Splay Tree | Set 2 (Insert)

Difficulty Level : **Medium** • Last Updated : 11 Aug, 2021

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:

```
          100                  [20]                             25
          /  \                   \                             /  \
        50    200                 50                          20    50
       /            insert(25)   /  \        insert(25)            /  \
      40            ======>    30    100     ========>           30   100
```

```
#include <bits/stdc++.h>
using namespace std;

// An AVL tree node
class node
{
    public:
    int key;
    node *left, *right;
};

/* Helper function that allocates
a new node with the given key and
    NULL left and right pointers. */
node* newNode(int key)
{
    node* Node = new 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.
node *rightRotate(node *x)
{
    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.
node *leftRotate(node *x)
{
    node *y = x->right;
    x->right = y->left;
```

```
// If key is not present, then it
// brings the last accessed item at
// root. This function modifies the
// tree and returns the new root
node *splay(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 after else
            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;
```

```
                // 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
    node *insert(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
        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;
        }

        // If root's key is smaller, make
        // root as left child of newnode
        // and copy the right child of root to newnode
        else
        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(node *root)
    {
        if (root != NULL)
        {
```

```
/* Driver code*/
int main()
{
    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);
    cout<<"Preorder traversal of the modified Splay tree is \n";
    preOrder(root);
    return 0;
}

// This code is contributed by rathbhupendra
```

C

```
// This code is adopted from http://algs4.cs.princeton.edu/33balanced/SplayBST.ja
#include<stdio.h>
#include<stdlib.h>

// An AVL tree node
truct node

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

```
    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)
        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 else
            // 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;
```

```
        if (root->key > k)
        {
            newnode->right = root;
            newnode->left = root->left;
            root->left = NULL;
        }

        // 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;
    }
```

```
class GFG{

// An AVL tree node
static class node
{

    int key;
    node left, right;
};

/* Helper function that allocates
a new node with the given key and
    null left and right pointers. */
static node newNode(int key)
{
    node Node = new 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.
static node rightRotate(node x)
{
    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.
static node leftRotate(node x)
{
    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
```

```
        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 after else
                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.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
    static node insert(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
        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;
        }

        // 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
    }
```

```
    if (root != null)
    {
        System.out.print(root.key+" ");
        preOrder(root.left);
        preOrder(root.right);
    }
}

/* Driver code*/
public static void main(String[] args)
{
    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);
    System.out.print("Preorder traversal of the modified Splay tree is \n");
    preOrder(root);
}
}


// This code is contributed by Rajput-Ji
```

C#

```
using System;

public class node
{
  public int key;
  public node left, right;
}

public class GFG{

  /* Helper function that allocates
a new node with the given key and
    null left and right pointers. */
  static node newNode(int key)
  {
    node Node = new node();
    Node.key = key;
  // See the diagram given above.
  static node rightRotate(node x)
  {
    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.
  static node leftRotate(node x)
  {
    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
  static node splay(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
        // 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
  static node insert(node root, int k)
  {
    // Simple Case: If tree is empty
    if (root == null) return newNode(k);

    // Bring the closest leaf node to root
```

```
        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;
        }

        // 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
      static void preOrder(node root)
      {
        if (root != null)
        {
          Console.Write(root.key+" ");
          preOrder(root.left);
          preOrder(root.right);
        }
      }

      /* Driver code*/
      static public void Main ()
      {

        node root = newNode(100);
        root.left = newNode(50);
        root.right =  newNode(200);
        root.left.left =  newNode(40);
        root.left.left.left =  newNode(30);
```

```
// This code is contributed by patel2127.
```

Javascript

```
<script>

    // An AVL tree node
    class Node {
        constructor(val) {
            this.key = val;
            this.left = null;
            this.right = null;
        }
    }

    /*
     Helper function that allocates a new
     node with the given key and null left
     and right pointers.
     */
     function newNode(key) {
        var node = new 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.
     function rightRotate( x) {
        var 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.
     function leftRotate( x) {
        var y = x.right;
        x.right = y.left;
        y.left = x;
    // brings the last accessed item at
    // root. This function modifies the
    // tree and returns the new root
     function splay( root , 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 after else
                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);
                // 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
     function insert( root , 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
        var 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;
        }

        // 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
        if (root != null) {
            document.write(root.key + " ");
            preOrder(root.left);
            preOrder(root.right);
        }
    }

    /* Driver code */

        var 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);

        document.write(
        "Preorder traversal of the modified Splay tree is <br/>"
        );
        preOrder(root);

// This code contributed by umadevi9616

</script>
```

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

Like

Next >

Splay Tree | Set 1 (Search)

RECOMMENDED ARTICLES Page : 1 2 3

01 **Splay Tree | Set 3 (Delete)**
13, Jun 17

02 **Splay Tree | Set 1 (Search)**
14, Jan 14

03 **Red-Black Tree | Set 2 (Insert)**
16, Feb 14

04 **K Dimensional Tree | Set 1 (Search and Insert)**
31, Oct 14

05 **Insert Operation in B-Tree**
23, Apr 13

06 **B-Tree Insert without aggressive splitting**
03, May 19

07 **How to insert Strings into an AVL Tree**
11, Oct 21

08 **Convert a Generic Tree(N-array Tree) to Binary Tree**
29, Oct 20

Article Contributed By :

GeeksforGeeks

Improved By : rathbhupendra, Akanksha_Rai, Rajput-Ji, umadevi9616, patel2127

Article Tags : Self-Balancing-BST, splay-tree, Advanced Data Structure

Report Issue

Start Your Coding Journey Now! Login Register

Business
Finance
Lifestyle

C#
SQL

Web Development

Web Tutorials
Django Tutorial
HTML
CSS
JavaScript
Bootstrap

Contribute

Write an Article
Improve an Article
Pick Topics to Write
Write Interview Experience
Internships
Video Internship

▲