

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 (Wednesday, finish **Today**)
- ▶ Left-leaning red-black trees (**Today**)
- ▶ “Wrap-up” BST (next week Monday)
- ▶ Begin dynamic programming (next week Wednesday)

Today:

- ▶ Lab retrospective
- ▶ Finishing Traditional RB
- ▶ LLRB context and definition
- ▶ LLRB invariant and cases
- ▶ Performance comparison among AVL, TrRB, and LLRB

Why invariants?

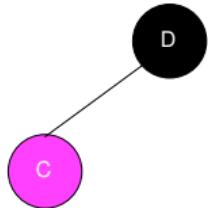
- ▶ An invariant is a constraint we put on our code to help us guarantee something about it.
- ▶ The general invariant for BSTs guarantees the correctness of our find algorithm.
- ▶ The invariants for AVL trees and RB trees guarantee logarithmic-time operations.

A stronger constraint is both a stronger constraint to *maintain* and a stronger constraint to *assume*.

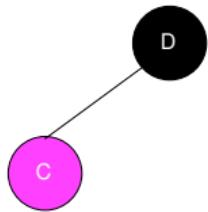
A **left-leaning** red-black tree is a binary tree (usually a BST) that is either empty or it is rooted at node T such that

- ▶ T is either red or black.
- ▶ Both of T 's children are roots of **left-leaning** red-black trees.
- ▶ T 's right child is black.
- ▶ If T is red, then its left child is black.
- ▶ The **left-leaning** 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 children if T is red.

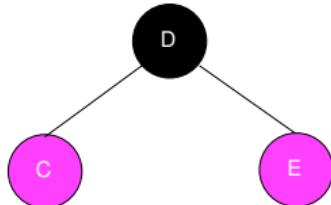
Left-leaning



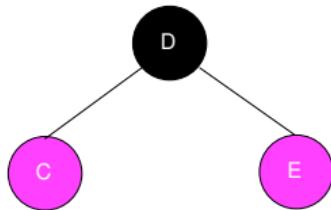
Traditional



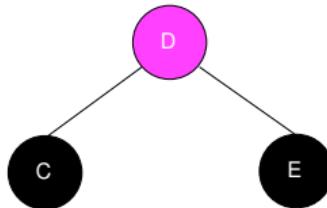
Left-leaning



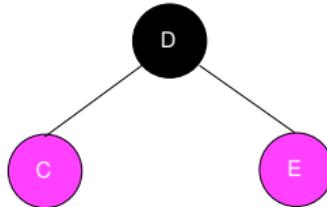
Traditional



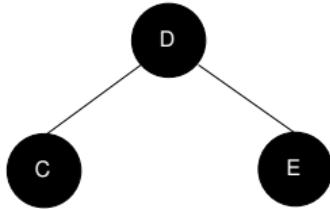
Left-leaning



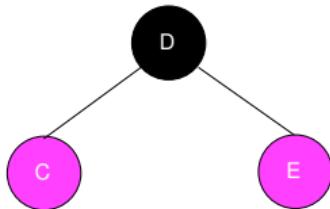
Traditional



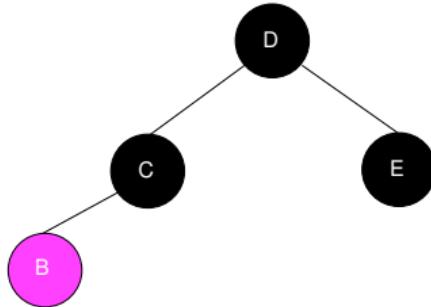
Left-leaning



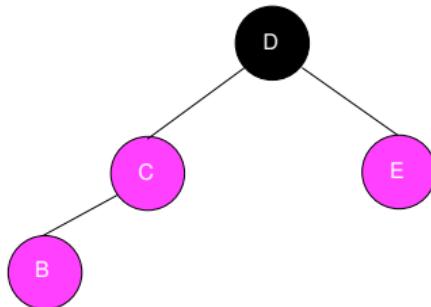
Traditional



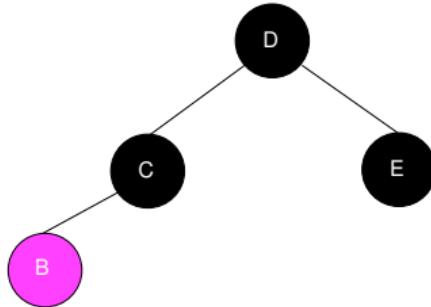
Left-leaning



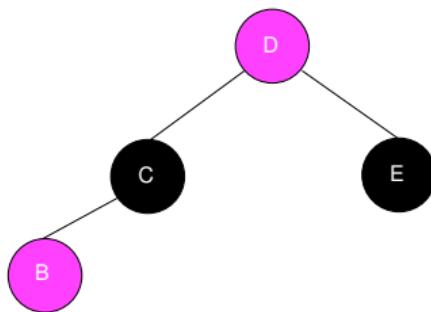
Traditional



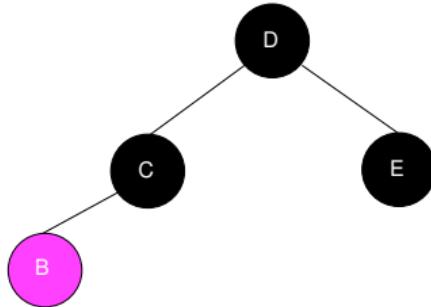
Left-leaning



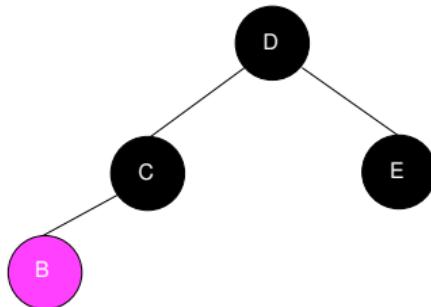
Traditional



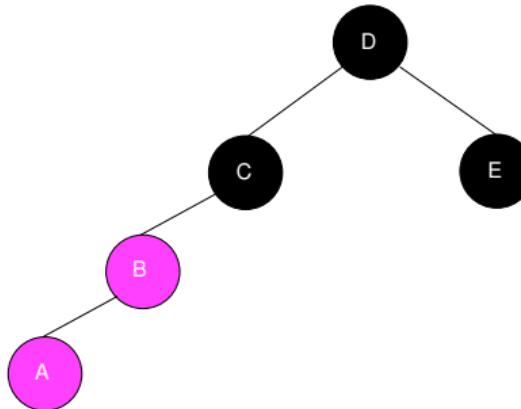
Left-leaning



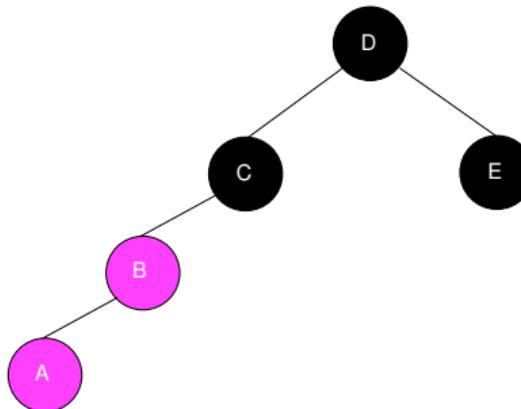
Traditional



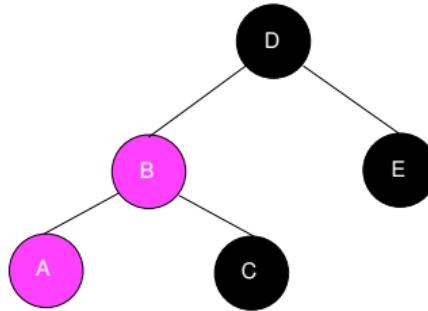
Left-leaning



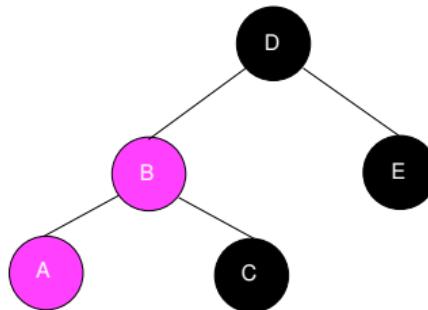
Traditional



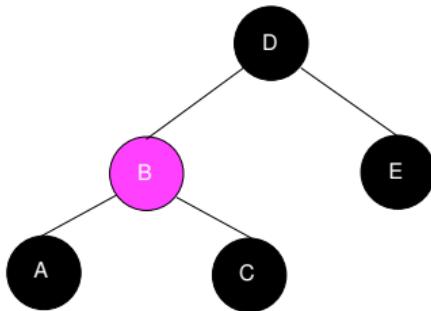
Left-leaning



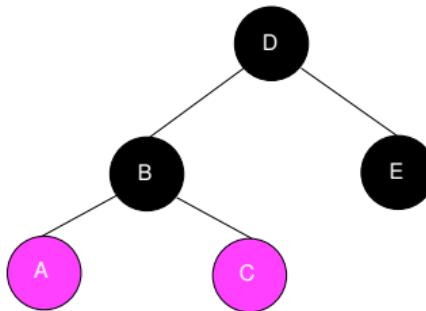
Traditional



Left-leaning



Traditional



Potential violations

Ignorant node

Inconsistent backheight

Red null

Double red

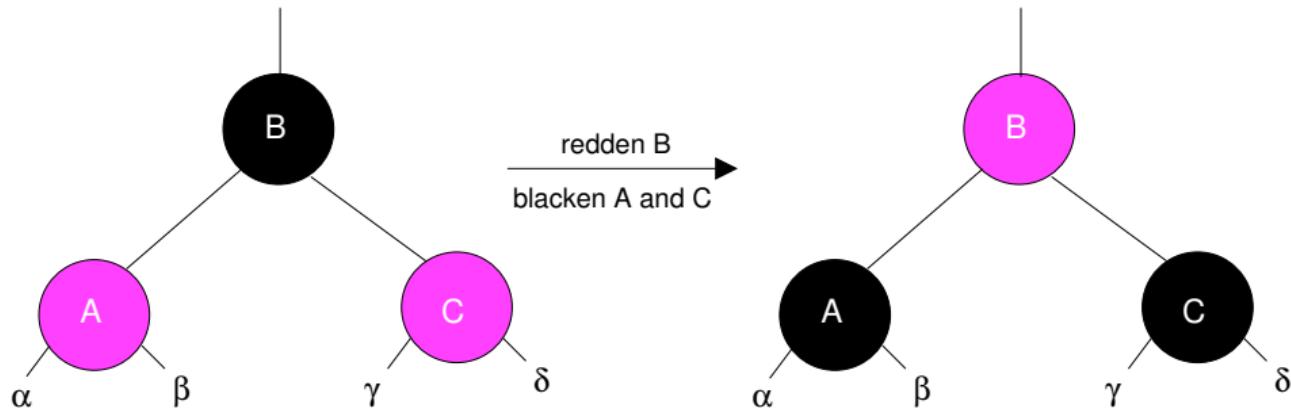
Right red

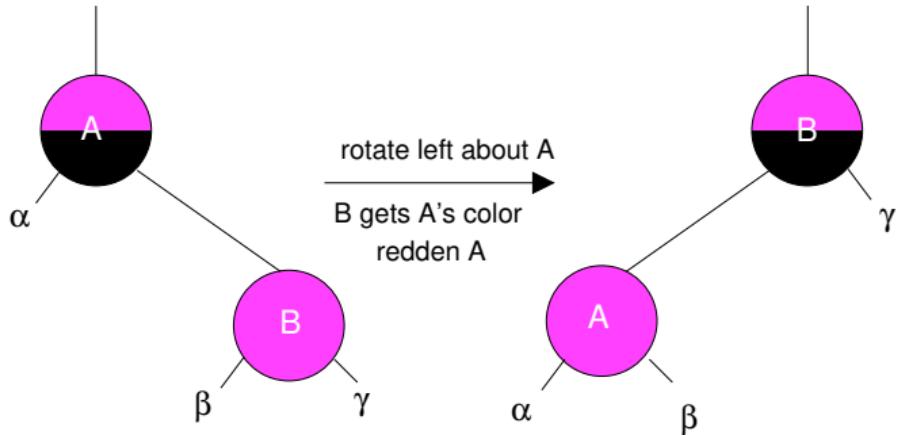
} shouldn't happen

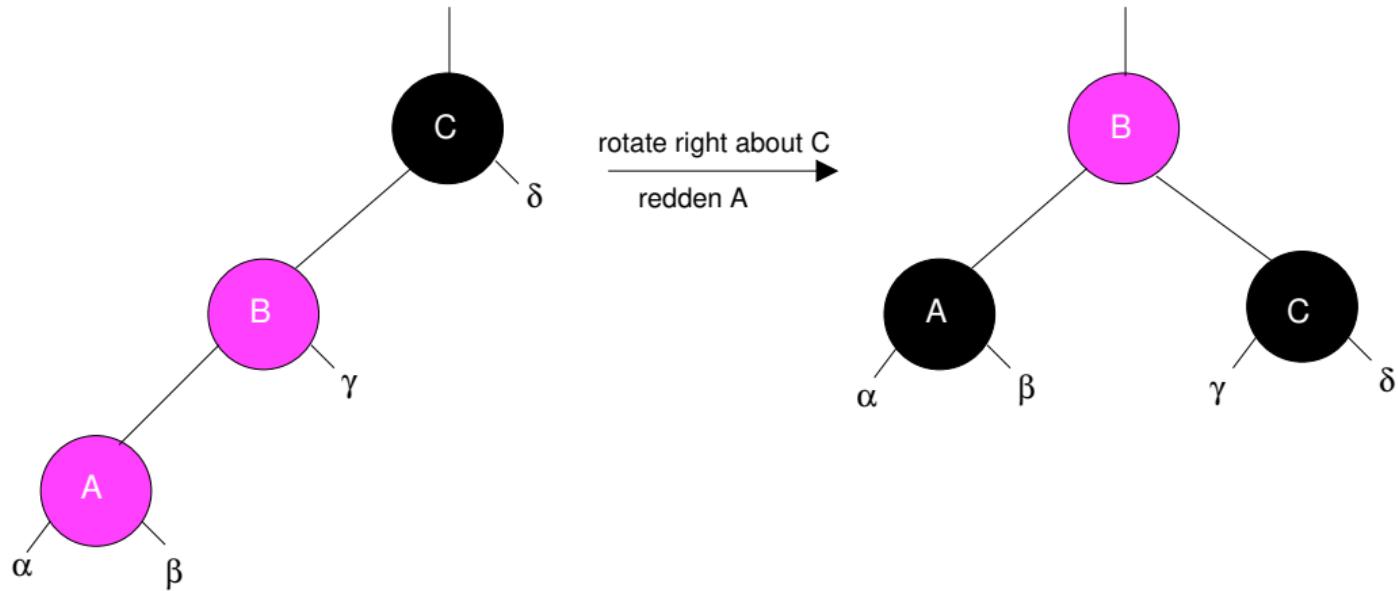
} fix when they happen

Invariant 28 (Postconditions of `RealNode.put()` with LLRB Balancer.) 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 its left child is red, **which can happen only if x is a left child.**
- (d) **The subtree rooted at y has no right-red violations.**





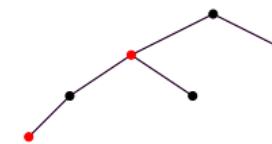


Blackheight

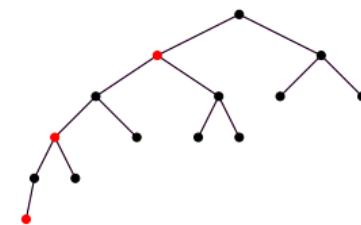
1



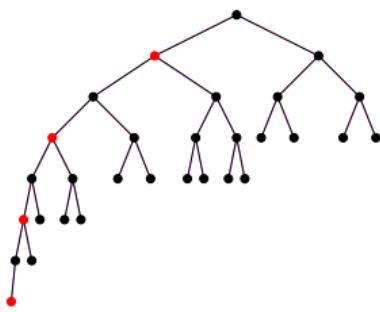
2



3



4



Height

2

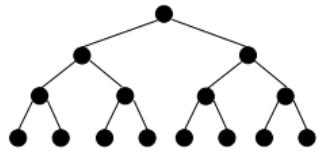
Nodes

4

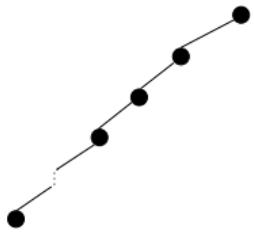
6

6

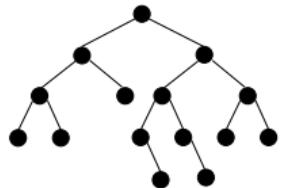
8



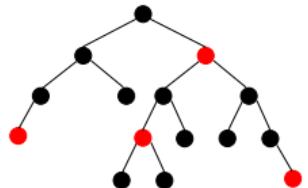
Height: 3
Leaves: 8
Total depth: 34



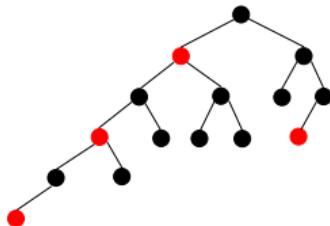
Height: 14
Leaves: 1
Total depth: 105



Height: 4
Leaves: 7
Total depth: 36



Height: 4
Leaves: 7
Total depth: 37



Height: 5
Leaves: 7
Total depth: 38

	After puts			After removals		
	Height	Leaf %	Total depth	Height	Leaf %	Total depth
Unbalanced	32	33.3%	134507	28	16.8%	61207
	31	33.2%	127865	26	17.0%	58171
	30	33.1%	129037	26	16.9%	58610
	28	33.5%	124463	26	17.3%	56086
	32	33.4%	136730	28	16.9%	62092
AVL	16	43.2%	100327	14	21.5%	46088
	15	42.9%	100395	14	21.1%	46028
	15	42.8%	100341	14	21.1%	46028
	15	42.8%	100282	14	21.3%	45973
	15	43.0%	100582	14	21.2%	46097
Traditional RB	16	42.8%	101948	16	21.5%	46729
	16	42.9%	101226	15	21.4%	46344
	16	43.1%	101525	15	21.5%	46462
	16	42.7%	101680	16	21.5%	46572
	16	42.9%	101292	15	21.4%	46338
Left-leaning RB	18	42.8%	102288	18	21.6%	46950
	19	42.9%	102860	16	21.3%	46774
	18	43.1%	101949	17	21.5%	46691
	18	42.7%	102011	17	21.6%	46938
	19	42.9%	102552	16	21.4%	46764