G54AAD

L10 Red Black Trees

This lecture: PFDS Ch 3.3

Red Black Trees

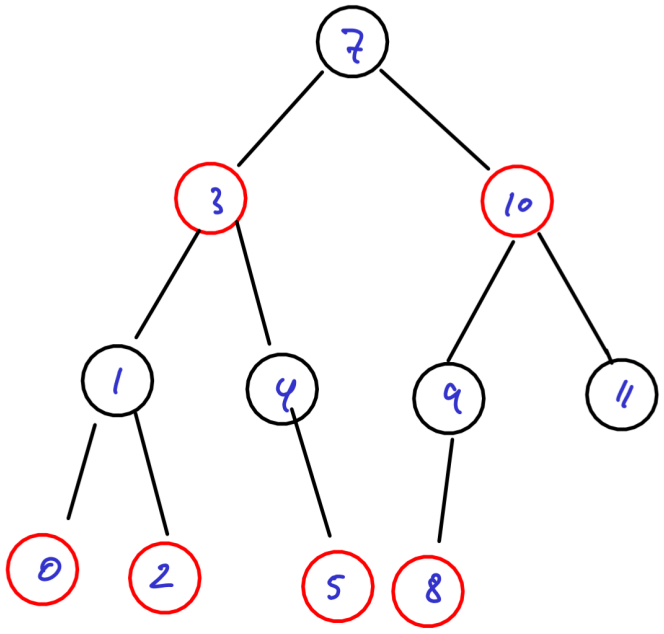
The binary search trees we studied in L4 may be highly unbalanced. Red Black trees are binary search trees that remain approximately balanced.

Def: A **Red Black tree** is a binary search tree where each node is coloured either red or black such that

(1) No red node has a red child.

(2) Every path from the root to the leaf contains the same number of black nodes.

Ex:



How balanced are Red Black Trees?

Lemma

The maximum depth of a node in a Red Black Tree of size n is no more than

$$2\lfloor \log(n+1) \rfloor.$$

How can we prove this?

1) What information do we have?

Let s be the length of the shortest path,
l be the length of the longest path,
k be the number of black nodes along any path.

By (2) $k \leq s \leq l \leq 2k$, so $2s \geq l$

How large is l?

Since each path contains at least s nodes, the number of nodes in the tree is $n \geq 2^{s+1} - 1$.

Hence

$$\begin{aligned} 2\lfloor \log(n+1) \rfloor &\geq 2\lfloor \log(2^{s+1}) \rfloor \\ &= 2\lfloor s+1 \rfloor \\ &\geq 2\lfloor l/2 + 1 \rfloor \\ &\geq l. \end{aligned}$$

□

Operations on Red Black Trees

We assume empty nodes to be black.

```
data Colour = R | B
data RedBlackTree a = L | N Colour a
                      (RedBlackTree a)   -- left
                      (RedBlackTree a)   -- right
```

1) `member :: a -> RedBlackTree a -> RedBlackTree a`
`member x ts` ≡ does `ts` contain a node with value `x`?

`member` $O(\log n)$ by Lemma 1.
implemented as for binary search trees.

2) `insert :: a -> RedBlackTree a -> RedBlackTree a`
How do we manage to satisfy the two invariants?

- insert red leaf node
  => Number of black nodes unchanged, so invariant 2 still satisfied.
  => Invariant 1 may be violated in exactly one place in the tree.

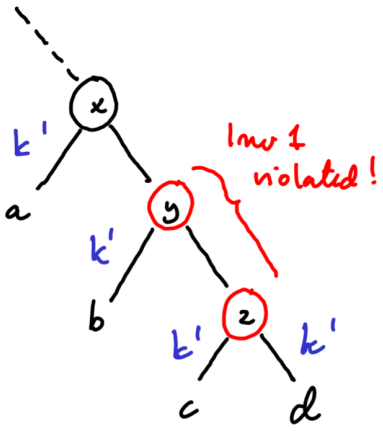
```
ins x E = RB R x E E
ins x (RB c y left right)
  | x ≤ y = balance (RB c y (ins x left) right)
  | x > y = balance (RB c y left (ins y right))
```

Makes sure that root is black.

```
insertTree x ts =
  RB B y left right
  where RB _ y left right = ins x ts.
```

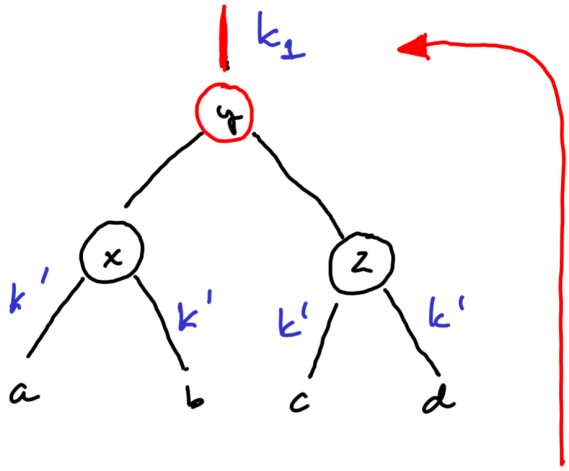
Assume that invariance 1 is violated.
What will the tree look like, and how can we rebalance it?

CASE 1:
BEFORE BALANCING



BALANCE REWRITING STEP

CASE 1:
AFTER BALANCING



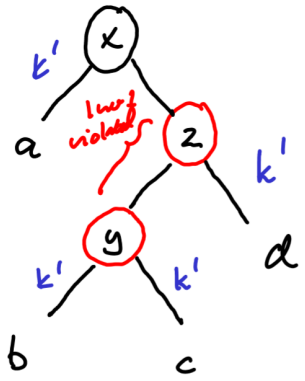
- Invariant 2 is not violated, so the number of black nodes along the paths from a, b, c, d must be the same, say k'.
- By the binary search tree property, we have

$$a \leq x \leq b \leq y \leq c \leq z \leq d$$

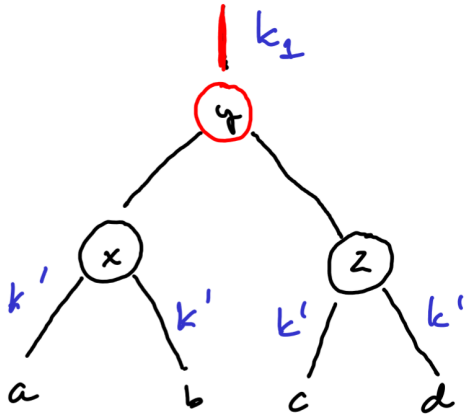
Rebalance along this node!

- After the balancing rewriting step, invariant 1 may be violated at one higher level in the tree.
- Invariant 2 is satisfied.

CASE 2



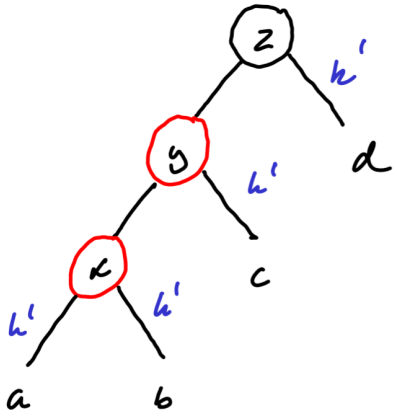
BALANCE REWRITING STEP →



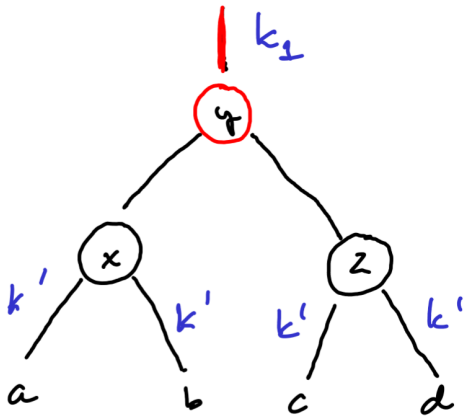
$a \leq x \leq b \leq y \leq c \leq z \leq d$

↑ Rebalance along this node!

CASE 3



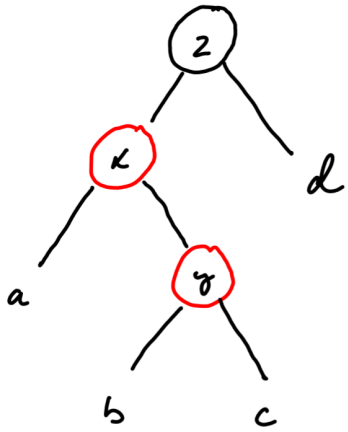
BALANCE REWRITING STEP →



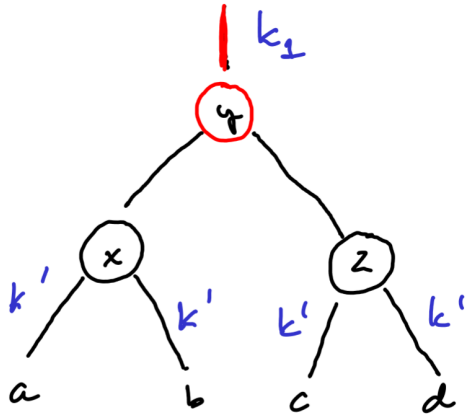
$a \leq x \leq b \leq y \leq c \leq z \leq d$

↑ Rebalance along this node!

CASE 4



BALANCE REWRITING STEP →



$$a \leq x \leq b \leq y \leq c \leq z \leq d$$

↑ Rebalance along this node!