Australian National University

COMP3600/6466 Algorithms

Lecture 19

S2 2016

Dr. Hassan Hijazi

Prof. Weifa Liang

Red Black Trees

WHAT ARE THEY?

Balanced Binary Search Trees
+
One additional attribute per node:
colour
Red or Black

Can be stored as a Boolean

Introduced by Bayer in 1972 (44 years!)

Red Black Trees

WHAT ARE THEY FOR?

Balanced Binary Search Trees Can be Unbalanced

$$O(h) = O(n)$$

Red Black Trees are balanced BSTs

$$\text{All operations are } O(\lg n)$$

Red Black Trees

Definition

Node attributes:

colour, left, right, p(parent), key and data

Five properties:

By definition of a Boolean

P1. Every node is either RED or BLACK.

Convention

P2. The root is BLACK.

P3. Every NIL leaf is BLACK.

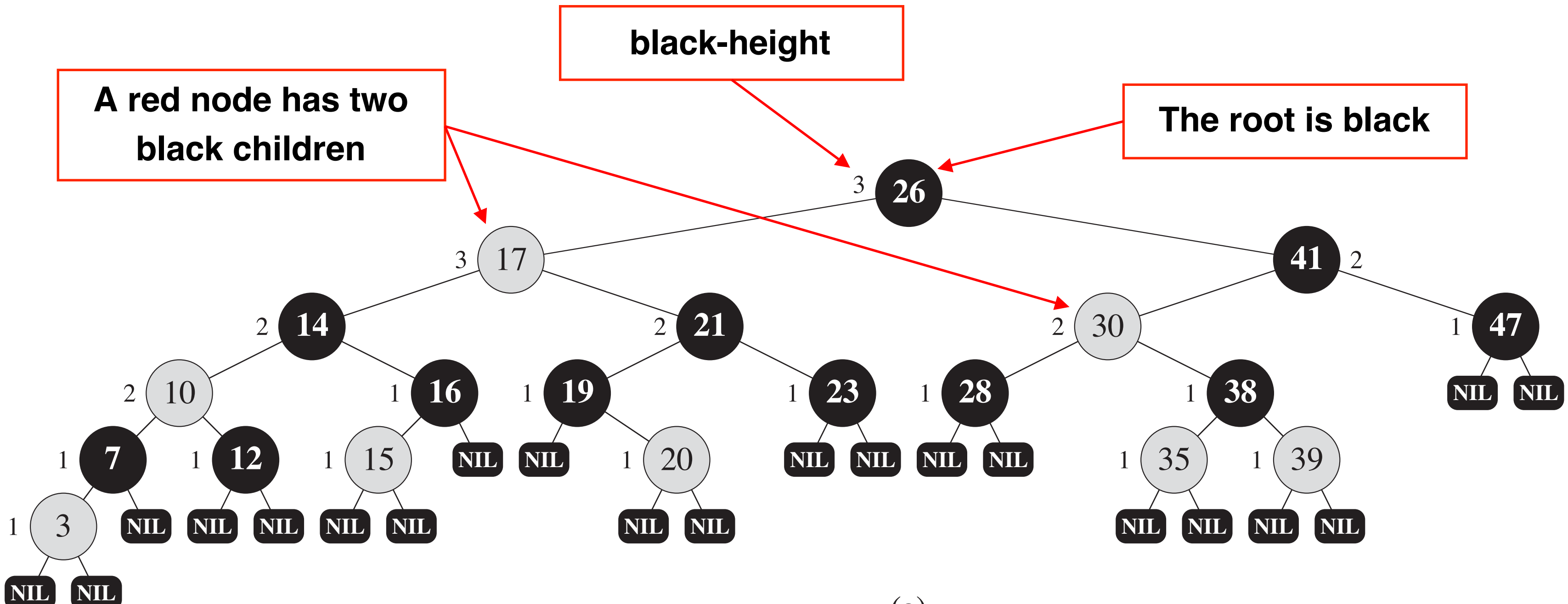
Main Properties

P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

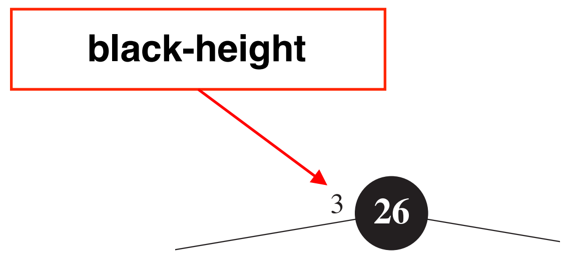
Representation



(a)

Red Black Trees

Black Height



The **black height** of a node x is the number of BLACK nodes on a path from x down to a leaf, **not counting x itself but counting the leaf.**

Notation: $bh(x)$

By Property 5, all paths leading to a leaf will return the same number of black nodes, thus

$bh(x)$ is unique

Red Black Trees

MAIN LEMMA

Lemma.
A red-black tree with n internal nodes has height h at most $2\log(n+1)$.

Proof outline:

We show that:

(1) The subtree rooted at any node x has at least $2^{bh(x)} - 1$ internal nodes.

(2) The tree height h is at most twice the black height of the tree's root.

Red Black Trees

(1) At least $2^{bh(x)} - 1$ internal nodes

Proof by induction

Let $ni(x)$ denote the number of internal nodes of the tree rooted at x.

Claim. $ni(x) \geq 2^{bh(x)} - 1, \ \forall x \in T$.

Base case. If x is a leaf, its black height $bh(x) = 0$ and the tree rooted at x contains 0 internal nodes, thus the claim is true:

$$ni(x) = 0 \geq 2^{bh(x)} - 1 = 2^0 - 1 = 0.$$

Induction. Consider a internal node x with two children l and r, we have:

$$bh(l) \geq bh(x) - 1 \text{ and } bh(r) \geq bh(x) - 1,$$

A child's colour can be red or black

By inductive hypothesis, $ni(l)$ and $ni(r) \geq 2^{bh(x)-1} - 1$, thus,

$$ni(x) \geq 2 \times (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1.$$

Red Black Trees

$$(2)\ h \leq 2 \times bh(T.root)$$

Proof. (2)

According to Property 4, at least half the nodes on any simple path from the root to a leaf (excluding the root) must be black. Therefore,

$$bh(T.root) \geq h/2$$

Combining (1) and (2)

General Proof.

$$bh(T.root) \geq h/2 \underset{\text{from (1)}}{\Longrightarrow} n \geq ni(T.root) \geq 2^{h/2}-1 \implies n+1 \geq 2^{h/2} \implies h \leq 2\lg(n+1)$$

Red Black Trees

CONSEQUENCES OF MAIN LEMMA

$$h \leq 2\lg(n+1)$$

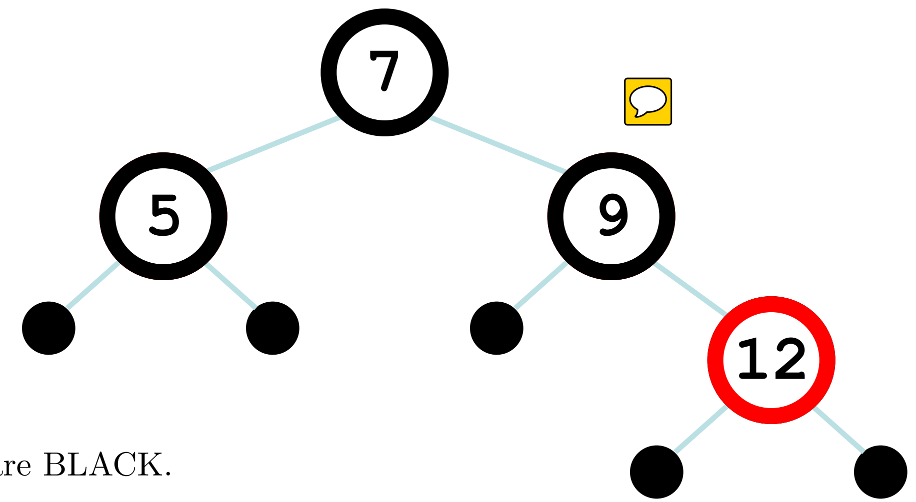
SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR

$$O(h) \implies O(\lg n)$$

Red Black Trees

What about INSERT and DELETE?

The Binary Search Tree algorithms for INSERT and DELETE might destroy properties P1 to P5

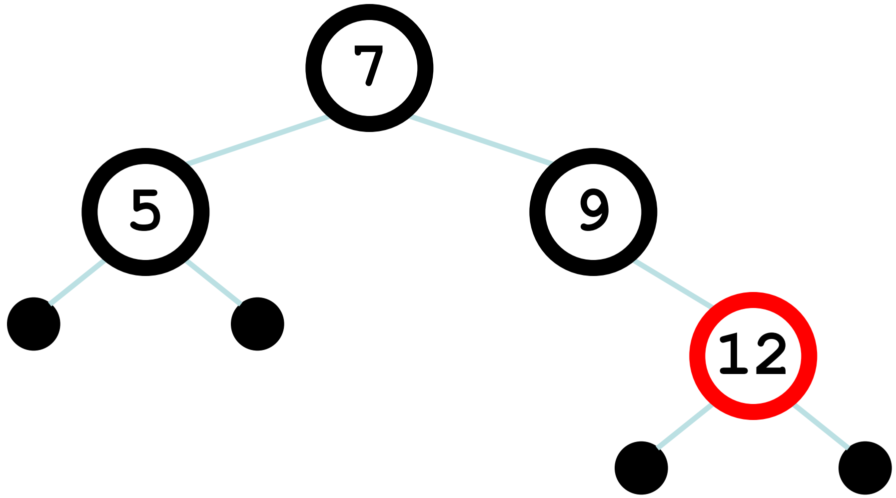


P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

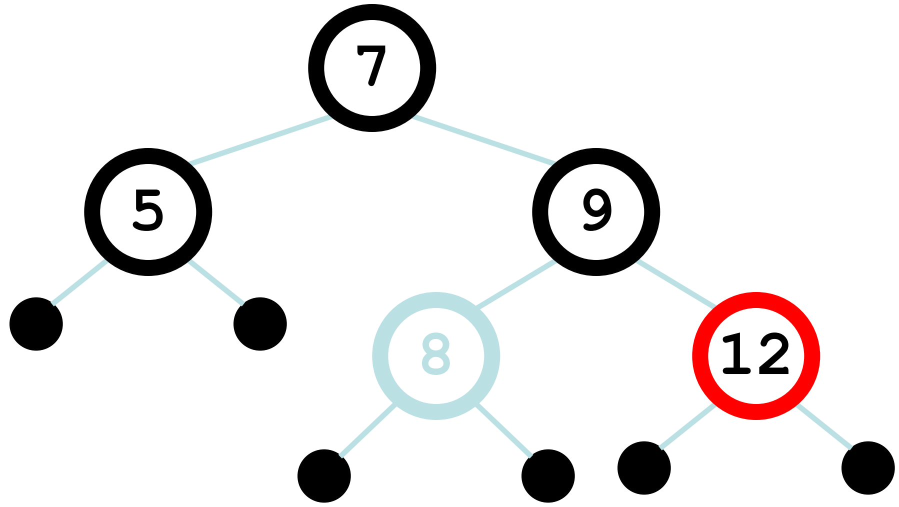
What about INSERT and DELETE?



INSERT 8

Red Black Trees

What about INSERT and DELETE?



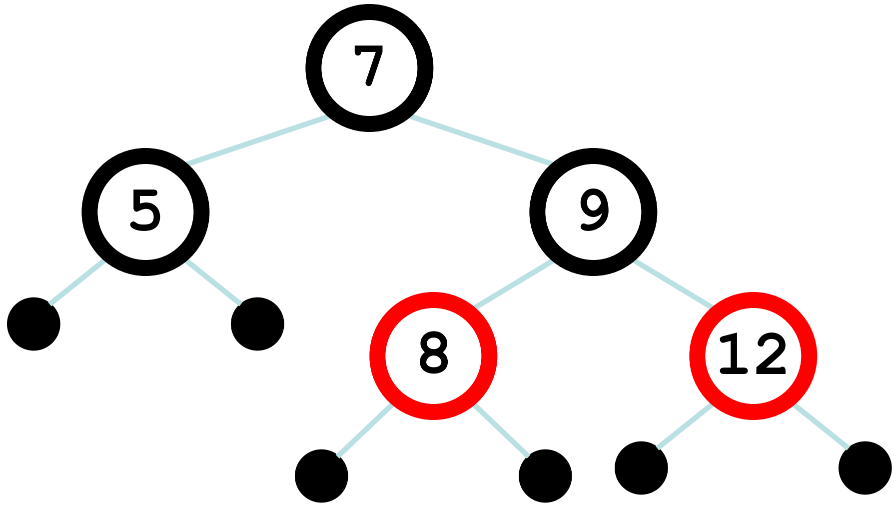
WHAT SHOULD I COLOR 8?

P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

What about INSERT and DELETE?



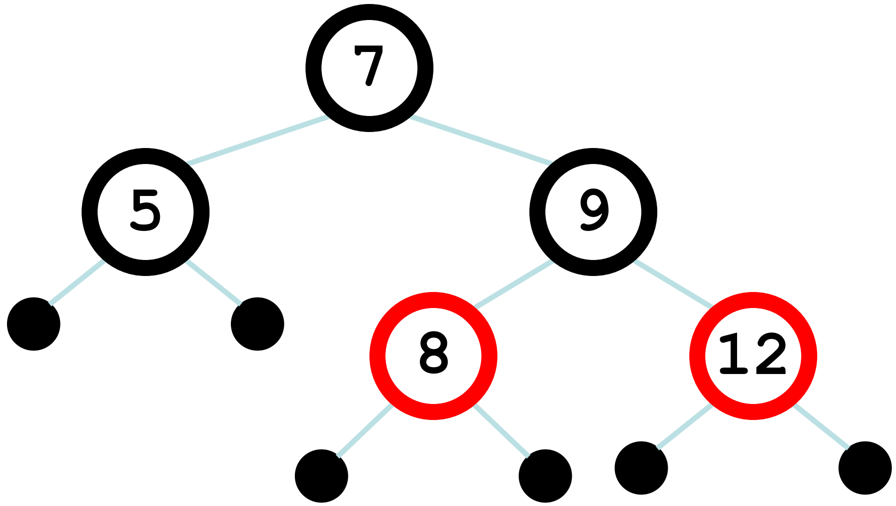
RED

P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

What about INSERT and DELETE?



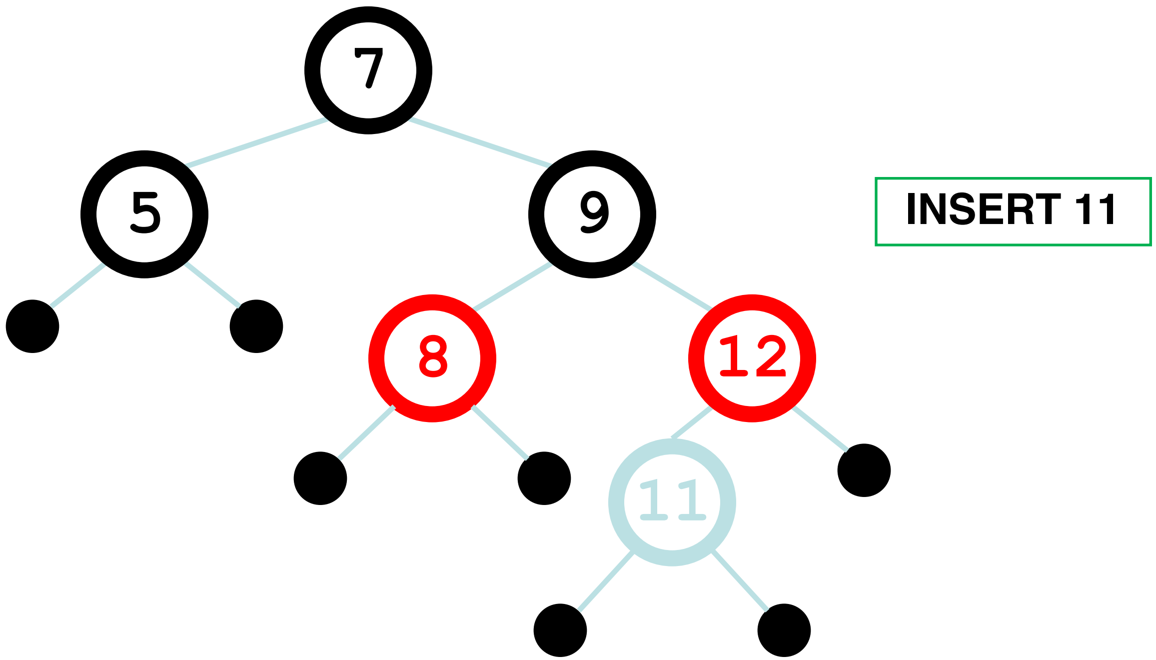
INSERT 11

P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

What about INSERT and DELETE?

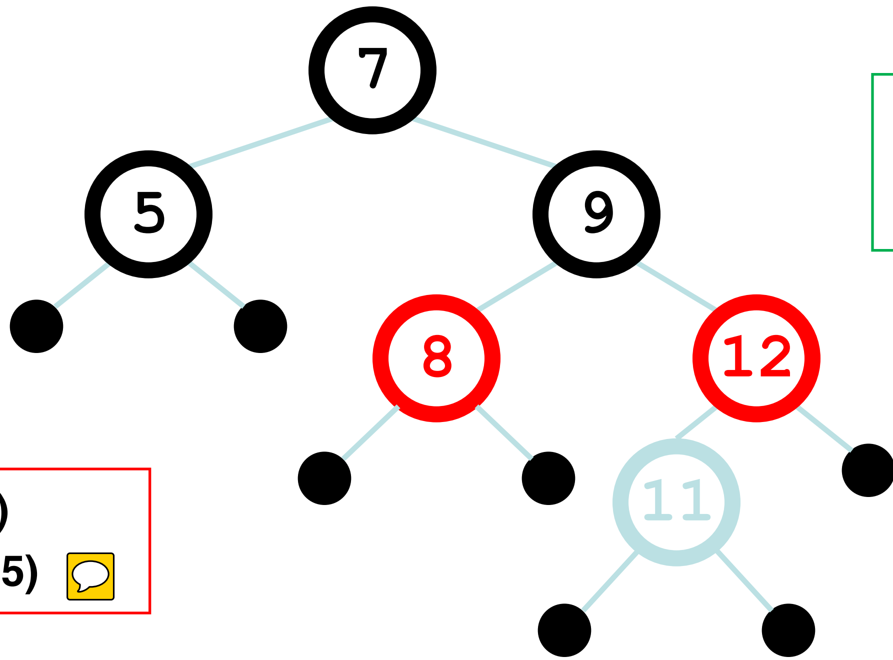


P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

What about INSERT and DELETE?



WHAT SHOULD I COLOR 11?

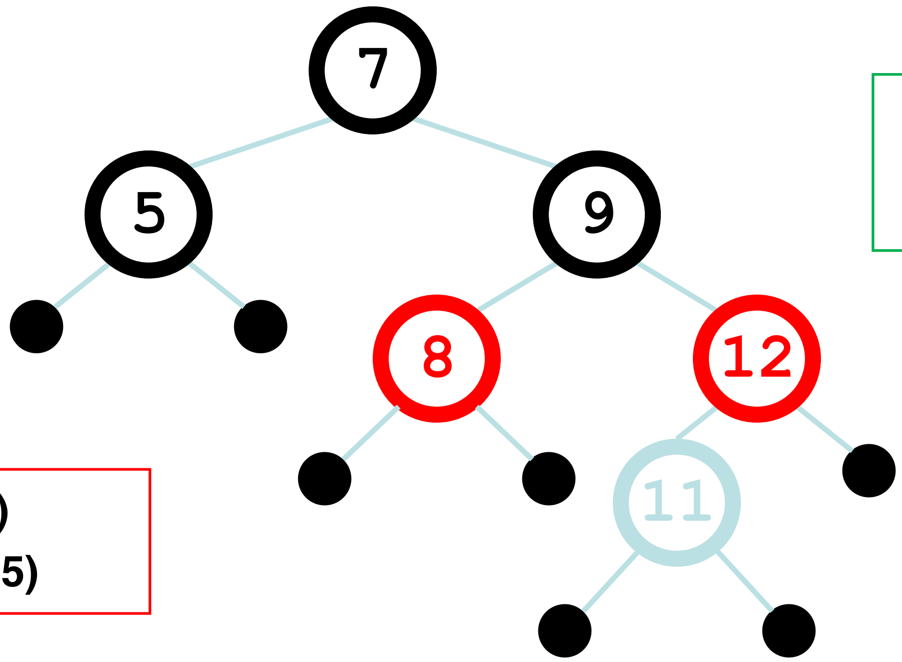
Can't be RED! (P4)
Can't be BLACK! (P5)

P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

ACTION 1. RECOLOR!



WHAT SHOULD I COLOR 11?

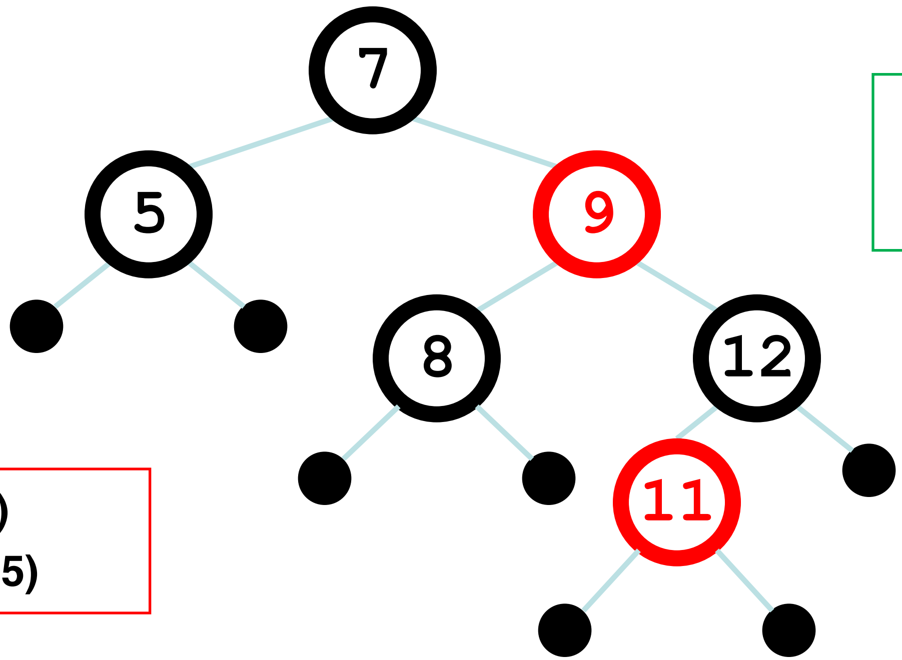
Can't be RED! (P4)
Can't be BLACK! (P5)

P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

ACTION 1. RECOLOR!



WHAT SHOULD I COLOR 11?

Can't be RED! (P4)
Can't be BLACK! (P5)

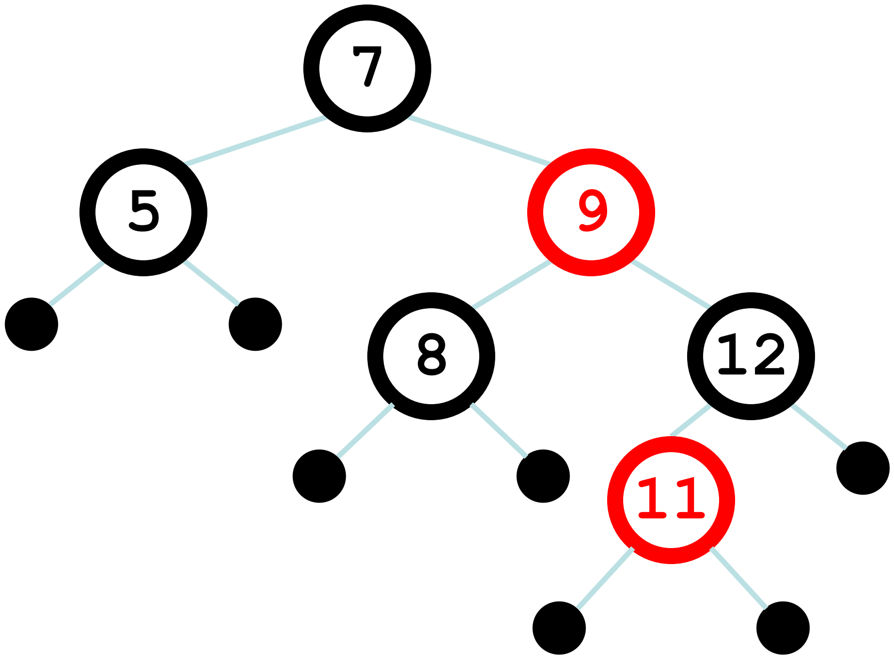
P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

What about INSERT and DELETE?

INSERT 10



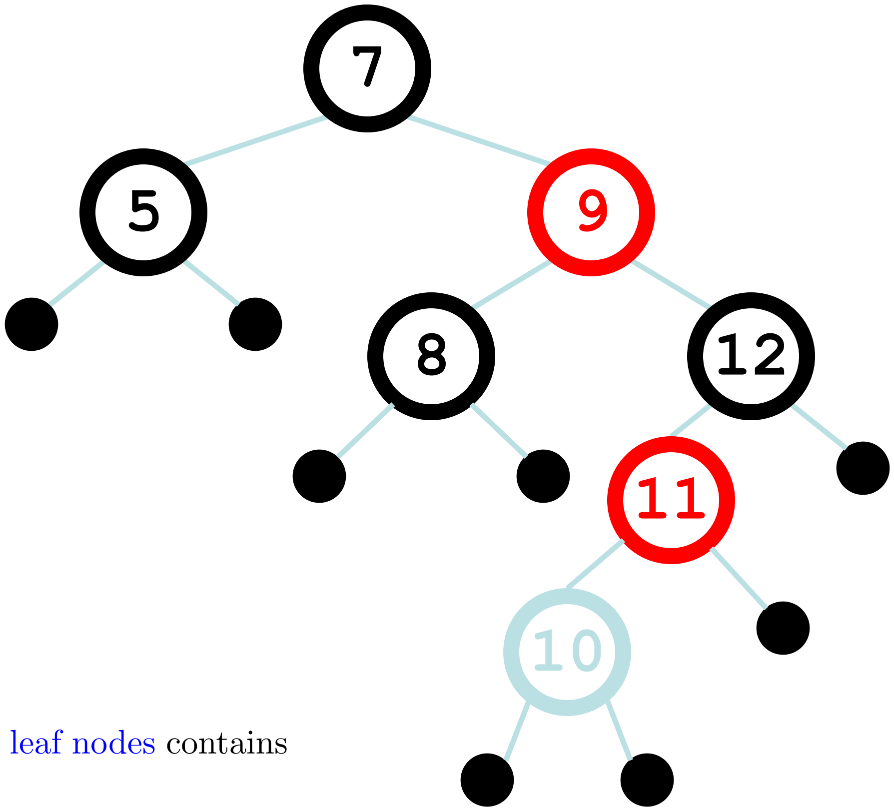
P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

What about INSERT and DELETE?

INSERT 10



P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

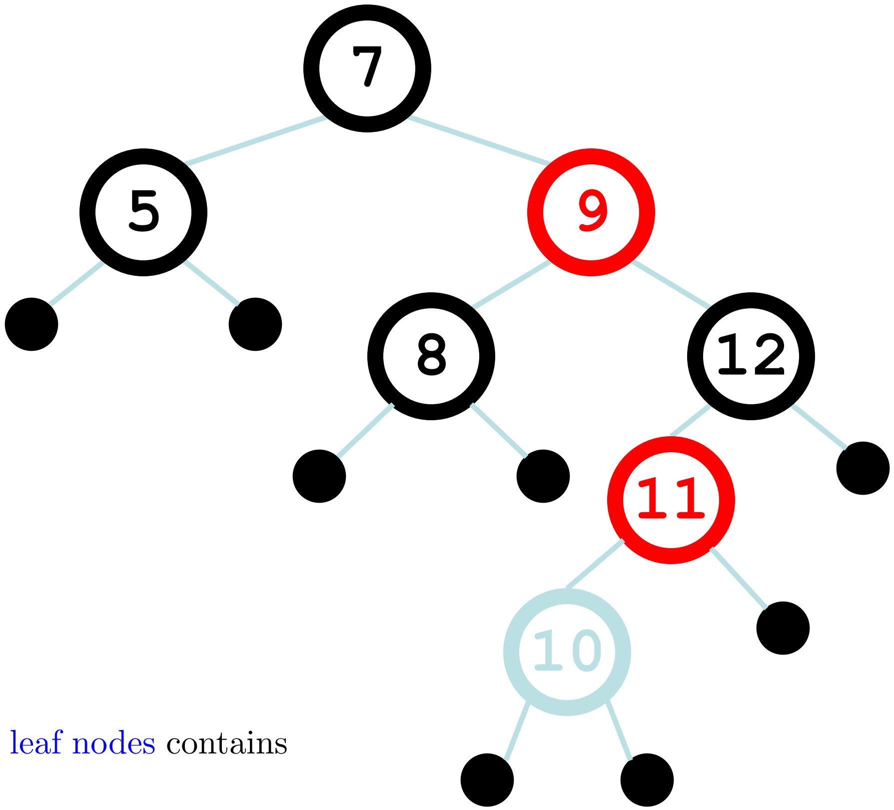
What about INSERT and DELETE?

WHAT SHOULD I COLOR 10?

Can't be RED! (P4)
Can't be BLACK! (P5)

Can I recolour?

NO! The Tree is too unbalanced



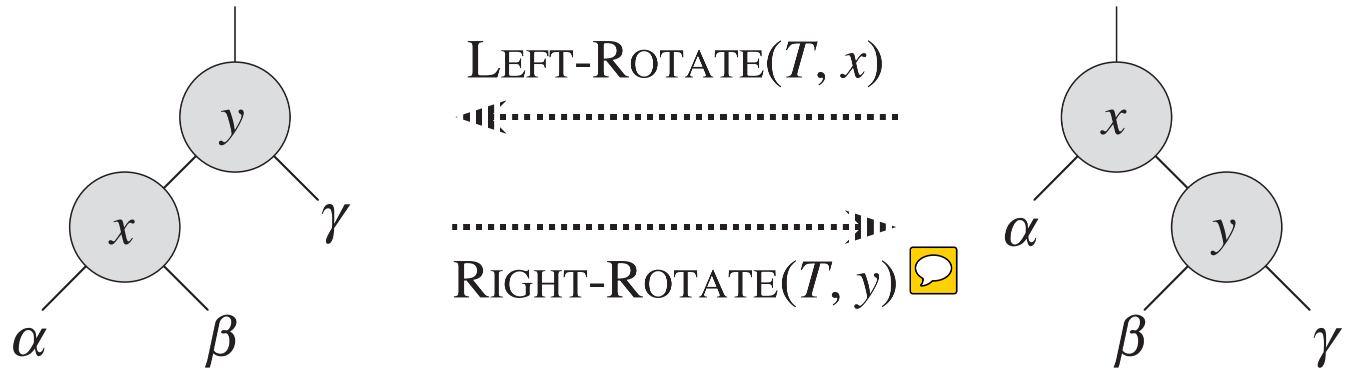
P4. If a node is RED, then both its children are BLACK.

P5. Each path from a node to **ANY** one of its descendant leaf nodes contains the same number of BLACK nodes.

Red Black Trees

ACTION 2. ROTATIONS

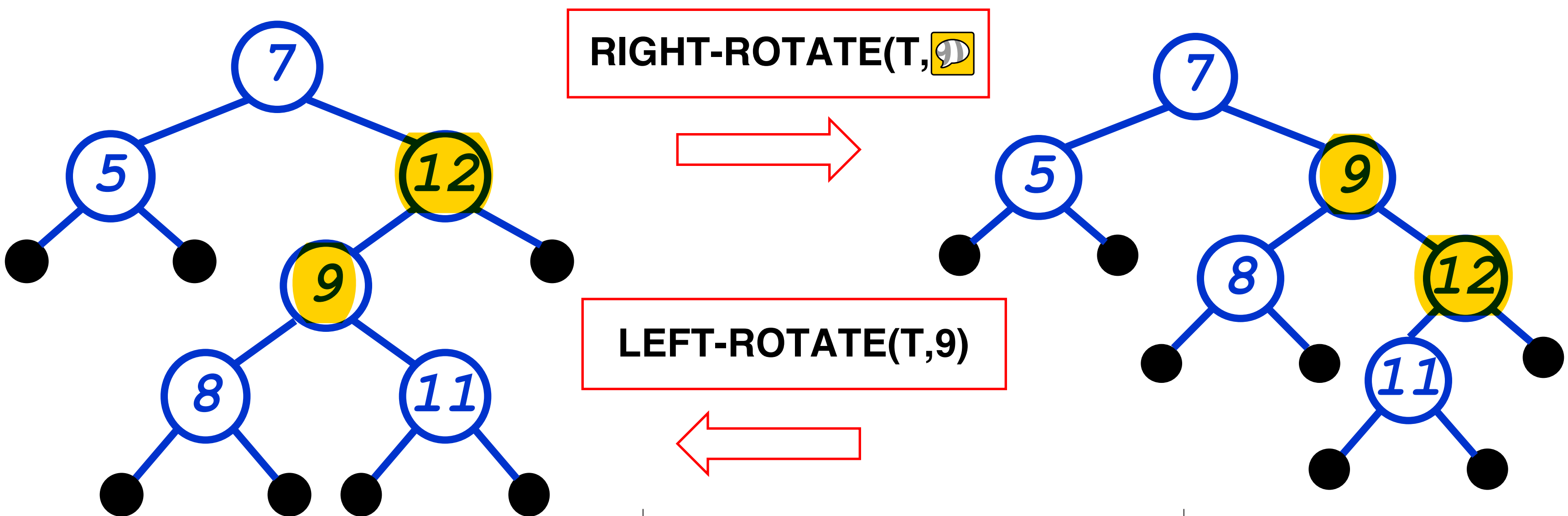
We need to preserve the 5 properties!

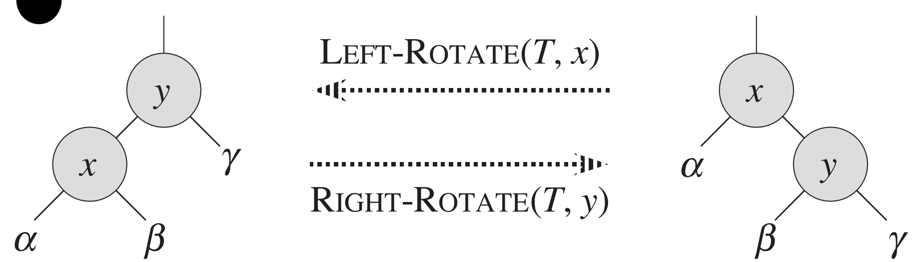


$$keys(\alpha) \leq x.key \leq keys(\beta) \leq y.key \leq keys(\gamma)$$

Red Black Trees

ACTION 2. ROTATIONS

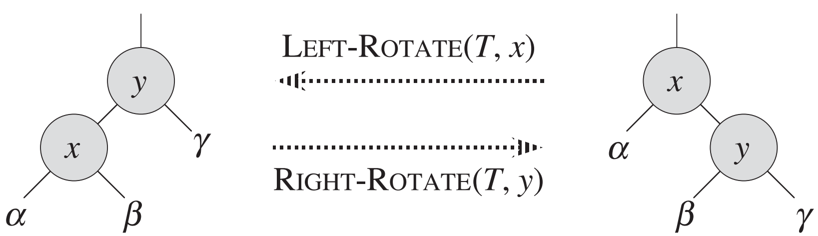




Red Black Trees

ACTION 2. ROTATIONS

```
LEFT-ROTATE(T, x)
  y = x.right                   // set y
  x.right = y.left              // turn y's left subtree into x's right subtree
  if y.left ≠ T.nil
      y.left.p = x
  y.p = x.p                     // link x's parent to y
  if x.p == T.nil
      T.root = y
  elseif x == x.p.left
      x.p.left = y
  else x.p.right = y
  y.left = x                    // put x on y's left
  x.p = y
```



Red Black Trees

What about INSERT?

NEW RULE: The inserted element is initially coloured RED

INSERT might destroy properties P2 or P4

2 fixing actions:

1. RECOULOUR

2. ROTATIONS

Red Black Trees

ACTION 1. RECOLOUR!

CASE 0: z is a RED root

COLOR IT BLACK!

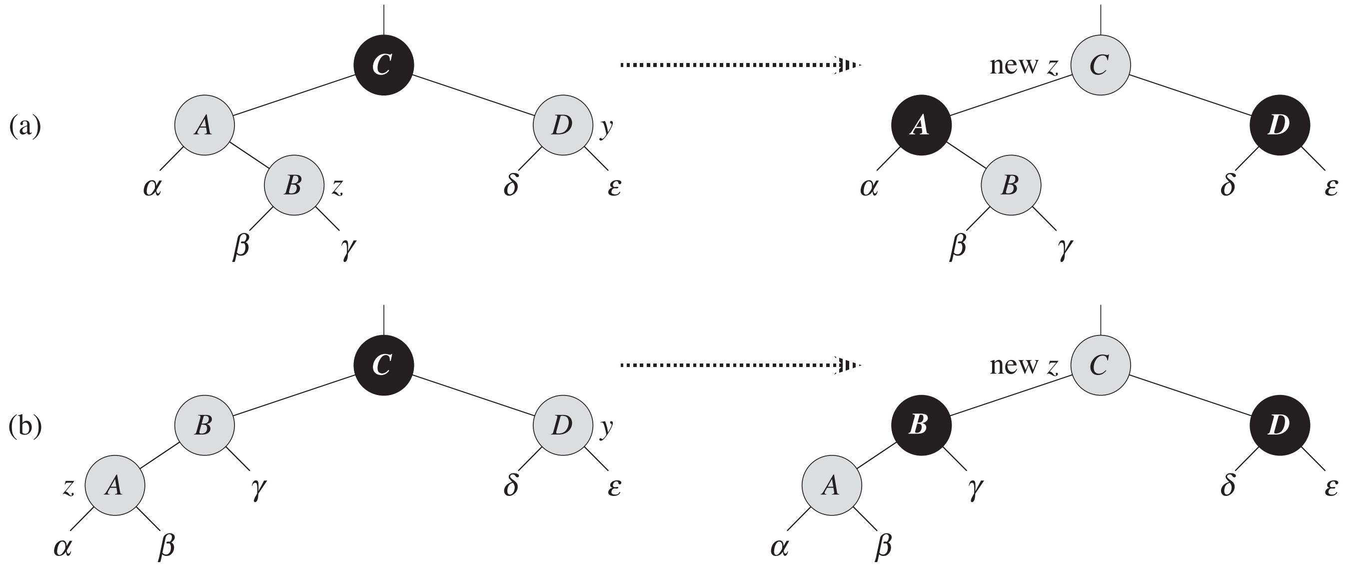
Definitely fixes the problem

Red Black Trees

ACTION 1. RECOLOUR!

We have a RED node z which has a RED parent

CASE 1: z has a RED uncle



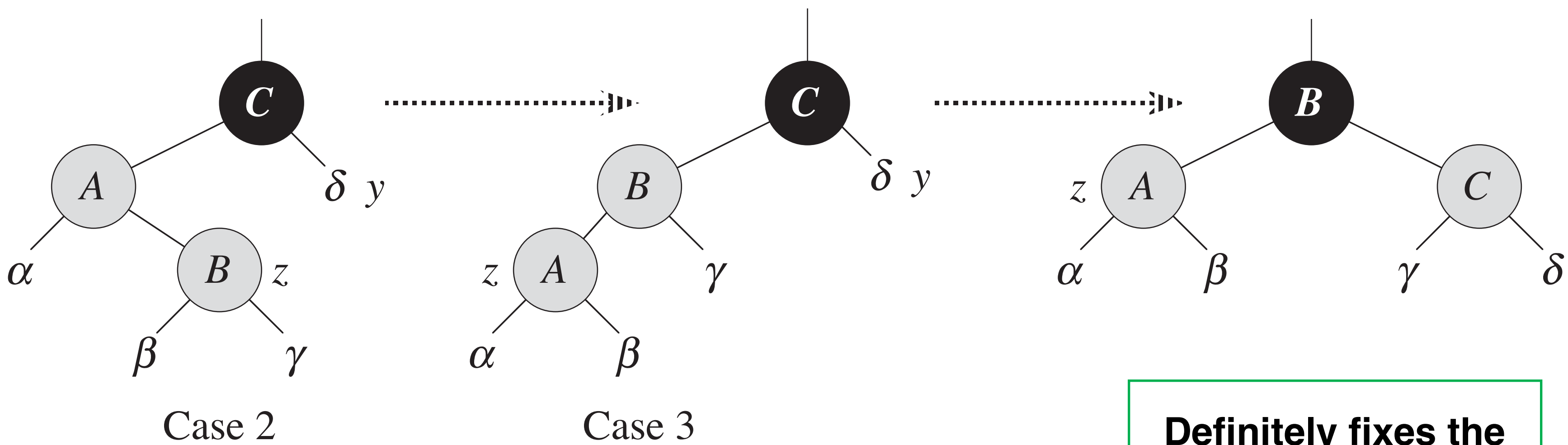
Either fixes the problem or moves it one level up

Red Black Trees

ACTION 2. ROTATION!

We have a RED node z which has a RED parent

CASE 2 ad 3: z has a BLACK uncle



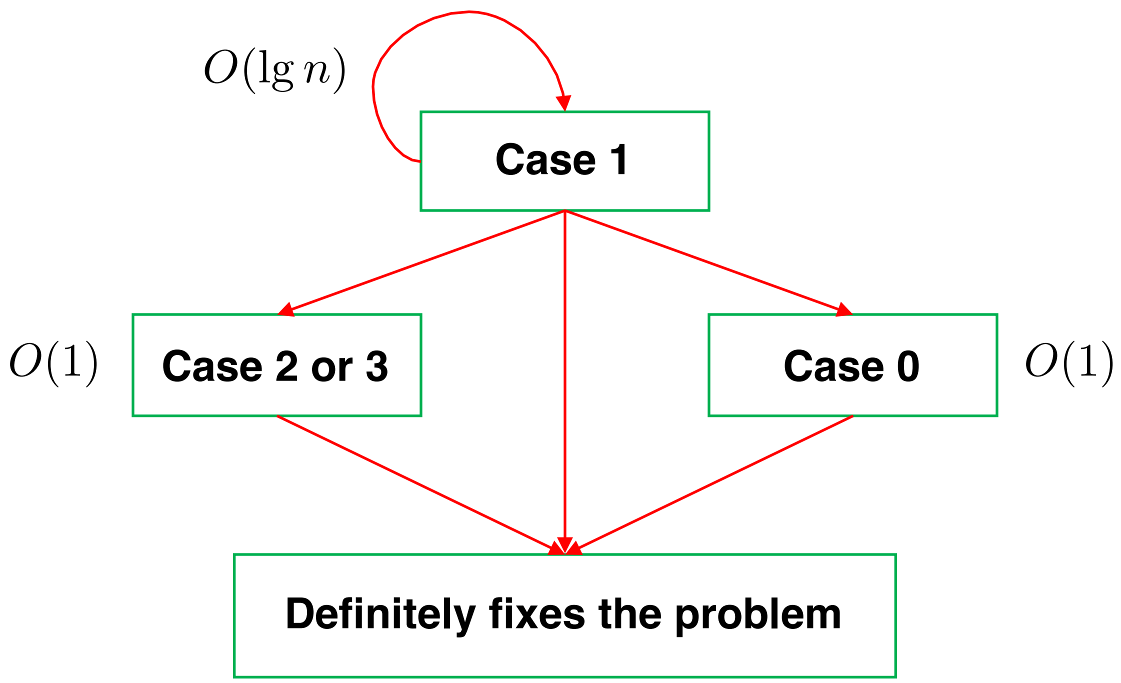
z is a right child

z is a left child

Definitely fixes the problem

Red Black Trees

INSERTION ACTIONS



$$O(\lg n)$$