mathit{data}, \mathit{node.right})$
    end if

Adding

```
        Update heights and balance factors
        if subtree is out of balance then
            Balance subtree
            node ← new root of subtree
            Update heights and balance factors of node
        end if
        return node
    end if
end procedure
```

(Removing is done in a similar way.)

Subtree Update Method

- Remember this from BSTs?

Subtree Update Method

- Remember this from BSTs?
- This method will make coding an AVL tree easier because you can easily update the left/right child of the parent node after doing a rotation.

Performance

- The average case of insertion, search, and deletion for an AVL tree are all $O(\log n)$.

Performance

- The average case of insertion, search, and deletion for an AVL tree are all $O(\log n)$.
- If the data items are added in sequential order, then the rotations that would be done would prevent the tree from turning into a linked list. Therefore, the worst case is also $O(\log n)$.