

Splay Tree | Set 1 (Search)

The worst case time complexity of Binary Search Tree (BST) operations like search, delete, insert is $O(n)$. The worst case occurs when the tree is skewed. We can get the worst case time complexity as $O(\log n)$ with [AVL](#) and Red-Black Trees.

Can we do better than AVL or Red-Black trees in practical situations?

Like [AVL](#) and Red-Black Trees, Splay tree is also [self-balancing BST](#). The main idea of splay tree is to bring the recently accessed item to root of the tree, this makes the recently searched item to be accessible in $O(1)$ time if accessed again. The idea is to use locality of reference (In a typical application, 80% of the access are to 20% of the items). Imagine a situation where we have millions or billions of keys and only few of them are accessed frequently, which is very likely in many practical applications.

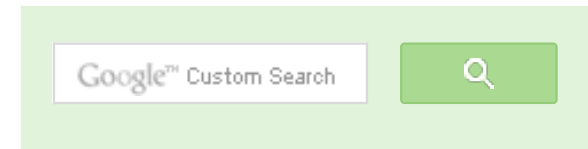
All splay tree operations run in $O(\log n)$ time on average, where n is the number of entries in the tree. Any single operation can take $\Theta(n)$ time in the worst case.

Search Operation

The search operation in Splay tree does the standard BST search, in addition to search, it also splays (move a node to the root). If the search is successful, then the node that is found is splayed and becomes the new root. Else the last node accessed prior to reaching the NULL is splayed and becomes the new root.

There are following cases for the node being accessed.

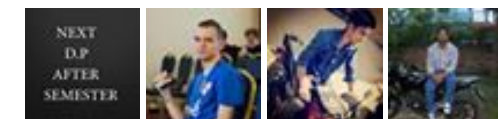
- 1) Node is root** We simply return the root, don't do anything else as the accessed node is already root.
- 2) Zig: Node is child of root** (the node has no grandparent). Node is either a left child of root (we



GeeksforGeeks



52,731 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

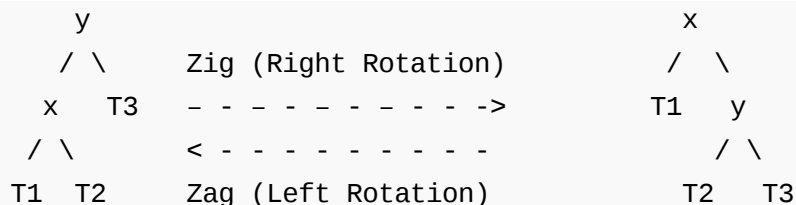
[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

[Geometric Algorithms](#)

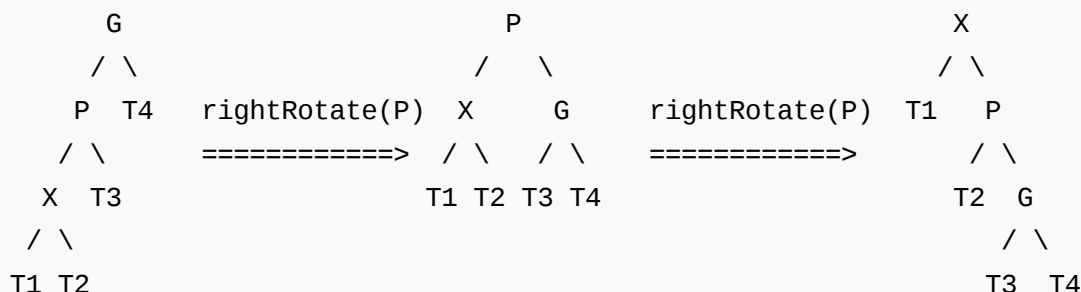
do a right rotation) or node is a right child of its parent (we do a left rotation).
T1, T2 and T3 are subtrees of the tree rooted with y (on left side) or x (on right side)



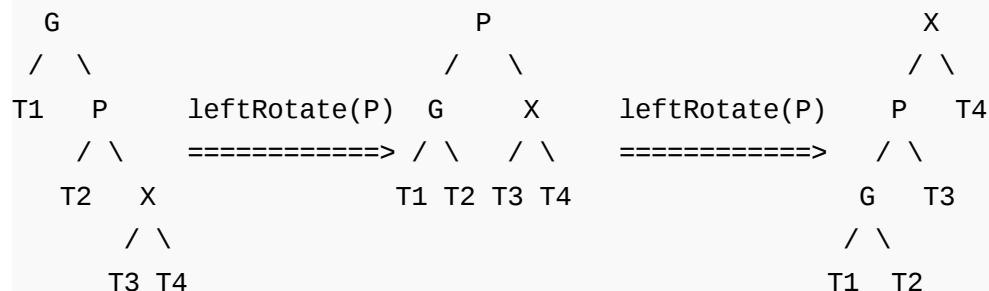
3) Node has both parent and grandparent. There can be following subcases.

.....**3.a) Zig-Zig and Zag-Zag** Node is left child of parent and parent is also left child of grand parent (Two right rotations) OR node is right child of its parent and parent is also right child of grand parent (Two Left Rotations).

Zig-Zig (Left Left Case):



Zag-Zag (Right Right Case):



.....**3.b) Zig-Zag and Zag-Zig** Node is left child of parent and parent is right child of grand parent (Left Rotation followed by right rotation) OR node is right child of its parent and parent is left child of grand parent (Right Rotation followed by left rotation).

Zig-Zag (Left Right Case):



Integrated

Desktop & Mobile Device
Management

ManageEngine
Desktop Central

Download

Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

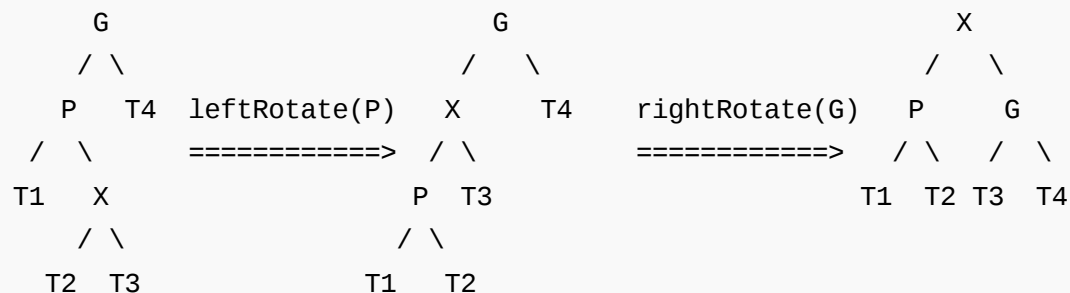
[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

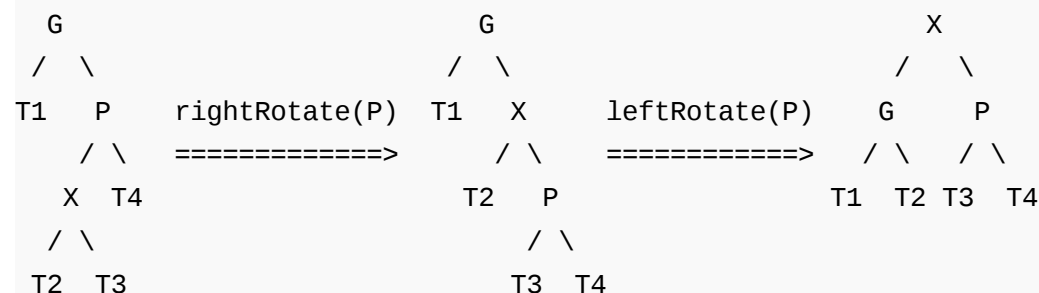
[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)

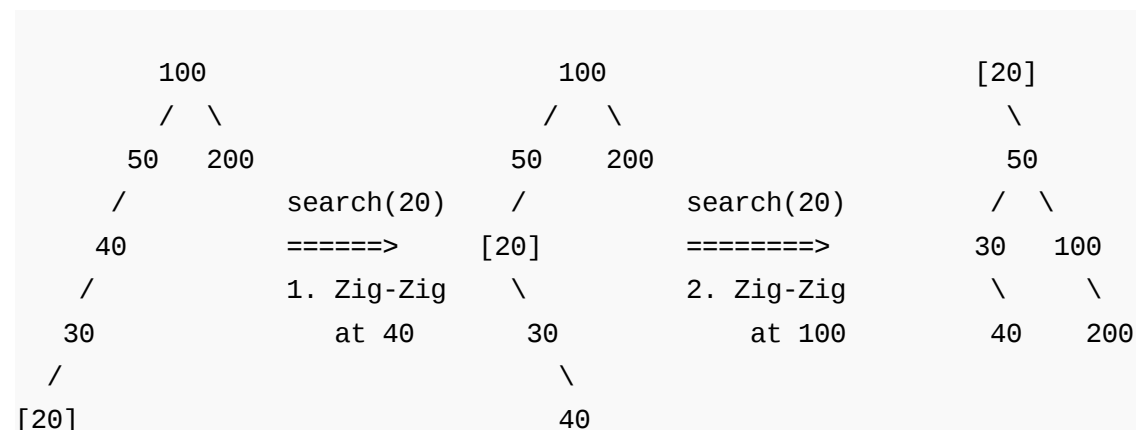
Zig-Zag (Left Right Case):



Zag-Zig (Right Left Case):



Example:



The important thing to note is, the search or splay operation not only brings the searched key to root, but also balances the BST. For example in above case, height of BST is reduced by 1.

Implementation:

// The code is adopted from <http://goo.gl/SDH9hH>

Integrated Desktop & Mobile Device Management



- App Management
- Policy Management
- Patch Management
- Software Deployment

Download

ManageEngine
Desktop Central

```
#include<stdio.h>
#include<stdlib.h>
```

```
// An AVL tree node
```

```
struct node
{
    int key;
    struct node *left, *right;
};
```

```
/* Helper function that allocates a new node with the given key and
   NULL left and right pointers. */
```

```
struct node* newNode(int key)
{
    struct node* node = (struct node*)malloc(sizeof(struct node));
    node->key = key;
    node->left = node->right = NULL;
    return (node);
}
```

```
// A utility function to right rotate subtree rooted with y
// See the diagram given above.
```

```
struct node *rightRotate(struct node *x)
{
    struct node *y = x->left;
    x->left = y->right;
    y->right = x;
    return y;
}
```

```
// A utility function to left rotate subtree rooted with x
// See the diagram given above.
```

```
struct node *leftRotate(struct node *x)
{
    struct node *y = x->right;
    x->right = y->left;
    y->left = x;
    return y;
}
```

```
// This function brings the key at root if key is present in tree.
// If key is not present, then it brings the last accessed item at
// root. This function modifies the tree and returns the new root
```

```
struct node *splay(struct node *root, int key)
{
    // Base cases: root is NULL or key is present at root
    if (root == NULL || root->key == key)
```



Recent Comments

affizerv Your example has two 4s on row 3,
that's why it...

[Backtracking | Set 7 \(Sudoku\)](#) · 27 minutes ago

RVM Can someone please elaborate this Qs
from above...

[Flipkart Interview | Set 6](#) · 47 minutes ago

Vishal Gupta I talked about as an Interviewer
in general,...

[Software Engineering Lab, Samsung Interview | Set
2](#) · 47 minutes ago

@meya Working solution for question 2 of
4f2f round....

[Amazon Interview | Set 53 \(For SDE-1\)](#) · 1 hour ago

sandeep void rearrange(struct node *head)
{...

Given a linked list, reverse alternate nodes and
append at the end · 2 hours ago

Neha I think that is what it should return as,
in...

Find depth of the deepest odd level leaf node · 2
hours ago

AdChoices

[► Binary Tree](#)

[► Red Black Tree](#)

```

    return root;

// Key lies in left subtree
if (root->key > key)
{
    // Key is not in tree, we are done
    if (root->left == NULL) return root;

    // Zig-Zig (Left Left)
    if (root->left->key > key)
    {
        // First recursively bring the key as root of left-left
        root->left->left = splay(root->left->left, key);

        // Do first rotation for root, second rotation is done after
        root = rightRotate(root);
    }
    else if (root->left->key < key) // Zig-Zag (Left Right)
    {
        // First recursively bring the key as root of left-right
        root->left->right = splay(root->left->right, key);


        // Do first rotation for root->left
        if (root->left->right != NULL)
            root->left = leftRotate(root->left);
    }

    // Do second rotation for root
    return (root->left == NULL)? root: rightRotate(root);
}
else // Key lies in right subtree
{
    // Key is not in tree, we are done
    if (root->right == NULL) return root;

    // Zag-Zig (Right Left)
    if (root->right->key > key)
    {
        // Bring the key as root of right-left
        root->right->left = splay(root->right->left, key);

        // Do first rotation for root->right
        if (root->right->left != NULL)
            root->right = rightRotate(root->right);
    }
    else if (root->right->key < key) // Zag-Zag (Right Right)
    {


```

AdChoices 

► [Computer Geeks](#)

► [Tree Trees](#)

► [Tree Structure](#)

AdChoices 

► [Tree View](#)

► [Tree Root](#)

► [In Memory Tree](#)

```

        // Bring the key as root of right-right and do first rotation
        root->right->right = splay(root->right->right, key);
        root = leftRotate(root);
    }

    // Do second rotation for root
    return (root->right == NULL)? root: leftRotate(root);
}

// The search function for Splay tree. Note that this function
// returns the new root of Splay Tree. If key is present in tree
// then, it is moved to root.
struct node *search(struct node *root, int key)
{
    return splay(root, key);
}

// A utility function to print preorder traversal of the tree.
// The function also prints height of every node
void preOrder(struct node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

/* Driver program to test above function*/
int main()
{
    struct node *root = newNode(100);
    root->left = newNode(50);
    root->right = newNode(200);
    root->left->left = newNode(40);
    root->left->left->left = newNode(30);
    root->left->left->left->left = newNode(20);

    root = search(root, 20);
    printf("Preorder traversal of the modified Splay tree is \n");
    preOrder(root);
    return 0;
}

```

Output:

Preorder traversal of the modified Splay tree is

20 50 30 40 100 200

Summary

- 1) Splay trees have excellent locality properties. Frequently accessed items are easy to find. Infrequent items are out of way.
- 2) All splay tree operations take $O(\log n)$ time on average. Splay trees can be rigorously shown to run in $O(\log n)$ average time per operation, over any sequence of operations (assuming we start from an empty tree)
- 3) Splay trees are simpler compared to **AVL** and Red-Black Trees as no extra field is required in every tree node.
- 4) Unlike **AVL tree**, a splay tree can change even with read-only operations like search.

Applications of Splay Trees

Splay trees have become the most widely used basic data structure invented in the last 30 years, because they're the fastest type of balanced search tree for many applications.

Splay trees are used in Windows NT (in the virtual memory, networking, and file system code), the gcc compiler and GNU C++ library, the sed string editor, Fore Systems network routers, the most popular implementation of Unix malloc, Linux loadable kernel modules, and in much other software (Source: <http://www.cs.berkeley.edu/~jrs/61b/lec/36>)

We will soon be discussing insert and delete operations on splay trees.

References:

<http://www.cs.berkeley.edu/~jrs/61b/lec/36>

<http://www.cs.cornell.edu/courses/cs3110/2009fa/recitations/rec-splay.html>

<http://courses.cs.washington.edu/courses/cse326/01au/lectures/SplayTrees.ppt>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Explore New Frontiers in Data.

HPCC Systems is Big Data Processing and Analytics
Open Source. Proven. Trusted.

 LexisNexis® [Learn More](#) 

Related Topics:

- [Print a Binary Tree in Vertical Order | Set 2 \(HashMap based Method\)](#)
- [Print Right View of a Binary Tree](#)
- [Red-Black Tree | Set 3 \(Delete\)](#)
- [Construct a tree from Inorder and Level order traversals](#)
- [Print all nodes at distance k from a given node](#)
- [Print a Binary Tree in Vertical Order | Set 1](#)
- [Interval Tree](#)
- [Check if a given Binary Tree is height balanced like a Red-Black Tree](#)



13

 Tweet

3



1

Writing code in comment? Please use [ideone.com](#) and share the link here.

9 Comments

GeeksforGeeks

Sort by Newest ▼





can the algorithm...



Siva Krishna • 4 months ago

I think here only Zig-Zag is mentioned and Zag-Zig is treated as a another case Typo?

^ | v • Reply • Share ›



GeeksforGeeks → Siva Krishna • 4 months ago

Siva Krishna & G4GFan,

Thanks for your inputs. We have updated the name from Zig-Zag to Za

^ | v • Reply • Share ›



G4GFan → Siva Krishna • 4 months ago

Yes it is correct. You can verify the same in the link mentioned in below

^ | v • Reply • Share ›



Siva Krishna → G4GFan • 4 months ago

I didn't get you. Are they both same or not(i.e can be called with

^ | v • Reply • Share ›



G4GFan → Siva Krishna • 4 months ago

Nope both are diff. zig->R and zag->L,
hence zig-zig->RR zag-zag->LL zig-zag->RL zag-zig->

^ | v • Reply • Share ›



Kartik → Siva Krishna • 4 months ago

Siva, as far as I know they are called same. You can co

^ | v • Reply • Share ›



G4GFan • 4 months ago

For a quick view even <http://lcm.csa.iisc.ernet.in/d...> is good



^ | v • Reply • Share ›



Siva Krishna • 4 months ago

nice one. I think there is a mistake in the Zag example figure. It is correct in the properly, please make a change to it.

1 ^ | v • Reply • Share ›



Kartik ➔ Siva Krishna • 4 months ago

Thanks for pointing this out. We have corrected the figure.

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team