Red-Black Tree

"Introduction to Algorithms" (Cormen et al, 3rd ed.), Chapter 13

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

For example,

```
import (
    "fmt"

    "go.etcd.io/etcd/pkg/v3/adt"
)

func main() {
    ivt := adt.NewIntervalTree()
    ivt.Insert(NewInt64Interval(510, 511), 0)
    ivt.Insert(NewInt64Interval(82, 83), 0)
    ivt.Insert(NewInt64Interval(830, 831), 0)
    ...
```

After inserting the values `510`, `82`, `830`, `11`, `383`, `647`, `899`, `261`, `410`, `514`, `815`, `888`, `972`, `238`, `292`, `953`.

red-black-tree-01-insertion.png

Deleting the node `514` should not trigger any rebalancing:

red-black-tree-02-delete-514.png

Deleting the node `11` triggers multiple rotates for rebalancing:

red-black-tree-03-delete-11.png red-black-tree-04-delete-11.png red-black-tree-05-delete-11.png red-black-tree-06-delete-11.png red-black-tree-07-delete-11.png red-black-tree-08-delete-11.png red-black-tree-09-delete-11.png

Try yourself at https://www.cs.usfca.edu/~galles/visualization/RedBlack.html.