A 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. They are 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. There exists an inconsistency in definitions as to the direction of rotations. Some say that the direction of a rotation depends on the side which the tree nodes are shifted upon whilst others say that it depends on which child takes the root’s place (opposite of the former). This article takes the approach of the side where the nodes get shifted to.

Illustration

The right rotation operation as shown in the image above 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 symmetric operation is the left rotation which results in a movement in an anti-clockwise direction (the left rotation shown above is rooted at P).

Detailed Illustration

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

Using the terminology of Root, Pivot for the parent node of the subtrees to rotate, RS for rotation side upon which to rotate and OS for opposite side of rotation. In the above diagram for the root Q, the RS is C and the OS is P. 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.

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:

```
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:

Left Rotation of node P:
Let Q be P’s right child.
Set Q to be the new root.
Set P’s right child to be Q’s left child.
Set Q’s left child to be P.
Right Rotation of node Q:

Let P be Q's left child.
Set P to be the new root.
Set Q's left child to be P's right child.
Set P's right child to be 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, 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 while the side opposite the rotation decreases its height by 1. Therefore, one can strategically apply rotations to make the difference in height between left and right sides at X decrease 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.

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 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.

It is an open problem whether there exists a polynomial time algorithm for calculating rotation distance. However, Daniel Sleator, Robert Tarjan and William Thurston showed that the rotation distance between any two n-node trees (for n ≥ 11) is at most 2n − 6, and that infinitely many pairs of trees are this far apart.

External links

- [Java applets demonstrating tree rotations](http://www.cs.cmu.edu/~aboek/rotations/)
- [Java applets demonstrating tree rotations](http://www.cs.cmu.edu/~aboek/rotations/)
- [The AVL Tree Rotations Tutorial](http://www.cs.cmu.edu/~aboek/rotations/)

See also

- AVL trees, red-black trees, and splay trees. Index of binary search tree data structures that use rotations to maintain balance.
- treaps, a partially ordered set in which the elements can be defined as binary trees and the ordering between elements is defined by two

The source of this article is [wikipedia](https://en.wikipedia.org), the free encyclopedia. The text of this article is licensed under the [GFDL](https://en.wikipedia.org/wiki/GNU_Free_Documentation_License).
There are 4 cases in all, choosing which one is made by seeing the direction of the first 2 nodes from the unbalanced node to the newly inserted node and matching them to the top most row.

**Root** is the initial parent before a rotation and **Pivot** is the child to take the root’s place.