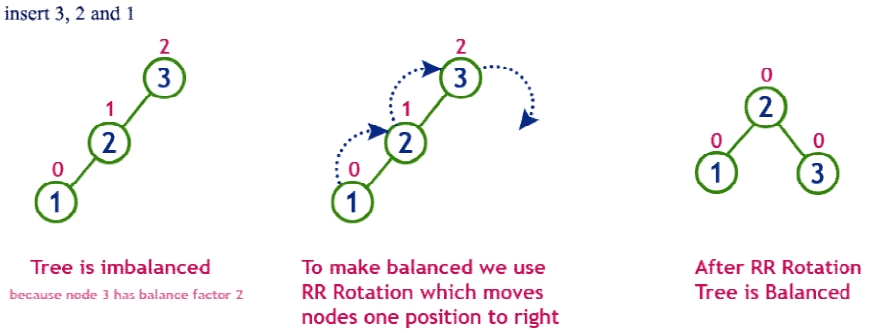
DS LAB Assignment 2

AVL Tree

Roll number :- 214101022

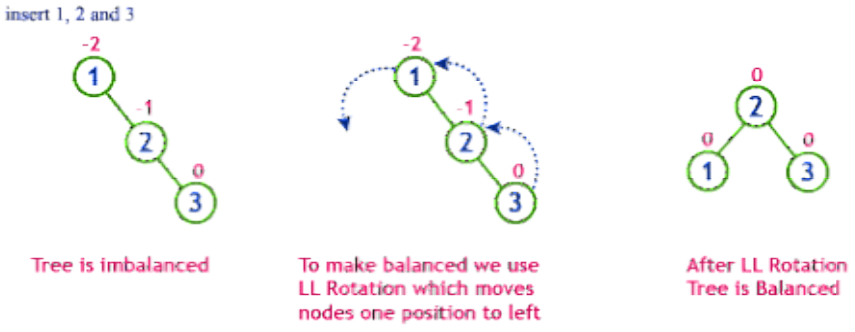
ROTATIONS:-

(1) Rotate right:-
Example,



- In the above graph, bf of node 3 is 2. Bf of 2 is 1.
- Suppose this is a subtree of a tree with root node 3.
- Now the function will create a temp pointer to point at node 2.
- The right child of 2 becomes left child of 3 and 3 becomes right child of 2.
- As the subtree becomes balanced, the bf of root node 2 and node 3 becomes 0.

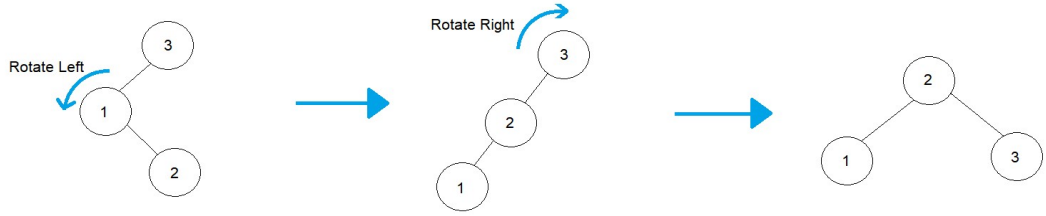
(2) Rotate left :-
Example,



- In the above graph, bf of node 1 is -2. Bf of 2 is -1.
- Suppose this is a subtree of a tree with root node 1.
- Now the function will create a temp pointer to point at node 2.

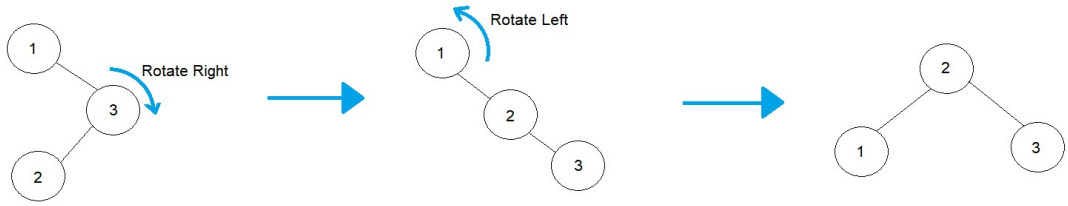
- The left child of 2 becomes right child of 1 and 1 becomes left child of 2.
- As the subtree becomes balanced, the bf of root node 2 and node 1 becomes 0.

(3) Rotate left right:-
Example,



- In this example there are two rotations. First left rotation at node 1 and then right rotation at node 2. We will create two temp pointers pointing at 1 and 2.
- Bf of 3 is 2. Bf of 1 is -1. (Condition for left right rotation).
- In left rotation, the left subtree of 2 becomes right subtree of 1 and 1 becomes left child of 2.
- If bf of 2 is less than or equal to zero then bf of 1 is 0 else bf of 1 is -1.(After left rotation)
- In right rotation, the right subtree of 2 becomes left subtree of 3 and 3 becomes right child of 2.
- If bf of 2 is greater than or equal to zero then bf of 3 is 0 else bf of 3 is -1.(After right rotation)
- As we add one left node and one right node to 2, the bf will remain same.

(4) Rotate right left:-
Example,



- In this example there are two rotations. First right rotation at node 3 and then left rotation at node 2. We will create two temp pointers pointing at 2 and 3.
- Bf of 1 is -2. Bf of 3 is 1. (Condition for right left rotation).
- In right rotation, the right subtree of 2 becomes left subtree of 3 and 3 becomes right child of 2.

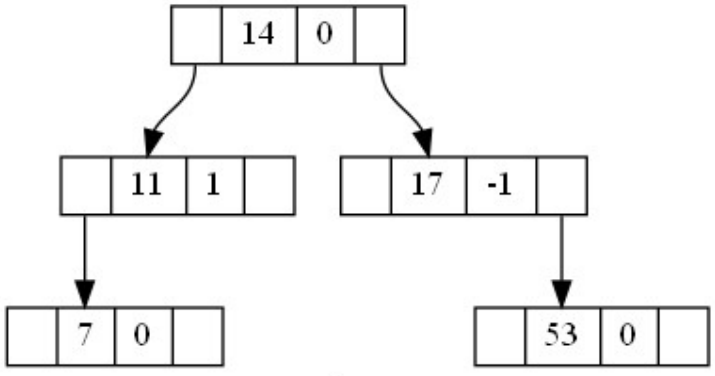
- If bf of 2 is greater than or equal to zero then bf of 3 is 0 else bf of 3 is -1.(After right rotation)
- In left rotation, the left subtree of 2 becomes right subtree of 1 and 1 becomes left child of 2.
- If bf of 2 is less than or equal to zero then bf of 1 is 0 else bf of 1 is -1.(After left rotation)
- As we add one left node and one right node to 2, the bf will remain same

---

INSERTION

- Lets say the new element is k. Search for the element in the tree. If the element is already present in the tree then return that, element is already present.
- If the element is not present in the tree then traverse from root node to the position where node is going to be inserted.
- While traversing from root to leaf node push all the references of nodes in the stack.
- Now the new node will we added to the leaf node and we will call it current node.
- As we added all the nodes in the stack, traverse back using those nodes.
- If the current node is left child of parent node then bf++ for parent node else bf--.

- Case 1 : If the parent node bf becomes 0 after above step, it means this subtree is balanced and we don't need to go further up.
- Case 2 : If the parent node bf becomes 1 or -1 then, continue the pop operation in stack.
- Case 3 : If the parent node bf becomes 2 or -2 then, apply rotation depending upon bf of parent node and current node.
- After rotation the new subtree becomes balanced and we will end the function.
Example, the initial tree is,



tree

All the rotations happened will be printed.

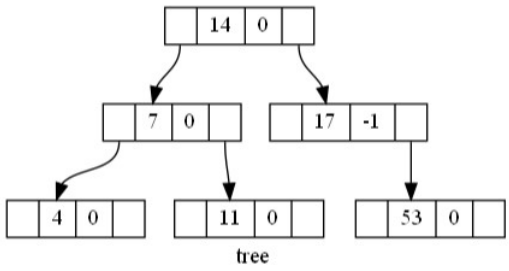
Now we are going to add elements in tree given above. At first we will add 4.

```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

1
Enter the value you want to insert: 4
Rotate_right


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

4
Tree Printed Successfully! Please check the image.png fi
An image.png file is generated in same folder
```



tree

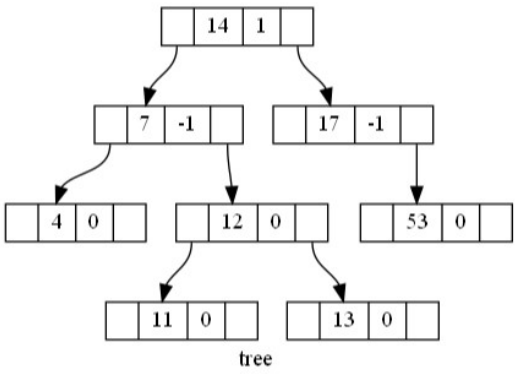
Here **Rotate right** operation happened and we will get new balanced tree. Now we will add 13 and 12 in the above tree.

```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

1
Enter the value you want to insert: 13


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

1
Enter the value you want to insert: 12
Rotate_right_left
```



tree

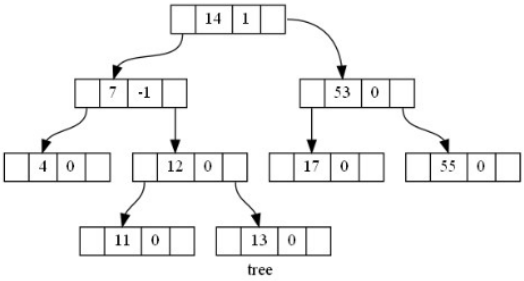
After entering 13 and 12 we can see **Rotate right left** operation to balance the tree. Now we will add 55 in the above tree.

```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

1
Enter the value you want to insert: 55
Rotate_left


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

4
Tree Printed Successfully! Please check the image.png file.
An image.png file is generated in same folder
```



tree

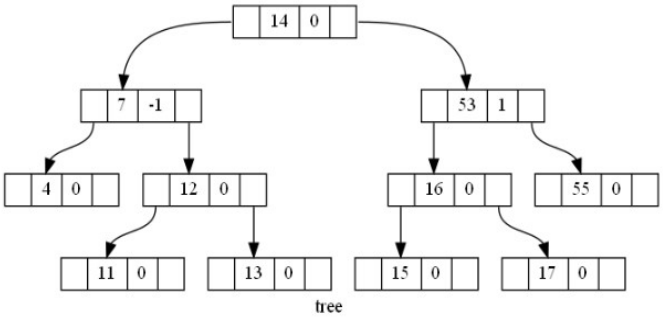
Here we can see **Rotate left** operation after adding 55 to the tree. Now we will add 15 and 16 in the tree.

```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

1
Enter the value you want to insert: 15


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

1
Enter the value you want to insert: 16
Rotate_left_right
```



tree

Here we can see the **Rotate left right** operation. Now the new tree will be balanced.

We have covered all four rotate operations in insertion.

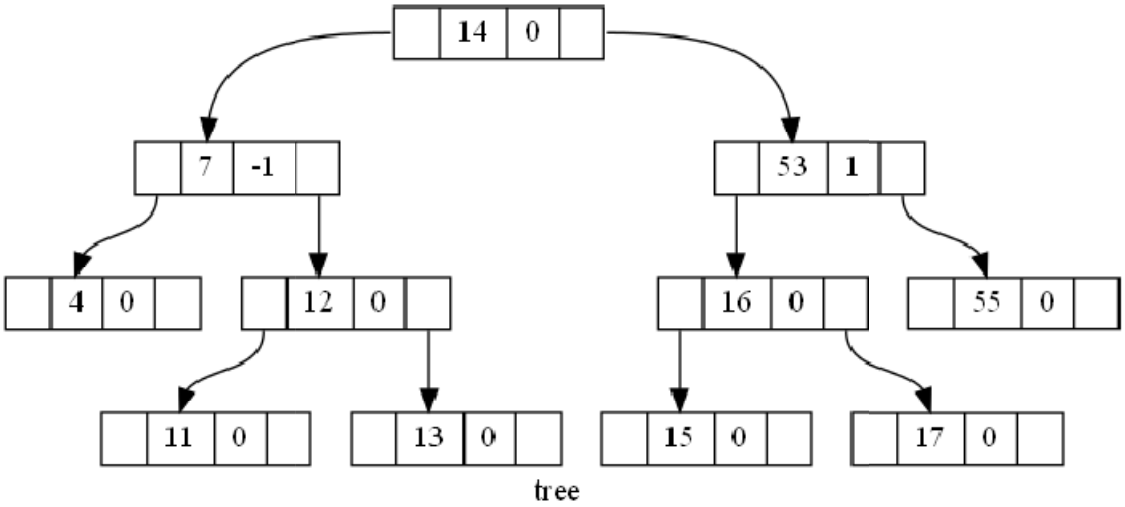
DELETION

- Search for the element in the tree. If the element is present then we move further else return, element is not present.
- Here also we are going to use stack.
- Push the pointers of nodes in the stack from root node to delete node.
- If the right child of delete node is null, then join its parent node to its left child.
- If the right child is not null, then find the inorder successor of delete node.
- While finding inorder successor, push all nodes in stack from delete to successor.
- Now swap the key values.
- Now the inorder successor becomes the delete node.
- If the delete node is leaf node then keep a pointer on it. If delete node is not leaf node then, join right child of delete node to the left child parent node.
- Now perform pop operation on stack.
- If the current node is left child of parent node then bf-- else bf++.

- Case 1: If the bf of parent node becomes 1 or -1 then return the function. It means height of the tree is not changed.
- Case 2: If the bf of parent node becomes 0 then, keep performing pop operation on stack. It means height of the tree may changed.
- Case 3: If the bf of parent node becomes 2 or -2 then, check balance factors of parent node and current node and perform rotate operation. Now the bf of new root node of subtree becomes 0. So keep performing pop operation on stack.

- Pop operations on stack will go on till bf of parent node is 1 or -1 or it reaches the root node

Example,



tree

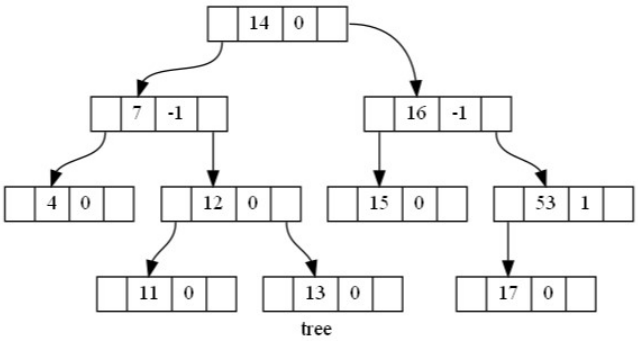
```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

2
Enter the value you want to delete: 55
Rotate_right


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

4
Tree Printed Successfully! Please check the image.png file
An image.png file is generated in same folder


Enter the number to perform the operation given below:
```



tree

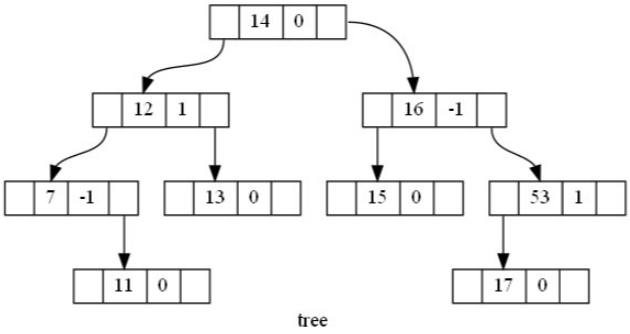
As we can see we have deleted 55 form above tree. **Rotate right** operation being performed.

```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

2
Enter the value you want to delete: 4
Rotate_left


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

4
Tree Printed Successfully! Please check the image.png file.
An image.png file is generated in same folder
```



tree

As we can see we have deleted 4 form above tree. **Rotate left** operation being performed.

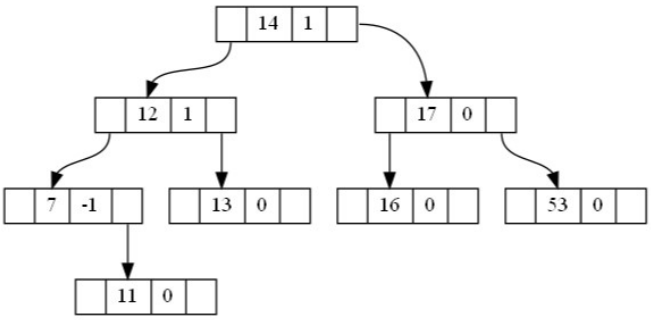
As we can see we have deleted 15 form above tree. **Rotate right left** operation being performed.

```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

2
Enter the value you want to delete: 15
Rotate_right_left


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

4
Tree Printed Successfully! Please check the image.png file.
An image.png file is generated in same folder
```



tree

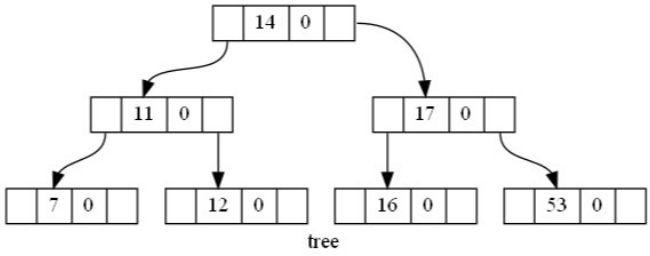
As we can see we have deleted 13 form above tree. **Rotate left right** operation being performed.

```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

2
Enter the value you want to delete: 13
Rotate_left_right


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

4
Tree Printed Successfully! Please check the image.png file.
An image.png file is generated in same folder
```



tree

SEARCH

- Given tree is the balanced binary search tree.
- If key is equal to current node return element is present in tree.
- If key is less than current node move to left child.
- If key is greater than current node move to right child.
- If current node is null then, element is not present in tree.

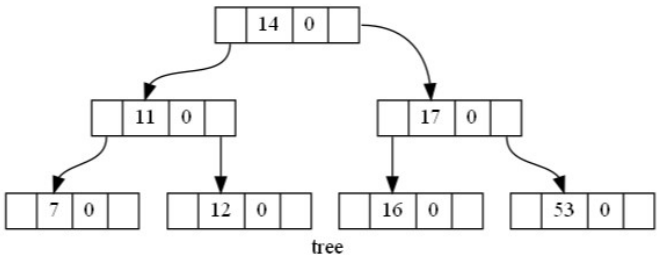
```
Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

3
Enter the value you want to search: 16
key is present in tree


Enter the number to perform the operation given below:
1. Insertion
2. Deletion
3. Search
4. Print
5. Exit

3
Enter the value you want to search: 15
key is not present in tree


Enter the number to perform the operation given below:
1. Insertion
```



tree

PRINT

- For the print operation press 4 and enter.
- A dot file will generate in same folder.
- An image.png file will be generated directly.
- System call for image.dot file: "dot -Tpng image.dot -o image.png".