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.

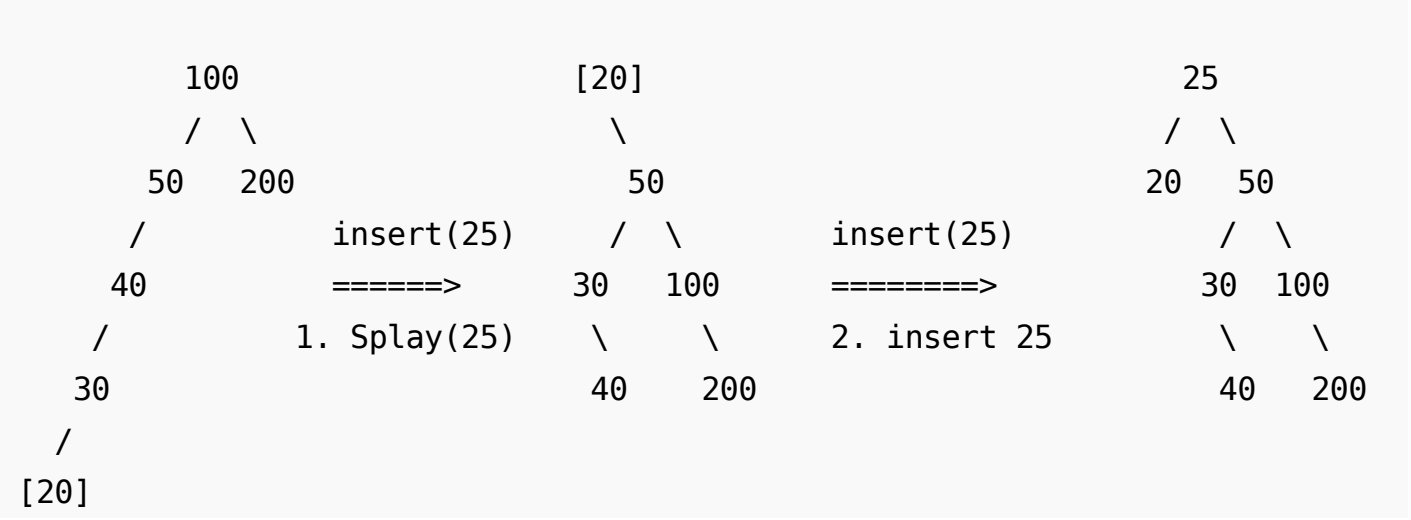
2) Splay the given key k. If k is already present, then it becomes the new root. If not present, then last accessed leaf node becomes the new root.

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:




```
// This code is adopted from http://algs4.cs.princeton.e
#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 gi
    NULL left and right pointers. */
struct node* newNode(int key)
{
    struct node* node = (struct node*)malloc(sizeof(stru
    node->key   = key;
    node->left  = node->right  = NULL;
    return (node);
}

// A utility function to right rotate subtree rooted wit
// 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
// 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 presen
// If key is not present, then it brings the last access
// root.  This function modifies the tree and returns th
struct node *splay(struct node *root, int key)
{
    // Base cases: root is NULL or key is present at roo
    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 o
            root->left->left = splay(root->left->left, k

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

            // 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(r
    }
    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,

            // Do first rotation for root->right
            if (root->right->left != NULL)
                root->right = rightRotate(root->right);
        }
        else if (root->right->key < key)// Zag-Zag (Righ
        {
            // Bring the key as root of right-right and
            root->right->right = splay(root->right->righ
            root = leftRotate(root);
        }

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

// Function to insert a new key k in splay tree with giv
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 chil
    // of newnode and copy the left child of root to new
    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 ne
    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
// 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);
    }
}

/* Drier 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 tre
    preOrder(root);
    return 0;
}
```

Output:

```
Preorder traversal of the modified Splay tree is
25 20 50 30 40 100 200
```