

[← Previous \(red-black-trees.html\)](#)[Next → \(comparison-of-search-trees.html\)](#)

Splay Tree Datastructure

Splay tree is another variant of a binary search tree. In a splay tree, recently accessed element is placed at the root of the tree. A splay tree is defined as follows...

Splay Tree is a self - adjusted Binary Search Tree in which every operation on element rearranges the tree so that the element is placed at the root position of the tree.

In a splay tree, every operation is performed at the root of the tree. All the operations in splay tree are involved with a common operation called "Splaying".

Splaying an element, is the process of bringing it to the root position by performing suitable rotation operations.

In a splay tree, splaying an element rearranges all the elements in the tree so that splayed element is placed at the root of the tree.

By splaying elements we bring more frequently used elements closer to the root of the tree so that any operation on those elements is performed quickly. That means the splaying operation automatically brings more frequently used elements closer to the root of the tree.

Every operation on splay tree performs the splaying operation. For example, the insertion operation first inserts the new element using the binary search tree insertion process, then the newly inserted element is splayed so that it is placed at the root of the tree. The search operation in a splay tree is nothing but searching the element using binary search process and then splaying that searched element so that it is placed at the root of the tree.

In splay tree, to splay any element we use the following rotation operations...

Rotations in Splay Tree

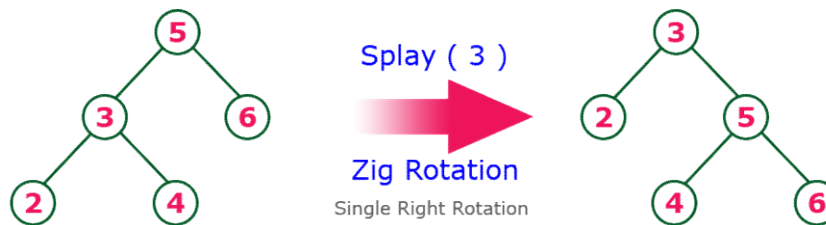
- 1. Zig Rotation
- 2. Zag Rotation
- 3. Zig - Zag Rotation

- 4. Zag - Zag Rotation
- 5. Zig - Zag Rotation
- 6. Zag - Zig Rotation

Example

Zig Rotation

The **Zig Rotation** in splay tree is similar to the single right rotation in AVL Tree rotations. In zig rotation, every node moves one position to the right from its current position. Consider the following example...



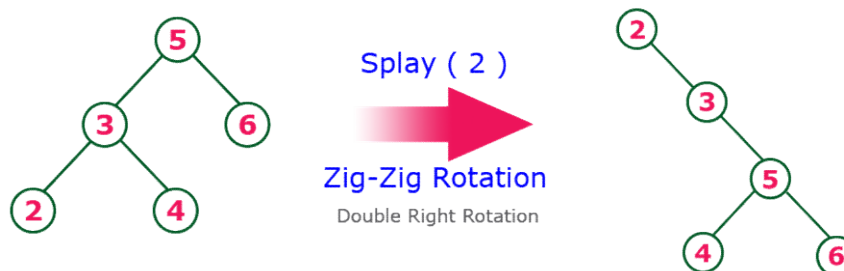
Zag Rotation

The **Zag Rotation** in splay tree is similar to the single left rotation in AVL Tree rotations. In zag rotation, every node moves one position to the left from its current position. Consider the following example...



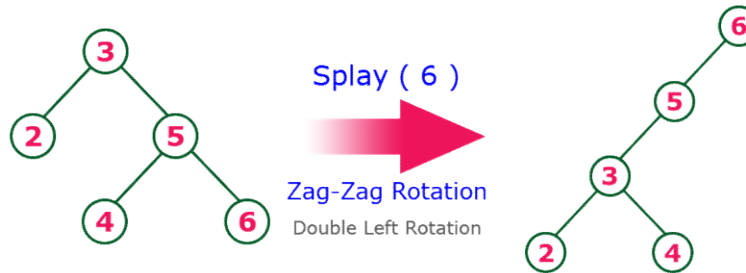
Zig-Zig Rotation

The **Zig-Zig Rotation** in splay tree is a double zig rotation. In zig-zig rotation, every node moves two positions to the right from its current position. Consider the following example...



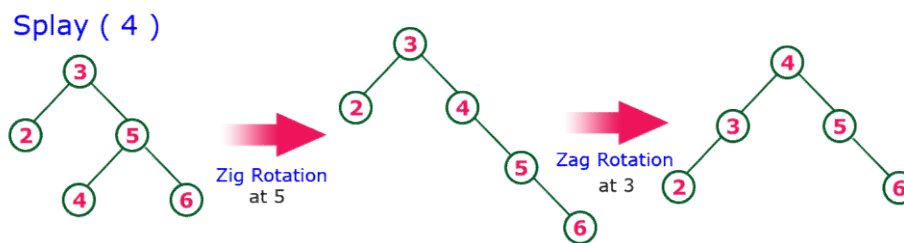
Zag-Zag Rotation

The **Zag-Zag Rotation** in splay tree is a double zag rotation. In zag-zag rotation, every node moves two positions to the left from its current position. Consider the following example...



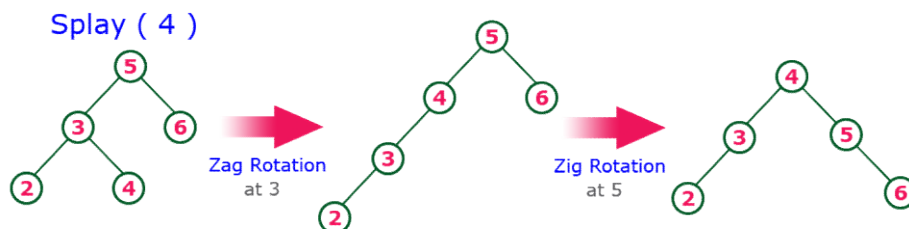
Zig-Zag Rotation

The **Zig-Zag Rotation** in splay tree is a sequence of zig rotation followed by zag rotation. In zig-zag rotation, every node moves one position to the right followed by one position to the left from its current position. Consider the following example...



Zag-Zig Rotation

The **Zag-Zig Rotation** in splay tree is a sequence of zag rotation followed by zig rotation. In zag-zig rotation, every node moves one position to the left followed by one position to the right from its current position. Consider the following example...



Every Splay tree must be a binary search tree but it is need not to be balanced tree.

Insertion Operation in Splay Tree

The insertion operation in Splay tree is performed using following steps...

Step 1 - Check whether tree is Empty.

Step 2 - If tree is Empty then insert the **newNode** as Root node and exit from the operation.

Step 3 - If tree is not Empty then insert the newNode as leaf node using Binary Search tree insertion logic.

Step 4 - After insertion, **Splay** the **newNode**

Deletion Operation in Splay Tree

The deletion operation in splay tree is similar to deletion operation in Binary Search Tree. But before deleting the element, we first need to **splay** that element and then delete it from the root position. Finally join the remaining tree using binary search tree logic.

[← Previous \(red-black-trees.html\)](#)

[Next → \(comparison-of-search-trees.html\)](#)

Place your ad here

Place your ad here

[Courses \(../courses.html\)](#) | [Downloads \(../downloads.html\)](#) | [About Us \(../authors.html\)](#) | [Contact Us \(../contact-us.html\)](#)

Website designed by Rajinikanth B

   
(<https://www.youtube.com/channel/UC9YHZpCPtRqt>)
(<https://www.facebook.com/rajnib>) (<https://plus.google.com/11720603518006305>) (<https://twitter.com/rajnib>)