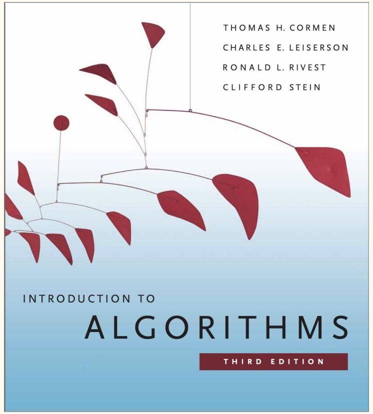
Data Structures and Algorithms

Tutorial 7. Random BST. Red-Black Trees

Today's topic is covered in detail in

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. **Introduction to Algorithms.** The MIT Press 2009.

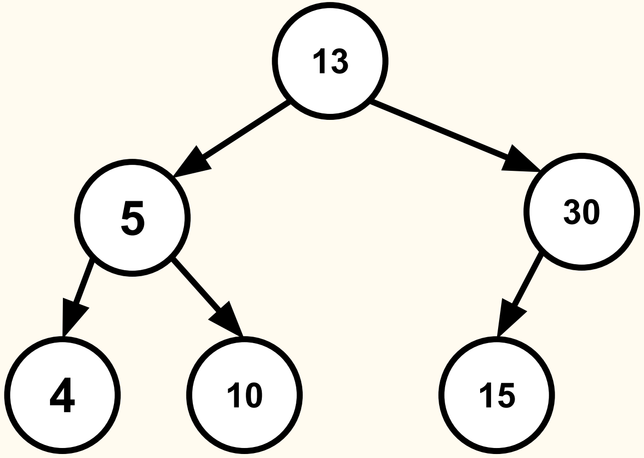


Objectives

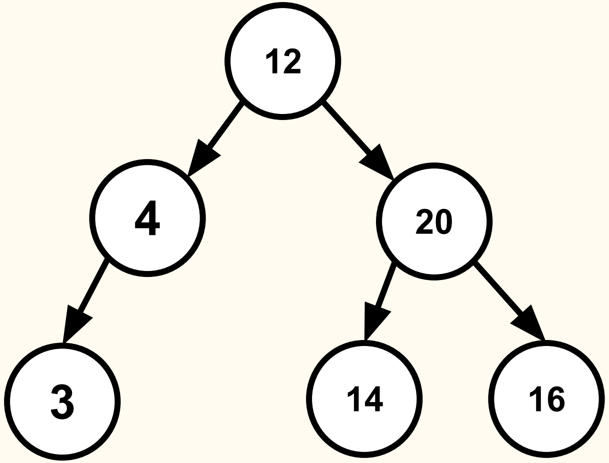
- Recap: binary search tree
- Height of a randomly build BST
- Red-Black Trees: invariant, insertion, deletion

Red-Black Trees: exercise

Exercise 7.0. Which of the following are valid BSTs?



A



B

Height of a randomly built BST

Theorem. The expected height of a randomly built binary search tree on n distinct keys is $O(\log n)$.

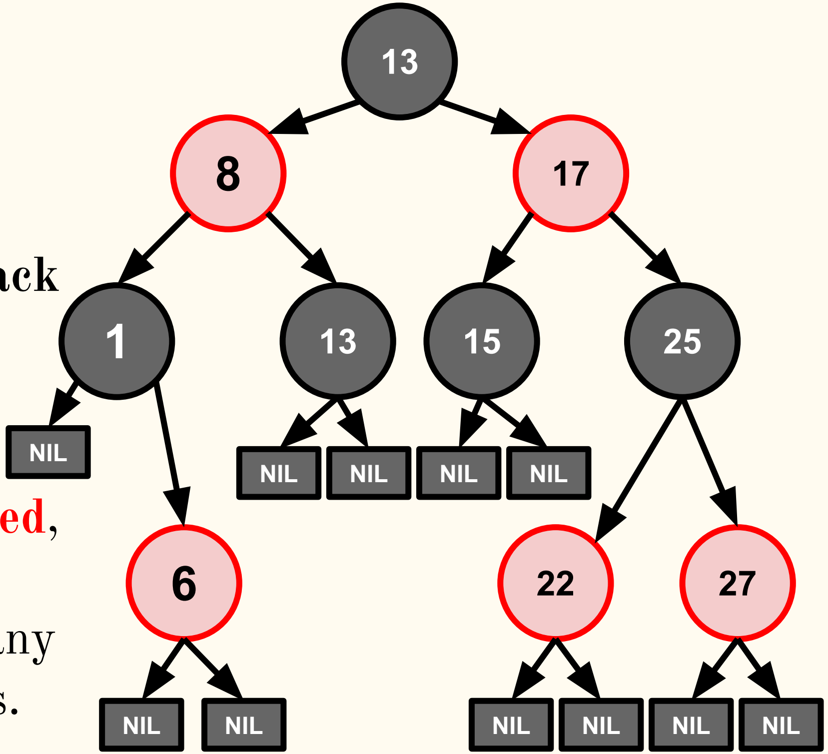
Randomly build BST: exercise

Exercise 7.1. Show that **randomly build** BST on n keys is not the same as **randomly chosen** BST on n key. Hint: consider $n = 3$.

Red-Black Trees: invariant

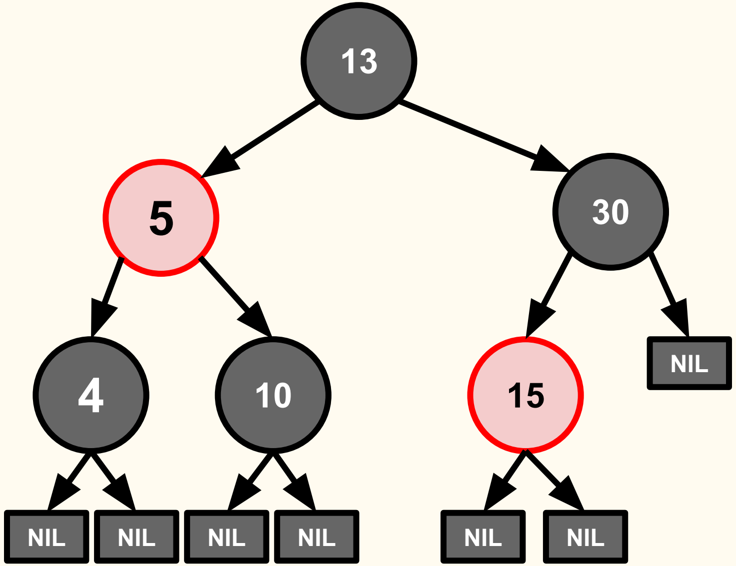
Red-Black Tree is a type of self-balancing BST:

1. Each node is either red or **black** (this information is stored in each node).
2. The root is always **black**.
3. Every leaf (NIL) is **black**.
4. If a node is red, then both its children are **black**.
5. For each node, all paths from this node to any leaf contain the same number of **black** nodes.

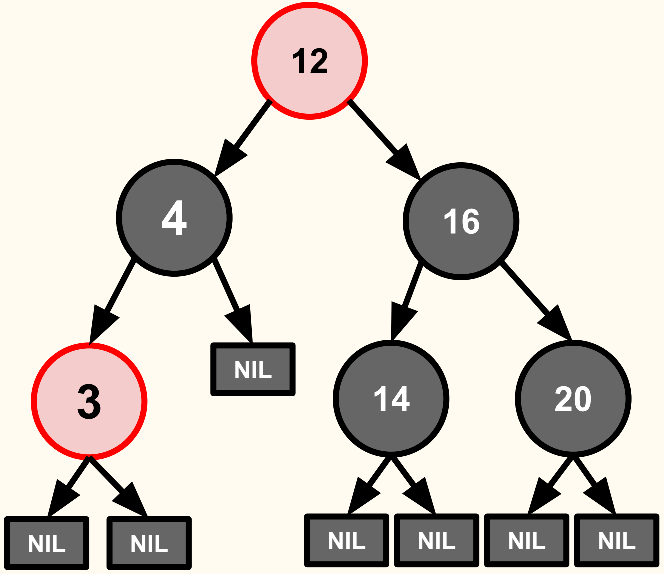


Red-Black Trees: exercise

Exercise 7.1. Which of the following are valid RBTs?



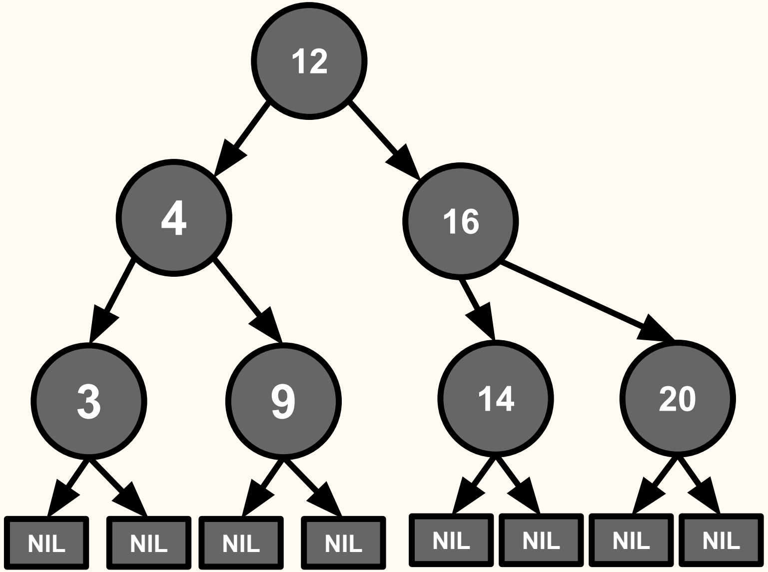
A



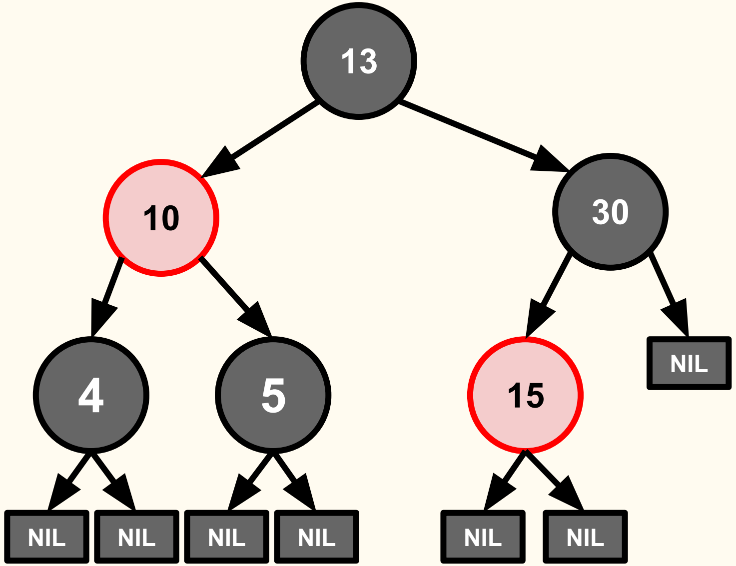
B

Red-Black Trees: exercise

Exercise 7.1. Which of the following are valid RBTs?



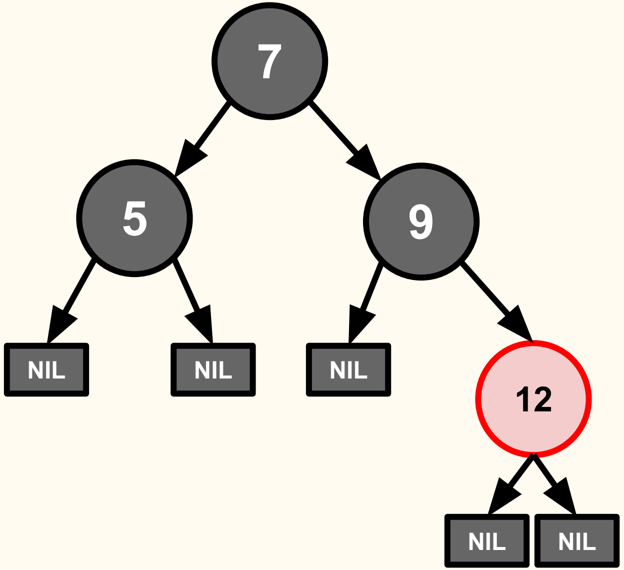
C



D

Red-Black Trees: exercise

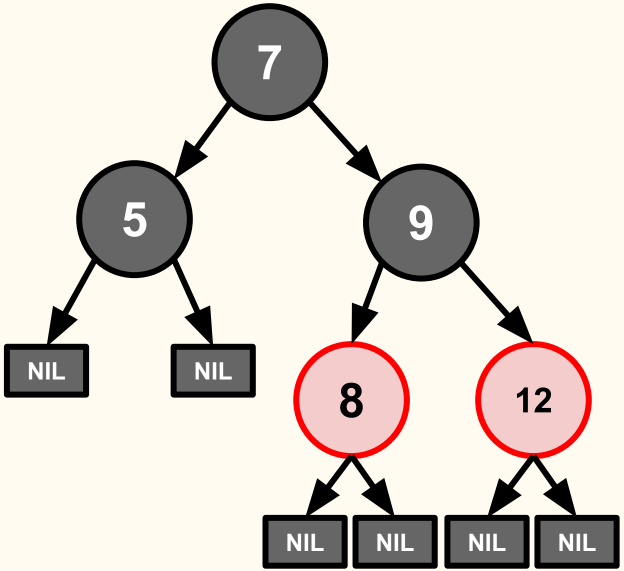
Exercise 7.2.
Insert keys 8, 11, 10 in this RBT. What color should the new nodes have?



Valid RBT

Red-Black Trees: exercise

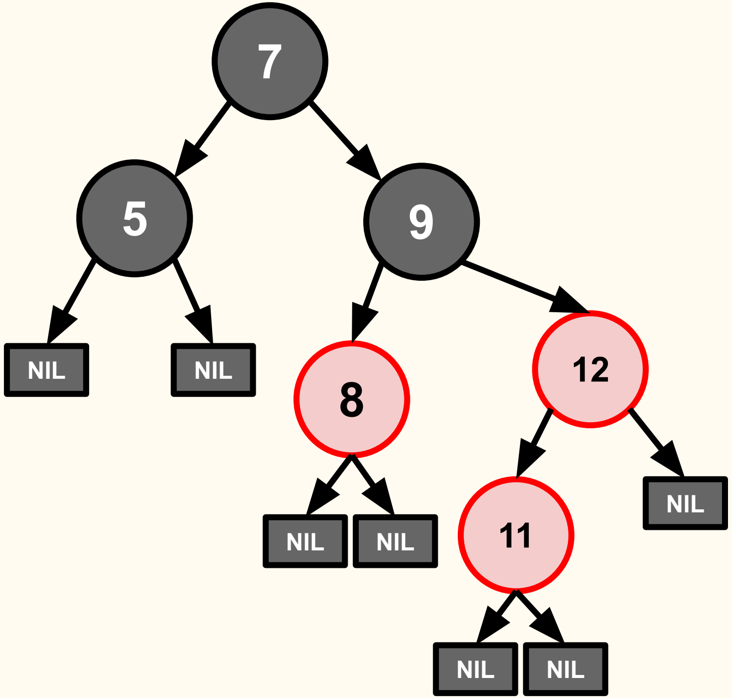
Exercise 7.2.
Insert keys 8, 11, 10 in this RBT. What color should the new nodes have?



Valid RBT

Red-Black Trees: exercise

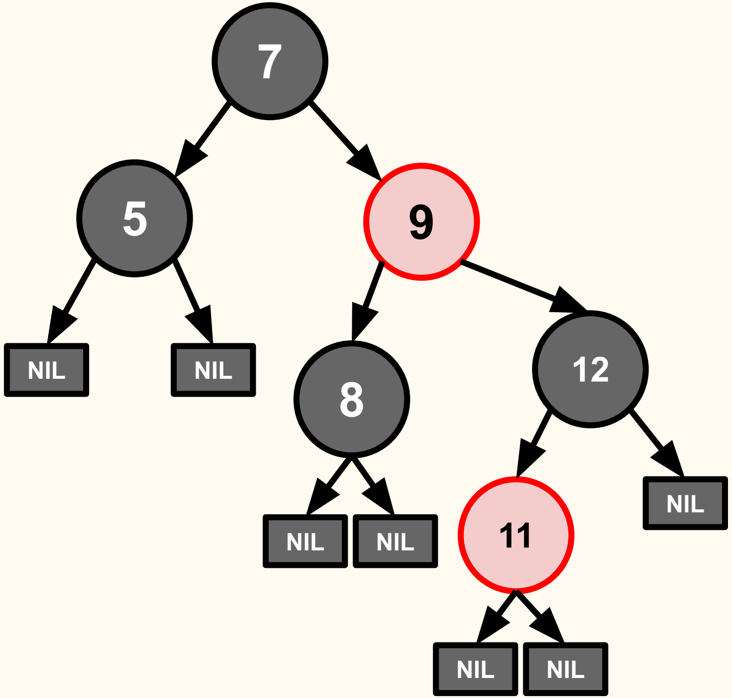
Exercise 7.2.
Insert keys 8, 11, 10 in this RBT. What color should the new nodes have?



Invalid RBT

Red-Black Trees: exercise

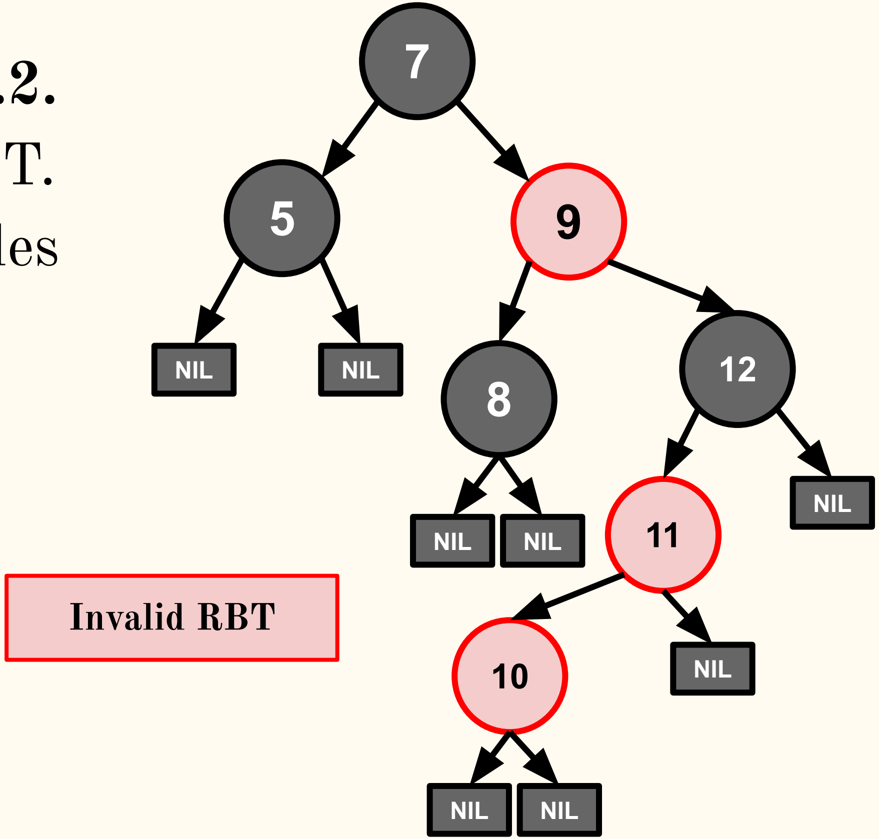
Exercise 7.2.
Insert keys 8, 11, 10 in this RBT. What color should the new nodes have?



Valid RBT

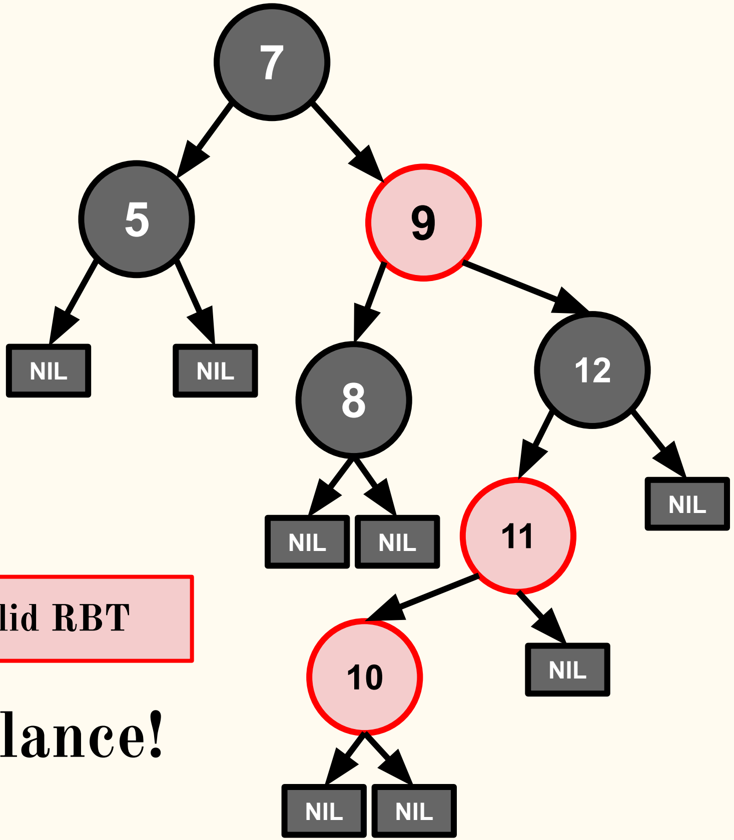
Red-Black Trees: exercise

Exercise 7.2.
Insert keys 8, 11, 10 in this RBT. What color should the new nodes have?



Red-Black Trees: exercise

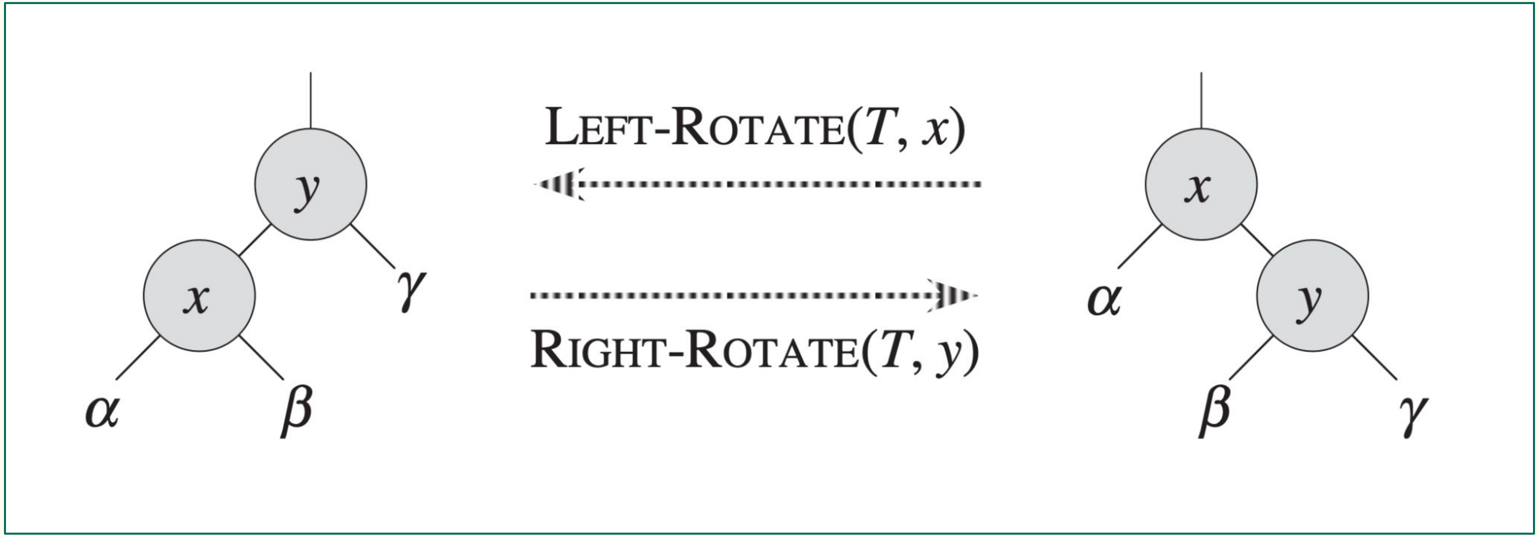
Exercise 7.2.
Insert keys 8, 11, 10 in this RBT. What color should the new nodes have?



Invalid RBT

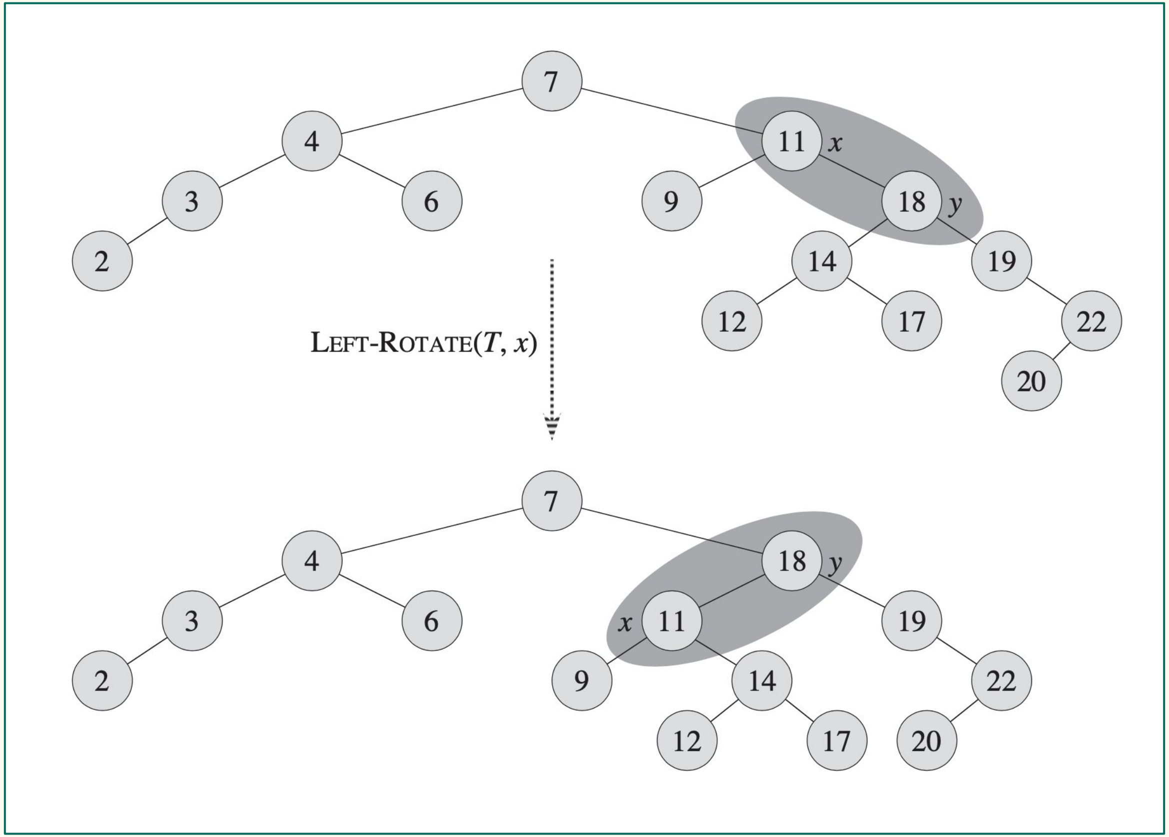
Cannot fix by recoloring. Have to rebalance!

BST rotations



Idea: change the shape of the tree, preserving BST property.

BST rotations: example

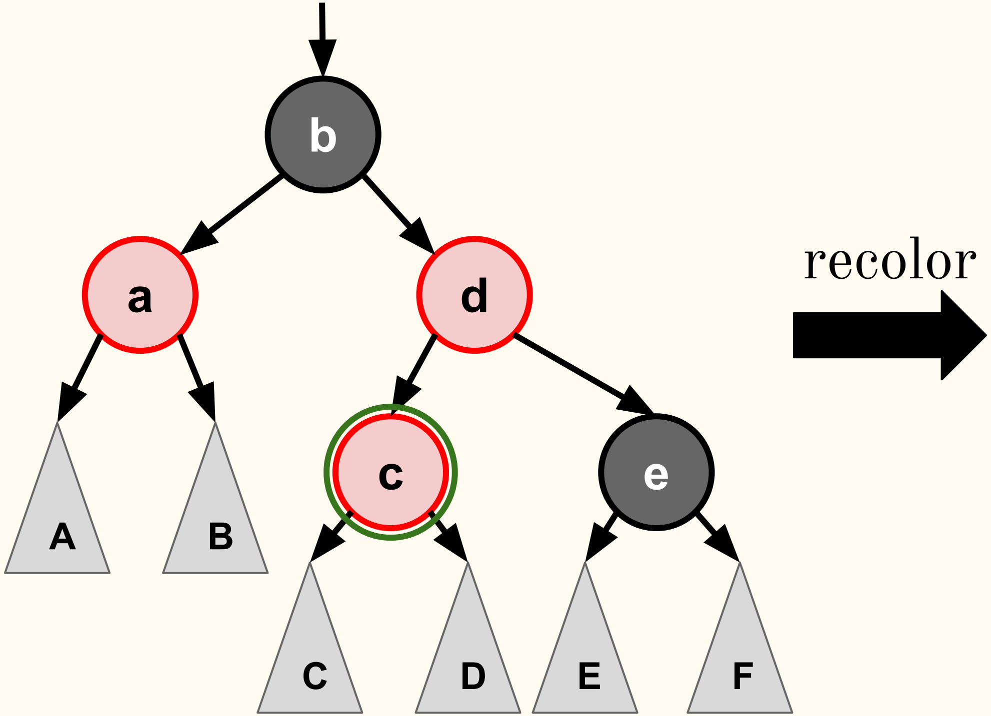


Red-Black Trees: insertion

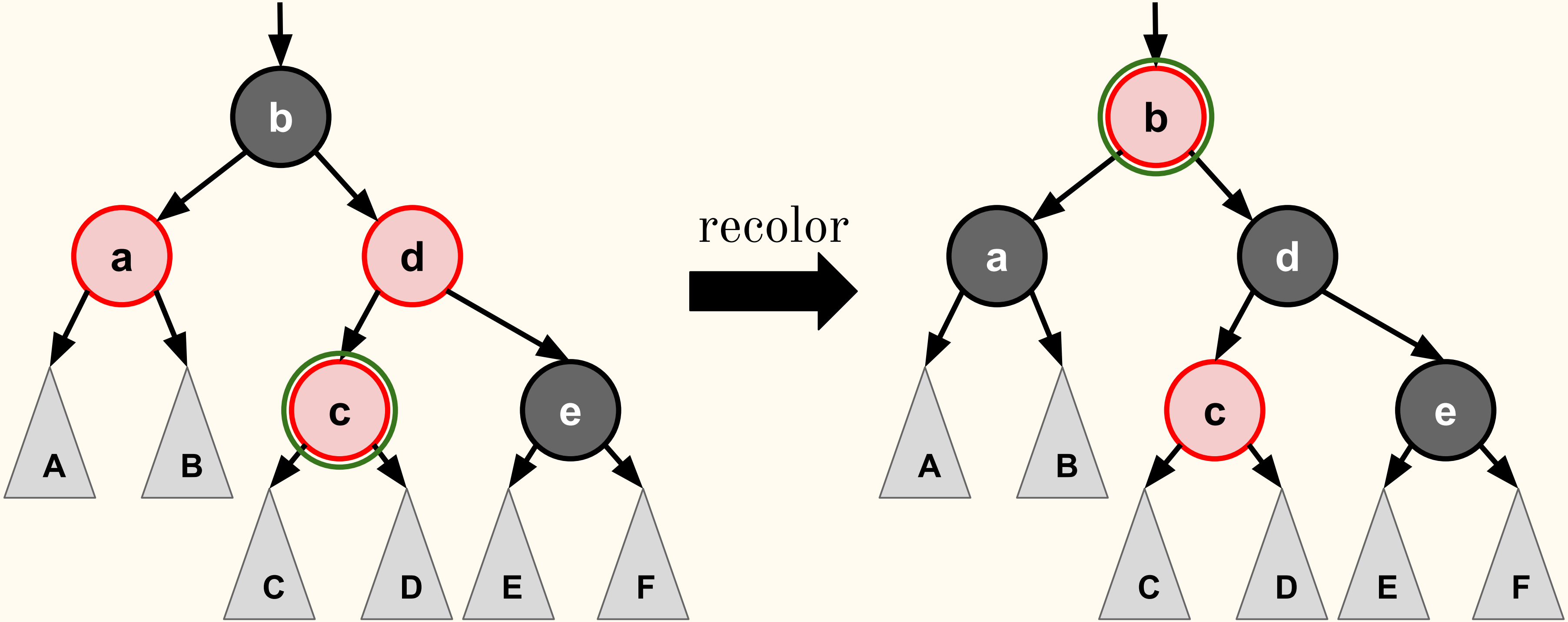
The idea for insertion is simple:

1. Insert a new **red** node using the default BST insertion
2. Fix the tree, recursively raising from the new node:
   a. If current node's parent is **black** — stop
   b. If current node's parent and uncle are **red** — recolor and go up
   c. If current node's parent is **red** and uncle is **black**, then rotate, recolor, and go up

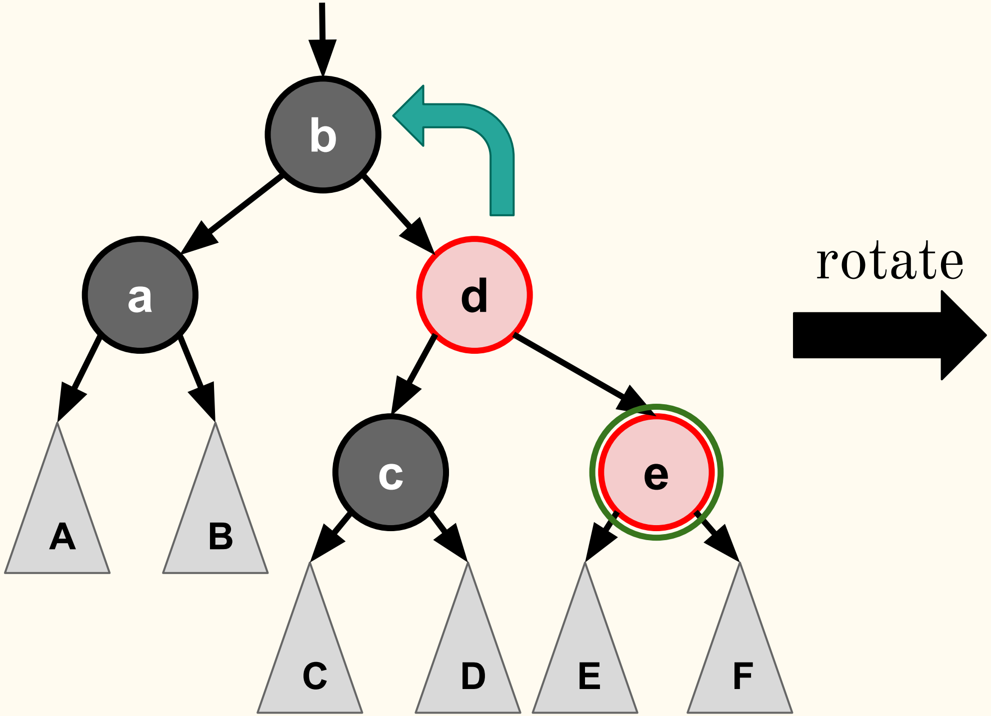
Red-Black Trees: insertion (case 1)



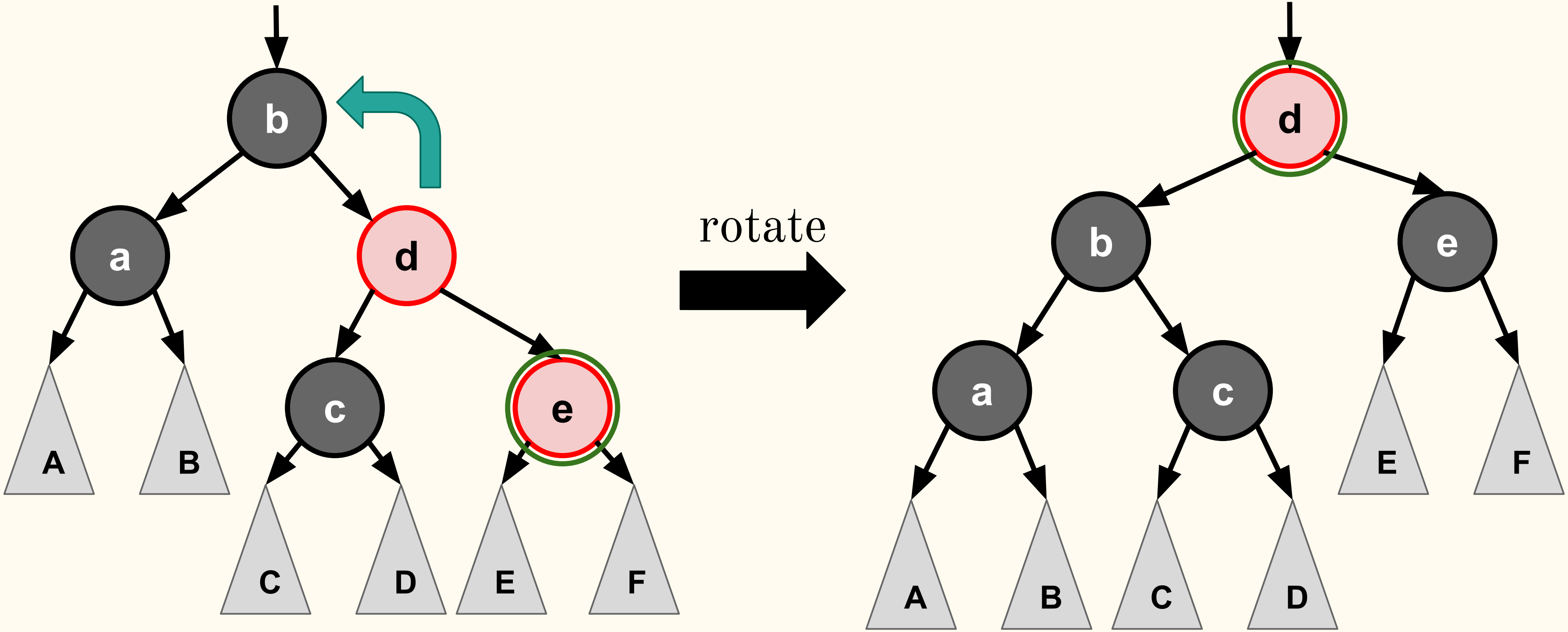
Red-Black Trees: insertion (case 1)



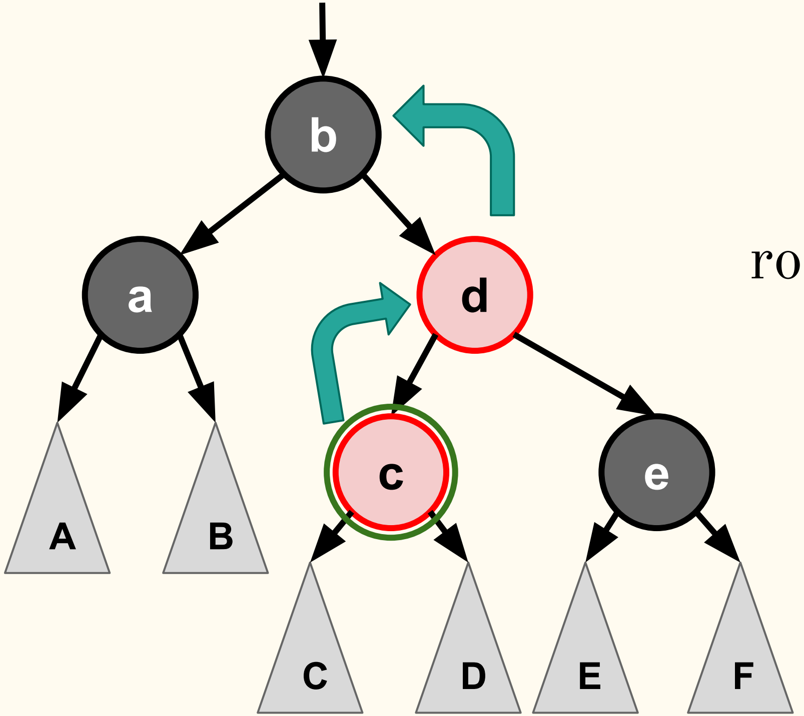
Red-Black Trees: insertion (case 2)



Red-Black Trees: insertion (case 2)

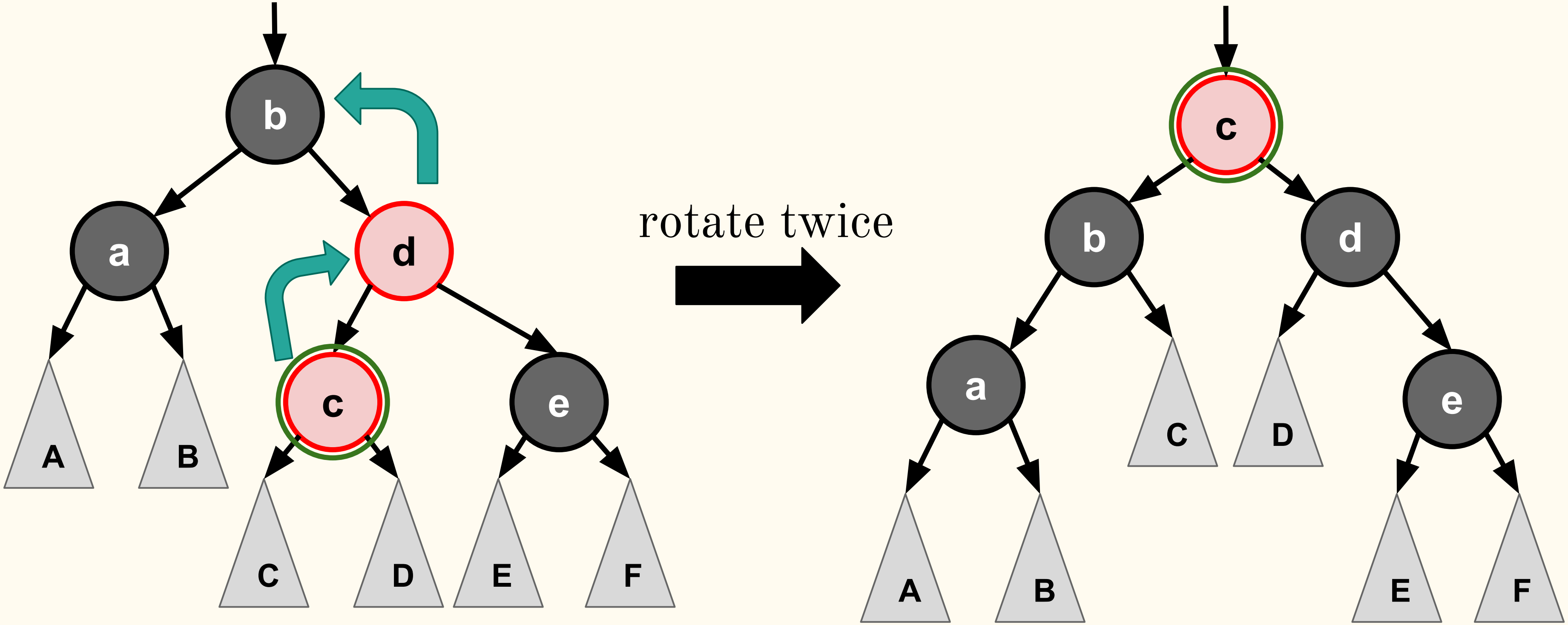


Red-Black Trees: insertion (case 3)



rotate twice

Red-Black Trees: insertion (case 3)



Red-Black Trees: insertion time complexity

The time complexity of insertion into a Red-Black Tree is

?

Red-Black Trees: insertion time complexity

The time complexity of insertion into a Red-Black Tree is

$$O(\log n)$$

Red-Black Trees: insertion (exercise)

Exercise 7.3. Delete these keys in order from the given RBT:

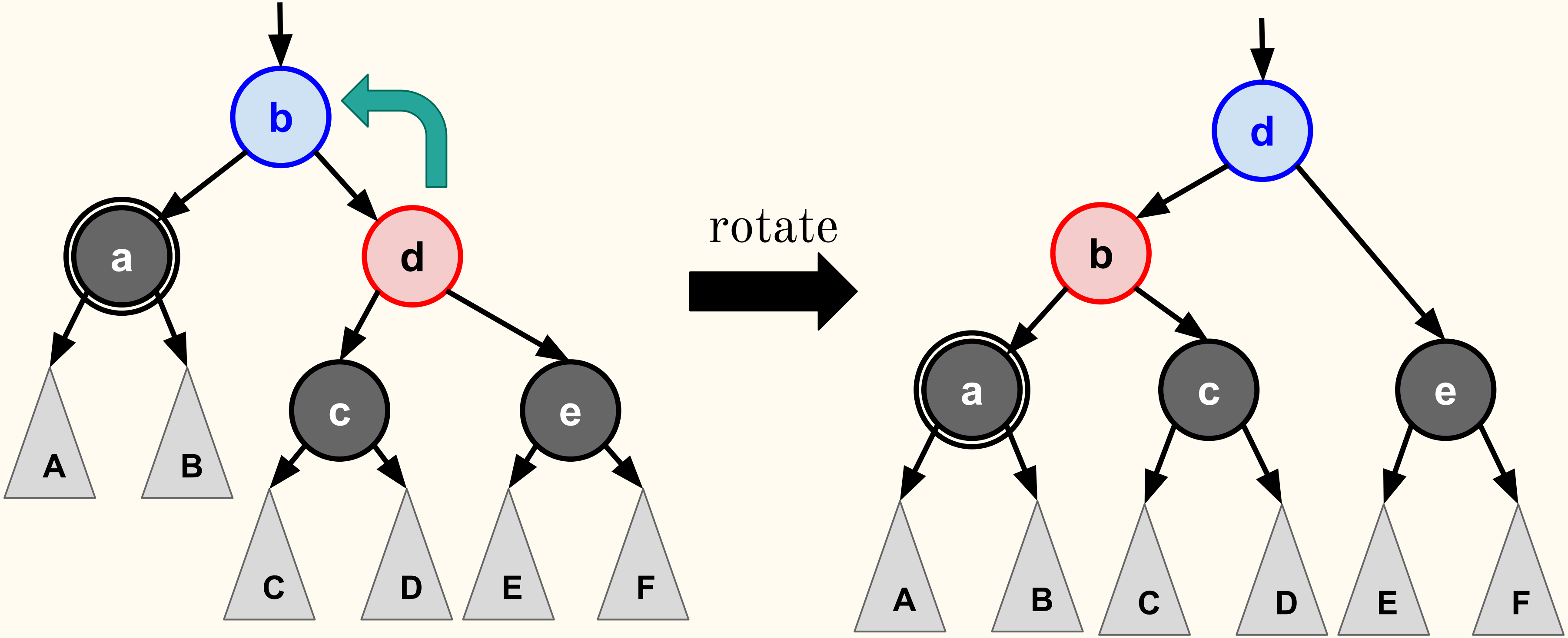
$$8, \quad 12, \quad 19, \quad 31, \quad 38, \quad 41$$

Red-Black Trees: deletion

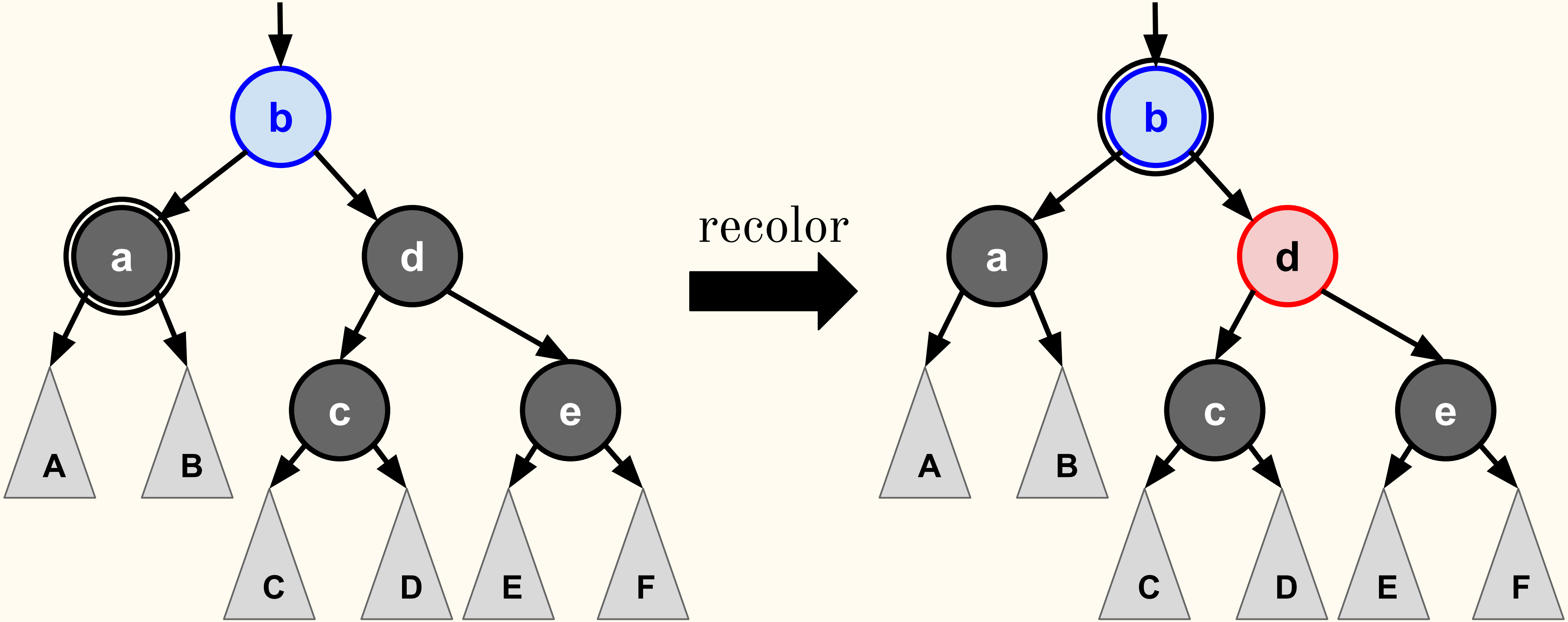
The idea for deletion is simple:

1. Delete a node using the default BST deletion
2. If deleted leaf node was **red**, nothing has to be adjusted.
3. Otherwise, fix the tree, recursively raising from the deleted node:
   a. If current node is **red** — stop
   b. Examine current node's sibling and nephews to perform rotation and recoloring correspondingly

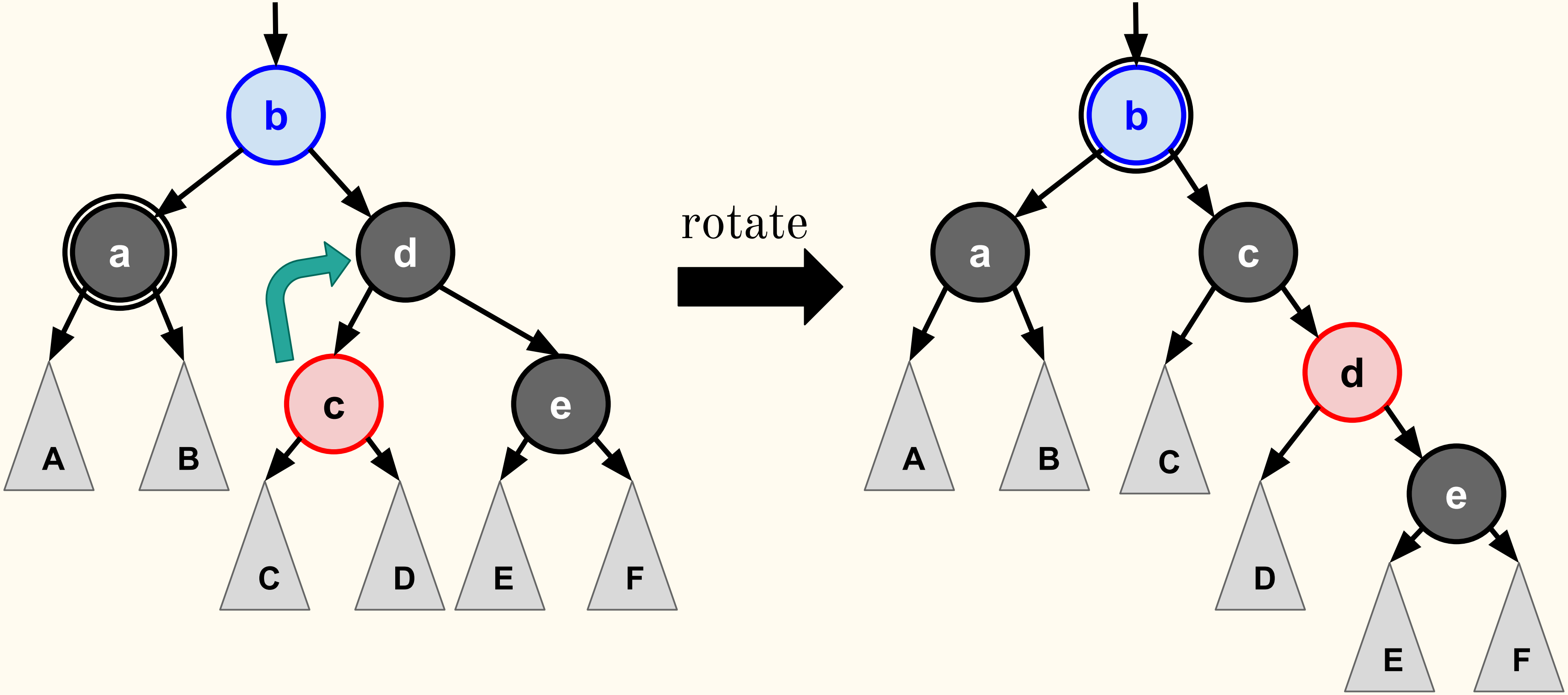
Red-Black Trees: deletion (case 1)



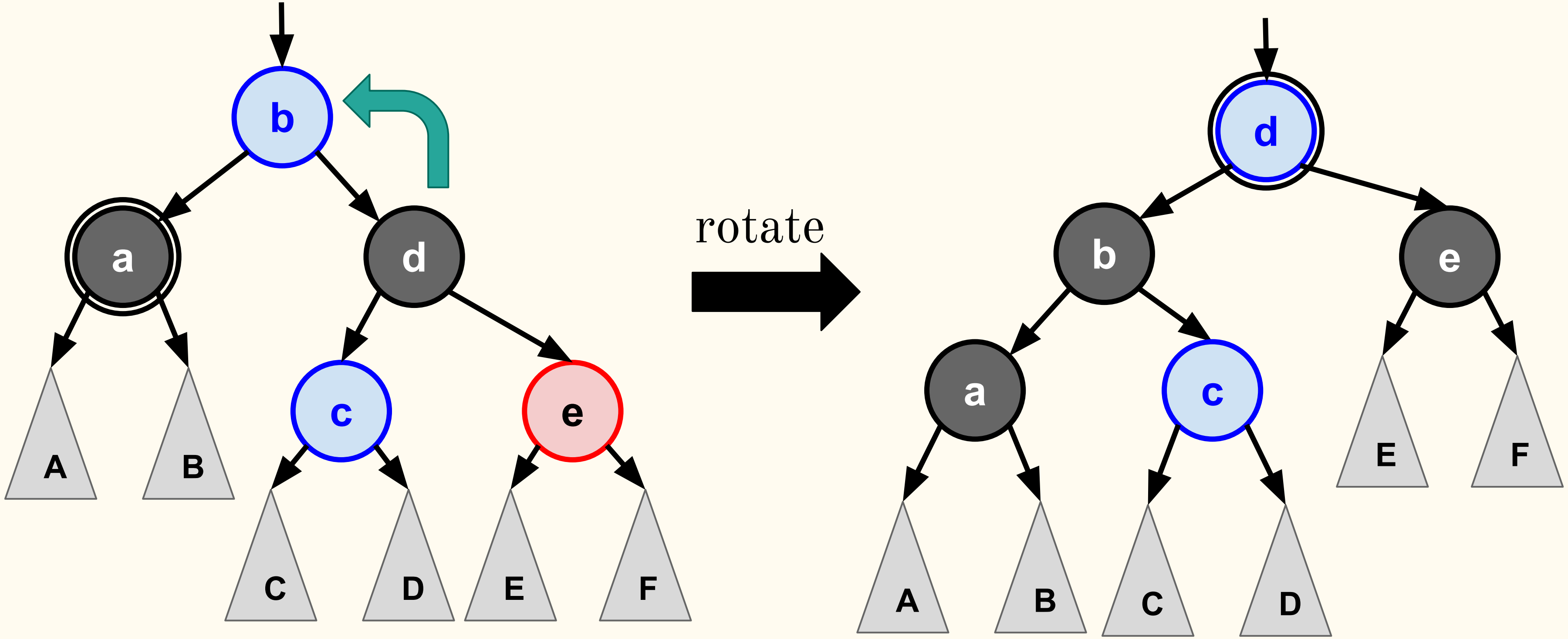
Red-Black Trees: deletion (case 2)



Red-Black Trees: deletion (case 3)



Red-Black Trees: deletion (case 4)



Red-Black Trees: deletion time complexity

The time complexity of deletion from a Red-Black Tree is

?

Red-Black Trees: deletion time complexity

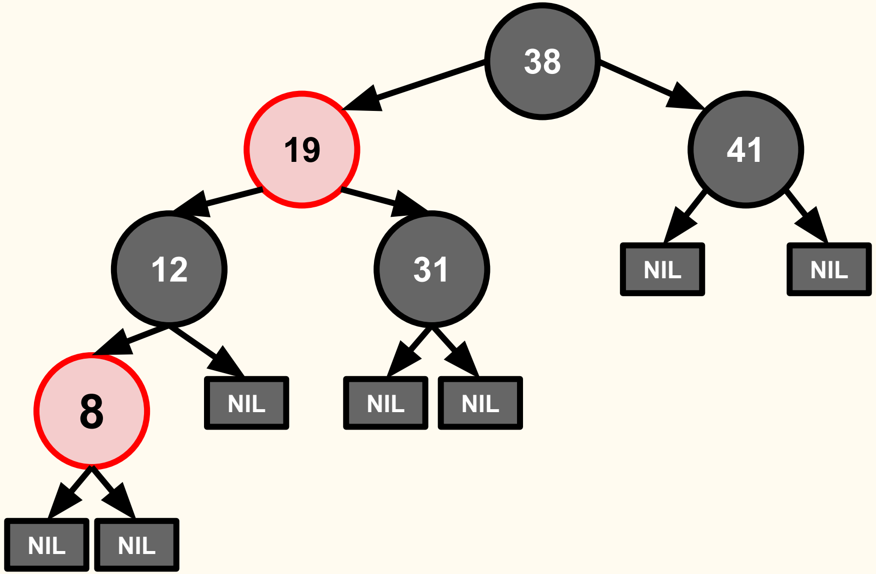
The time complexity of deletion from a Red-Black Tree is

$$O(\log n)$$

Red-Black Trees: deletion (exercise)

Exercise 7.4. Build a RBT by inserting these keys in order:

$$11, \quad 19, \quad 8, \quad 16, \quad 17, \quad 31, \quad 26, \quad 41, \quad 61$$



Summary

- Randomly build BST has expected height of $\Theta(\log n)$
- Red-Black tree is a kind of self-balancing BST
  - Height of an RBT is $O(\log n)$
  - Insertion into an RBT takes $O(\log n)$
  - Deletion from an RBT takes $O(\log n)$

Summary

- Randomly build BST has expected height of $\Theta(\log n)$
- Red-Black tree is a kind of self-balancing BST
  - Height of an RBT is $O(\log n)$
  - Insertion into an RBT takes $O(\log n)$
  - Deletion from an RBT takes $O(\log n)$

See you next week!