

Splay Tree | Set 2 (Insert)

It is recommended to refer following post as prerequisite of this post.

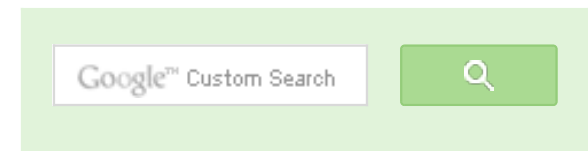
Splay Tree | Set 1 (Search)

As discussed in the [previous post](#), Splay tree is a self-balancing data structure where the last accessed key is always at root. The insert operation is similar to Binary Search Tree insert with additional steps to make sure that the newly inserted key becomes the new root.

Following are different cases to insert a key k in splay tree.

- 1) Root is NULL: We simply allocate a new node and return it as root.
- 2) **Splay** the given key k. If k is already present, then it becomes the new root. If not present, then last accessed leaf node becomes the new root.
- 3) If new root's key is same as k, don't do anything as k is already present.
- 4) Else allocate memory for new node and compare root's key with k.
 -**4.a)** If k is smaller than root's key, make root as right child of new node, copy left child of root as left child of new node and make left child of root as NULL.
 -**4.b)** If k is greater than root's key, make root as left child of new node, copy right child of root as right child of new node and make right child of root as NULL.
- 5) Return new node as new root of tree.

Example:



GeeksforGeeks



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



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

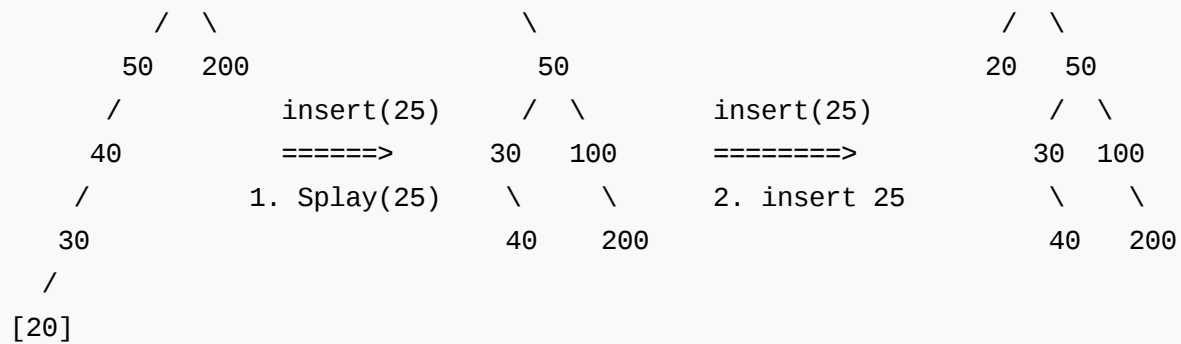
[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

[Geometric Algorithms](#)



```
// This code is adopted from http://algs4.cs.princeton.edu/33balanced/
#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)
{

```



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



Integrated Desktop & Mobile Device Management



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

Download 

ManageEngine
Desktop Central



```

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)
        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 aft
            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;
    }
}

```

```
// Zig-Zag (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)
{
    // 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);
}
```

```
// Function to insert a new key k in splay tree with given root
struct node *insert(struct node *root, int k)
{
    // Simple Case: If tree is empty
    if (root == NULL) return newNode(k);

    // Bring the closest leaf node to root
    root = splay(root, k);

    // If key is already present, then return
    if (root->key == k) return root;

    // Otherwise allocate memory for new node
    struct node *newnode = newNode(k);

    // If root's key is greater, make root as right child
    // of newnode and copy the left child of root to newnode
    if (root->key > k)
    {
        newnode->right = root;
        newnode->left = root->left;
        root->left = NULL;
    }
}
```

Recent Comments

affiszerv 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](#)

[▶ Java Tree](#)

[▶ Tree Diagram](#)

```

// If root's key is smaller, make root as left child
// of newnode and copy the right child of root to newnode
else
{
    newnode->left = root;
    newnode->right = root->right;
    root->right = NULL;
}

return newnode; // newnode becomes new root
}

```

```

// 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 = insert(root, 25);
    printf("Preorder traversal of the modified Splay tree is \n");
    preOrder(root);
    return 0;
}

```

Output:

Preorder traversal of the modified Splay tree is

25 20 50 30 40 100 200

This article is compiled by **Abhay Rathi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

AdChoices 

[► Insert Java](#)

[► Red Black Tree](#)

[► Tree Balancing](#)

AdChoices 

[► XML Tree Viewer](#)

[► Remove Tree Root](#)

[► Tree Structure](#)



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](#)



11



Tweet

4



1

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

Sort by Newest ▼



Join the discussion...

**lizhe** • 2 months ago

Thank you so much. This code is really easy to understand.
btw: Can you explain how to remove a key?

^ | ▼ • Reply • Share ›

**h** • 3 months ago

how is the tree you have shown in figure height balanced? I dont think it is height wrong.

^ | ▼ • Reply • Share ›

**Kartik** ➔ **h** • 2 months ago

Please note that the splay trees give average performance as Logn, than certain operations.

^ | ▼ • Reply • Share ›



Subscribe



Add Disqus to your site