Using insertion method introduced in the lecture, prove that if n>1, the tree contains at least one red node.

Below is the inserting algorithm in the lecture slides.

```
RB-INSERT(T, z)
y = T.nil
x = T.root
while x ≠ T.nil
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == T.nil
    T.root = z
elseif z.key < y.key
    y.left = z
else y.right = z
z.left = T.nil
z.right = T.nil
z.color = RED
RB-INSERT-FIXUP(T, z)
```

The new inserted node always has the red color. The only thing that might change this situation is the following RB_INSERT_FIXUP function.

```
RB-INSERT-FIXUP(T, z)
while z.p.color == RED
    if z.p == z.p.p.left
        y = z.p.p.right
        if y.color == RED
            z.p.color = BLACK          Case 1
            y.color = BLACK
            z.p.p.color = RED
            z = z.p.p
        else if z == z.p.right
                z = z.p                Case 2
                LEFT-ROTATE(T, z)
            z.p.color = BLACK          Case 3
            z.p.p.color = RED
            RIGHT-ROTATE(T, z.p.p)
    else (same as then clause
            with "right" and "left" exchanged)
T.root.color = BLACK
```

There are three cases under the condition that z.p.color is RED, therefore, when z.p.color is black, the tree will not be changed as well as the color of the new inserted node.

Let's consider the three cases that may change colors, in every case, each time a node's color being changed to black, there is always a node changes to RED. Therefore, no matter how many times the tree is being fixed in this function, one RED node is always remained.