Chapter 5, Binary search trees:

- Binary search trees; the balanced BST problem (fall-break eve; finished last week Friday)
- AVL trees (last week Friday and this week Monday)
- Traditional red-black trees (**Today**)
- Left-leaning red-black trees (Friday)
- "Wrap-up" BST (next week Monday)

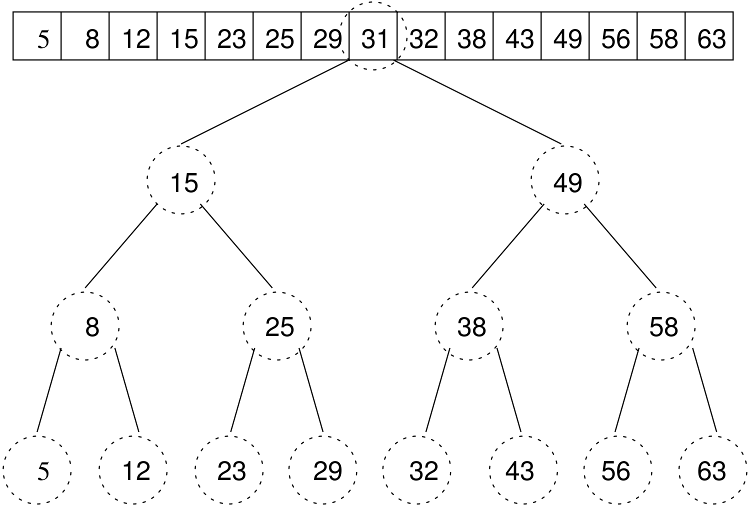
Today:

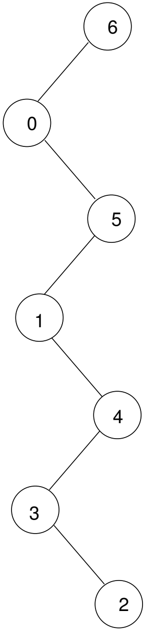
- Red-black trees in context
- Definition and examples
- Codebase details
- Cases for put-fixup
- Analysis

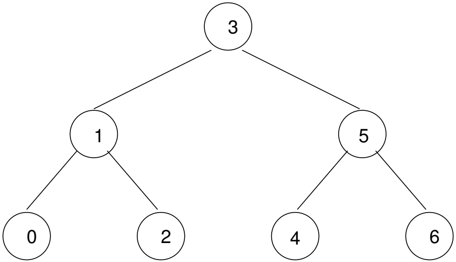
https://www.beyourownbirder.com

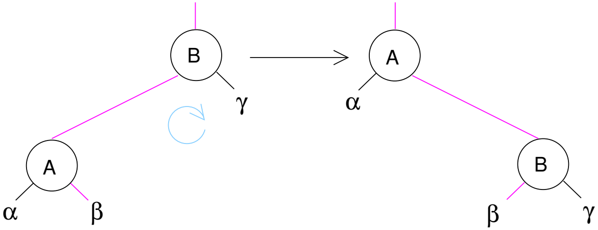
TRADITIONAL Red-Black Trees

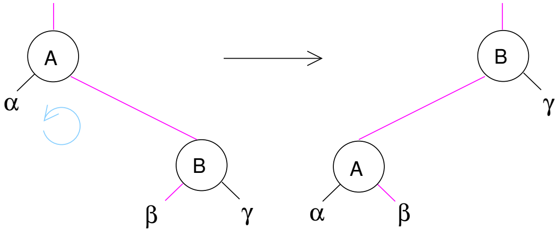
Image swiped from http://www.cheeseandchickpeas.com/tomato-halloumi-salad/

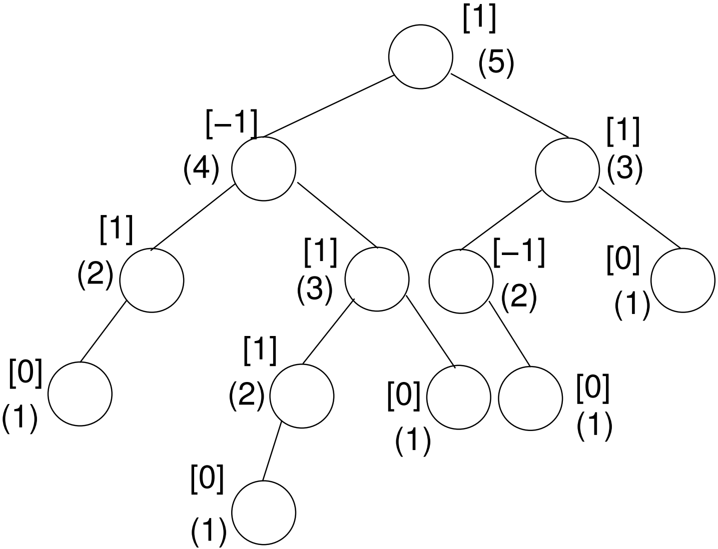

5 8 12 15 23 25 29 31 32 38 43 49 56 58 63
15
49
8
25
38
58
5
12
23
29
32
43
56
63


6
0
5
1
4
3
2

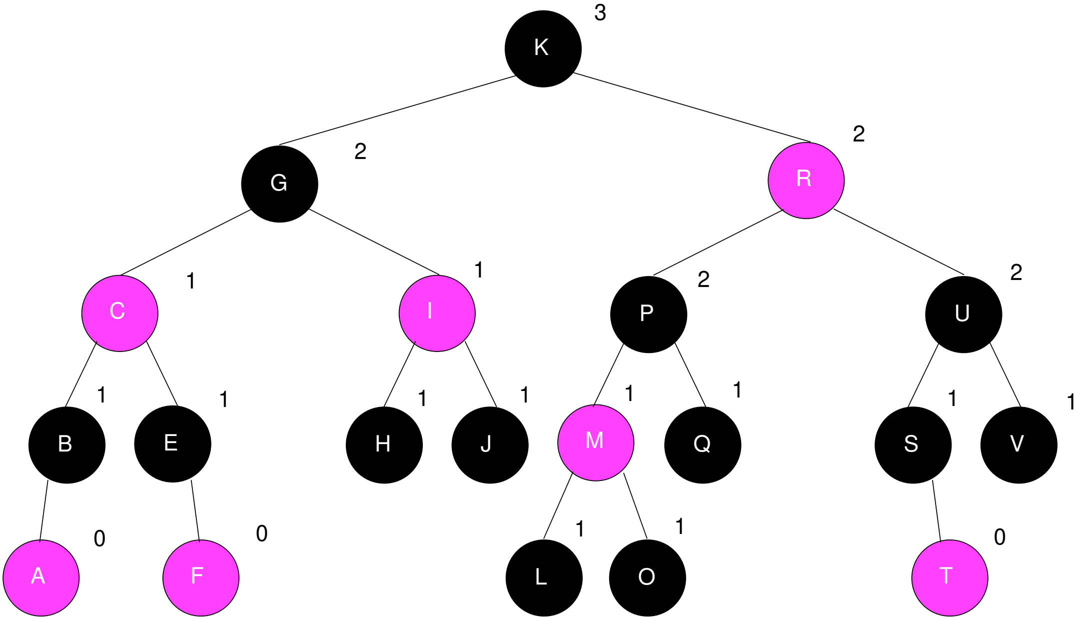

3
1
5
0
2
4
6

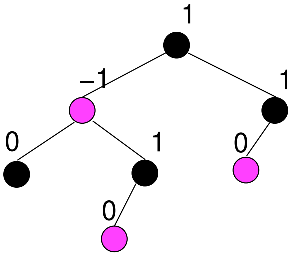

B
γ
A
α
β
A
α
B
β
γ

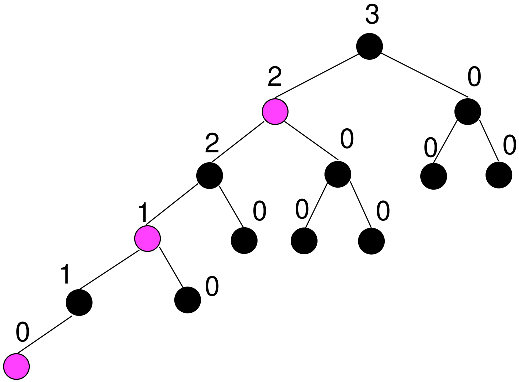

A
α
B
β
γ
B
γ
A
α
β



[1]
(5)
[–1]
(4)
[1]
(3)
[1]
(2)
[1]
(3)
[–1]
(2)
[0]
(1)
[0]
(1)
[1]
(2)
[0]
(1)
[0]
(1)
[0]
(1)

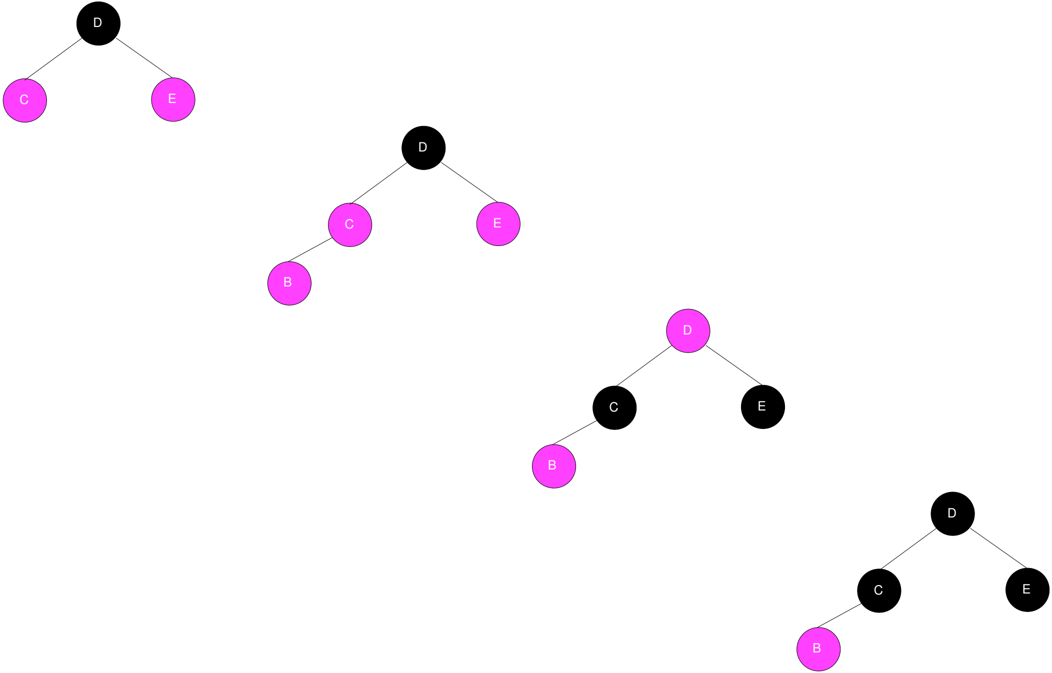

K
3
G
2
R
2
C
1
I
1
P
2
U
2
B
1
E
1
H
1
J
1
M
1
Q
1
S
1
V
1
A
0
F
0
L
1
O
1
T
0

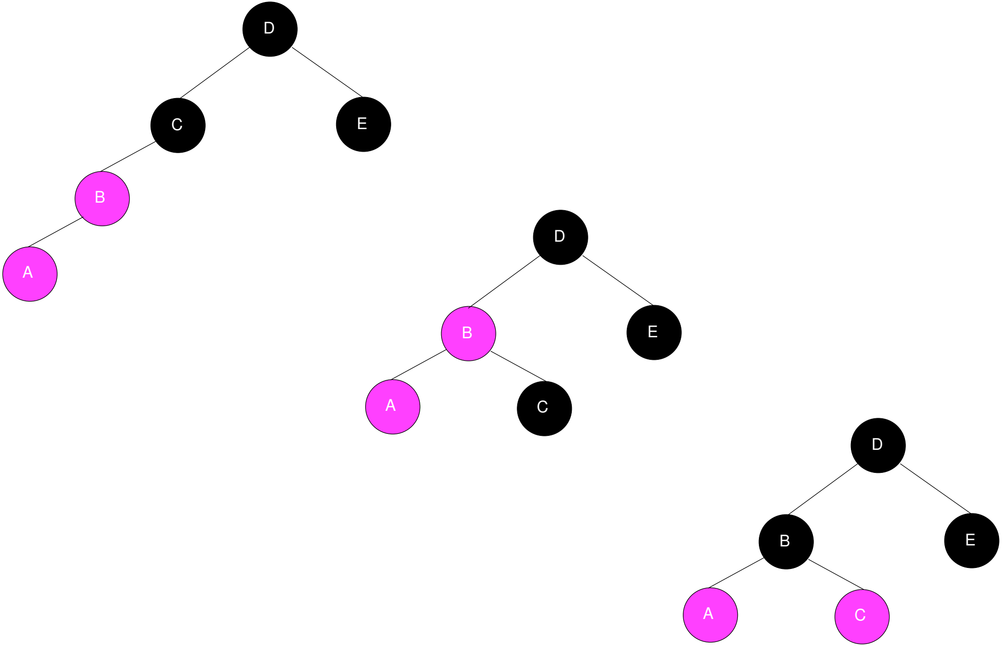

1
−1
1
0
1
0
0


3
2
0
2
0
0
0
1
0
0
0
1
0
0

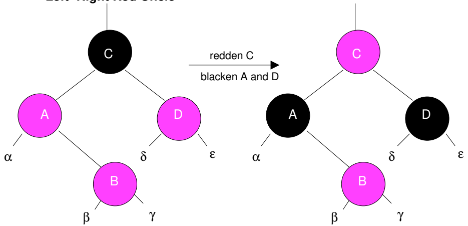
A red-black tree is a binary tree (usually a BST) that is either empty or it is rooted at node T such that

- T is either red or black.
- Both of T's children are roots of red-black trees.
- If T is red, then both its children are black.
- The red-black trees rooted at its children have equal blackheight; moreover, the blackheight of the tree rooted at T is one more than the blackheight of its children if T is black or equal to that of its chidren if T is red.

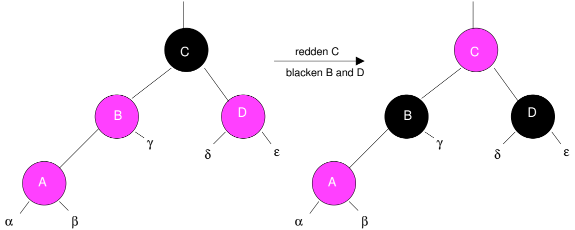

D
C
E
D
C
E
B
D
C
E
B
D
C
E
B

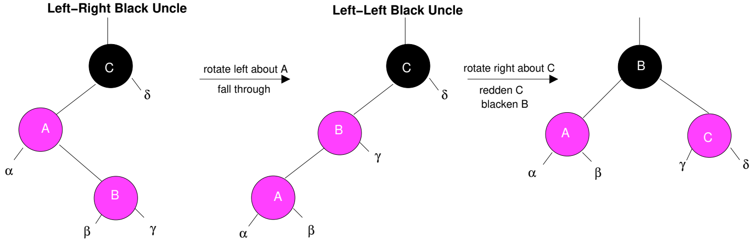

D
C
E
B
A
D
B
E
A
C
D
B
E
A
C

Left–Right Red Uncle

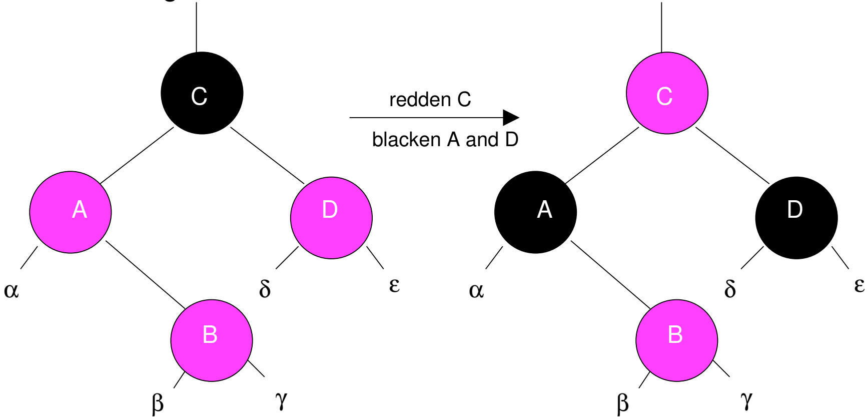


Left–Left Red Uncle

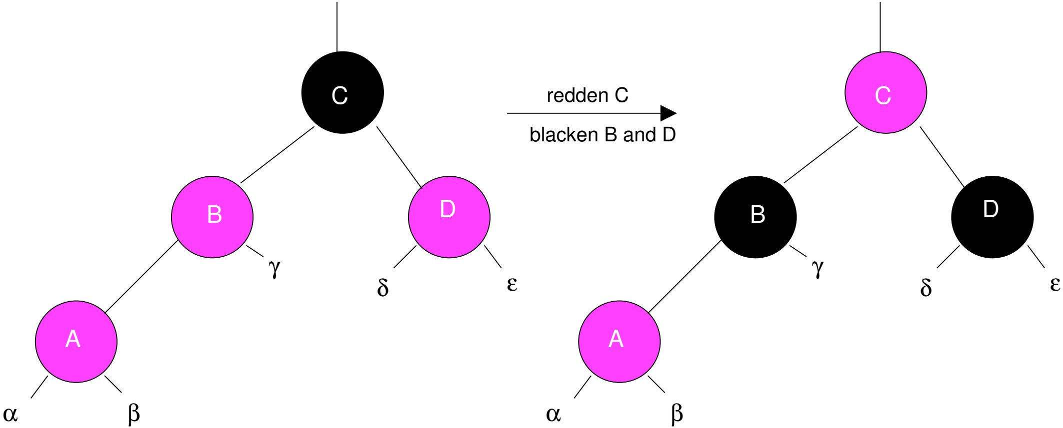


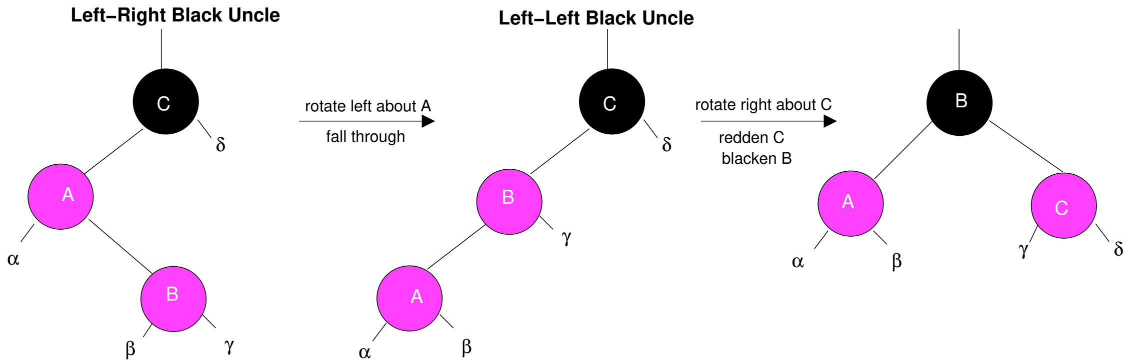


Left–Right Red Uncle



Left–Left Red Uncle




Left–Right Black Uncle
C
δ
A
α
B
β
γ
rotate left about A
fall through
Left–Left Black Uncle
C
δ
B
γ
A
α
β
rotate right about C
redden C
blacken B
B
A
α
β
C
γ
δ

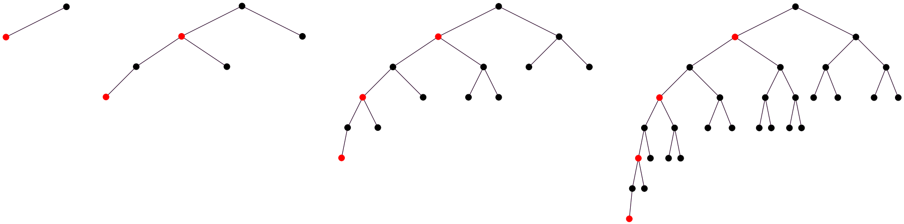
Invariant 26 (Postconditions of RealNode.put() with TradRBBalancer.) Let x be the root of a subtree on which `put()` is called and let y be the node returned, that is, the root of the resulting subtree.

(a) The subtree rooted at y has a consistent black height.

(b) The black height of subtree rooted at y is equal to the original black height of the subtree rooted at x.

(c) The subtree rooted at y has no double-red violations except, possibly, both y and one of its children is red.

Blackheight	1	2	3	4
Height	2	4	6	8
Nodes	2	6	14	30



AVL trees	(Traditional) red-black trees
$h \leq 1.44 \lg n$	$h \leq 2 \lg(n+2) - 2$
The difference between the longest routes to leaves in the two subtrees is no greater than 1.	The longest route to any leaf is no greater than twice the shortest route to any leaf.
Stronger constraint, more aggressive rebalancing, more balanced tree, more work spent rebalancing.	Looser constraint, less aggressive rebalancing, less balanced tree, less work spent rebalancing.

Coming up:

Do **BST rotations** *project (due Mon, Oct 27)—nothing to turn in*
Do **AVL trees** *project (due Fri, Oct 31)*
Do **Traditional RB** *project (due Wed, Nov 5)*

Due **Mon, Nov 3** *(end of day)—but spread it out*
Read Sections 5.(4-6)
Do Exercise 5.13
Take quiz (red-black trees)