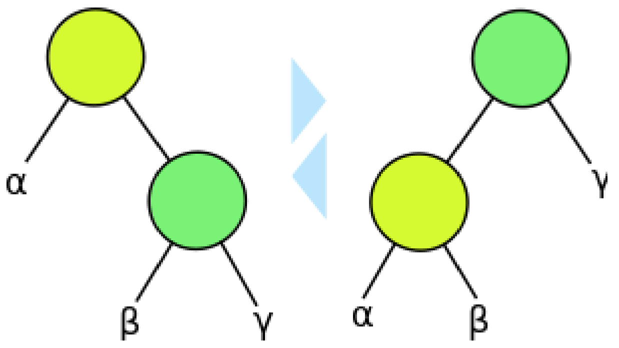
WIKIPEDIA

# Tree rotation

In discrete mathematics, **tree rotation** is an operation on a binary tree that changes the structure without interfering with the order of the elements. A tree rotation moves one node up in the tree and one node down. It is used to change the shape of the tree, and in particular to decrease its height by moving smaller subtrees down and larger subtrees up, resulting in improved performance of many tree operations.

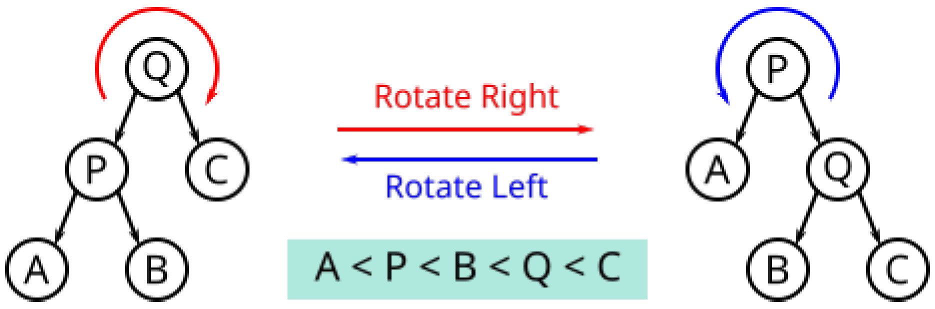


Generic tree rotations.

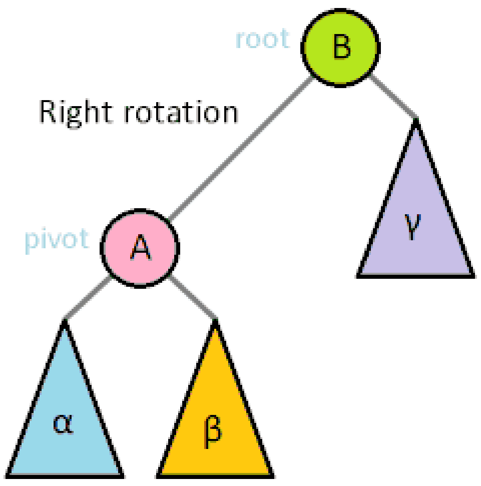
There exists an inconsistency in different descriptions as to the definition of the **direction of rotations**. Some say that the direction of rotation reflects the direction that a node is moving upon rotation (a left child rotating into its parent's location is a right rotation) while others say that the direction of rotation reflects which subtree is rotating (a left subtree rotating into its parent's location is a left rotation, the opposite of the former). This article takes the approach of the directional movement of the rotating node.

## Contents

## Illustration



The right rotation operation as shown in the adjacent image is performed with *Q* as the root and hence is a right rotation on, or rooted at, *Q*. This operation results in a rotation of the tree in the clockwise direction. The inverse operation is the left rotation, which results in a movement in a counter-clockwise direction (the left rotation shown above is rooted at *P*). The key to understanding how a rotation functions is to understand its constraints. In particular the order of the leaves of the tree (when read left to right for example) cannot change (another way to think of it is that the order that the leaves would be visited in an in-order traversal must be the same after the operation as before). Another constraint is the main property of a binary search tree, namely that the right child is greater than the parent and the left child is less than the parent. Notice that the right child of a left child of the root of a sub-tree (for example node B in the diagram for the tree rooted at Q) can become the left child of the root, that itself becomes the right child of the "new" root in the rotated sub-tree, without violating either of those constraints. As you can see in the diagram, the order of the leaves doesn't change. The opposite operation also preserves the order and is the second kind of rotation.



Assuming this is a binary search tree, as stated above, the elements must be interpreted as variables that can be compared to each other. The alphabetic characters to the left are used as placeholders for these variables. In the animation to the right, capital alphabetic characters are used as variable placeholders while lowercase Greek letters are placeholders for an entire set of variables. The circles represent individual nodes and the triangles represent subtrees. Each subtree could be empty, consist of a single node, or consist of any number of nodes.

## Detailed illustration

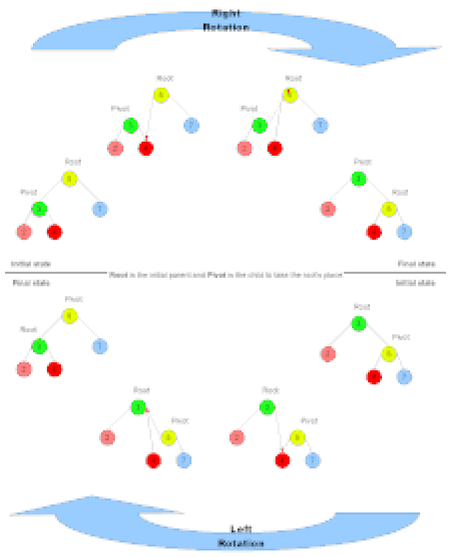
When a subtree is rotated, the subtree side upon which it is rotated increases its height by one node while the other subtree decreases its height. This makes tree rotations useful for rebalancing a tree.

Consider the terminology of **Root** for the parent node of the subtrees to rotate, **Pivot** for the node which will become the new parent node, **RS** for the side of rotation and **OS** for the opposite side of rotation. For the root Q in the diagram above, **RS** is C and **OS** is P. Using these terms, the pseudo code for the rotation is:

```
Pivot = Root.OS
Root.OS = Pivot.RS
Pivot.RS = Root
Root = Pivot
```

This is a constant time operation.

The programmer must also make sure that the root's parent points to the pivot after the rotation. Also, the programmer should note that this operation may result in a new root for the entire tree and take care to update pointers accordingly.


Right Rotation
Root
Pivot
Initial state
Final state
Root is the initial parent and Pivot is the child to take the root's place
Left Rotation

Pictorial description of how rotations are made.

## Inorder invariance

The tree rotation renders the inorder traversal of the binary tree invariant. This implies the order of the elements are not affected when a rotation is performed in any part of the tree. Here are the inorder traversals of the trees shown above:

```
Left tree: ((A, P, B), Q, C)        Right tree: (A, P, (B, Q, C))
```

Computing one from the other is very simple. The following is example Python code that performs that computation:

```python
def right_rotation(treenode):
    left, Q, C = treenode
    A, P, B = left
    return (A, P, (B, Q, C))
```

Another way of looking at it is:

Right rotation of node Q:

```
Let P be Q's left child.
Set Q's left child to be P's right child.
[Set P's right-child's parent to Q]
Set P's right child to be Q.
[Set Q's parent to P]
```

Left rotation of node P:

```
Let Q be P's right child.
Set P's right child to be Q's left child.
[Set Q's left-child's parent to P]
Set Q's left child to be P.
[Set P's parent to Q]
```

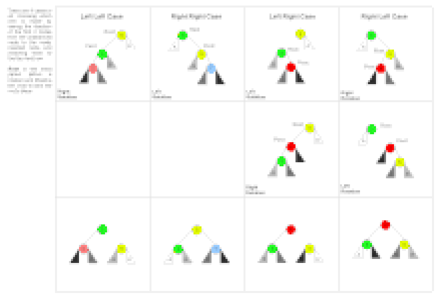
All other connections are left as-is.

There are also *double rotations*, which are combinations of left and right rotations. A *double left* rotation at X can be defined to be a right rotation at the right child of X followed by a left rotation at X; similarly, a *double right* rotation at X can be defined to be a left rotation at the left child of X followed by a right rotation at X.

Tree rotations are used in a number of tree data structures such as AVL trees, red-black trees, WAVL trees, splay trees, and treaps. They require only constant time because they are *local* transformations: they only operate on 5 nodes, and need not examine the rest of the tree.

## Rotations for rebalancing

A tree can be rebalanced using rotations. After a rotation, the side of the rotation increases its height by 1 whilst the side opposite the rotation decreases its height similarly. Therefore, one can strategically apply rotations to nodes whose left child and right child differ in height by more than 1. Self-balancing binary search trees apply this operation automatically. A type of tree which uses this rebalancing technique is the AVL tree.



Pictorial description of how rotations cause rebalancing in an AVL tree.

## Rotation distance

The rotation distance between any two binary trees with the same number of nodes is the minimum number of rotations needed to transform one into

**Unsolved problem in computer science**:

*Can the rotation distance between two binary trees be* the other. With this distance, the set of *n*-node binary trees becomes a metric space: the distance is symmetric, positive when given two different trees, and satisfies the triangle inequality.

*computed in polynomial time?*

(more unsolved problems in computer science)

It is an open problem whether there exists a polynomial time algorithm for calculating rotation distance.

Daniel Sleator, Robert Tarjan and William Thurston showed that the rotation distance between any two $n$-node trees (for $n \geq 11$) is at most $2n - 6$, and that some pairs of trees are this far apart as soon as $n$ is sufficiently large.[1] Lionel Pournin showed that, in fact, such pairs exist whenever $n \geq 11$.[2]

## See also

- AVL tree, red-black tree, and splay tree, kinds of binary search tree data structures that use rotations to maintain balance.
- Associativity of a binary operation means that performing a tree rotation on it does not change the final result.
- The Day–Stout–Warren algorithm balances an unbalanced BST.
- Tamari lattice, a partially ordered set in which the elements can be defined as binary trees and the ordering between elements is defined by tree rotation.

## References

1. Sleator, Daniel D.; Tarjan, Robert E.; Thurston, William P. (1988), "Rotation distance, triangulations, and hyperbolic geometry", *Journal of the American Mathematical Society*, **1** (3): 647–681, doi:10.2307/1990951 (https://doi.org/10.2307%2F1990951), JSTOR 1990951 (https://www.jstor.org/stable/1990951), MR 0928904 (https://www.ams.org/mathscinet-getitem?mr=0928904).
2. Pournin, Lionel (2014), "The diameter of associahedra", *Advances in Mathematics*, **259**: 13–42, arXiv:1207.6296 (https://arxiv.org/abs/1207.6296), doi:10.1016/j.aim.2014.02.035 (https://doi.org/10.1016%2Fj.aim.2014.02.035), MR 3197650 (https://www.ams.org/mathscinet-getitem?mr=3197650).

## External links

- The AVL Tree Rotations Tutorial (http://fortheloot.com/public/AVLTreeTutorial.rtf) (RTF) by John Hargrove