```
//tag nodes
Node Node.LEFT
Node Node.RIGHT
```

Algorithm next(Node n, int x)

Input: A node n in a fixed red black tree T of integers and an integer x to be inserted

Output: Either the left or right child of n (if not null) on the path to the insertion point for x; if the child to be returned is null, one of the tag nodes is returned instead

```
if n is black and both its children are red, perform a color flip; if the color flip causes the color of n to
    become red and the color of the parent of n is also red, perform Rebalance
if (x > n.value)
    if (n.right is null) return Node.RIGHT
    else return n.right
if (x < n.value)
    if (n.left is null) return Node.LEFT
    else return n.left
```

Algorithm insert(int x)

Input: An int x, to be inserted into a red black tree T

Output: T, after x has been inserted

```
If the root of T is null, create a black node containing x; return T
Node n <- root
boolean inserted <- false
while (not inserted) do
   nextNode <- next(n)
   if (nextNode == Node.RIGHT or nextNode == Node.LEFT)
        create red node r
        place x inside r
        if(nextNode == Node.RIGHT)  n.right <- r
        else n.left <- r
        inserted <- true
   else //nextNode not null
       n <- nextNode

//a red node containing x has been added at insertion point
if parent of new red node is also red, do Rebalance
```