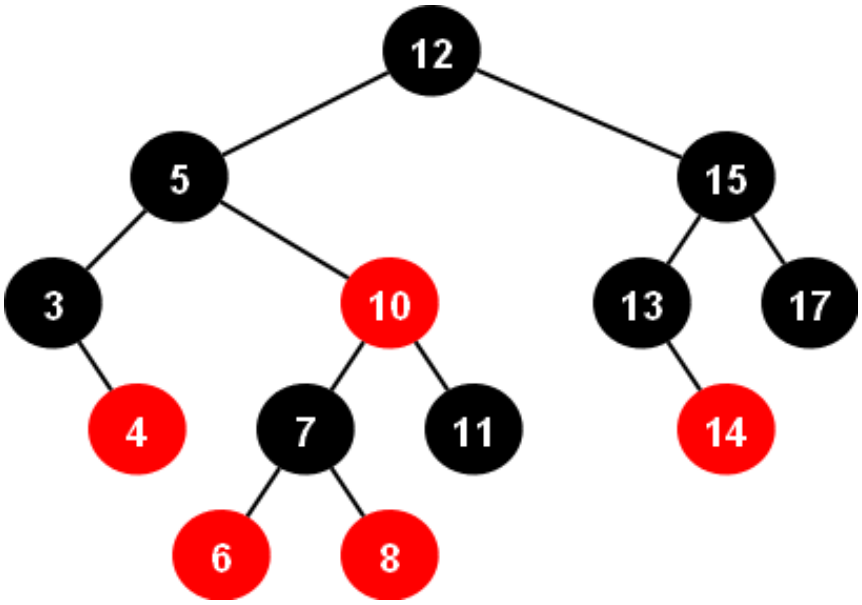
Data structures and Algorithms

Red-Black Tree



ID	NAME
19016063	Omar Khairat Mohamed
19015941	Abdelaziz Mohamed Abdelaziz

Overview

A red black tree is a kind of self-balancing binary search tree in computer science. Each node of the binary tree has an extra bit, and that bit is often interpreted as the color (red or black) of the node. These color bits are used to ensure the tree remains approximately balanced during insertions and deletions. Balance is preserved by painting each node of the tree with one of two colors in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

RBTree Class methods

rightRotate

It rotates the given node as parameter in the right direction and return the node that takes its place.

leftRotate

It rotates the given node as parameter in the left direction and return the node that takes its place.

getRoot

It returns the root of the given Red black tree.

isEmpty

It returns whether the given tree is Empty or not.

clear

It clears all keys in the given tree.

search

It returns the node associated with the given key. There is an overloading of this method that takes the root as an additional parameter and It searches for the node recursively by putting root as child subtree

contains

It returns true if the tree contains the given key and false otherwise.

insert

It inserts the given key in the tree while maintaining the red black tree properties and if the key is already present in the tree, update its value. It takes key and value as parameter.

Delete

It deletes the node associated with the given key and it returns true in case of success and false otherwise.

Subtree_first

It brings the first node in subtree inorder traversal where it returns INode wrapper to the first node in subtree inorder traversal.

Subtree_last

It brings the last node in subtree inorder traversal where it returns INode wrapper to the last node in subtree inorder traversal.

fixInsert

It makes a color balance and the necessary rotations to keep up with red black tree properties. In the first case if the parent is black then we exit while in the second case the parent is red and the uncle is red we then check if the grandparent is not root to recolor it. In the third case the parent is red and uncle is black thus we have to make rotations according to the position of the child and parent.

assignParent

It puts the node as child to its parent with parameters node which is the node to be put as child and isLeft which detects if the node should be put on left.

fixDelete

It removes the double black node by applying cases to keep the red-black tree properties. If the node is root we pass it as we remove the double black color. If the node's sibling is red we swap the parent and sibling color and rotate the parent in the node direction but first we remove node. If he node's sibling is black we see the color of sibling's children if there is children and if one child is missing we set its color as black. We distinguish the sibling children according to their nearness to the double black node. If the two children are black we make the sibling red and add black to parent if it is black it become double thus we remove double black node. If the sibling near child is red while the other far one is black we swap the color of the sibling and its near child and we rotate the parent in direction opposite to the double black node direction moreover we apply the next case. In the final case if the far child is red we

swap the parent and sibling's colors, rotate the parent in double black node direction and finally delete double black node.

deleteNode

It brings the node to be deleted by putting it as leaf to be deleted by the delete method.

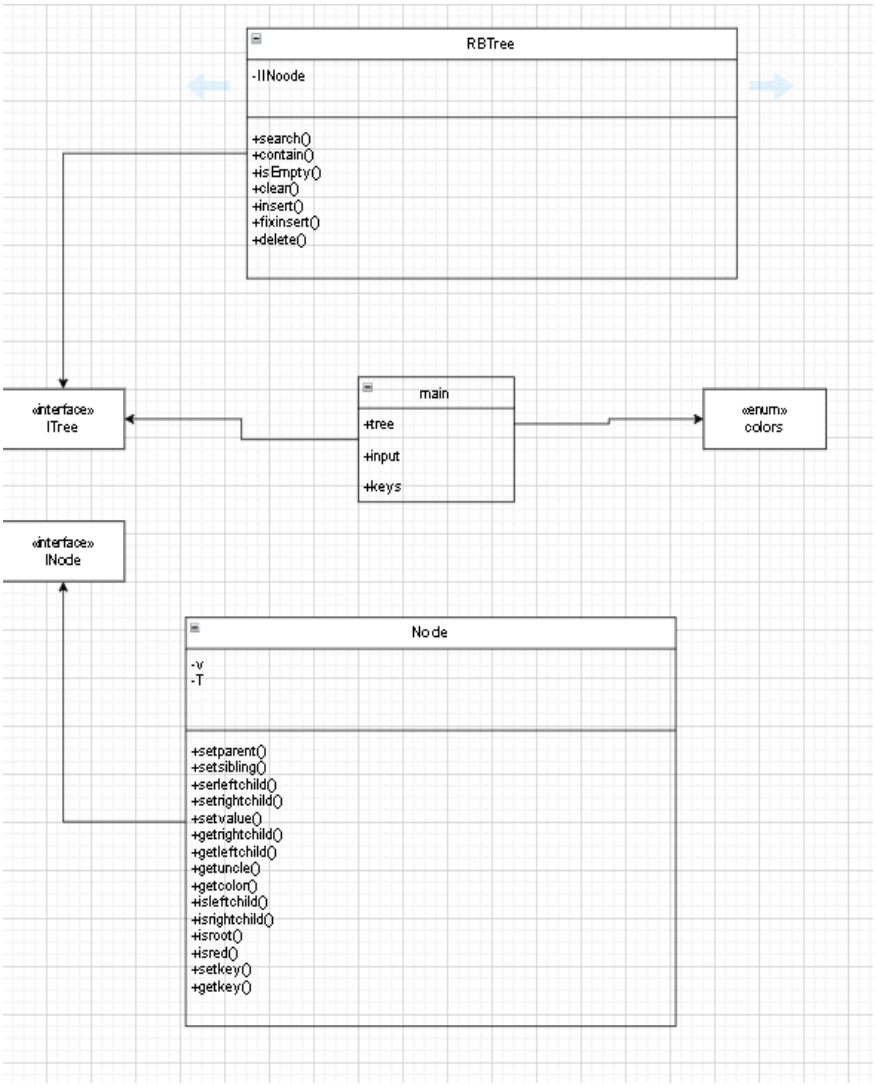
Successor

It gets the given node successor.

Predecessor

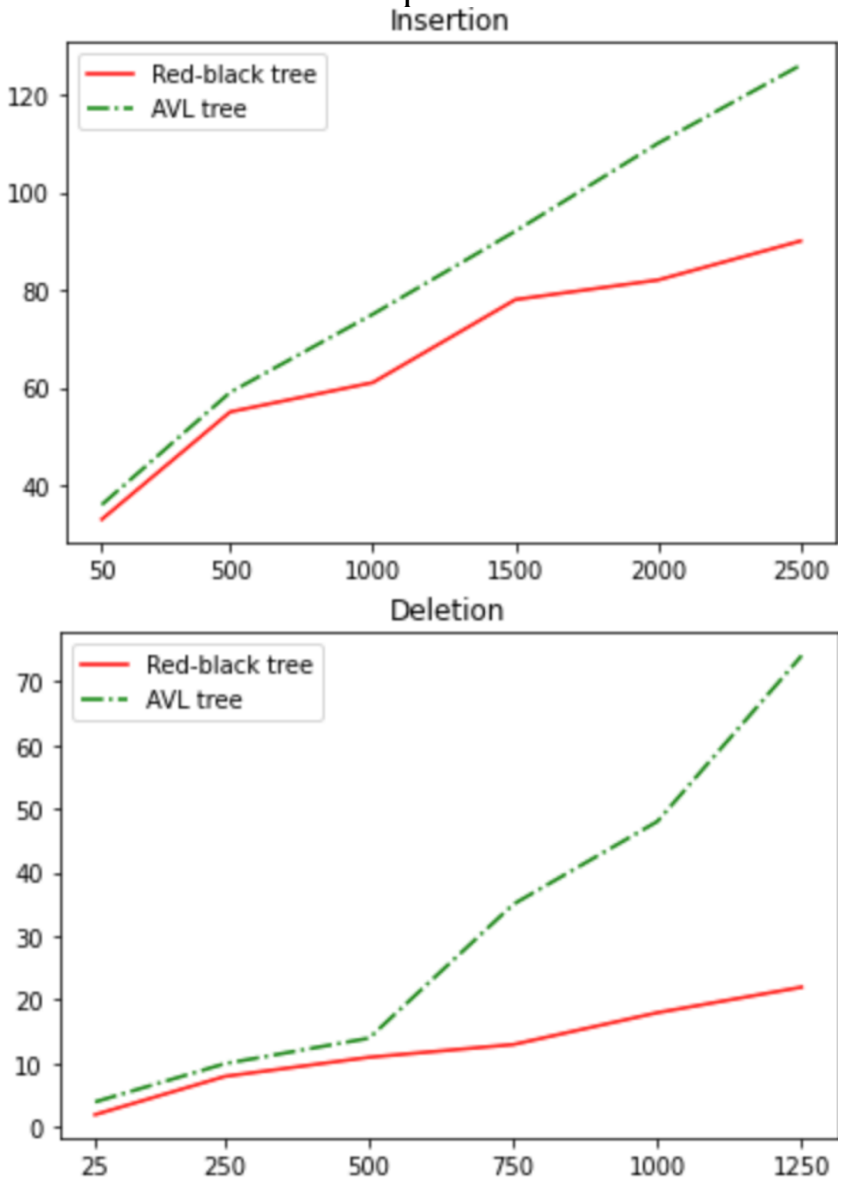
It gets the given node predecessor.

UML



Comparison with AVL tree

We compare the time taken by AVL tree and red-balck tree by insertion and deletion on different samples of nodes of different size



We notice red-black tree provide faster insertion and removal operations than AVL trees as fewer rotations are done due to relatively relaxed balancing.

Sample Runs

```
Enter the keys to build tree
1 2 3 4 5 6 7 8
Enter the values corresponding to the given keys
1 2 3 4 5 6 7 8
Time to insert all nodes: 3
*************************
1
parent: red 2
*************************
2
parent: black 4
leftChild: black 1
rightChild: black 3
*************************
3
parent: red 2
*************************
4
leftChild: red 2
rightChild: red 6
*************************
5
parent: red 6
*************************
6
```

```
6
parent: black 4
leftChild: black 5
rightChild: black 7
************************
7
parent: red 6
rightChild: red 8
************************
8
parent: black 7
************************
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
3
Enter the key
2
true
Time Taken: 0
```

```
Enter the key
2
true
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
4
Enter the key
2
false
Time Taken: 1

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
```

```
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
1
Enter the key
90
Enter the corresponding value
58
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
5
4

if you want to complete operations write yes
```

```
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
2
Enter the key to search for
90
58
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
3
Enter the key
6
true
Time Taken: 0
```

```
if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
3
Enter the key
6
true
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
2
Enter the key to search for
6
Not found
```

```
Enter the keys to build tree
44 12 45 33
Enter the values corresponding to the given keys
2 3 77 100
Time to insert all nodes: 4
************************
44
leftChild: black 12
rightChild: black 45
************************
12
parent: black 44
rightChild: red 33
************************
45
parent: black 44
************************
33
parent: black 12
************************
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
```

```
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
4
Enter the key
1000
false
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
1
Enter the key
1000
Enter the corresponding value
244
Time Taken: 0
```

```
244
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
4
Enter the key
1000
true
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
3
```

```
3. Delete
4. Contain
5. getRoot
3
Enter the key
4000
false
Time Taken: 0

if you want to complete operations write yes
yes
Select the number of the method you want to test:
1. Insert
2. Search
3. Delete
4. Contain
5. getRoot
1
Enter the key
1000
Enter the corresponding value
77
Time Taken: 0

if you want to complete operations write yes
```