What is a Red-Black Tree?

This lesson is an introduction to Red-Black trees, their properties, and the total time they take to perform the operations of insertion, deletion, and searching. We will also do a small comparison between AVL and Red-Black Trees.

We'll cover the following

Introduction#

Red-Black Trees are another type of self-balancing Binary Search Tree, but with some additions: the nodes in Red-Black Trees are colored either red or black. Colored nodes help with re-balancing the tree after insertions or deletions. We will go through the insertion and deletion functions of Red-Black trees just like we did with AVL Trees previously.

Properties of Red-Black Trees#

- Every node is either **Red** or **Black** in color
- The root is always colored **Black**

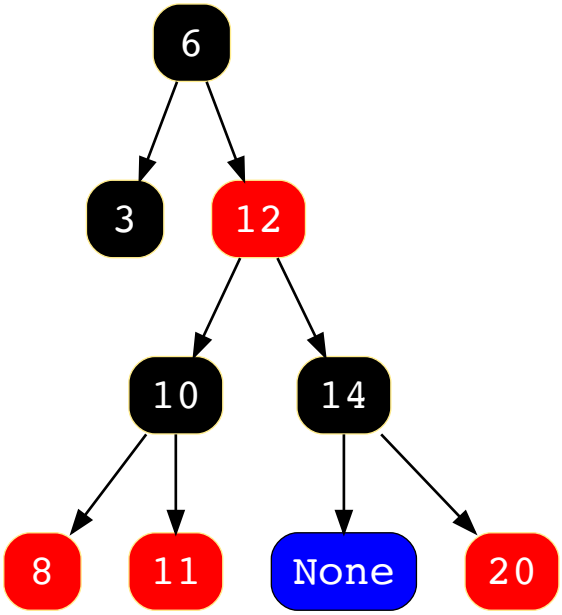
- Two **Red** nodes cannot be adjacent, i.e., No red parent can have a red child and vice versa
- Each path from the root to None contains the same number of **Black** colored nodes
- The color of *None* nodes is considered **Black**

From the perspective of implementation, our node class will contain an addition of a *boolean* variable that will store the information about the color of a node. Here is a basic structure of a Node which will be used to build a Red-Black tree.

```
class Node:
  def __init__(self,val):
    self.val = val
    self.leftChild = None
    self.rightChild = None
    isRed = None # True if Node is RedColored else false
```

Node Class

Here is an example of a valid Red Black Tree:



Time Complexity#

Balancing the tree doesn't result in a tree being perfectly balanced, but it is good enough to make time complexity of basic operations like searching, deletion, and insertion to be around $O(logn)$.

AVL vs. Red-Black Trees#

Although AVL Trees are technically more 'balanced' than Red-Black Trees, AVL Trees take more rotations during insertion and deletion operations than Red-Black Trees. So, if you have search-intensive applications where insertion and deletion are not that frequent, use AVL Trees, otherwise, use Red-Black Trees.

As the above operations involve a series of steps and cases to follow to fulfill the property of Red-Black Trees and to keep the Trees balanced, we will look into each operation of insertion and individually in the next lessons.

Back
AVL Deletion

Next
Red-Black Tree Insertion