Red-Black Tree Example

Every red-black tree is also a search tree. Red-black trees also have the following properties, which enable us to achieve a better runtime for some operations:

- red edges go from node to its left child ("left-leaning")
- there are no nodes with two red edges
- each path from root to leaf has the same number of black edges ("perfectly black balanced")

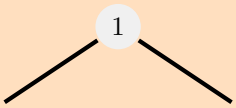
The basic idea of operations on red-black trees is to restore the above-mentioned properties after the operation by means of rotatins, color changes, push-ups and push-ups. When to use which operation is outlined in the lecture slides[1] and lecture document[2]

[1] (`https://lec.inf.ethz.ch/DA/2024/slides/daLecture10.en.pdf`)
[2] (`https://lec.inf.ethz.ch/DA/2024/lecture_notes/dsaln10.pdf`)

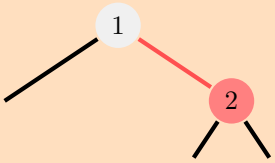
Beispiel "Insertion into Red Black Tree"

Insert the numbers $1, \ldots, 7$ one after the other into an (initially empty) red-black tree and draw the tree after each step.

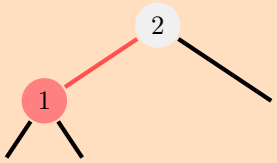
`insert(1)`



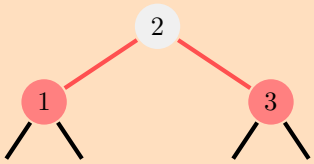
`insert(2)`: add



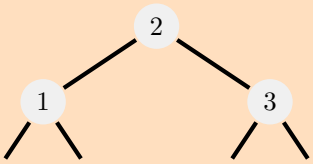
`insert(2)`: rotate_left
(because right child is red)



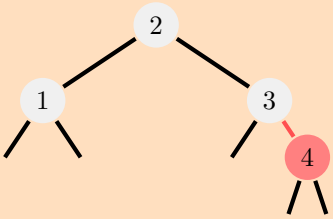
`insert(3)`: add



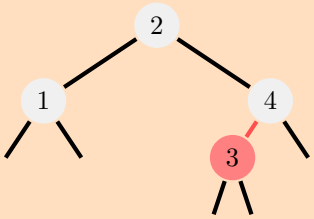
`insert(3)`: push_up
(because two children are red)



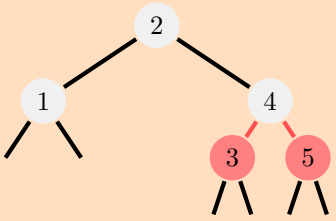
`insert(4)`: add



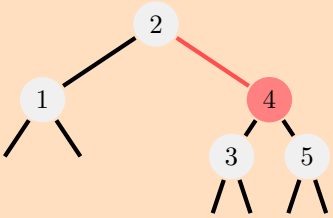
`insert(4)`: rotate_left
(because right child is red)



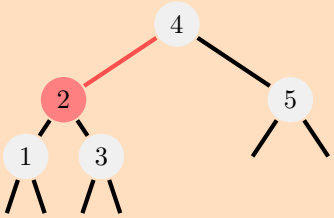
`insert(5)`: add



`insert(5)`: push_up
(because two children are red)

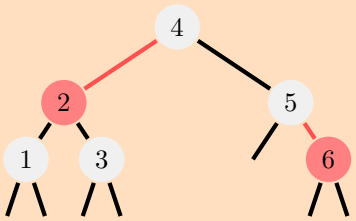


`insert(5)`: rotate_left
(because right child is red)

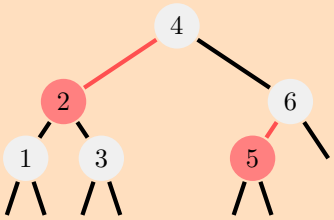


Beispiel "Insertion into Red Black Tree (continued)"

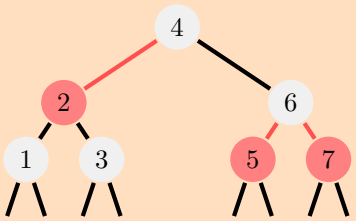
`insert(6): add`



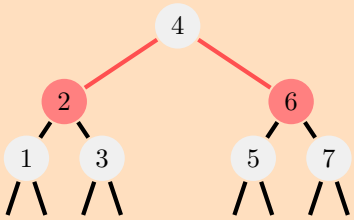
`insert(6): rotate_left`
(because right child is red)



`insert(7): add`



`insert(7): push_up`
(because two children are red)



`insert(7): push_up`
(because two children are red)