Balanced Search Trees

Written by Rachel Singh, last updated Nov 28, 2018

About

There are several types of Balanced Search Trees we can use to ensure that the data we have in a data structure stay sorted, and that the tree stays balanced.

Topics

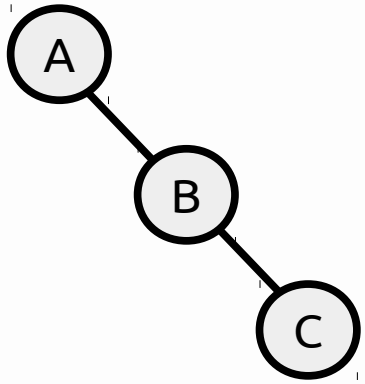
1. What is a Balanced Search Tree?
2. What is an AVL Tree?
3. Balancing the AVL Tree
4. Efficiency of the AVL Tree

What is a Balanced Search Tree?

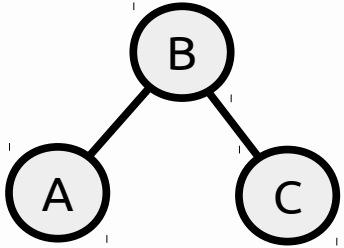
1. What is a balanced search tree?

Remember that with a Binary Search Tree, we store all nodes with "less" values to the left, and all nodes with "greater" values to the right.

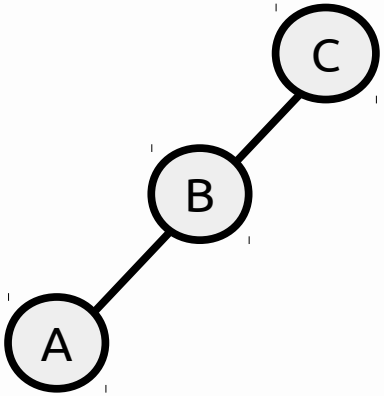
Because of this, depending on the order you insert your data, the tree can become pretty linear.



Insert: A, B, C

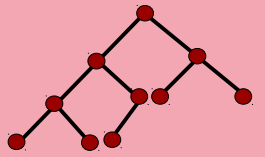


Insert: B, A, C



Insert: C, B, A

Notes



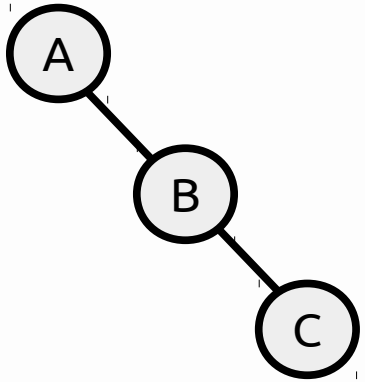
A complete binary tree of height ***h***…

- All nodes at level ***h – 2*** have two children each.
- When a node at level ***h – 1*** has children, all sibling nodes to its left have two children, and
- When a node at level ***h – 1*** has one child, it is a left child.

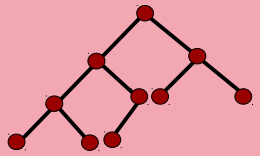
From Data Abstraction & Problem Solving Walls and Mirrors, 7th ed, by Carrano & Henry, pg 451

1. What is a balanced search tree?

If our binary search tree ends up unbalanced off to one side, this kills its efficiency – its efficiency approaches closer to linear time, O(n), instead of O(log n).



Notes



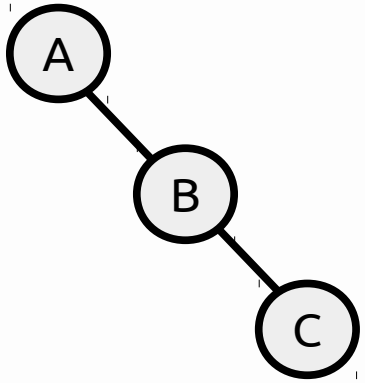
A complete binary tree of height ***h***…

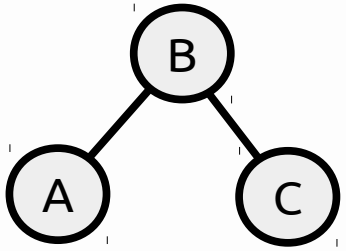
- All nodes at level ***h – 2*** have two children each.
- When a node at level ***h – 1*** has children, all sibling nodes to its left have two children, and
- When a node at level ***h – 1*** has one child, it is a left child.

From Data Abstraction & Problem Solving Walls and Mirrors, 7th ed, by Carrano & Henry, pg 451

1. What is a balanced search tree?

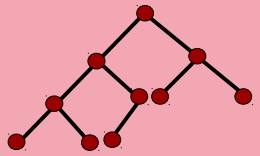
With a Balanced Search Tree (also known as a Self-balancing Binary Search Tree) are structures that deal with inserts and removals a bit more intelligently than a vanilla Binary Search Tree.





With the extra sophistication, these data structures basically “self-balance”, so that we can ensure that the structure is always in a balanced state as we use it.

Notes



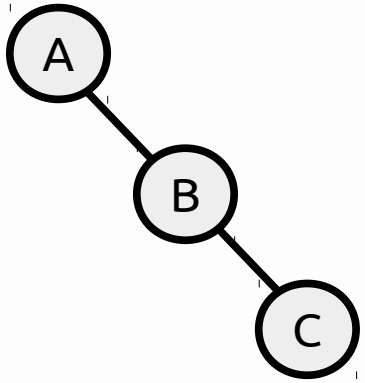
A complete binary tree of height ***h***…

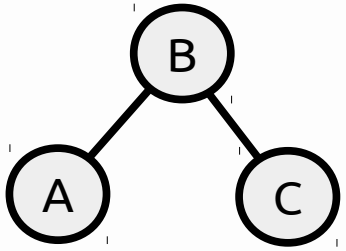
- All nodes at level ***h – 2*** have two children each.
- When a node at level ***h – 1*** has children, all sibling nodes to its left have two children, and
- When a node at level ***h – 1*** has one child, it is a left child.

From Data Abstraction & Problem Solving Walls and Mirrors, 7th ed, by Carrano & Henry, pg 451

1. What is a balanced search tree?

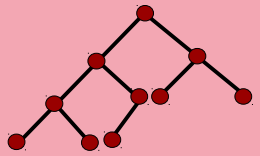
With a Balanced Search Tree (also known as a Self-balancing Binary Search Tree) are structures that deal with inserts and removals a bit more intelligently than a vanilla Binary Search Tree.





With the extra sophistication, these data structures basically “self-balance”, so that we can ensure that the structure is always in a balanced state as we use it.

Notes



A complete binary tree of height ***h***…

- All nodes at level ***h – 2*** have two children each.
- When a node at level ***h – 1*** has children, all sibling nodes to its left have two children, and
- When a node at level ***h – 1*** has one child, it is a left child.

From Data Abstraction & Problem Solving Walls and Mirrors, 7th ed, by Carrano & Henry, pg 451

1. What is a balanced search tree?

There are different types of Balanced Search Trees, but we will only cover one example.

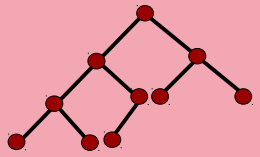
AVL Tree

2-3 Tree

Red-Black Tree

2-3-4 Tree

Notes



A complete binary tree of height ***h***…

- All nodes at level ***h – 2*** have two children each.
- When a node at level ***h – 1*** has children, all sibling nodes to its left have two children, and
- When a node at level ***h – 1*** has one child, it is a left child.

From Data Abstraction & Problem Solving Walls and Mirrors, 7th ed, by Carrano & Henry, pg 451

What is an AVL Tree?

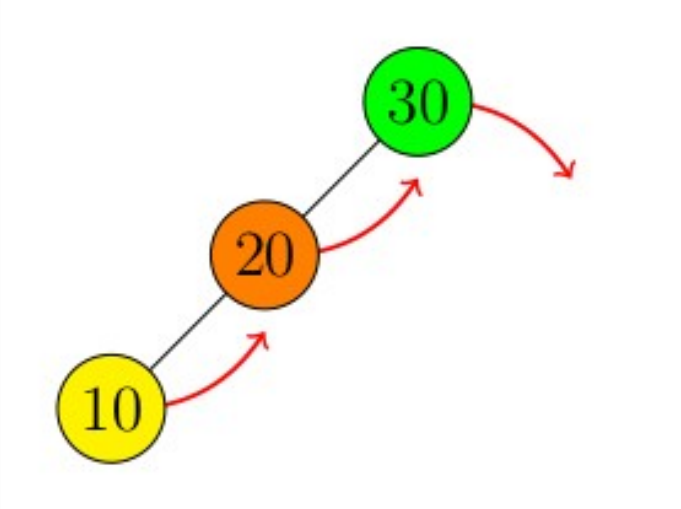
2. What is an AVL Tree?

For an AVL tree, the two subtrees of the root differ by no more than one. After an operation, if the height difference is more than one, we do rebalancing to fix the tree.

Notes

2. What is an AVL Tree?

Say that we have the following unbalanced tree.

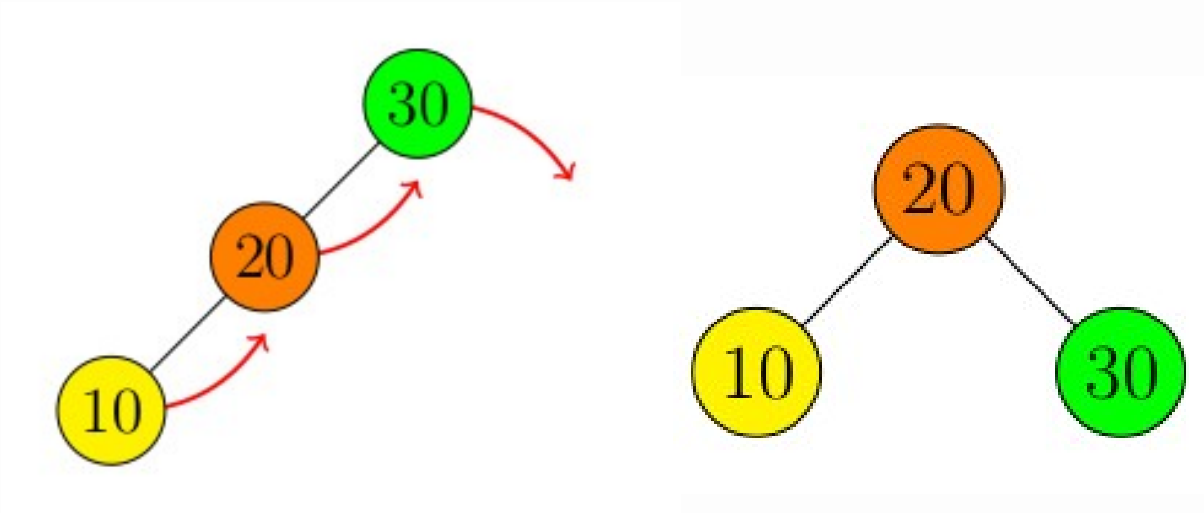


To balance it, we push the 20-node up, which pushes the 30-node down.

Notes

2. What is an AVL Tree?

Say that we have the following unbalanced tree.



To balance it, we push the 20-node up, which pushes the 30-node down.

Notes

2. What is an AVL Tree?

We need to review some terminology in order to be able to use AVL Trees:

Height of a node: The height of a node is the amount of edges on the longest path between that node and a leaf. A leaf node has a height of 0.

Height of a tree: The height of a tree is the height of its root node.

A height-balanced tree: A tree is height balanced (aka, simply balanced), if the height of any node's right subtree differs from the height of the node's left subtree by no more than 1.

Notes

Height of a node:
The amount of edges on the longest path between the node and a leaf. Leaf height = 0.

Height of a tree:
The height of the root node of the tree.

Height-balanced tree:
A tree is height-balanced if the height of any node's right subtree differs from the height of the left subtree by no more than 1.

2. What is an AVL Tree?

We need to review some terminology in order to be able to use AVL Trees:

Left subtree of n: According to the book, “In a binary tree, the left child of node n plus its descendants [is a left subtree of node n].”

However, to work with the definition of a balance factor from Wikipedia, we need to use Wikipedia’s definition of a subtree: “A subtree of a tree T is a tree consisting of a node T and all of its descendants in T .”

So for our purposes, Lef tSubtree(n) is going to be n plus its left descendants.

Notes

Height of a node:
The amount of edges on the longest path between the node and a leaf. Leaf height = 0.

Height of a tree:
The height of the root node of the tree.

Height-balanced tree:
A tree is height-balanced if the height of any node’s right subtree differs from the height of the left subtree by no more than 1.

2. What is an AVL Tree?

We need to review some terminology in order to be able to use AVL Trees:

Balance Factor: After an operation is performed on an AVL tree, the Balance Factor of each node is calculated. The equation for a node n's balance factor is:

BalanceFactor(n) = Height(RightSubtree(n)) – Height(Left Subtree(n))

If the **absolute value** of the Balance Factor is more than 1, then we need to rebalance. In other words, the balance factor for every node must be -1, 0, or 1.

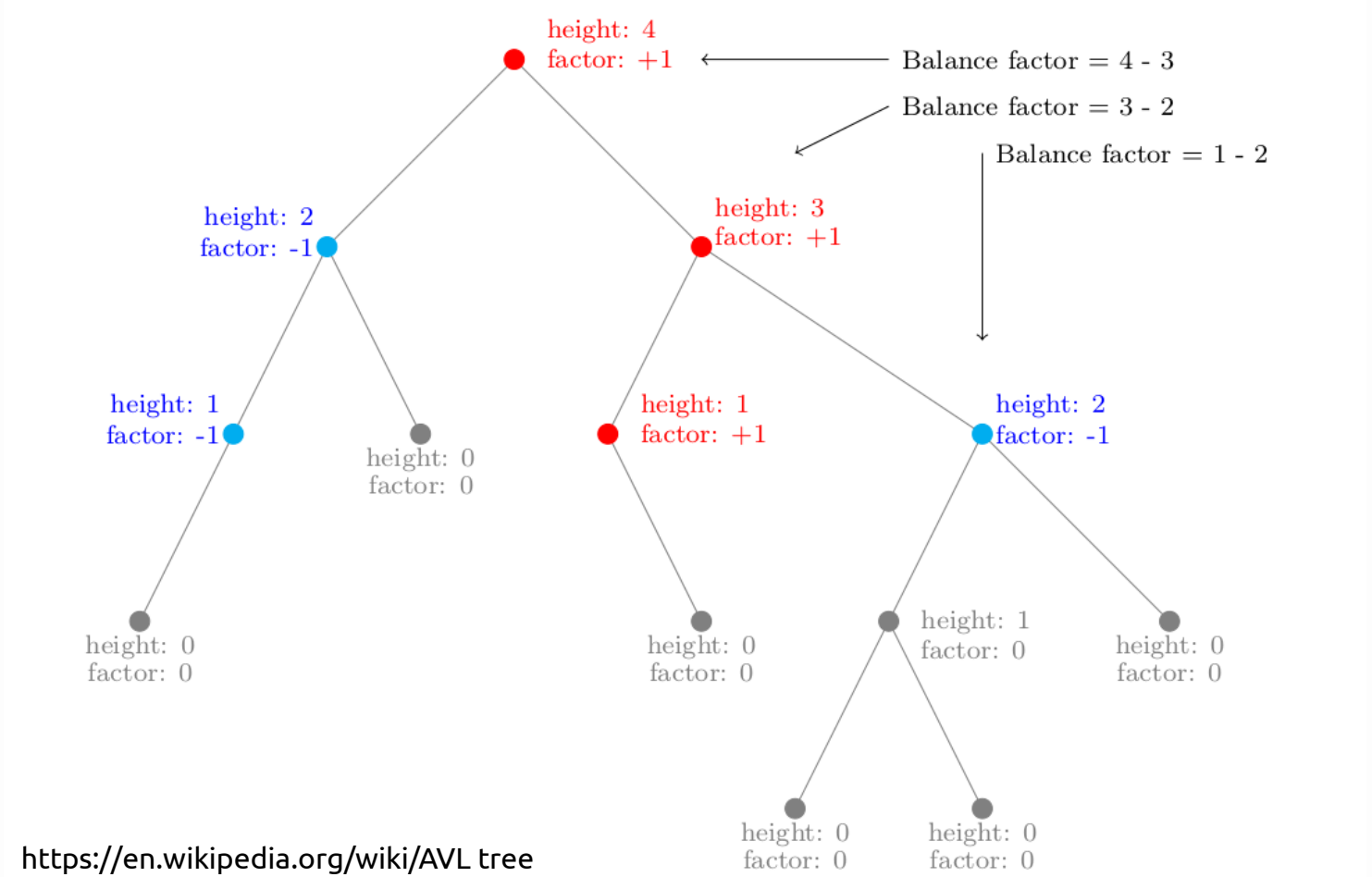
Notes

Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

2. What is an AVL Tree?



https://en.wikipedia.org/wiki/AVL tree

Notes

Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

Balancing the AVL Tree

3. Balancing the AVL Tree

We will need to do one of four types of rotations, based on how unbalanced the AVL tree is…

- Left Rotation (LL)
- Right Rotation (RR)
- Left-right Rotation / Double-left Rotation (LR)
- Right-left Rotation / Double-right Rotation (RL)

Notes

Balance Factor:
The balance factor of node n is:

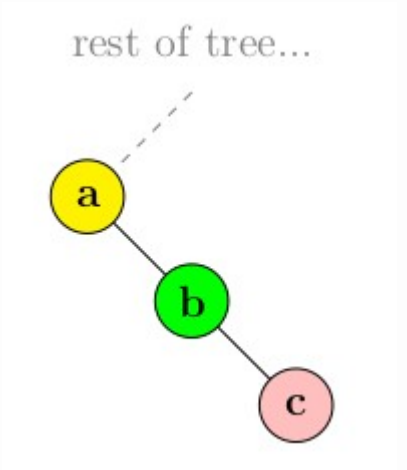
BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

Left Rotation:

Here we have a subtree that has a balance factor of +2. The left subtree of **a** is 0, and the right subtree of **a** is 2. The result is 2 – 0; So the balance factor is +2.



Notes

Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

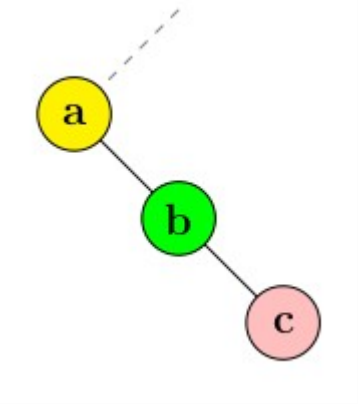
If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

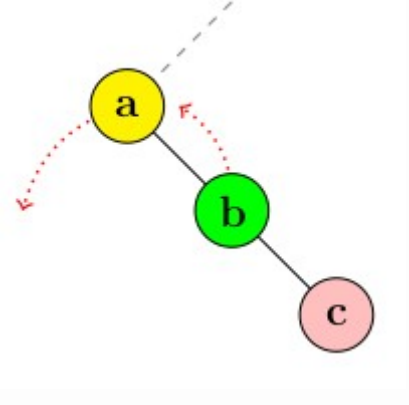
To adjust it, we will do the following steps to perform a **left rotation:**

1) **a's** right child, **b**, becomes the new root of this subtree.
2) If **b** has a left child, it becomes **a's** right child.
3) **a** becomes **b's** left child.

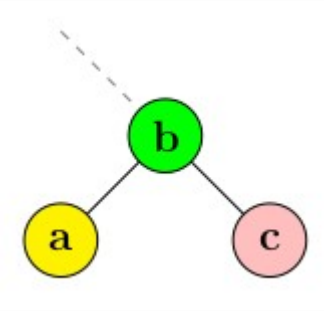
Original subtree



Trajectory of rotation



Rotated subtree



Notes

Balance Factor:
The balance factor of node n is:

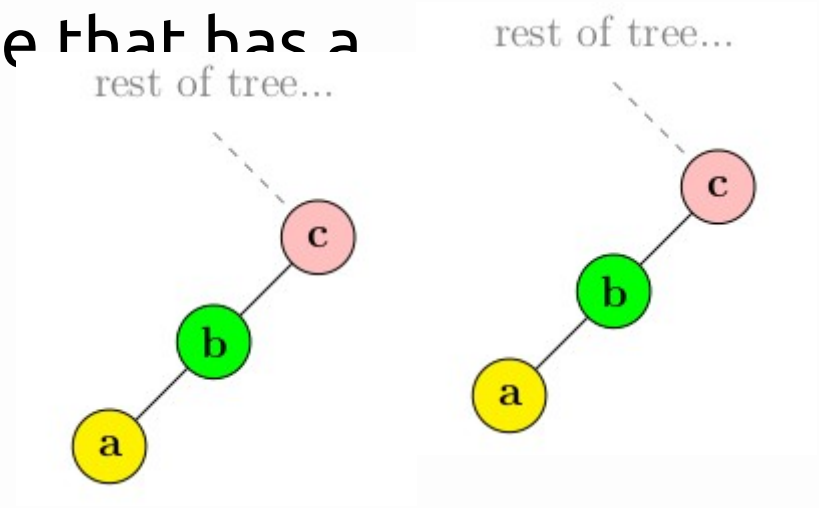
BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

Right Rotation:

Here we have a subtree that has a balance factor of -2.



Notes

Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

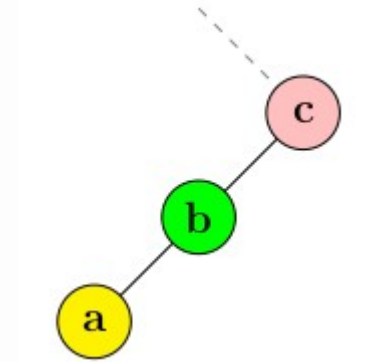
If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

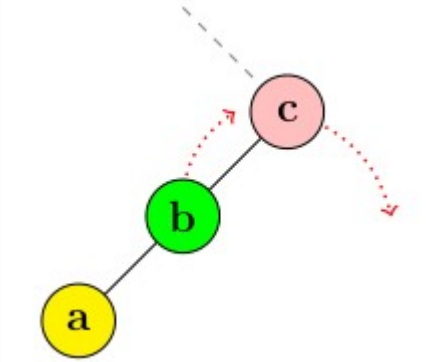
To adjust it, we will do the following steps to perform a **left rotation:**

1) **c's** left child, **b**, becomes the new root of this subtree.
2) If **b** has a right child, it becomes **c's** left child.
3) **c** becomes **b's** right child.

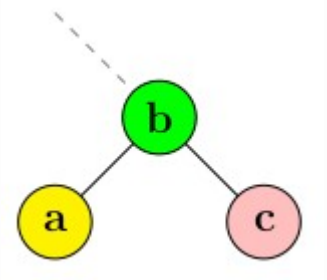
Original subtree



Trajectory of rotation



Rotated subtree



Notes

Balance Factor:
The balance factor of node n is:

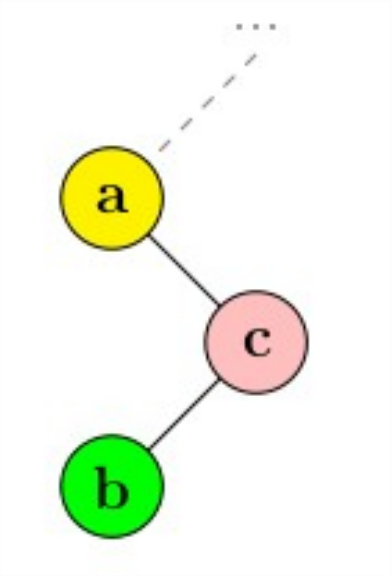
BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

Left-Right Rotation:

In some cases, we can't do a simple Left or Right rotation to balance a subtree, but need to do **two** rotations.



Notes

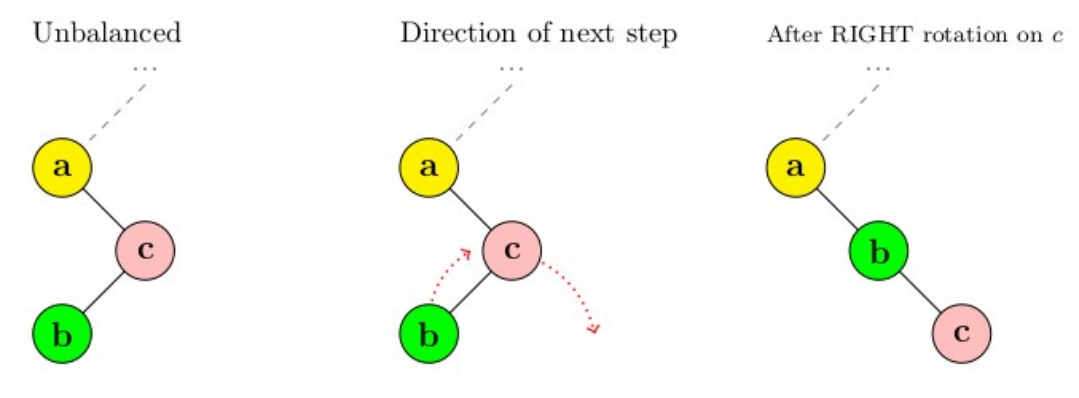
Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

First, we rotate the lower part of the subtree:



Notes

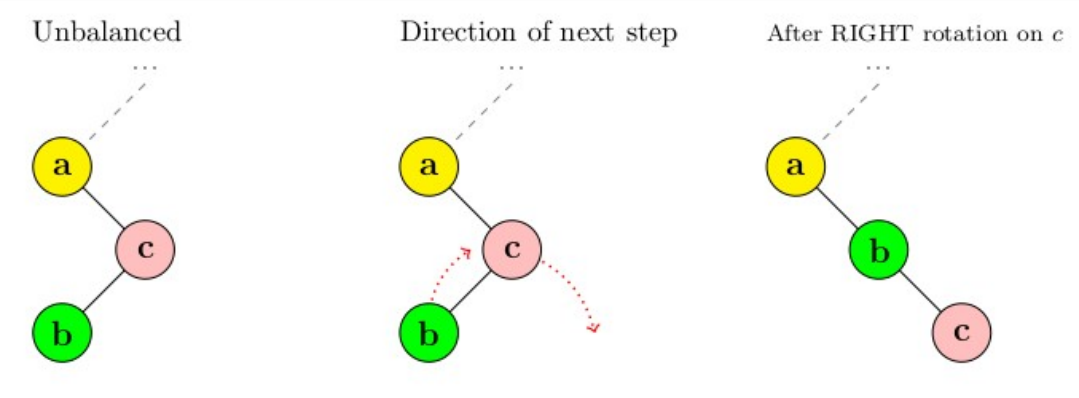
Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

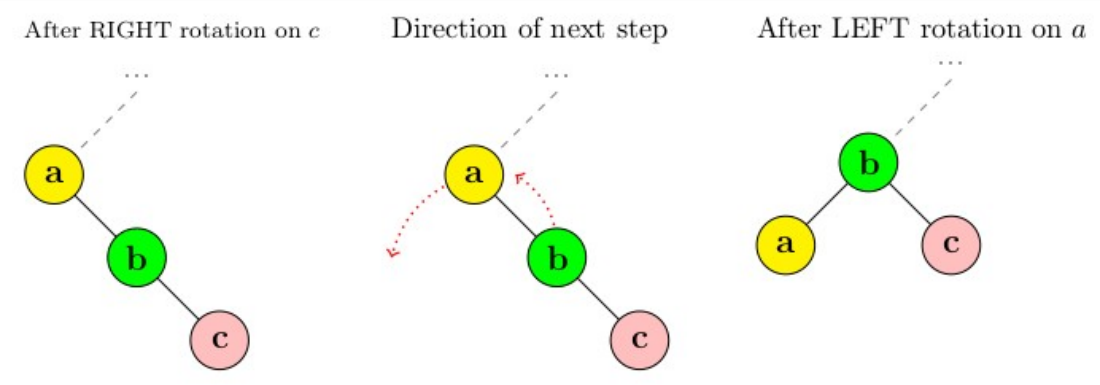
If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

First, we rotate the lower part of the subtree:



Then we rotate the upper part of the subtree:



Notes

Balance Factor:
The balance factor of node n is:

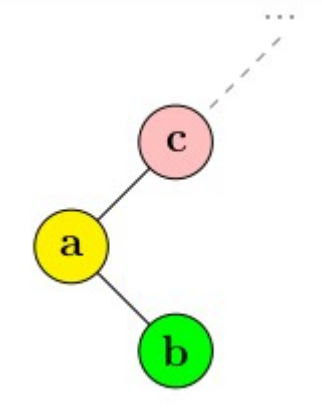
BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

Right-Left Rotation:

And, likewise, when we have a tree twisted the opposite way, we will again do a two-step rotation.



Notes

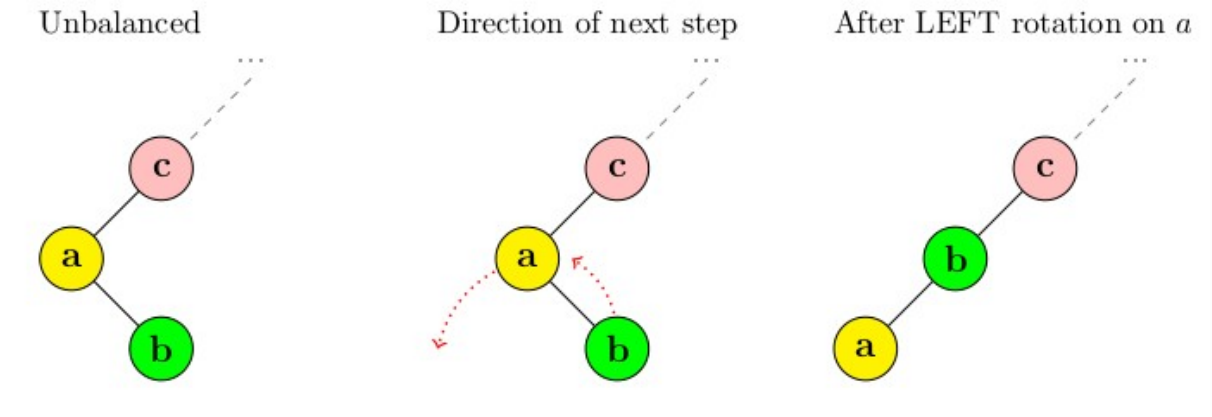
Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

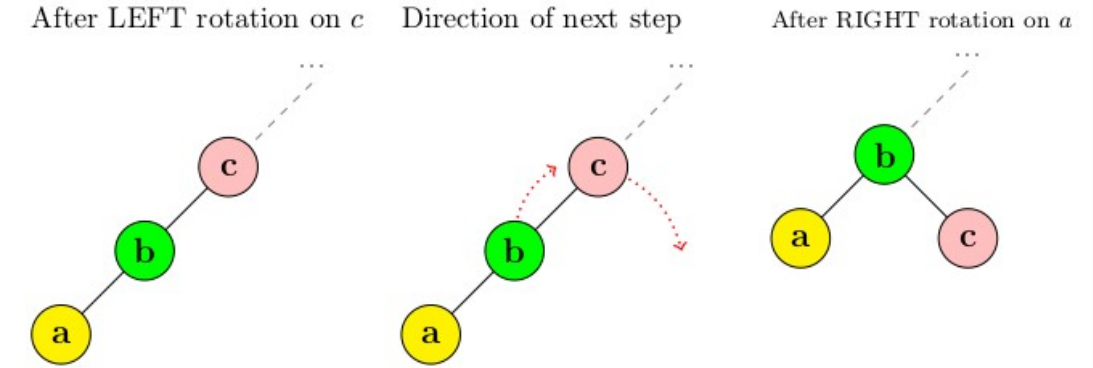
If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

First, we rotate the lower part of the subtree:



Then we rotate the upper part of the subtree:



Notes

Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

3. Balancing the AVL Tree

To try to make these rotations more clear, here's some pseudocode:

```
IF tree is right heavy
{
     IF tree's right subtree is left heavy
     {
          DO double left rotation (LR)
     }
     ELSE
     {
          DO single left rotation (LL)
     }
}
ELSE IF tree is left heavy
{
     IF tree's left subtree is right heavy
     {
          DO double right rotation (RL)
     }
     ELSE
     {
          DO single right rotation (RR)
     }
}
```

Notes

Balance Factor:
The balance factor of node n is:

BF(n) =
Height(RightSubtree(n)) – Height(LeftSubtree(n))

If the absolute value of the balance factor is more than 1, we need to rebalance the tree.

Efficiency of the AVL Tree

3. Efficiency of the AVL Tree

Operation	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

Notes

Conclusion

We aren't going to program a balanced search tree in this class, though you might do it later on in another course.

The main idea here is to have you understand how they work.