

CS202 - Algorithm Analysis

Balanced Tree Algorithms

Aravind Mohan

Allegheny College

April 16, 2023



Sedgewick 3.3 Red Black Trees

2-3 Tree Limitation

- **Too much** balancing, thereby certain insert takes longer.

RED BLACK Trees - An overview

- **Red-black** trees are a variation of binary search trees to ensure that the tree is balanced. A self balancing bst, that offers balancing in a much efficient time than other self balancing trees.
- **Height of the tree** is $O(\log(n))$ and hence the search, insert, and delete operations could be done in $O(\log(n))$ time complexity in the worst case.
- So what are we doing different from 2-3 Trees? We **aim** to **reduce** too much balancing act and thereby provide a better performance.

RED BLACK Trees - An overview

- **Where** is Red-black trees used?

They are used to implement a data structure called finite maps.

TreeMaps and SortedMaps are some examples of finite maps in Java Framework.

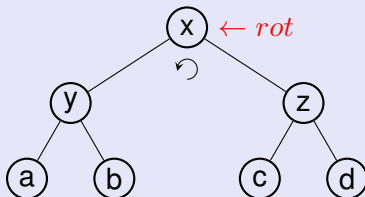
RB Trees - Properties

- Every node is either red or black.
- The **root** is black.
- Every leaf (nil) is black.
- If a node is red, then both its children are black.
- For each node, all paths from the node to descendant leaves contain the same number of **black** nodes.

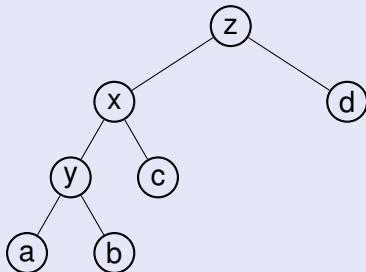
How does balancing work in RB Trees?

- Recoloring - change colors of the node
- Rotation - rotate left or rotate right to make the tree balanced. How does Rotation work?

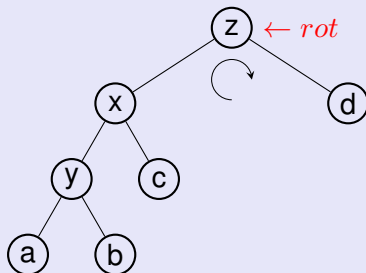
Left Rotation on x



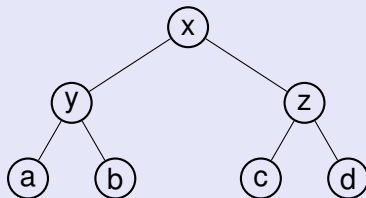
After Left Rotation



Right Rotation on z



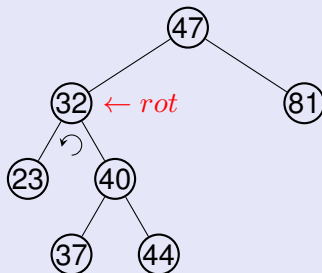
After Right Rotation



Left and right rotations are symmetrical.

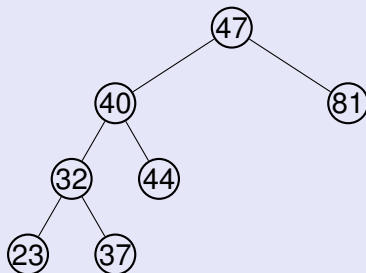
Example:

Left Rotation on 32



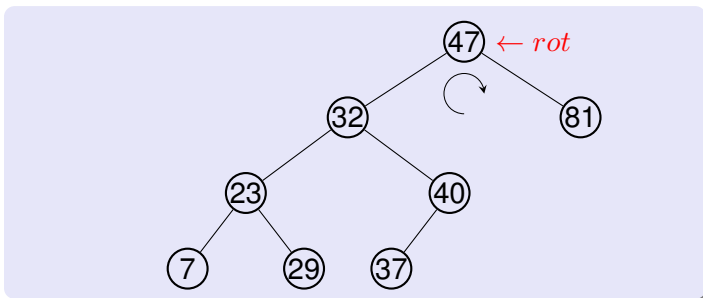
Example:

After Left Rotation



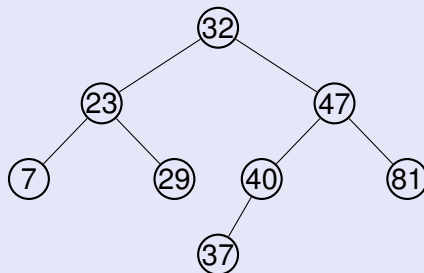
Example:

Right Rotation on 47



Example:

After Right Rotation

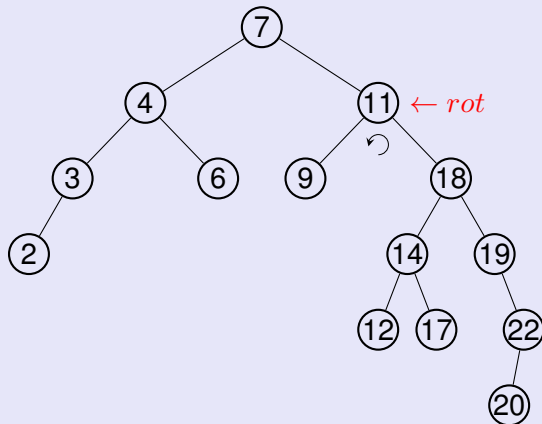


Solve Try out on your own 1 and 2.

Try out 1: Left Rotate 11.

(by using the tree from the next slide.)

RB Trees - Exercise



Try out 2: Right Rotate 18

(by using the tree from your Tryout1 solution)

Important Point:

RB Trees Insert and Delete operations.

Get used to Rotation Process. We will need to do a lot of rotations in Insert and Delete!

RB Trees - Properties (Recall)

- Every node is either red or black.
- The **root** is black.
- Every leaf (nil) is black.
- If a node is red, then both its children are black.
- For each node, all paths from the node to descendant leaves contain the same number of **black** nodes.

How to insert a new node into the Red Black Tree?

- Insert node z in RB tree:
 - 1 As in ordinary Binary Search Tree
 - 2 Node z should be colored red

Violation of property of RB tree

- After insertion using the rules outlined in the previous slide, further examination is required to identify if the RB tree properties are preserved or not.
- The violation is categorized into two broad categories:
 - 1 Case 1: Node z is the root node
 - 2 Case 2: Node z is a child of a red parent

Violation of property of RB tree (contd)

Case 1: Node z is the root node and red node.

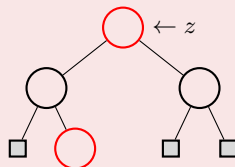
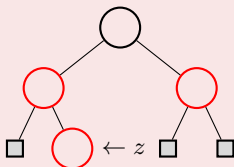


The fix for this case is:

- Change node z 's color to **Black**

Violation of property of RB tree (contd)

Case 2.1: Uncle of node z is red

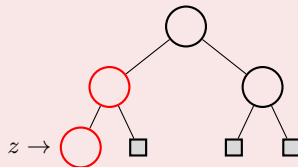
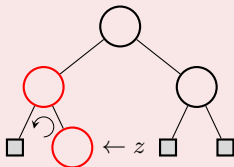


The fix for this case is:

- Change node z 's grandparent color to **Red** and make this node the new z
- Change node z 's parent and uncle color to **Black**
- Elevate the grandparent of node z to be the new **node z** (FIX UP)

Violation of property of RB tree (contd)

Case 2.2: Uncle of node z is black and z is the right child

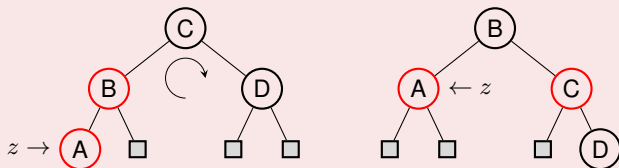


The fix for this case is:

- Left rotate node z 's parent node
- Change the position from right to left. This leads to Case 2.3
- If black uncle is on left, and z is left child, Right rotate parent.

Violation of property of RB tree (contd)

Case 2.3: Uncle of node z is black and z is the left child

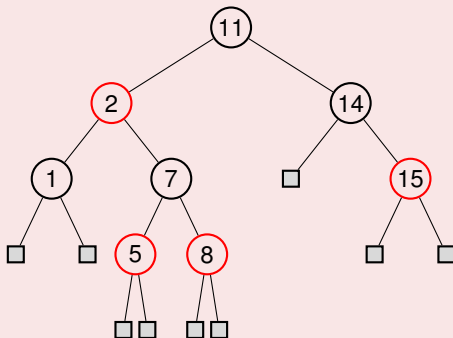


The fix for this case is:

- Change node z 's grandparent color to **Red** and parent color to **Black**
- Right rotate node z 's grandparent node
- Left rotate node z 's grandparent node, if black uncle is on left. Rotate towards the direction of uncle.

Okay, let's do an example?

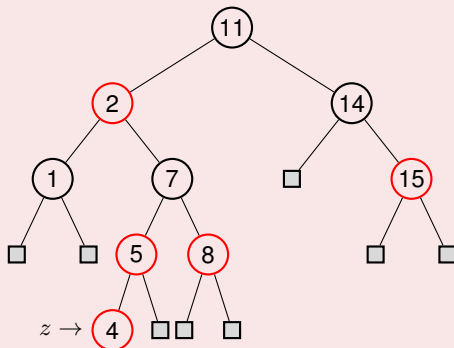
Insert a new node $z = 4$



So how to insert this new node?

Okay, let's do an example? [contd]

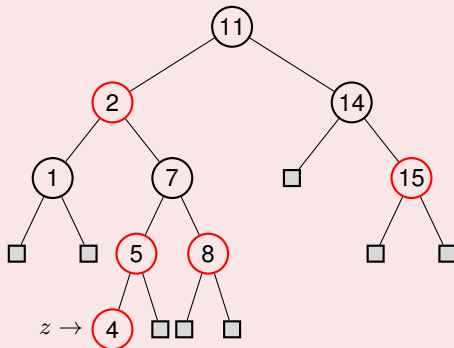
Step 1: Insert a new red color node $z = 4$



By using the same rule as BST

Okay, let's do an example? [contd]

Step 2: Violation at node $z = 4$



Okay, let's do an example? [contd]

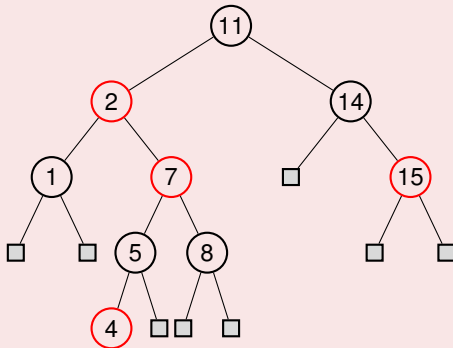
"Case 2.1 violation @ node z "

Step 2: Violation at node $z = 4$

- **Change node z 's grandparent color to Red**
- **Change node z 's parent and uncle color to Black**
- **Make grand parent of node z to be the new node z**

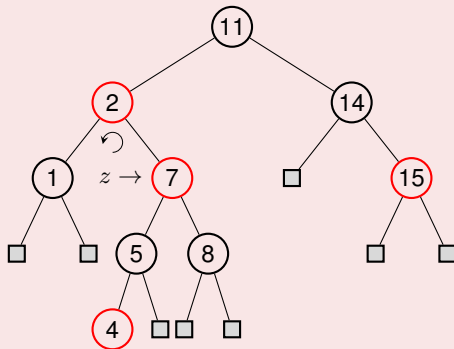
Okay, let's do an example? [contd]

Step 2: Violation at node $z = 4$



Okay, let's do an example? [contd]

Step 3: Violation at node $z = 7$



Okay, let's do an example? [contd]

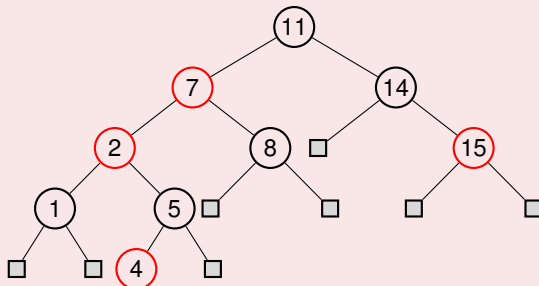
"Case 2.2 violation @ node z "

Step 3: Violation at node $z = 7$

- Left rotate at node z 's parent
- Change the position from right to left. This leads to Case 2.3

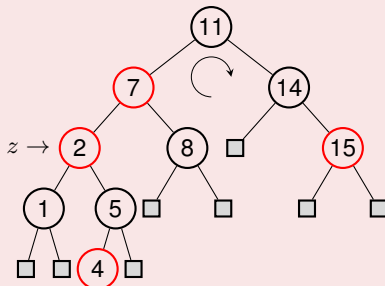
Okay, let's do an example? [contd]

Step 3: Violation at node $z = 7$



Okay, let's do an example? [contd]

Step 4: Violation at node $z = 2$



Okay, let's do an example? [contd]

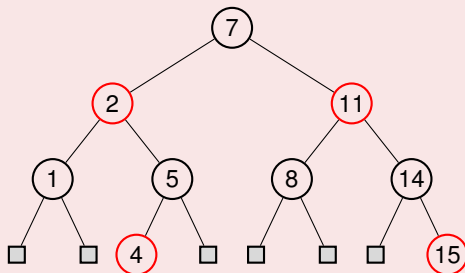
"Case 2.3 violation @ node z "

Step 4: Violation at node $z = 2$

- **Change node z 's grandparent color to Red and parent color to Black;**
- **Right rotate at node z 's grandparent**

Okay, let's do an example? [contd]

Step 4: Violation at node $z = 2$



- No more violation
- **All the path from the root node to the NIL nodes contains equal number of black nodes!**

Solve Try out 3 and 4 on your own!

Try out 3: Insert 4, 3, 6 (step by step)
(by using the tree from the above slide.)

Try out 4: Insert 9, 10, 13 (step by step)
(by using the tree from your Tryout1 solution)

How to delete a node in the Red Black Tree?

- **Step 1:** Perform BST deletion procedure.
- **Step 2:** If the node (z) to be deleted is red color, then just delete (z) from the tree.
- **Step 3:** If the node (z) to be deleted is black color, then color (z) as Double Black.

- **Double Black:** Apply a series of resolution rules to fix the violations. Cases are **Mirrored**.

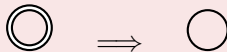
Double Black Node Resolution

Case 1:

- If (z) is a double black root node, then just convert the double black root to single black.

Double Black Node Resolution

Case 1:



Double Black Node Resolution

Case 2:

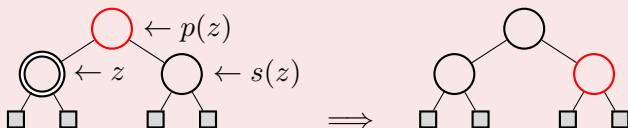
- If the sibling of the double black node $s(z)$ is black and the children of $s(z)$ are both black:
 - 1 **color** z as single **black**
 - 2 add **black** to the parent node $p(z)$
 - 3 **color** $s(z)$ as **red**

add black

- if $p(z)$ is originally black, then color $p(z)$ as **double black** and recursively **fix** the violation upwards
- if $p(z)$ is originally red, then **color** $p(z)$ as **black**

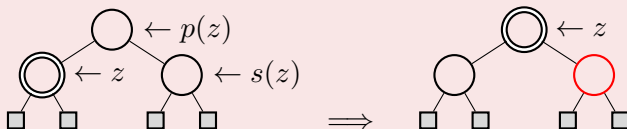
Double Black Node Resolution

Case 2 - (1):



Double Black Node Resolution

Case 2 - (2):

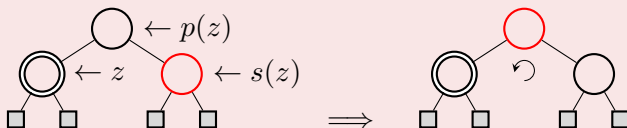


Case 3:

- If the sibling of the double black node $s(z)$ is red:
 - 1 **swap** colors of parent $p(z)$ and $s(z)$
 - 2 **rotate** $p(z)$ towards the direction of z
 - 3 **reapply** cases

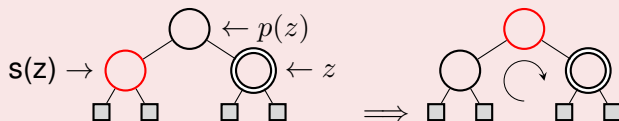
Double Black Node Resolution

Case 3 - (1):



Double Black Node Resolution

Case 3 - (2):



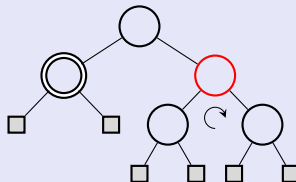
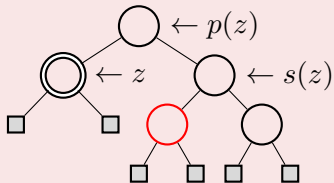
MIRRORED Case 3 - (1)

Case 4:

- If the sibling of the double black node $s(z)$ is black and the closest child of $s(z)$ from z is red, farthest child of $s(z)$ from z is black.
 - 1 **swap** colors of sibling $s(z)$ and the closest child of $s(z)$ from z
 - 2 **rotate** sibling $s(z)$ towards the opposite direction of z
 - 3 **apply** Case 5

Double Black Node Resolution

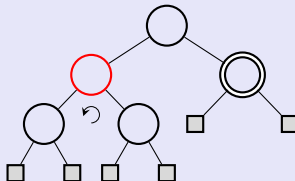
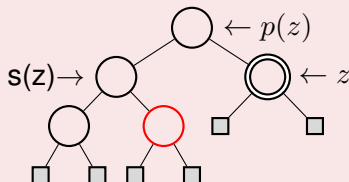
Case 4 - (1):



Double Black Node Resolution

Case 4 - (2):

MIRRORED Case 4 - (1)

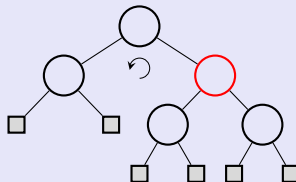
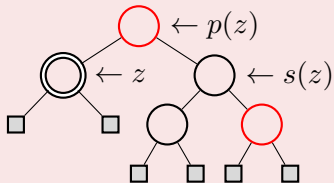


Case 5:

- If the sibling of the double black node $s(z)$ is black and the farthest child of $s(z)$ from z is red:
 - 1 **swap** colors of parent $p(z)$ and sibling $s(z)$
 - 2 **change** color of the farthest child of $s(z)$ from z , to black
 - 3 **rotate** parent $p(z)$ towards the direction of z
 - 4 **color** z as single black.

Double Black Node Resolution

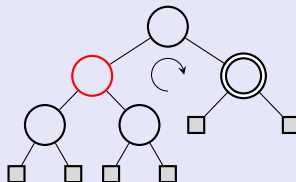
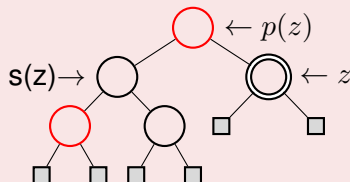
Case 5 - (1):



Double Black Node Resolution

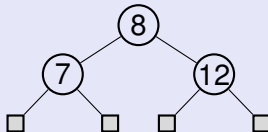
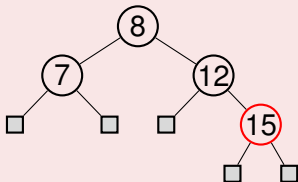
Case 5 - (2):

MIRRORED Case 5 - (1)



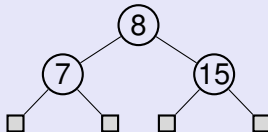
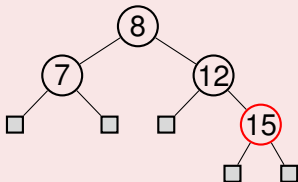
RB Delete Examples

Example 1: Delete 15



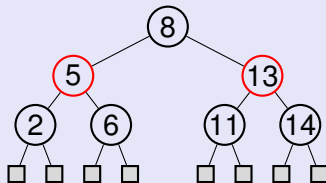
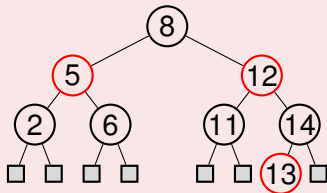
RB Delete Examples

Example 2: Delete 12



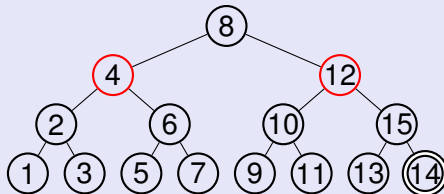
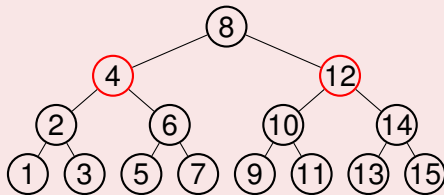
RB Delete Examples

Example 3: Delete 12



RB Delete Examples

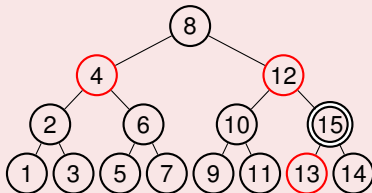
Example 4: Delete 14



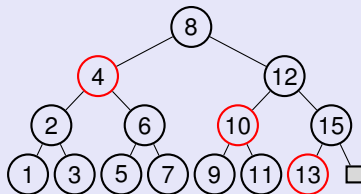
RB Delete Examples

Example 4: Delete 14 (contd)

Case 2

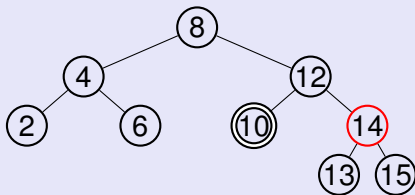
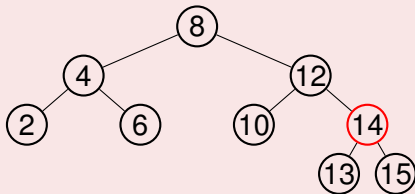


Case 2



RB Delete Examples

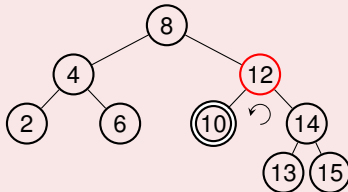
Example 5: Delete 10



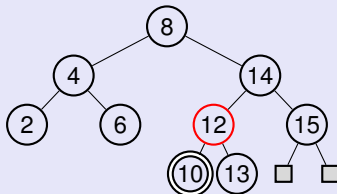
RB Delete Examples

Example 5: Delete 10 (contd)

Case 3

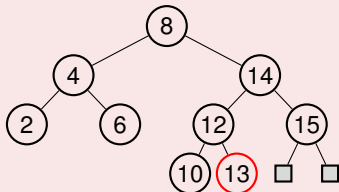


Case 3

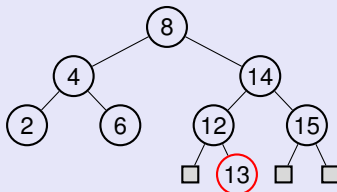


Example 5: Delete 10 (contd)

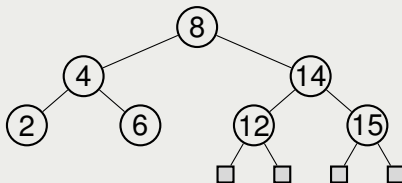
Case 2



Case 3



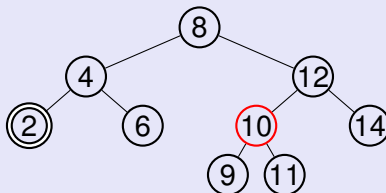
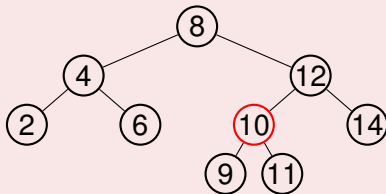
Try out 5: Delete 15 from the tree below:



Continue with more examples.

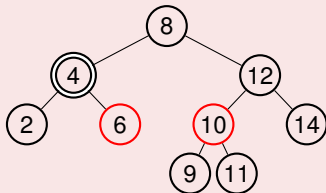
RB Delete Examples

Example 6: Delete 2

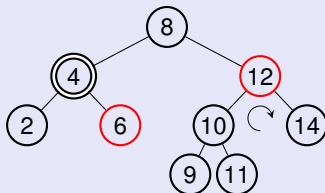


Example 6: Delete 2 (contd)

Case 2

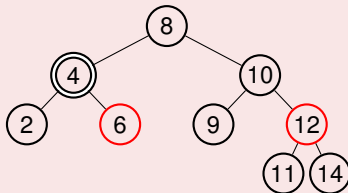


Case 4

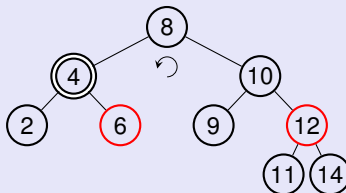


Example 6: Delete 2 (contd)

Case 4



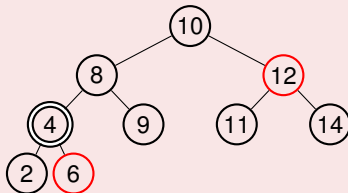
Case 5



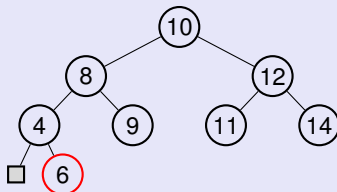
RB Delete Examples

Example 6: Delete 2 (contd)

Case 5

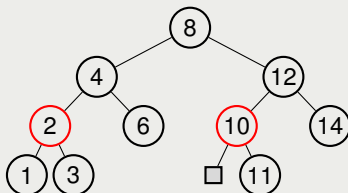


Case 5



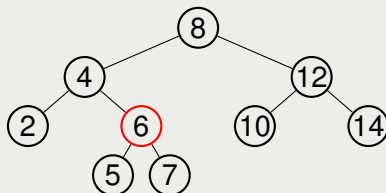
Try out 6 - delete from the tree below:

- **First** Delete 2
- **Second** Delete 3
from the tree produced after deleting 2

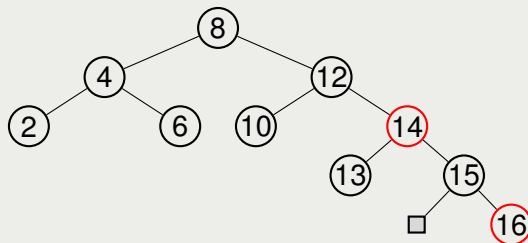


Try out 7

- Delete 14



RB Trees - Try out 8 Exercise



Delete 10, 4, 16, 15, 8, 6, 2

Sedgewick 3.3 RB Trees

Questions?

Please ask if there are any Questions!