Splay Trees

AJ Cronin, Brooke Stetson, Daniel Reynaldo, Nathan Osborne

What Does a Splay Tree Do?

- A Splay Tree has the following qualities:
  - Mostly self-balancing
  - The most recently touched node is the **root**
  - The tree is in **binary search tree** order

- The general idea is that recently searched/inserted items are brought to the root

- Utilizes Splaying(rotations) to bring the inserted/searched node to the root of the tree

- Splay Trees are space efficient, as they do not require extra variables for the Nodes or Tree.

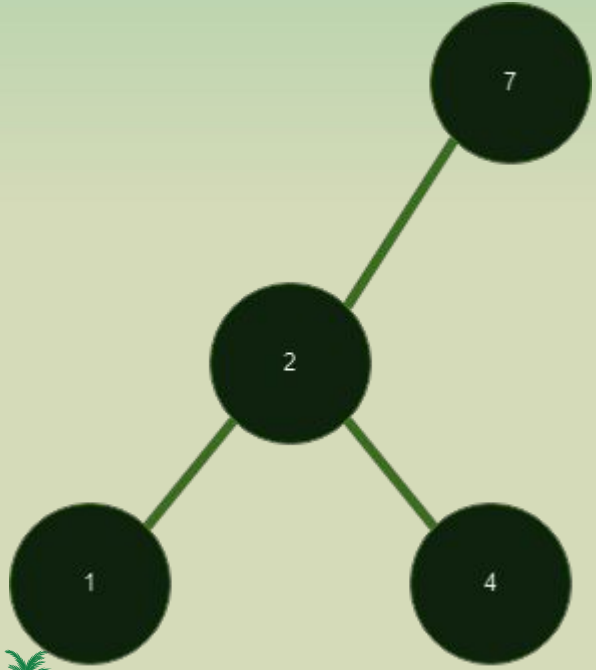
- However, they do not ensure balancing on every Splay, this can lead to Worst Case runtimes.

Splay Method

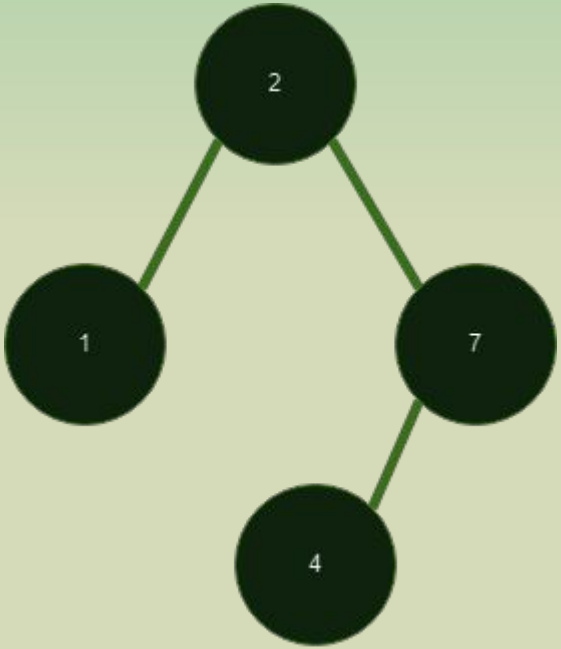
- Splay starts at the root of the tree and recursively looks for the desired value since it is in BST form.
- Once the element is found, we do a series of rotations to bring the splayed value to the root of the tree.
- There are 6 Different Main Types of Rotation:
  - Zig Rotation
  - Zag Rotation
  - Zig-Zig Rotation
  - Zag-Zag Rotation
  - Zig-Zag Rotation
  - Zag-Zag Rotation
- Splaying a value to become the new root of the tree allows for easy access when searching and removing because the desired element is already at the top of the tree.

Rotations

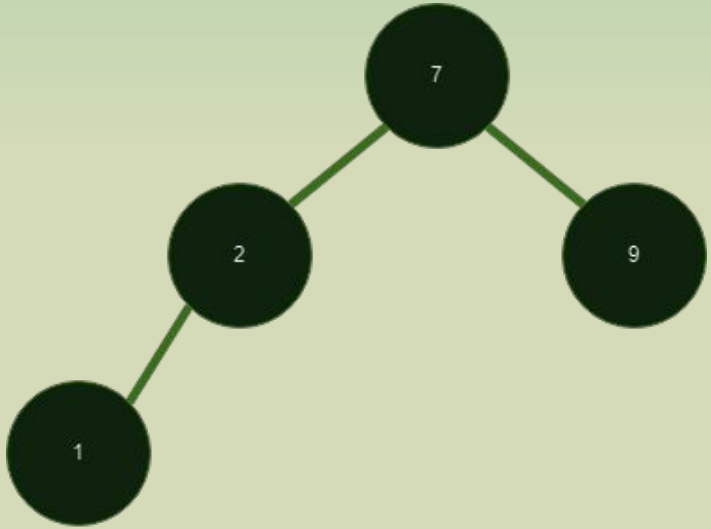
Zig (Right) Rotation



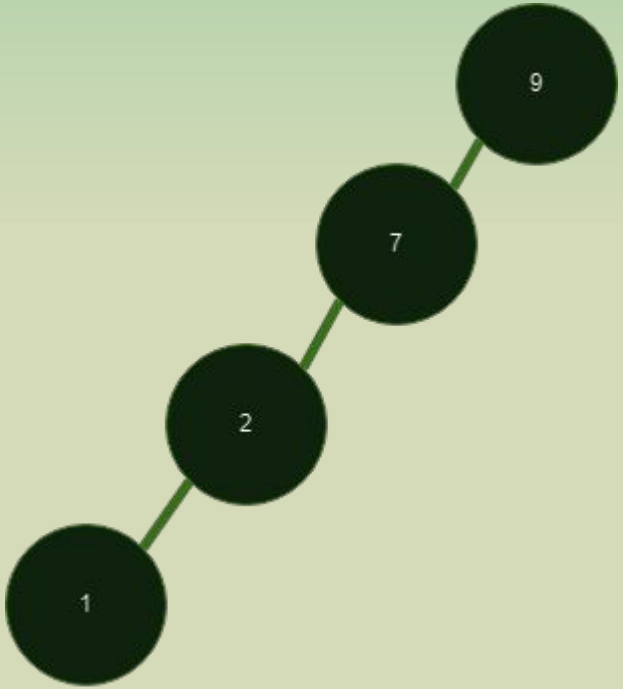
Splay(root, 2) ->



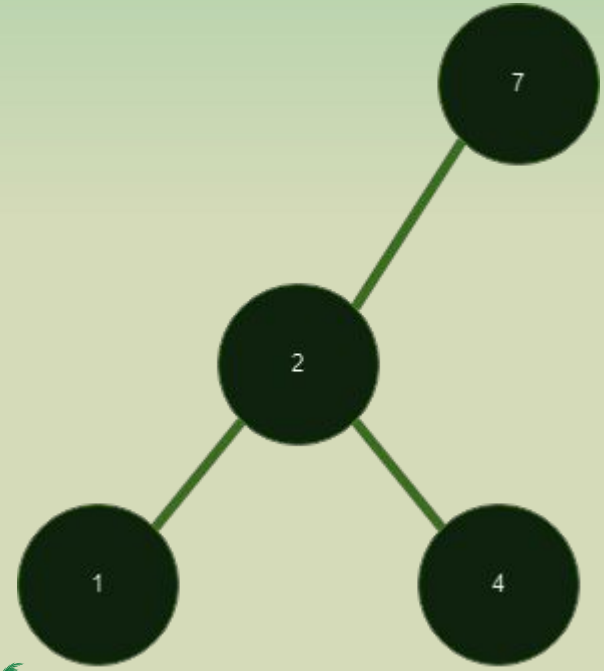
Zag (Left) Rotation



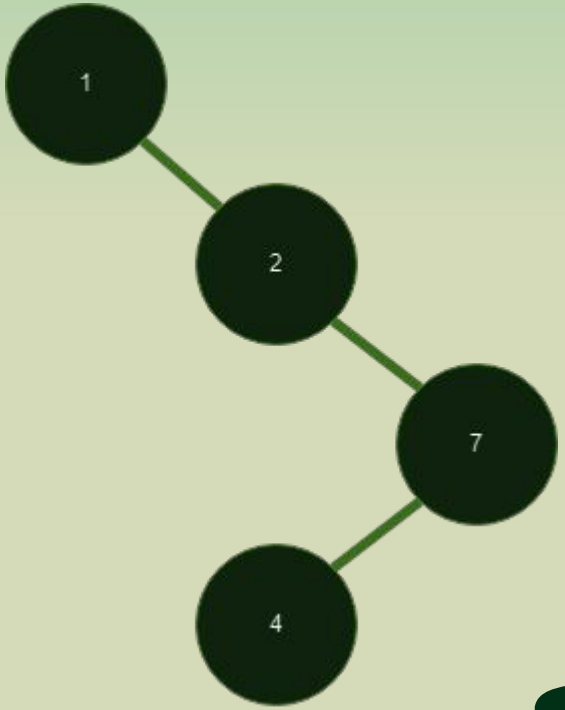
Splay(root, 9) ->



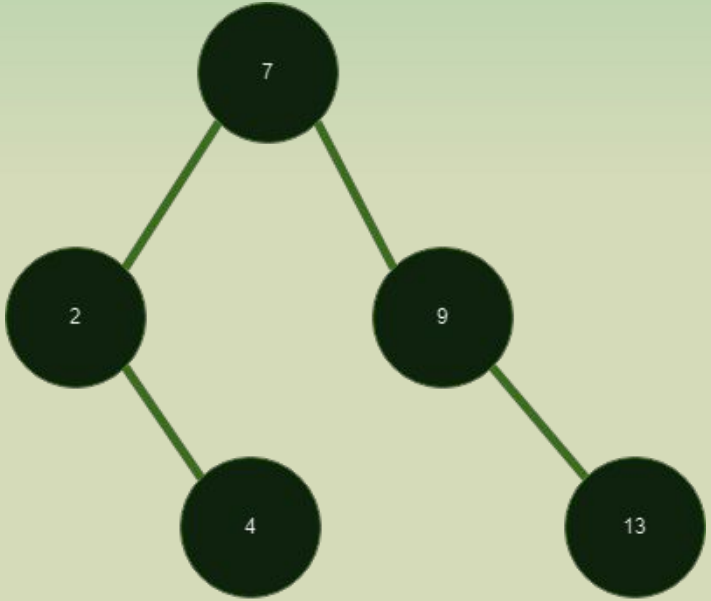
Zig-Zig (Double Right) Rotation



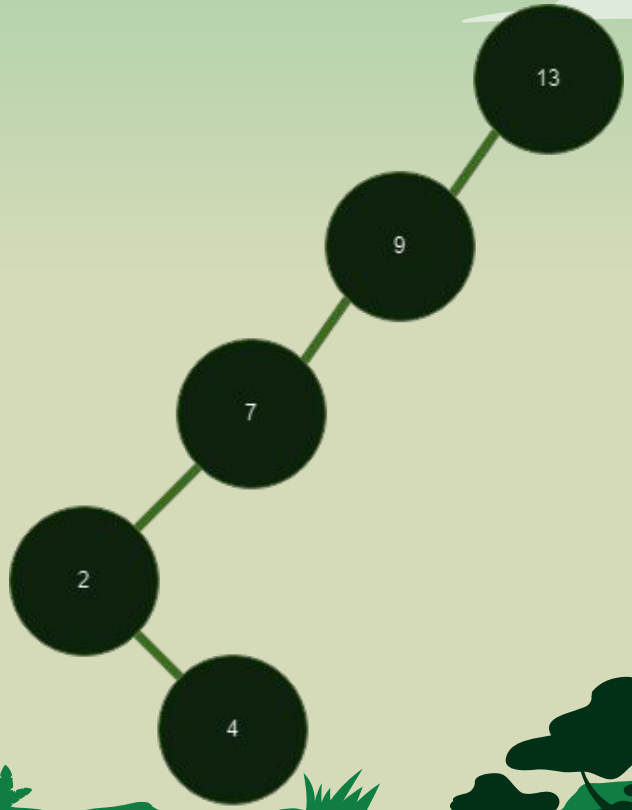
Splay(root, 1) ->



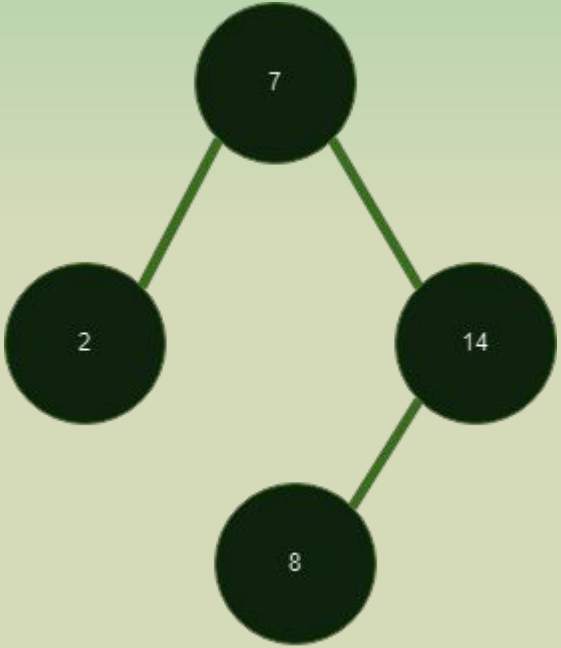
Zag-Zag (Double Left) Rotation



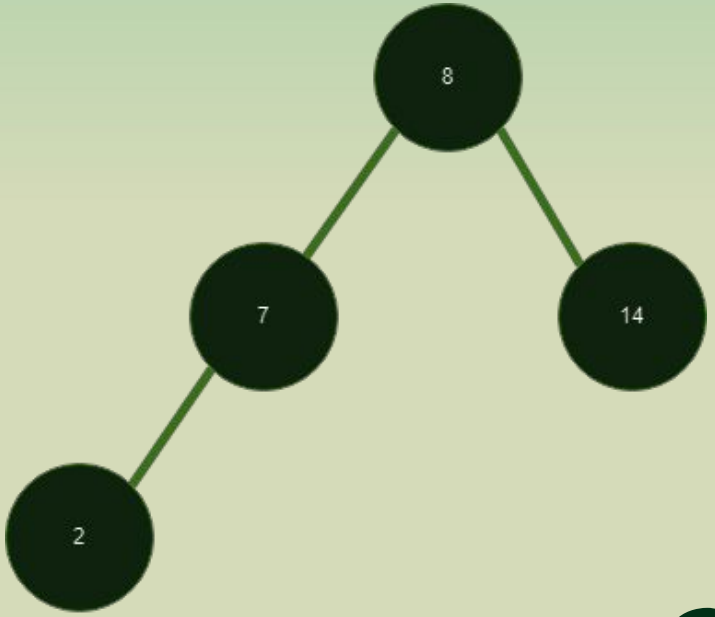
Splay(root, 13) ->



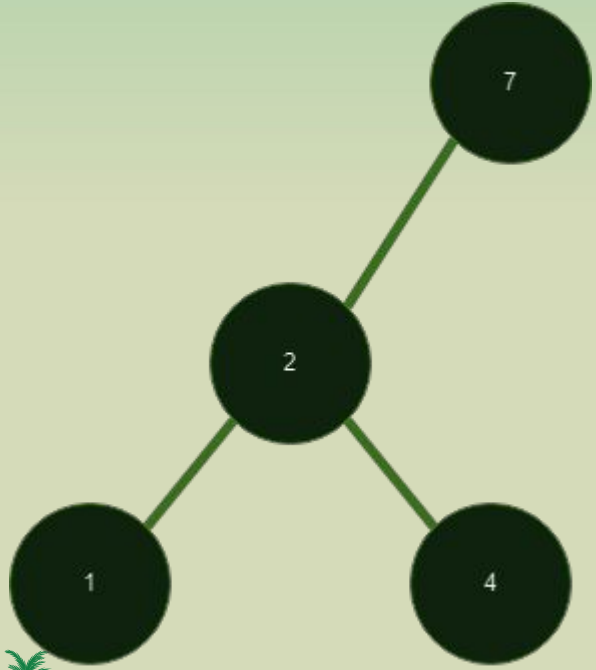
Zig-Zag (Right-Left) Rotation



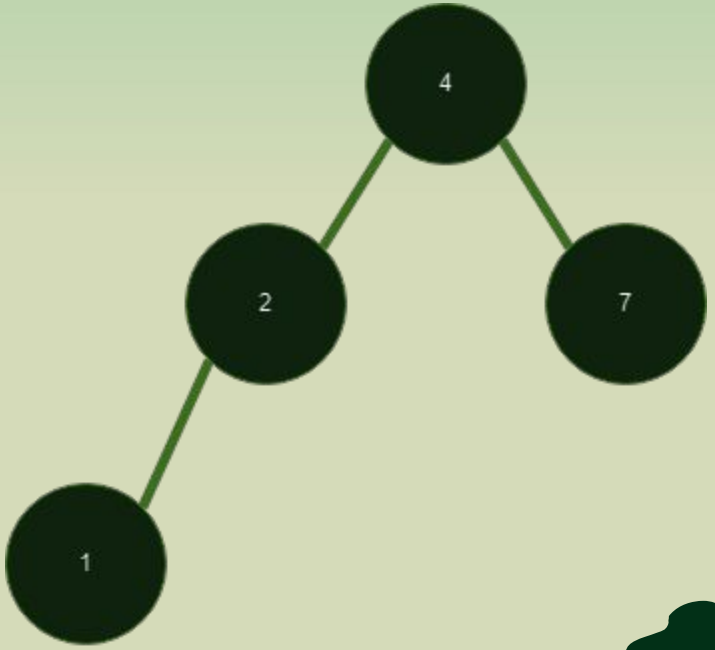
Splay(root, 8) ->



Zag-Zig (Left-Right) Rotation



Splay(root, 4) ->



Inserting

- Follows Binary Search Tree insertion
    - Branch left if value to be inserted is less than the current node
    - Branch right if the value is greater than the current node
- Once inserted, Splay to the root of the tree

Insert Runtime

- O(N) worst case runtime- all the nodes are to one side of the tree
- O(1) best case runtime- the tree was previously empty
- O(log N) average runtime- assuming a mostly-balanced tree in which the height of the tree is approximately log N

Example (Test Code): **insert 5**, insert 3, insert 10, insert 8, insert 2, insert 4, insert 7

```
}

public static void testInsertRoot4() {
      System.out.println("Test Insert 4\n------------");
      SplayTree tree = new SplayTree();


      // Test Insertion of Nodes. Print on each iteration to see that rotations are correct, and root
      // is the last element inserted
      testInsert(tree, 5);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 3);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 10);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 8);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 2);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 4);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 7);
      tree.printTree();
      tree.printInOrder();
```

OUTPUT

```
Insert: 5,    Root is now 5
-------
SUCCESS! Inserted value becomes root

 - PRINT TREE -
5|
   NULL
   NULL

InOrder: 5
```

ROOT —->

Right Child ->

Left Child ->

In Order traversal:

Example (Test Code): insert 5, **insert 3**, insert 10, insert 8, insert 2, insert 4, insert 7

```
}

public static void testInsertRoot4() {
      System.out.println("Test Insert 4\n------------");
      SplayTree tree = new SplayTree();


      // Test Insertion of Nodes. Print on each iteration to see that rotations are correct, and root
      // is the last element inserted
      testInsert(tree, 5);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 3);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 10);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 8);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 2);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 4);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 7);
      tree.printTree();
      tree.printInOrder();
```

OUTPUT

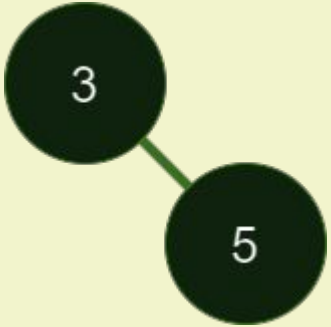
```
Insert: 3,    Root is now 3
-------
SUCCESS! Inserted value becomes root

 - PRINT TREE -
3|
   5|
      NULL
      NULL
   NULL

InOrder: 3 5
```

ROOT —->

Right Child ->

Left Child ->

In Order traversal:



Example (Test Code): insert 5, insert 3, **insert 10**, insert 8, insert 2, insert 4, insert 7

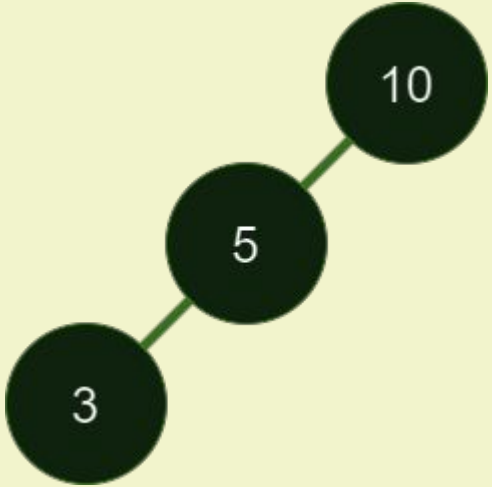
```java
}

public static void testInsertRoot4() {
        System.out.println("Test Insert 4\n------------");
        SplayTree tree = new SplayTree();


        // Test Insertion of Nodes. Print on each iteration to see that rotations are correct, and root
        // is the last element inserted
        testInsert(tree, 5);
        tree.printTree();
        tree.printInOrder();

        testInsert(tree, 3);
        tree.printTree();
        tree.printInOrder();

        testInsert(tree, 10);
        tree.printTree();
        tree.printInOrder();

        testInsert(tree, 8);
        tree.printTree();
        tree.printInOrder();

        testInsert(tree, 2);
        tree.printTree();
        tree.printInOrder();

        testInsert(tree, 4);
        tree.printTree();
        tree.printInOrder();

        testInsert(tree, 7);
        tree.printTree();
        tree.printInOrder();
```

OUTPUT

```
Insert: 10,    Root is now 10
-------
SUCCESS! Inserted value becomes root

 - PRINT TREE -
10|
   NULL
   5|
      NULL
      3|
         NULL
         NULL

InOrder: 3 5 10
```

ROOT —-> 10|
Right Child -> NULL
Left Child -> 5|

In Order traversal: InOrder: 3 5 10



Example (Test Code): insert 5, insert 3, insert 10, **insert 8**, insert 2, insert 4, insert 7

```
}

public static void testInsertRoot4() {
      System.out.println("Test Insert 4\n------------");
      SplayTree tree = new SplayTree();


      // Test Insertion of Nodes. Print on each iteration to see that rotations are correct, and root
      // is the last element inserted
      testInsert(tree, 5);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 3);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 10);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 8);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 2);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 4);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 7);
      tree.printTree();
      tree.printInOrder();
```

OUTPUT

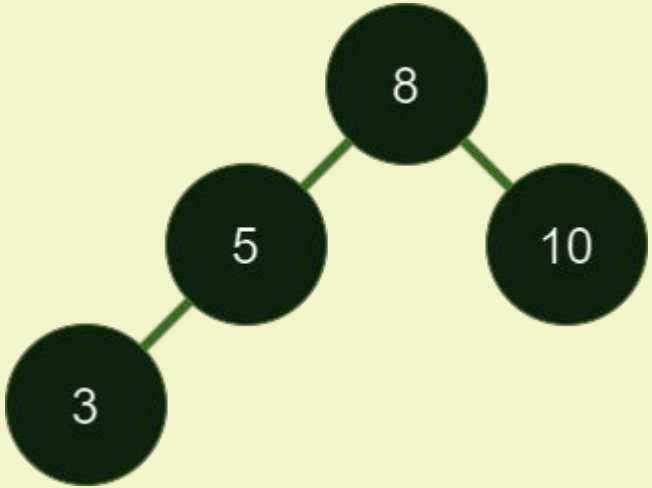
```
Insert: 8,     Root is now 8
-------
SUCCESS! Inserted value becomes root

 - PRINT TREE -
8|
   10|
      NULL
      NULL
   5|
      NULL
      3|
         NULL
         NULL

InOrder: 3 5 8 10
```

ROOT —-> 8|

Right Child -> 10|

Left Child -> 5|

In Order traversal:



Example (Test Code): insert 5, insert 3, insert 10, insert 8, **insert 2**, insert 4, insert 7

```
}

public static void testInsertRoot4() {
      System.out.println("Test Insert 4\n------------");
      SsplayTree tree = new SplayTree();


      // Test Insertion of Nodes. Print on each iteration to see that rotations are correct, and root
      // is the last element inserted
      testInsert(tree, 5);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 3);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 10);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 8);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 2);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 4);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 7);
      tree.printTree();
      tree.printInOrder();
```

OUTPUT

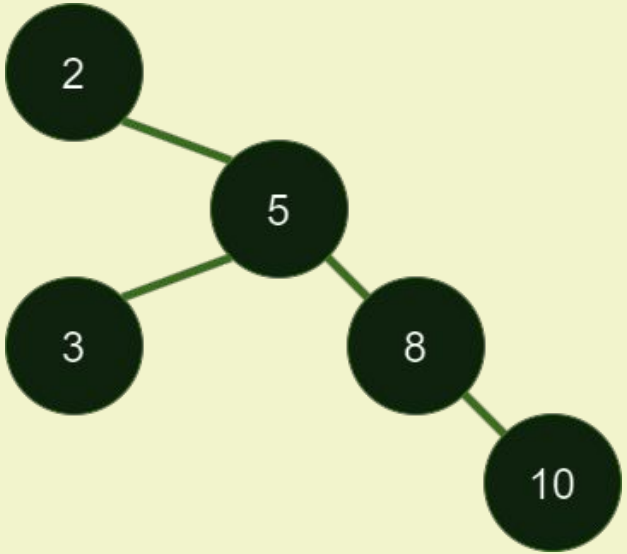
```
Insert: 2,    Root is now 2
-------
SUCCESS! Inserted value becomes root

 - PRINT TREE -
2|
   5|
      8|
         10|
            NULL
            NULL
         NULL
      3|
         NULL
         NULL
   NULL

InOrder: 2 3 5 8 10
```

ROOT --->

Right Child ->

Left Child ->

In Order traversal:



Example (Test Code): insert 5, insert 3, insert 10, insert 8, insert 2, **insert 4**, insert 7

```
}

public static void testInsertRoot4() {
      System.out.println("Test Insert 4\n------------");
      SplayTree tree = new SplayTree();


      // Test Insertion of Nodes. Print on each iteration to see that rotations are correct, and root
      // is the last element inserted
      testInsert(tree, 5);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 3);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 10);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 8);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 2);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 4);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 7);
      tree.printTree();
      tree.printInOrder();
```

OUTPUT

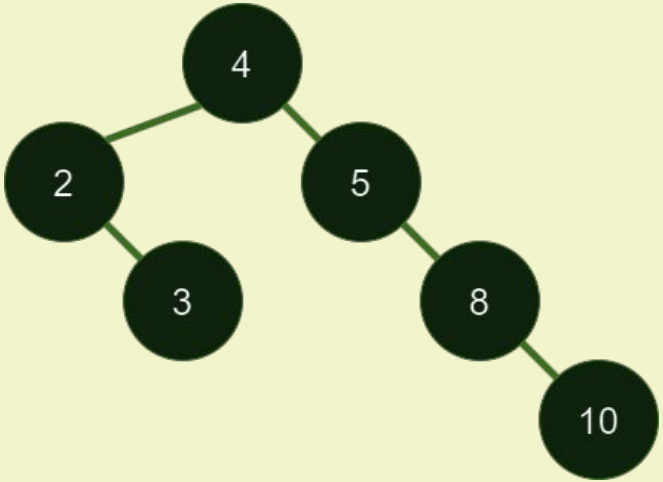
```
Insert: 4,    Root is now 4
-------
SUCCESS! Inserted value becomes root

 - PRINT TREE -
4|
   5|
      8|
         10|
            NULL
            NULL
         NULL
      NULL
   2|
      3|
         NULL
         NULL
      NULL
```

ROOT —->

Right Child ->

Left Child ->

In Order traversal: `InOrder: 2 3 4 5 8 10`

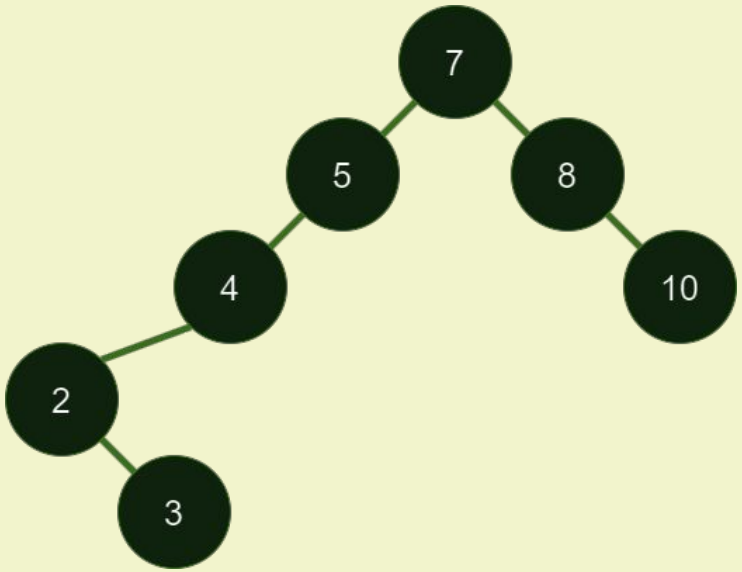


Example (Test Code): insert 5, insert 3, insert 10, insert 8, insert 2, insert 4, **insert 7**

```
}

public static void testInsertRoot4() {
      System.out.println("Test Insert 4\n------------");
      SplayTree tree = new SplayTree();


      // Test Insertion of Nodes. Print on each iteration to see that rotations are correct, and root
      // is the last element inserted
      testInsert(tree, 5);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 3);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 10);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 8);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 2);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 4);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 7);
      tree.printTree();
      tree.printInOrder();
```

OUTPUT

```
Insert: 7,    Root is now 7
-------
 - PRINT TREE -
ROOT -->7|
    8|
       10|
Right Child ->
          NULL
          NULL
       NULL
Left Child  -> 5|
       NULL
       4|
          NULL
          2|
             3|
                NULL
                NULL
             NULL

In Order traversal: 2 3 4 5 7 8 10
```



Searching

- The searching function of a splay tree locates a value and brings it's node to the root
- This function works by calling splay to get the value at the root
    - The splay function does not change the tree if the value is not in it
- Because of this, if the root is not the value then the value is not in the tree

Search Runtime

- O(N) worst case runtime - entire tree must be traversed to find the value
- O(1) best case runtime - value is at the root
- O(log N) average runtime - assuming a mostly-balanced tree in which the height of the tree is approximately log N

Example (Test Code): Initial Tree

```java
public static void testSearch1() {

    System.out.println("Test Search1\n--------------------
    SplayTree tree = new SplayTree();

    // Test Search of Nodes. Print on each iteration to se
    // correct, and root
    // is the last element inserted
    testInsert(tree, 0);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 1);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 2);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 3);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 4);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 6);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 7);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 8);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 9);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 10);
    tree.printTree();
    tree.printInOrder();

    testInsert(tree, 5);
    tree.printTree();
    tree.printInOrder();
```

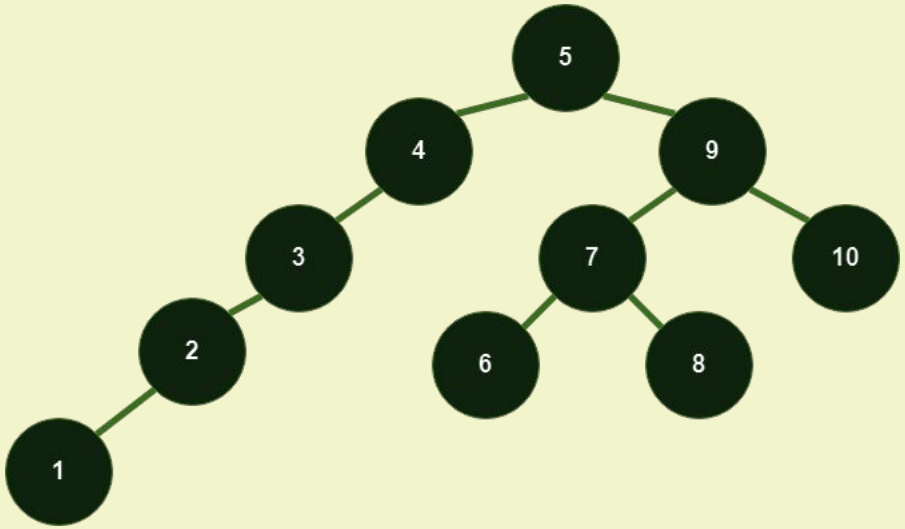
OUTPUT

ROOT —-> Right Child ->

Left Child ->

In Order traversal:

```
- PRINT TREE -
5|
   9|
      10|
         NULL
         NULL
      7|
         8|
            NULL
            NULL
         6|
            NULL
            NULL
   4|
      NULL
      3|
         NULL
         2|
            NULL
            1|
               NULL
               0|
                  NULL
                  NULL

InOrder: 0 1 2 3 4 5 6 7 8 9 10
```



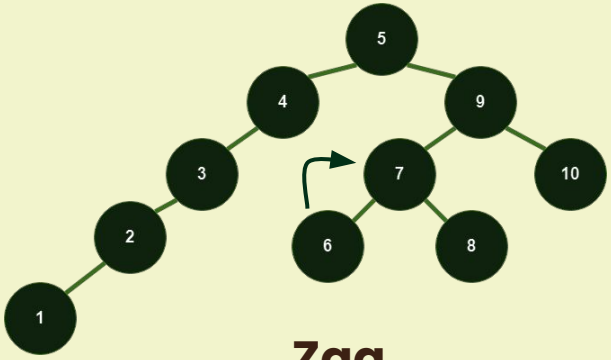
Example (Test Code): search 6

Steps:

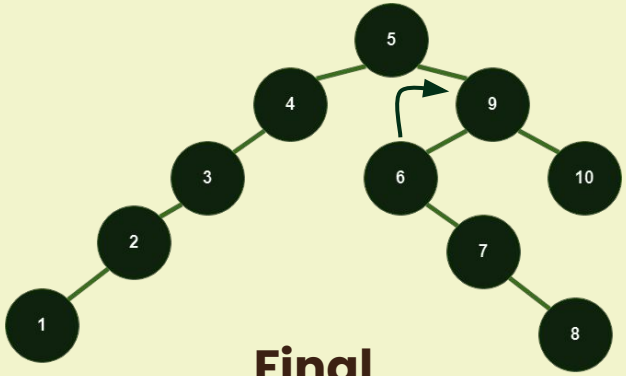
Zig-Zig
Zag

Zig-Zig

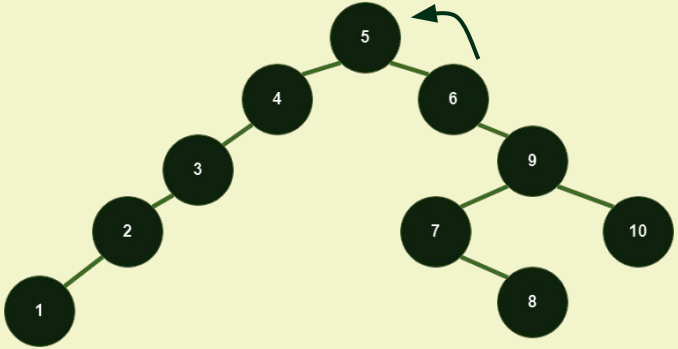
Zig 1



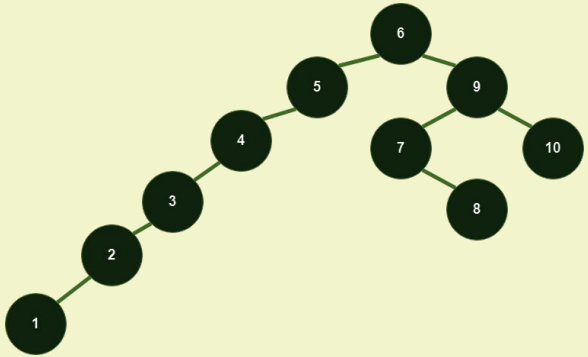
Zig 2



Zag



Final



Example (Test Code): search 6

```
testSearch(tree, 6);
tree.printTree();
tree.printInOrder();
```

OUTPUT

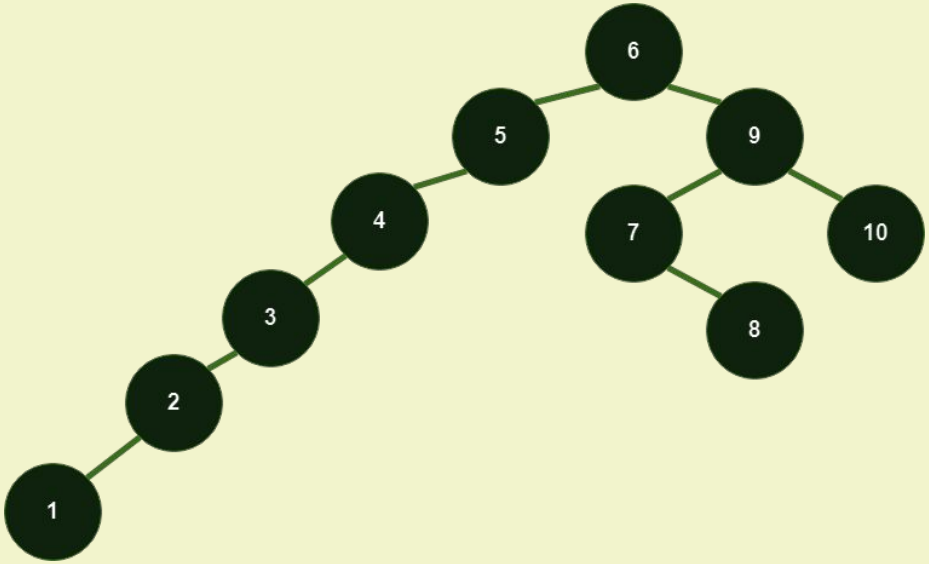
```
Search: 6,    Root is now 6
-------
SUCCESS! Searched value becomes root

 - PRINT TREE -
6|
   9|
      10|
         NULL
         NULL
      7|
         8|
            NULL
            NULL
         NULL
   5|
      NULL
      4|
         NULL
         3|
            NULL
            2|
               NULL
               1|
                  NULL
                  0|
                     NULL
                     NULL
```

ROOT —-> 6|

Right Child -> 9|

Left Child -> 5|

In Order traversal: `InOrder: 0 1 2 3 4 5 6 7 8 9 10`



Removing

- Locate the node to be removed using the Search operation, this will splay the node to the root of the tree
    - If the node has no children remove it from the tree
    - If the node has one child replace the node with its child
    - If the node has both children, find the maximum value in the left subtree of the removed node and splay it to the root, then set it's right child to the initial right child of the removed node.

Remove Runtime

- O(N) worst case runtime- entire tree must be traversed to find the value to be removed
- O(1) best case runtime- node to be removed is the root
- O(log N) average runtime- assuming a mostly-balanced tree in which the height of the tree is approximately log N

Example (Test Code): Initial Tree

```java
public static void testRemove1() {

      System.out.println("Test Remove1\n-----------------------------------");
      SsplayTree tree = new SplayTree();


      testInsert(tree, 8);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 1);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 11);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 7);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 14);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 0);
      tree.printTree();
      tree.printInOrder();

      testInsert(tree, 21);
      tree.printTree();
      tree.printInOrder();
```

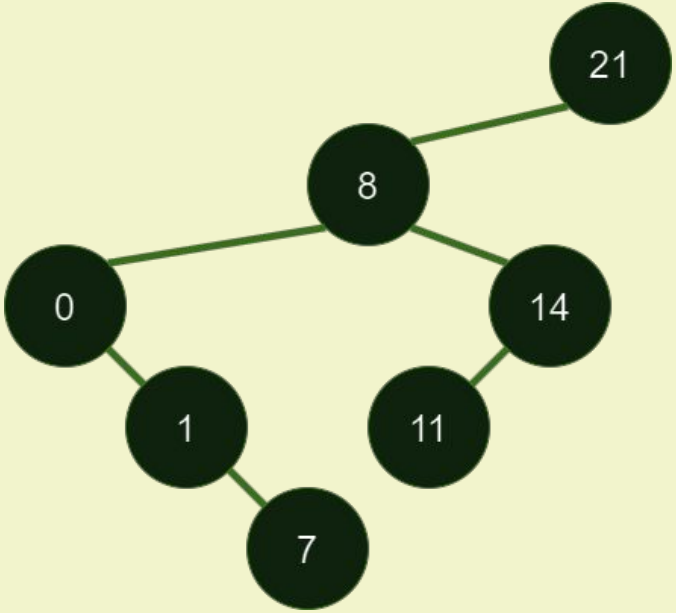
OUTPUT:

ROOT —->
Right Child ->
Left Child ->

```
Insert: 21,    Root is now 21
-------
SUCCESS! Inserted value becomes root

 - PRINT TREE -
21|
   NULL
   8|
      14|
         NULL
         11|
            NULL
            NULL
      0|
         1|
            7|
               NULL
               NULL
            NULL
         NULL
```

In Order traversal:

```
InOrder: 0 1 7 8 11 14 21
```

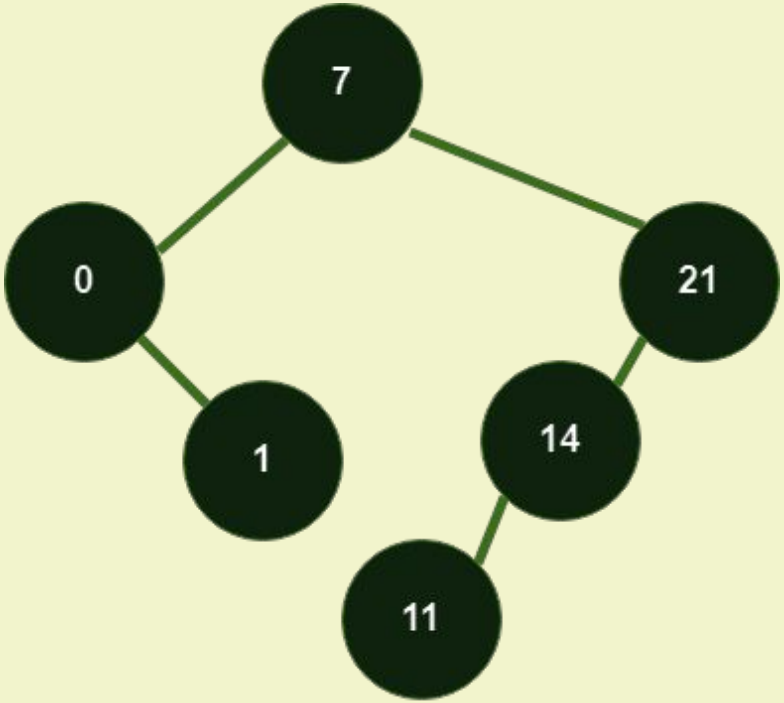


Example (Test Code): Remove 8, Remove 21, Remove 0, Remove 11, Remove 14, Remove 7, Remove 1

```
// Remove until empty
testRemove(tree, 8);
tree.printTree();
tree.printInOrder();

testRemove(tree, 21);
tree.printTree();
tree.printInOrder();

testRemove(tree, 0);
tree.printTree();
tree.printInOrder();

testRemove(tree, 11);
tree.printTree();
tree.printInOrder();

testRemove(tree, 14);
tree.printTree();
tree.printInOrder();

testRemove(tree, 7);
tree.printTree();
tree.printInOrder();

testRemove(tree, 1);
tree.printTree();
tree.printInOrder();
```

OUTPUT:

```
                    Remove: 8,    Root is now 7
                    -------
                     - PRINT TREE -
ROOT  —->           7|
Right Child ->       21|
                        NULL
                        14|
                           NULL
                           11|
                              NULL
                              NULL
Left Child  -> 0|
                        1|
                           NULL
                           NULL
                        NULL

In Order traversal:  0 1 7 11 14 21
                        ↑
                    Removed 8
```



Example (Test Code): Remove 8, **Remove 21**, Remove 0, Remove 11, Remove 14, Remove 7, Remove 1

```
// Remove until empty
testRemove(tree, 8);
tree.printTree();
tree.printInOrder();

testRemove(tree, 21);
tree.printTree();
tree.printInOrder();

testRemove(tree, 0);
tree.printTree();
tree.printInOrder();

testRemove(tree, 11);
tree.printTree();
tree.printInOrder();

testRemove(tree, 14);
tree.printTree();
tree.printInOrder();

testRemove(tree, 7);
tree.printTree();
tree.printInOrder();

testRemove(tree, 1);
tree.printTree();
tree.printInOrder();
```

OUTPUT:

```
Remove: 21,    Root is now 7
-------
 - PRINT TREE -
7|
  14|
     NULL
     11|
        NULL
        NULL
  0|
     1|
        NULL
        NULL
     NULL

0 1 7 11 14
```

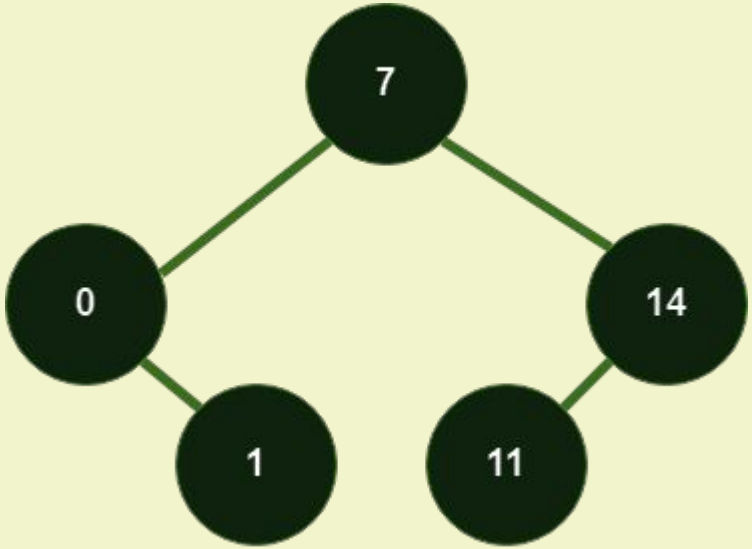
ROOT —-> 7|

Right Child -> 14|

Left Child -> 0|

In Order traversal: 0 1 7 11 14

↑ Removed 21



Example (Test Code): Remove 8, Remove 21, **Remove 0**, Remove 11, Remove 14, Remove 7, Remove 1

```
// Remove until empty
testRemove(tree, 8);
tree.printTree();
tree.printInOrder();

testRemove(tree, 21);
tree.printTree();
tree.printInOrder();

testRemove(tree, 0);
tree.printTree();
tree.printInOrder();

testRemove(tree, 11);
tree.printTree();
tree.printInOrder();

testRemove(tree, 14);
tree.printTree();
tree.printInOrder();

testRemove(tree, 7);
tree.printTree();
tree.printInOrder();

testRemove(tree, 1);
tree.printTree();
tree.printInOrder();
```

OUTPUT:

```
Remove: 0,    Root is now 7
-------
 - PRINT TREE -
7|
   14|
      NULL
      11|
         NULL
         NULL
   1|
      NULL
      NULL

1 7 11 14
```

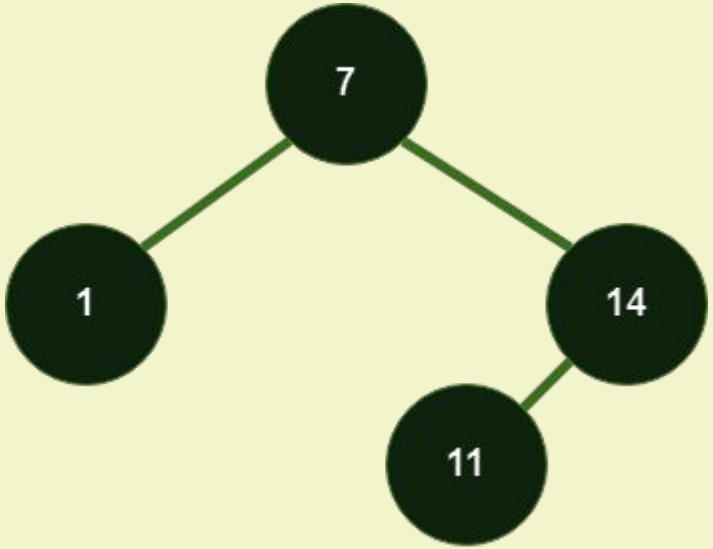
ROOT —-> 7|

Right Child -> 14|

Left Child -> 1|

In Order traversal: 1 7 11 14

Removed 0



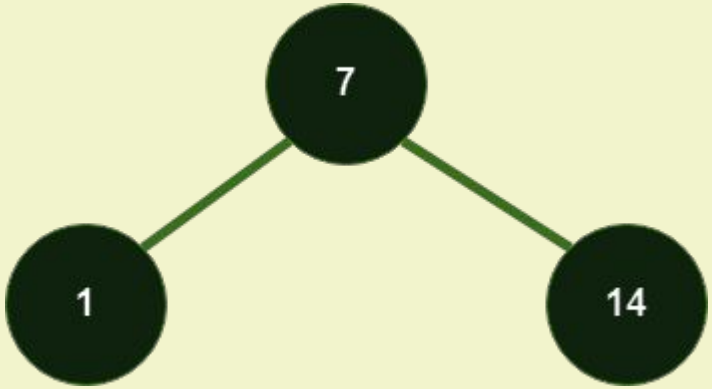
Example (Test Code): Remove 8, Remove 21, Remove 0, **Remove 11**, Remove 14, Remove 7, Remove 1

```
// Remove until empty
testRemove(tree, 8);
tree.printTree();
tree.printInOrder();

testRemove(tree, 21);
tree.printTree();
tree.printInOrder();

testRemove(tree, 0);
tree.printTree();
tree.printInOrder();

testRemove(tree, 11);
tree.printTree();
tree.printInOrder();

testRemove(tree, 14);
tree.printTree();
tree.printInOrder();

testRemove(tree, 7);
tree.printTree();
tree.printInOrder();

testRemove(tree, 1);
tree.printTree();
tree.printInOrder();
```

OUTPUT:

```
Remove: 11,    Root is now 7
-------
 - PRINT TREE -
ROOT —-> 7|
Right Child ->    14|
                     NULL
                     NULL
Left Child  ->    1|
                     NULL
                     NULL

In Order traversal: 1 7 14
```

Removed 11



Example (Test Code): Remove 8, Remove 21, Remove 0, Remove 11, **Remove 14**, Remove 7, Remove 1

```
// Remove until empty
testRemove(tree, 8);
tree.printTree();
tree.printInOrder();

testRemove(tree, 21);
tree.printTree();
tree.printInOrder();

testRemove(tree, 0);
tree.printTree();
tree.printInOrder();

testRemove(tree, 11);
tree.printTree();
tree.printInOrder();

testRemove(tree, 14);
tree.printTree();
tree.printInOrder();

testRemove(tree, 7);
tree.printTree();
tree.printInOrder();

testRemove(tree, 1);
tree.printTree();
tree.printInOrder();
```

OUTPUT:

```
Remove: 14,    Root is now 7
-------
 - PRINT TREE -
7|
   NULL
   1|
      NULL
      NULL

1 7
```

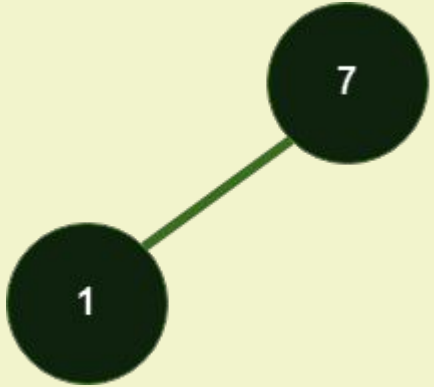
ROOT —-> 7|

Right Child -> NULL

Left Child -> 1|

In Order traversal: 1 7

↑ Removed 14



Example (Test Code): Remove 8, Remove 21, Remove 0, Remove 11, Remove 14, **Remove 7**, Remove 1

```
// Remove until empty
testRemove(tree, 8);
tree.printTree();
tree.printInOrder();

testRemove(tree, 21);
tree.printTree();
tree.printInOrder();

testRemove(tree, 0);
tree.printTree();
tree.printInOrder();

testRemove(tree, 11);
tree.printTree();
tree.printInOrder();

testRemove(tree, 14);
tree.printTree();
tree.printInOrder();

testRemove(tree, 7);
tree.printTree();
tree.printInOrder();

testRemove(tree, 1);
tree.printTree();
tree.printInOrder();
```

OUTPUT:

```
Remove: 7,    Root is now 1
-------
 - PRINT TREE -
1|
   NULL
   NULL

1
```

ROOT —->

Right Child ->

Left Child ->

In Order traversal:

↑

Removed 7

Example (Test Code): Remove 8, Remove 21, Remove 0, Remove 11, Remove 14, Remove 7, **Remove 1**

```
// Remove until empty
testRemove(tree, 8);
tree.printTree();
tree.printInOrder();

testRemove(tree, 21);
tree.printTree();
tree.printInOrder();

testRemove(tree, 0);
tree.printTree();
tree.printInOrder();

testRemove(tree, 11);
tree.printTree();
tree.printInOrder();

testRemove(tree, 14);
tree.printTree();
tree.printInOrder();

testRemove(tree, 7);
tree.printTree();
tree.printInOrder();

testRemove(tree, 1);
tree.printTree();
tree.printInOrder();
```

OUTPUT:

```
Remove: 1,    Root is now NULL
-------
 - PRINT TREE -
NULL
```

ROOT —-> NULL

↑

Removed 1

Thank You!