AVL Tree

CSE 332 - Section 4



Reminder

Meet the Staff ***due Oct 23!***

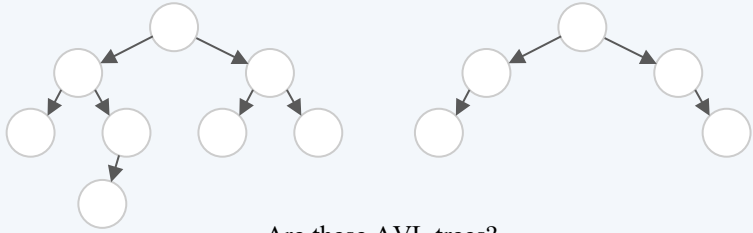
please come to our office hours!! 😄

Announcements

- Midterm is coming up!
  - Monday, **Nov 9th**
  - Review session:Friday, Nov 6th (during Lecture)
  - Study for midterm: section problems, past midterms. Have any questions? Go to OH or post on ed

AVL Trees

What are AVL Trees?



Are these AVL trees?

According to lecture:

An AVL Tree is a tree data structure with the following properties:

Structural Property

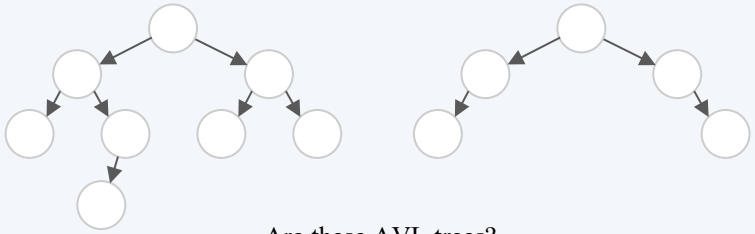
- *Each node has ⩽2 children.*
- *Heights of the left and right subtrees of every node differ by at most 1 .*

Order Property

- *All keys in the left subtree < node's key.*
- *All keys in the right subtree > node's key.*

Notice that this is just a Binary Search Tree (BST) with 1 additional property.

Wait, why?



Are these AVL trees?

- **Guaranteed balance**. AVL trees keep their height minimal after every insert/delete. This allows operations to stay efficient.

Data Structure	Find	Insert	Delete
AVL Tree	O(log n)	O(log n)	O(log n)
Unbalanced BST	O(h), h = height	O(h)	O(h)
Linked List	O(n)	O(n)	O(n)
Unsorted Array	O(n)	O(n)	O(n)
Sorted Array	O(log n) [binary search]	O(n)	O(n)

Recall the ADT ensures no duplicate elements, so we must search before inserting

Worksheet: Problem 0

Problem 0 (Questions)

What are the constraints on the data types you can store in an AVL tree?

When is an AVL tree preferred over another dictionary implementation, such as a HashMap?

Problem 0 (Answer 1)

What are the constraints on the data types you can store in an AVL tree?

- Keys must be of a comparable data type.
- Values can be any data type.

When is an AVL tree preferred over another dictionary implementation, such as a HashMap?

Problem 0 (Answer 2)

What are the constraints on the data types you can store in an AVL tree?

- Keys must be of a comparable data type.
- Values can be any data type.

When is an AVL tree preferred over another dictionary implementation, such as a HashMap?

- In an AVL tree, keys can be iterated over in sorted order.
- It’s easier to find next and previous key in AVL tree. (check ex4)

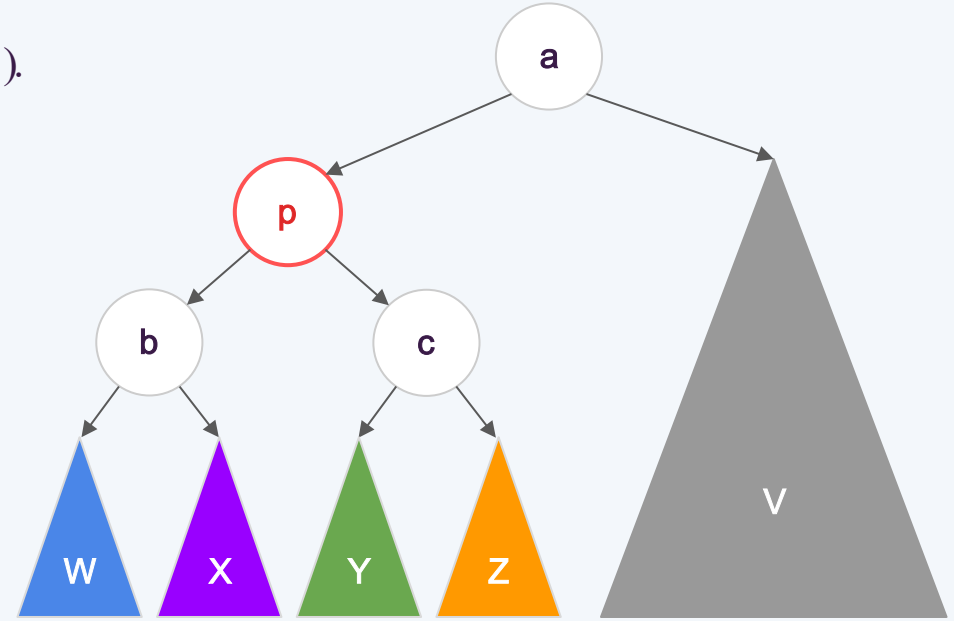
AVL Tree Rotations Microteach

AVL Tree Rotations

To fix an imbalance in an AVL tree after an insert:

1. Identify the *lowest* problem node p where the imbalance occurs.
2. Identify the insert location.
3. Execute the corresponding tree rotation(s).

Case	Insert Location	Tree Rotation(s)
1	Left subtree of Left child (W)	Single right rotation
2	Right subtree of Left child (X)	Double left-right rotation
3	Left subtree of Right child (Y)	Double right-left rotation
4	Right subtree of Right child (Z)	Single left rotation

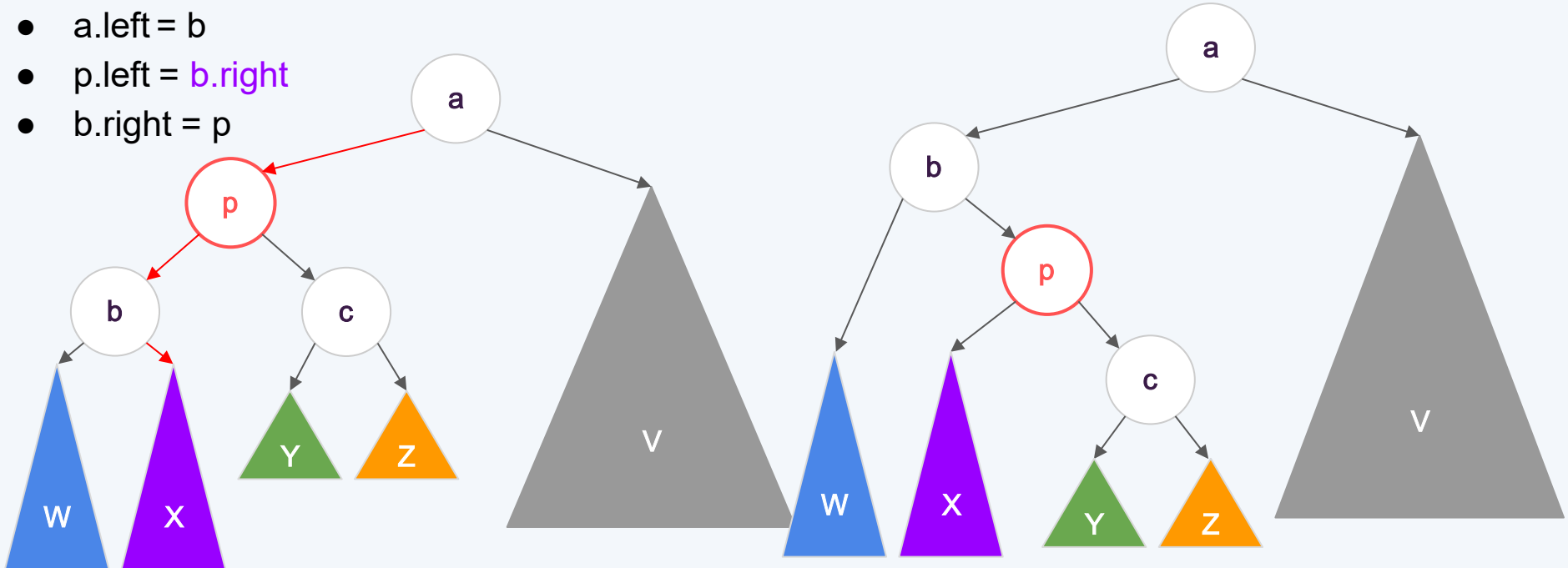


Right Rotation

p is the lowest problem node. Insert location was in w.

References that change:

- a.left = b
- p.left = b.right
- b.right = p

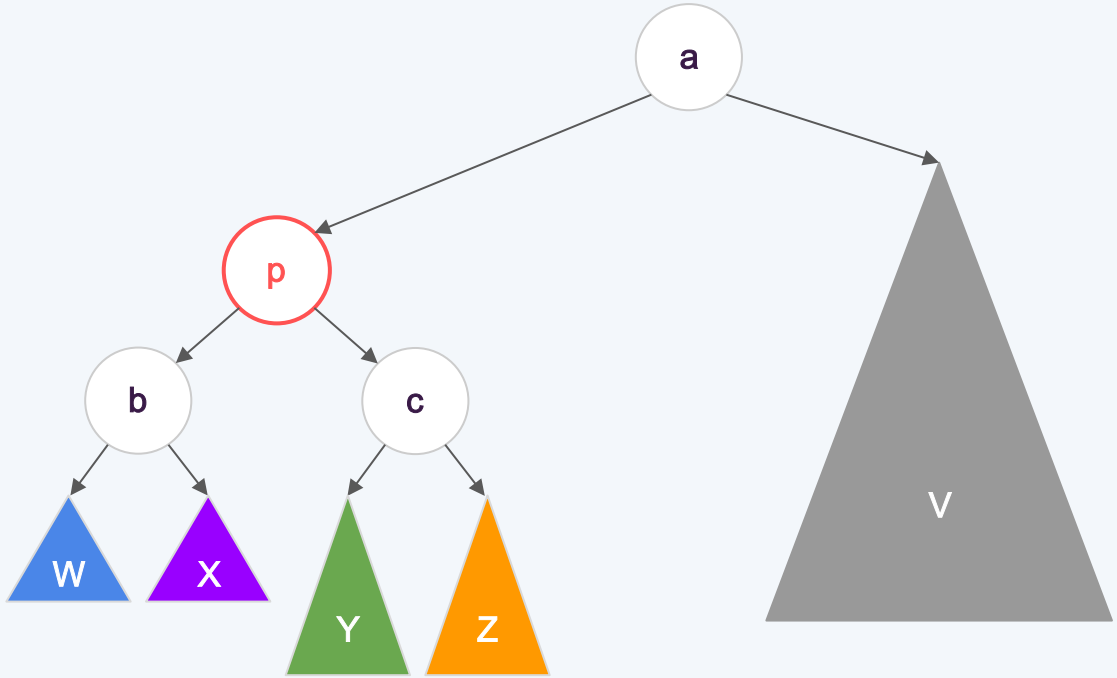


Problem 1 - Left Rotation

p is the lowest problem node. Insert location was in z.

References that change:

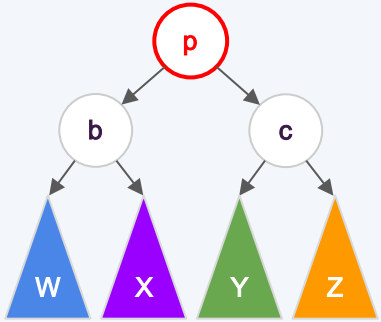
- a.left = c
- p.right = c.left
- c.left = p



Tip for ex4: remember to update height field in nodes

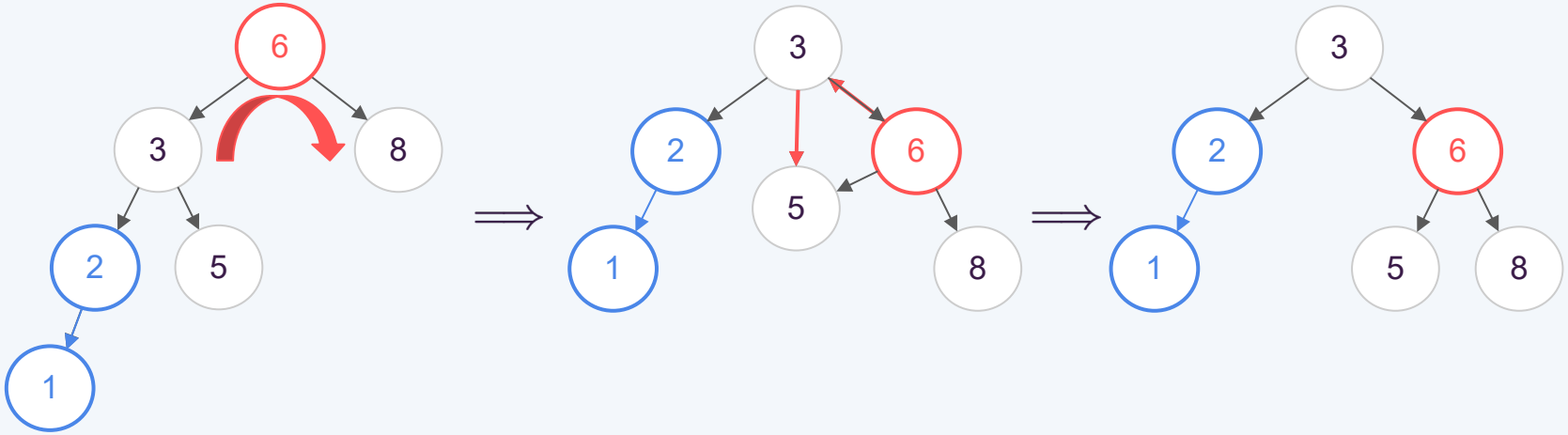
Case 1

Case	Insert Location	Tree Rotation(s)
1	Left subtree of Left child (W)	Single right rotation
2	Right subtree of Left child (X)	Double left-right rotation
3	Left subtree of Right child (Y)	Double right-left rotation
4	Right subtree of Right child (Z)	Single left rotation



Let's insert 1.

1. Identify the problem node p
2. Identify the insert location
3. Execute the tree rotation



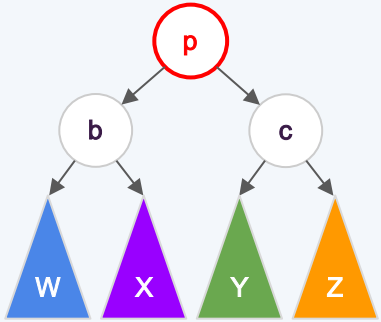
3.1. Identify bad edges

3.2. Remove bad edges

3.3. Replace edges

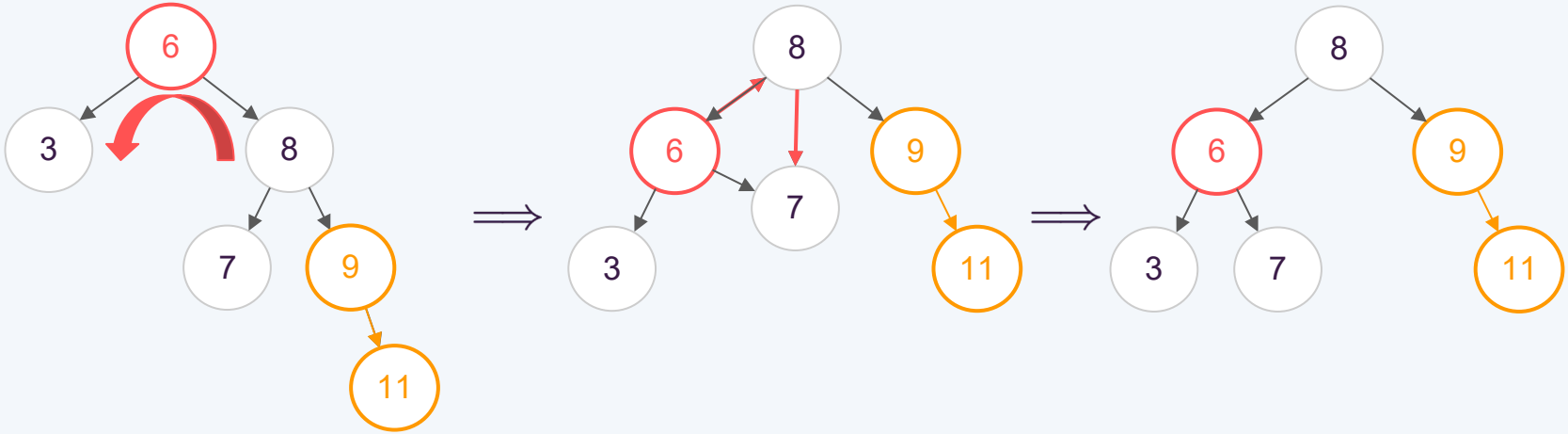
Case 4

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Let's insert 11.

1. Identify the problem node p
2. Identify the insert location
3. Execute the tree rotation



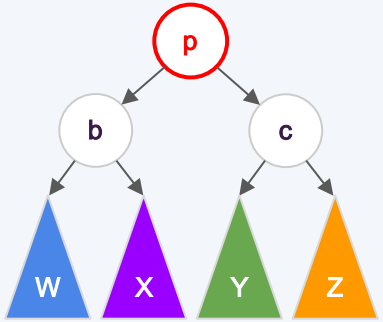
3.1. Identify bad edges

3.2. Remove bad edges

3.3. Replace edges

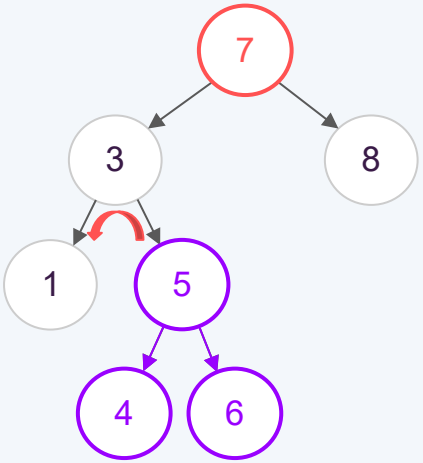
Case 2 (left)

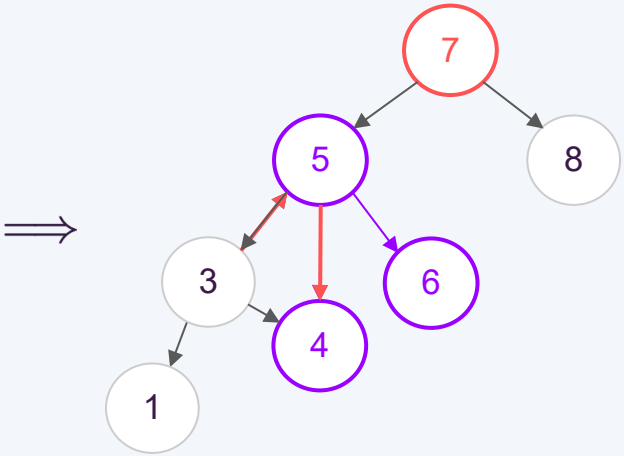
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

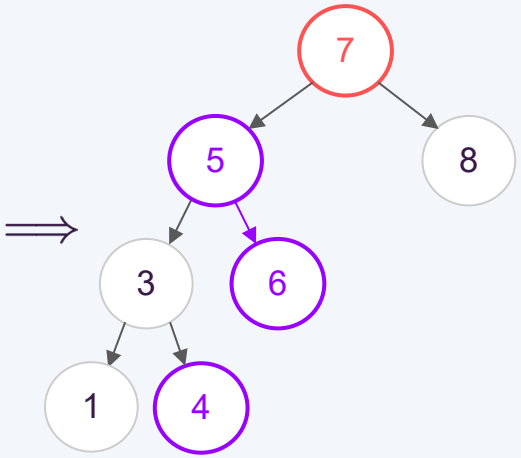


Let's insert either4 or 6.

1. Identify the problem nodep
2. Identify the insert location
3. Execute the first tree rotation
   For Case 2, we do the first rotation on nodeb







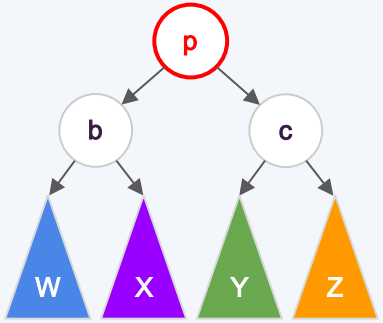
3.1. Identify bad edges

3.2. Remove bad edges

3.3. Replace edges

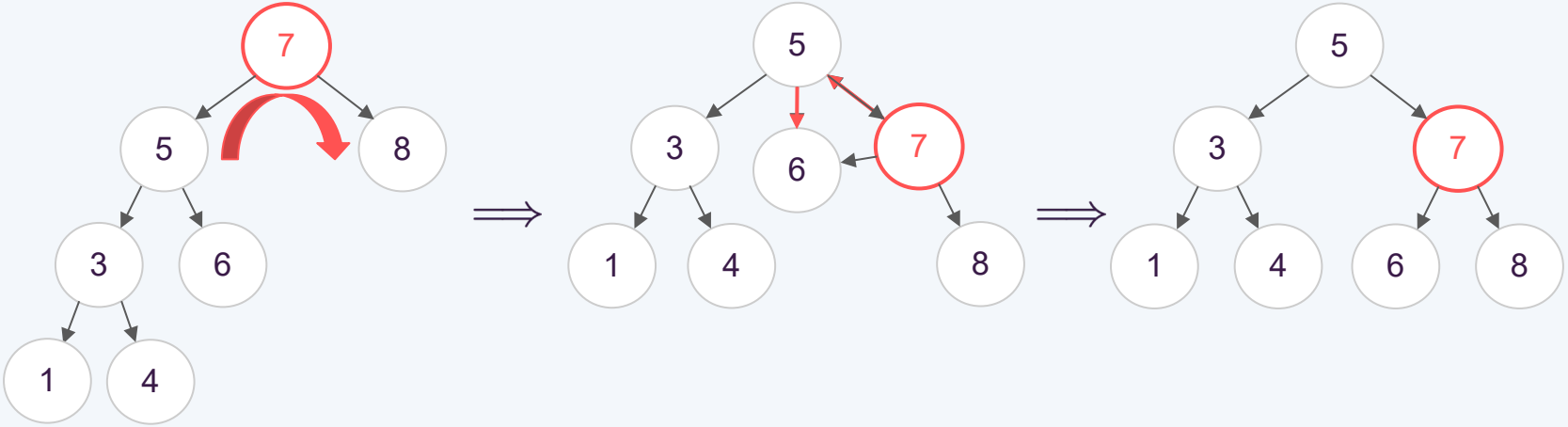
Case 2 (right)

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



This is now similar to Case 1!

4. Execute the second tree rotation
   We do the second rotation on node p



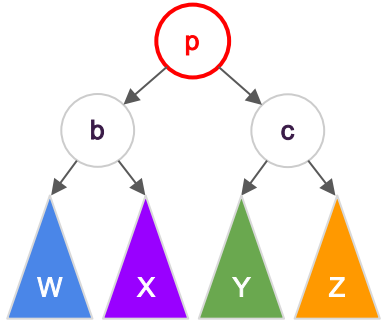
4.1. Identify bad edges

4.2. Remove bad edges

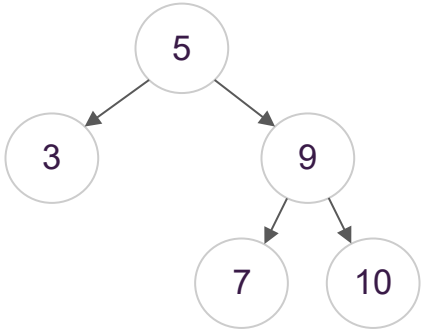
4.3. Replace edges

Case 3

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

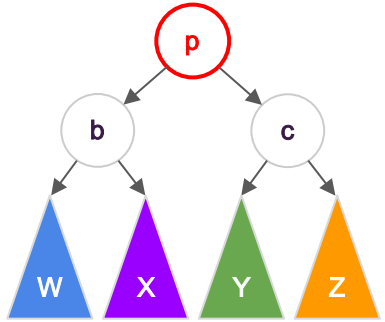


Let's insert either6 or 8.

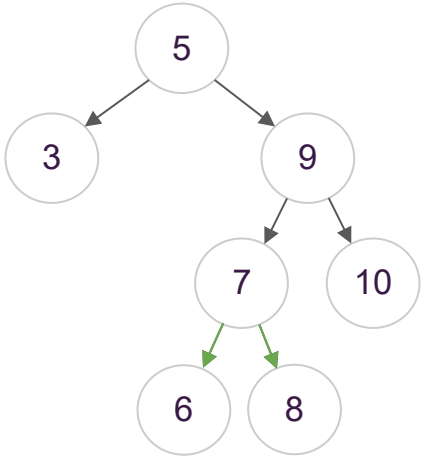


Case 3 – Step 1

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

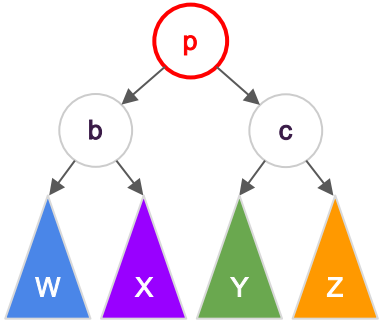


Let's insert either6 or 8.



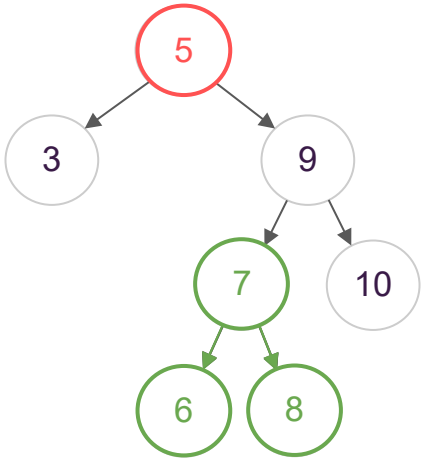
Case 3 – Step 2

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



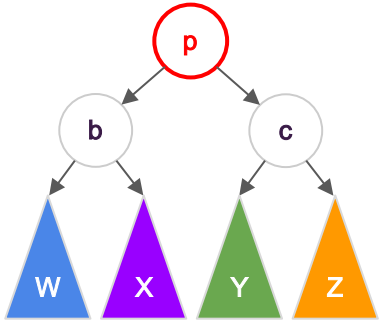
Let's insert either6 or 8.

1. Identify the problem nodep
2. Identify the insert location



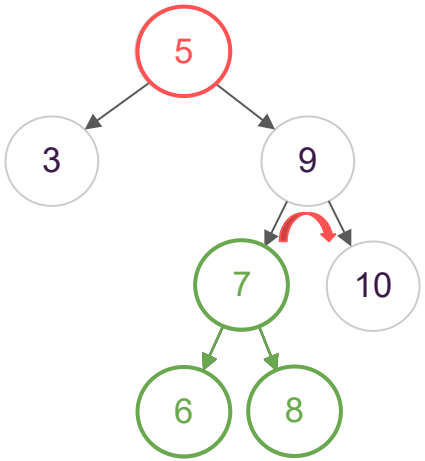
Case 3 – Step 3

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Let's insert either 6 or 8.

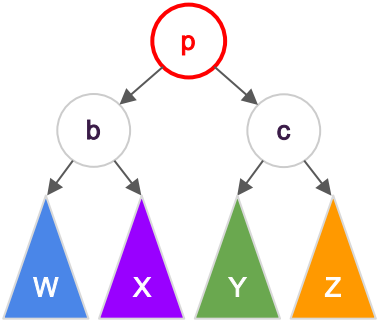
1. Identify the problem node **p**
2. Identify the insert location
3. Execute the first tree rotation
   For Case 3, we do the first rotation on node **c**



3.1. Identify bad edges

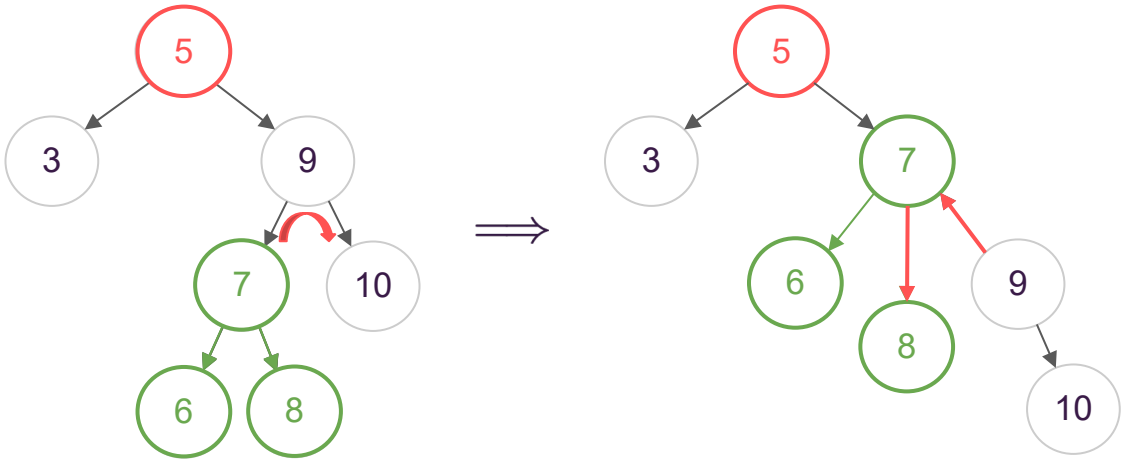
Case 3 – Step 4

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Let's insert either 6 or 8.

1. Identify the problem node **p**
2. Identify the insert location
3. Execute the first tree rotation
   For Case 3, we do the first rotation on node **c**

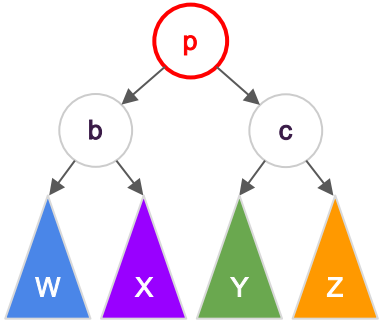


3.1. Identify bad edges

3.2. Remove bad edges

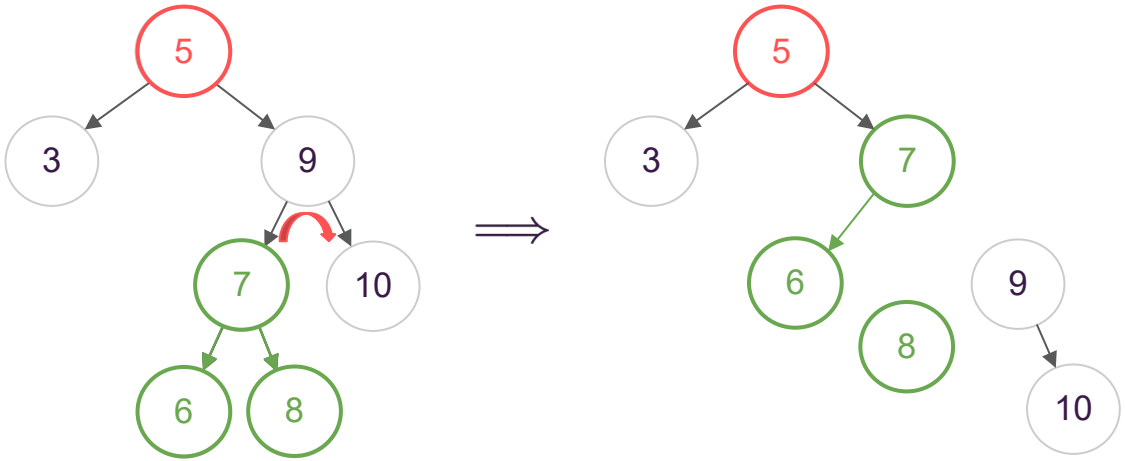
Case 3 – Step 5

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Let's insert either6 or 8.

1. Identify the problem nodep
2. Identify the insert location
3. Execute the first tree rotation
   For Case 3, we do the first rotation on nodec

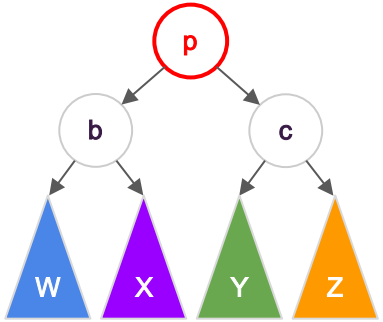


3.1. Identify bad edges

3.2. Remove bad edges

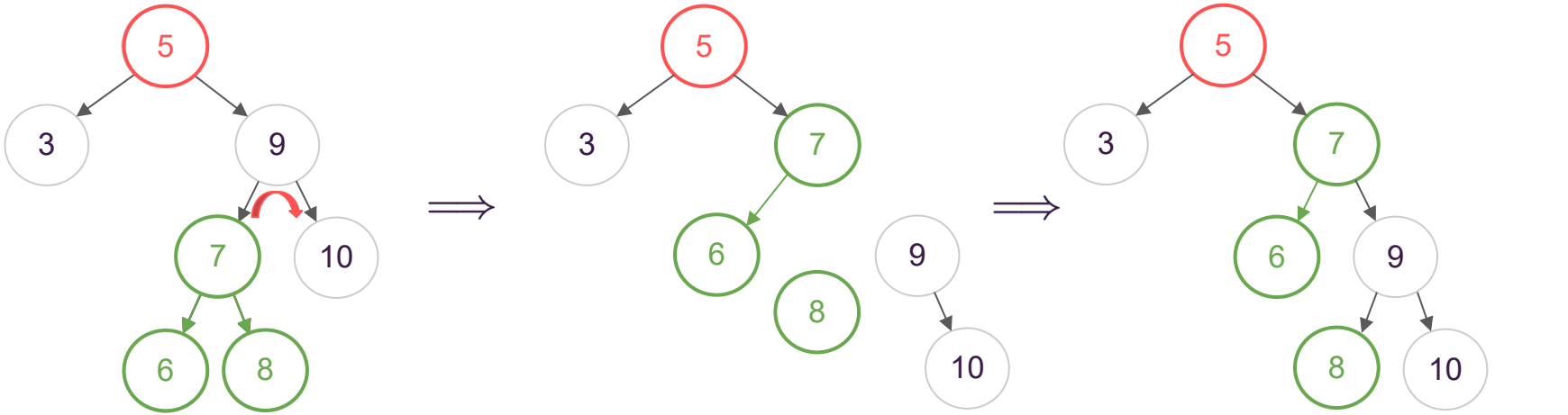
Case 3 – Step 6

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Let's insert either6 or 8.

1. Identify the problem nodep
2. Identify the insert location
3. Execute the first tree rotation
   For Case 3, we do the first rotation on nodec



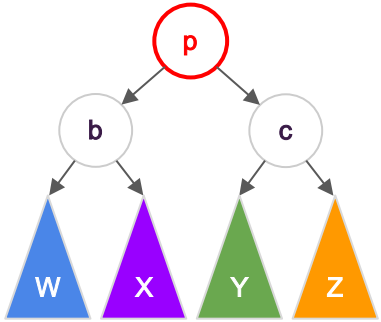
3.1. Identify bad edges

3.2. Remove bad edges

3.3. Replace edges

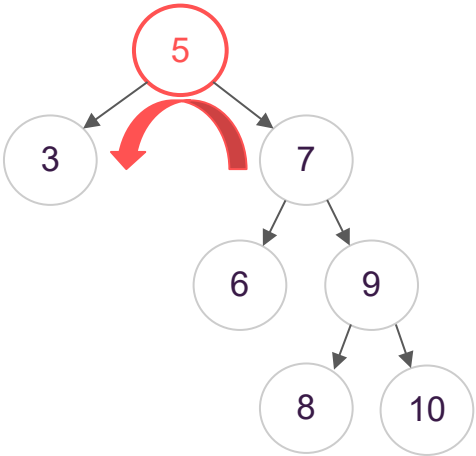
Case 3 – Step 7

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



This is now similar to Case 4!

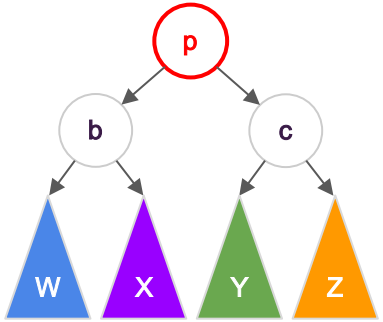
4. Execute the second tree rotation
   We do the second rotation on node p



4.1. Identify bad edges

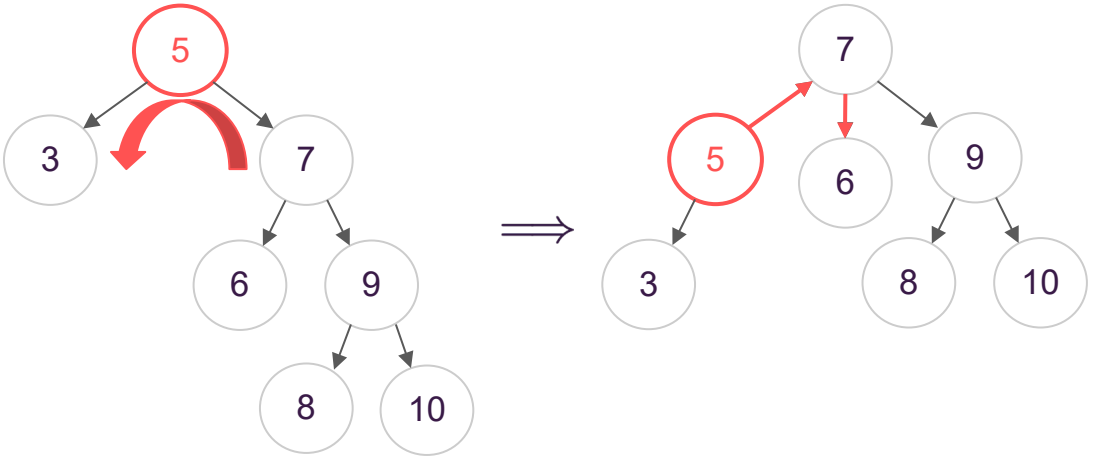
Case 3 – Step 8

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



This is now similar to Case 4!

4. Execute the second tree rotation
   We do the second rotation on node p

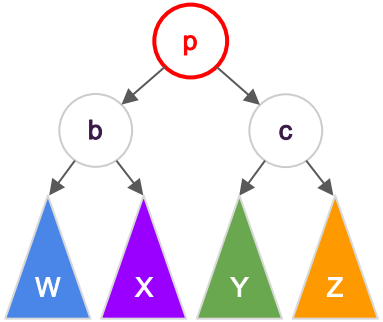


4.1. Identify bad edges

4.2. Remove bad edges

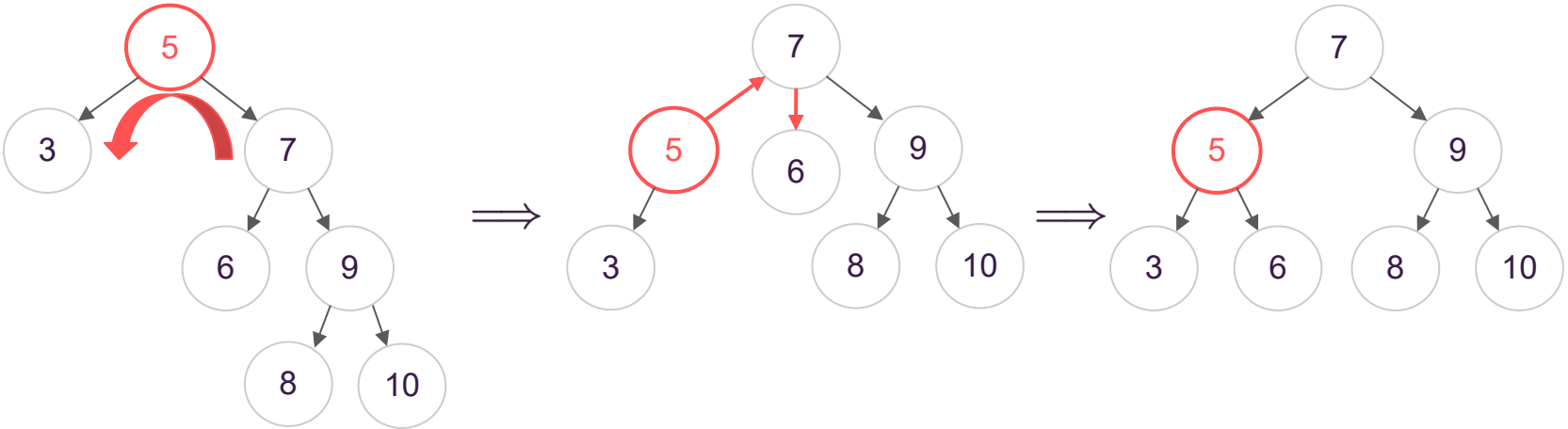
Case 3 – Step 9

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



This is now similar to Case 4!

4. Execute the second tree rotation
   We do the second rotation on node p



4.1. Identify bad edges

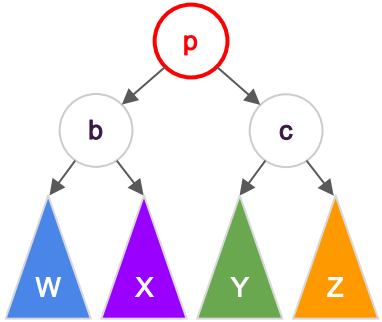
4.2. Remove bad edges

4.3. Replace edges

Problem 2 Walkthrough

Problem 2

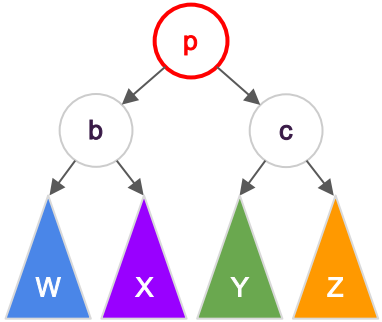
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

P2 – Step 1

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



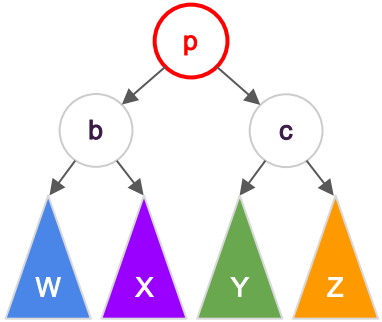
Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

No imbalances

10

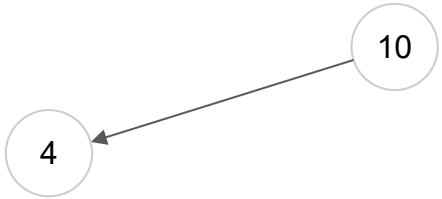
P2 – Step 3

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



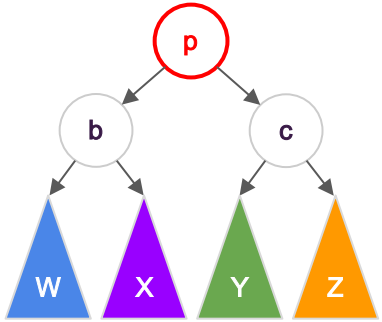
Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

No imbalances



P2 – Step 4

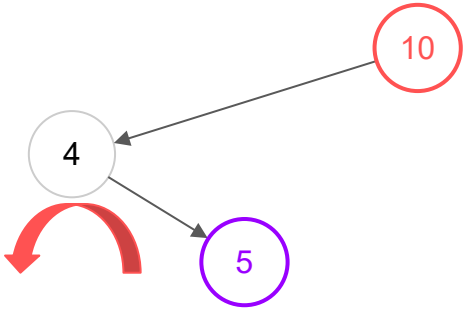
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

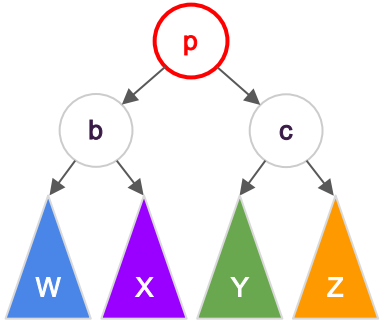
Imbalance at 10

- Case 2
- Rotate left at 4



P2 – Step 5

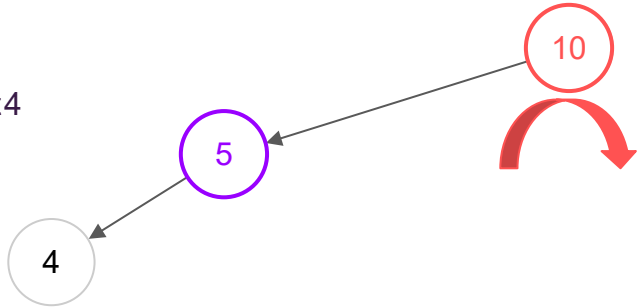
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

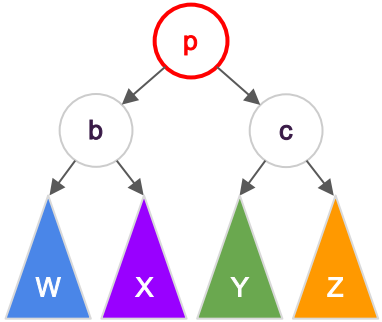
Imbalance at 10

- Case 2
- Rotate left at 4



P2 – Step 6

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

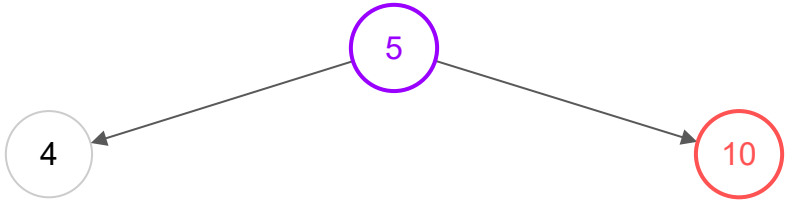


Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

Imbalance at 10

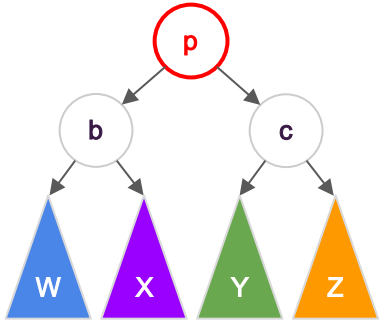
- Case 2
- Rotate left at 4
- Rotate right at 10

No more imbalances



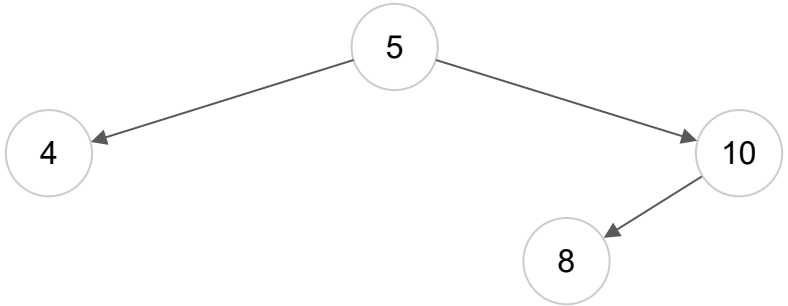
P2 – Step 7

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



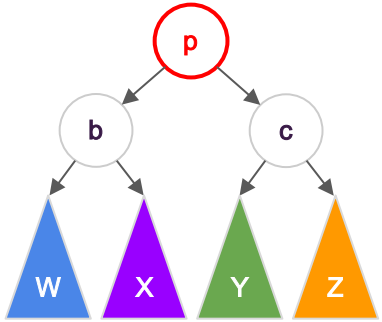
Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

No imbalances



P2 – Step 8

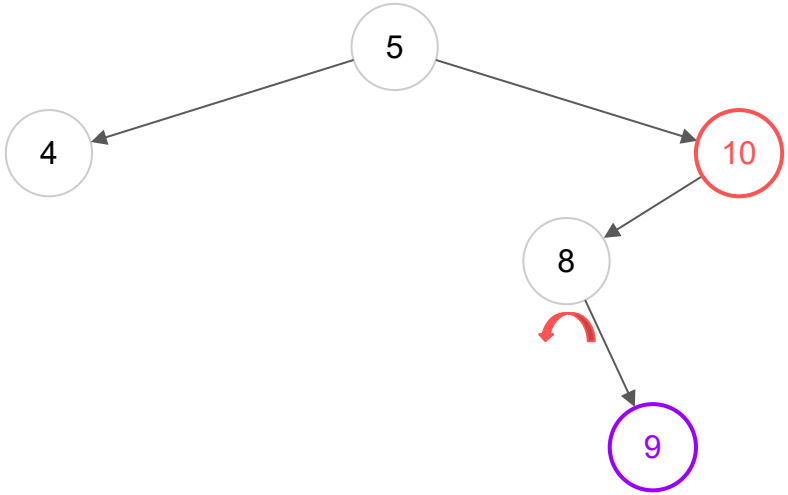
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

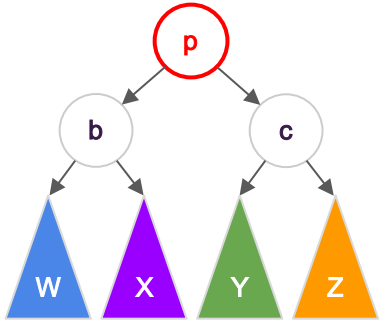
Imbalance at 10

- Case 2
- Rotate left at 8



P2 – Step 9

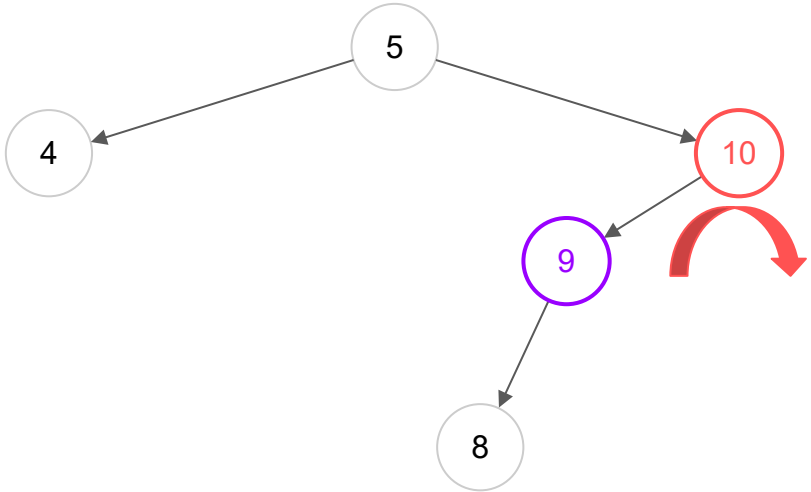
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

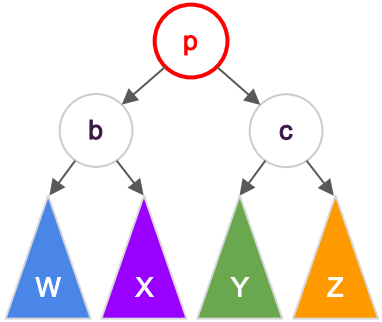
Imbalance at 10

- Case 2
- Rotate left at 8
- Rotate right at 10



P2 – Step 10

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

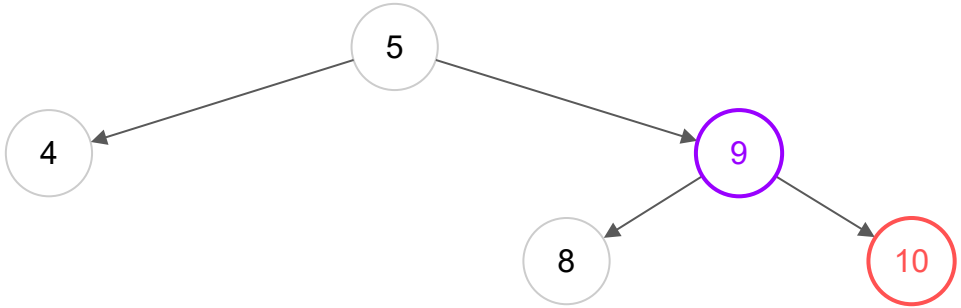


Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

Imbalance at 10

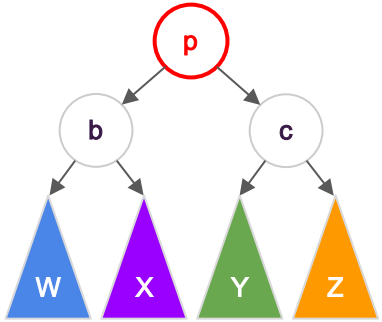
- Case 2
- Rotate left at 8
- Rotate right at 10

No more imbalances



P2 – Step 11

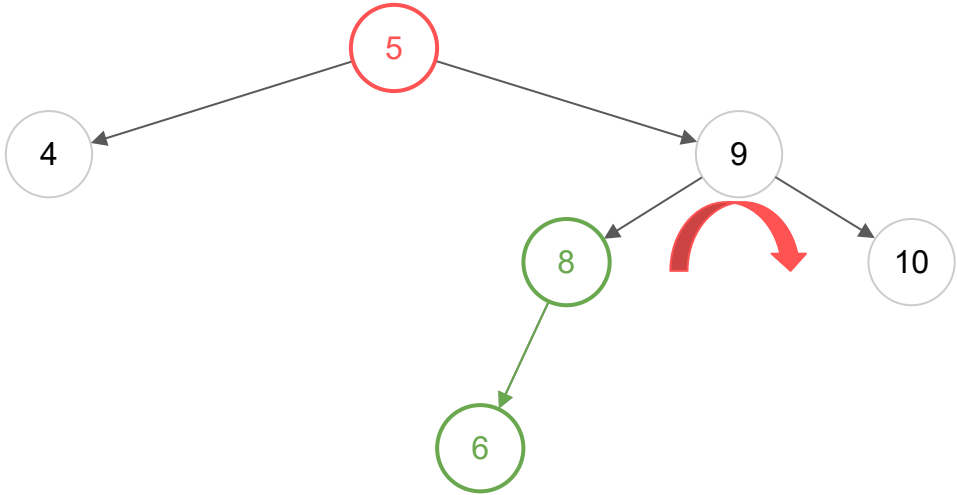
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

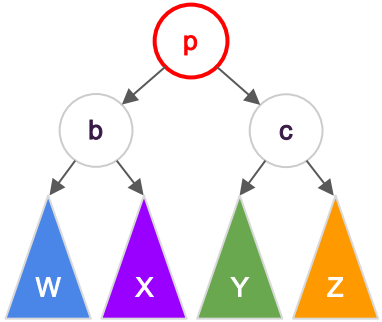
Imbalance at 5

- Case 3
- Rotate right at 9



P2 – Step 12

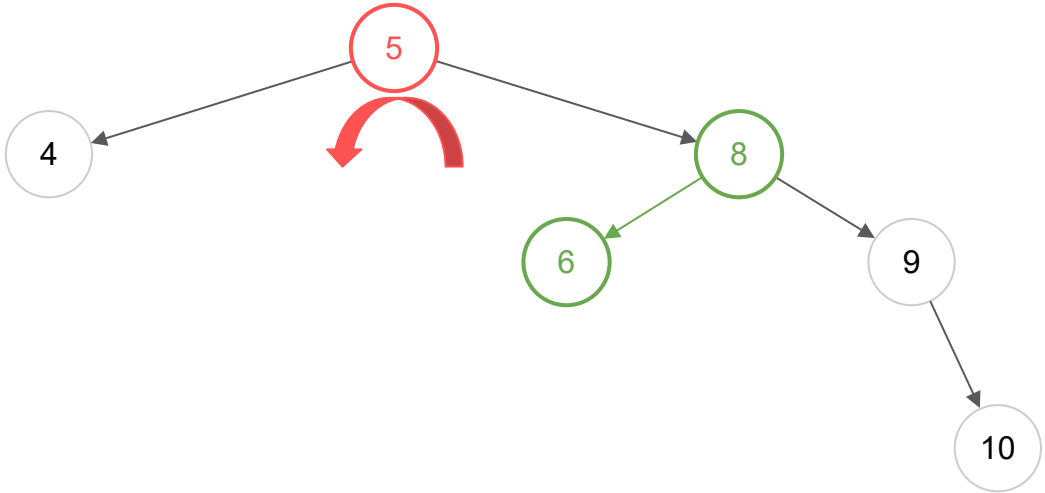
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

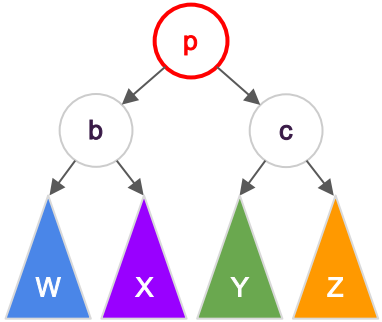
Imbalance at 5

- Case 3
- Rotate right at 9
- Rotate left at 5



P2 – Step 13

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

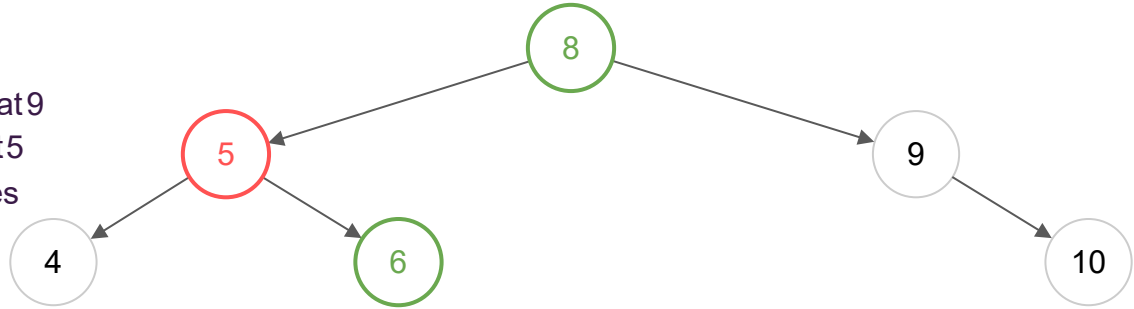


Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

Imbalance at 5

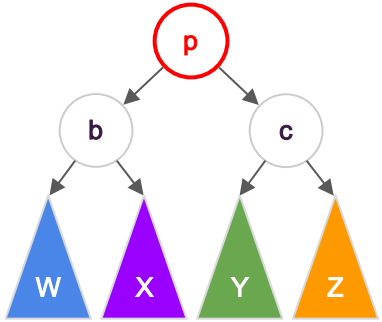
- Case 3
- Rotate right at 9
- Rotate left at 5

No more imbalances



P2 – Step 14 (pt 1)

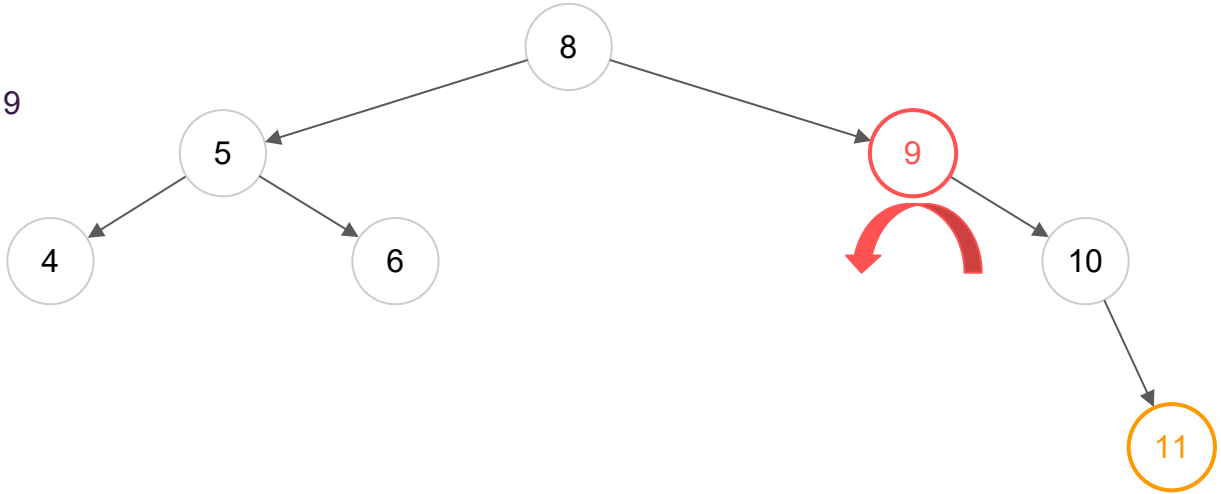
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

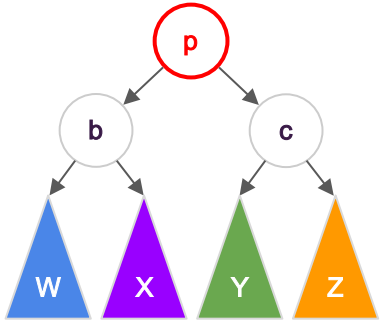
Imbalance at 9

- Case 4
- Rotate left at 9



P2 – Step 14 (pt 2)

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

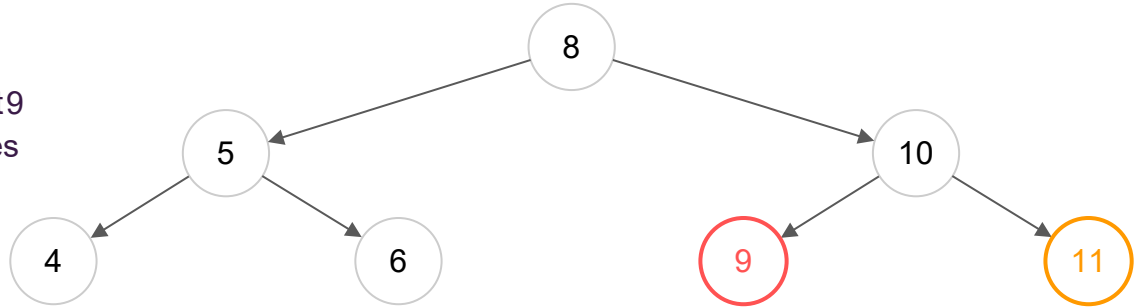


Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

Imbalance at 9

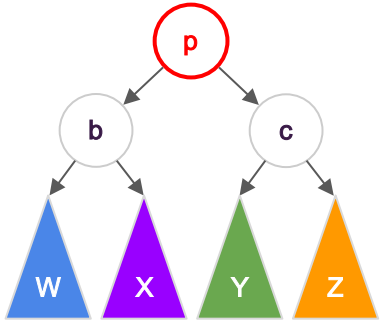
- Case 4
- Rotate left at 9

No more imbalances



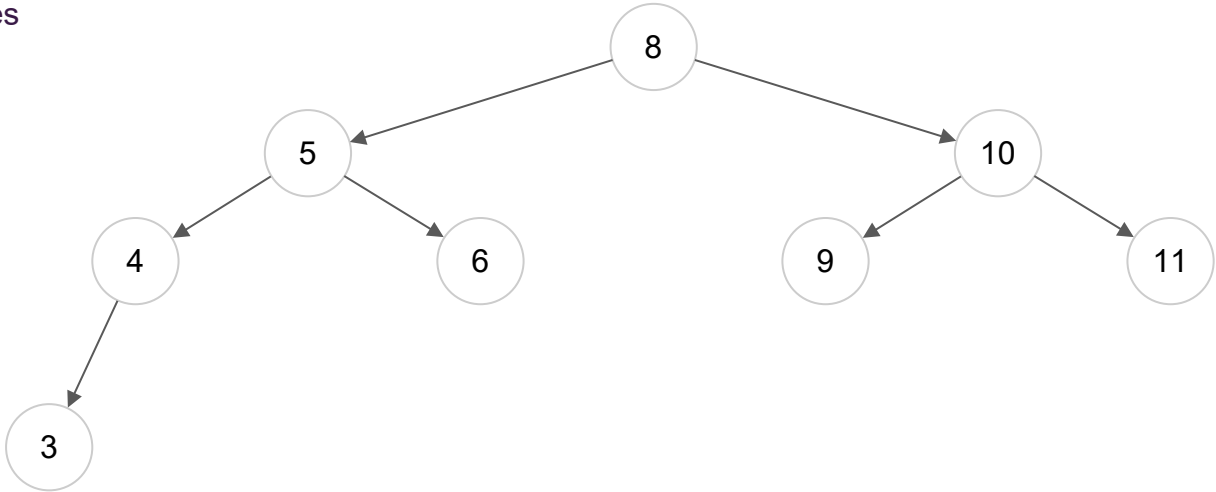
P2 – Step 15

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



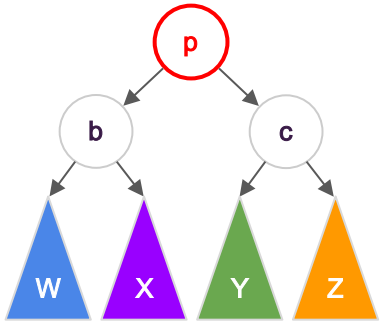
Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

No imbalances



P2 – Step 16

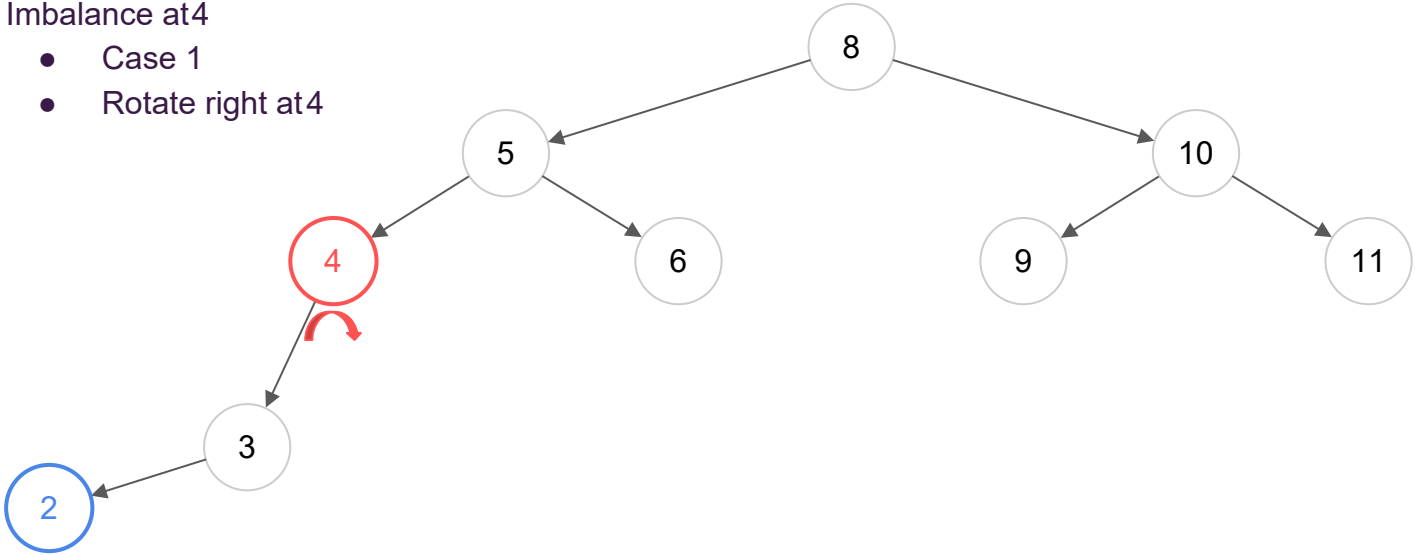
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

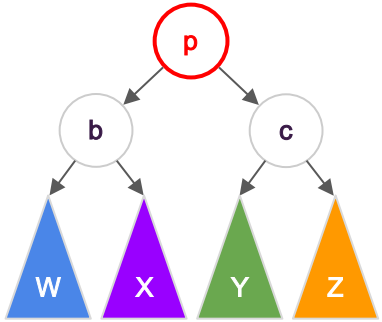
Imbalance at 4

- Case 1
- Rotate right at 4



P2 – Step 17

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

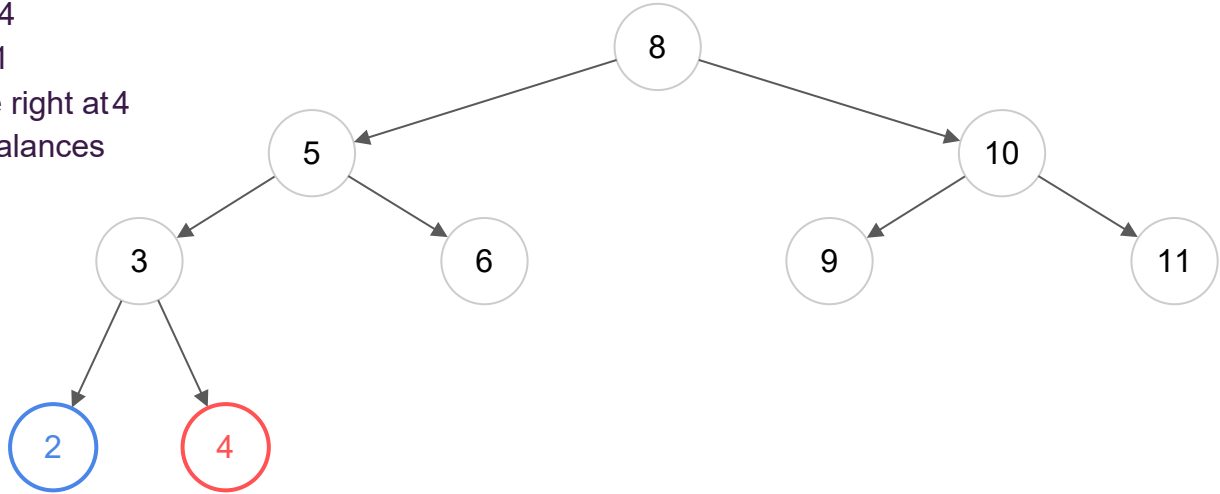


Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

Imbalance at 4

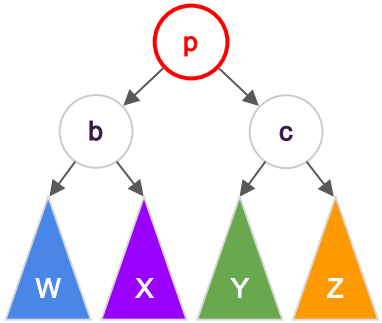
- Case 1
- Rotate right at 4

No more imbalances



P2 – Step 18

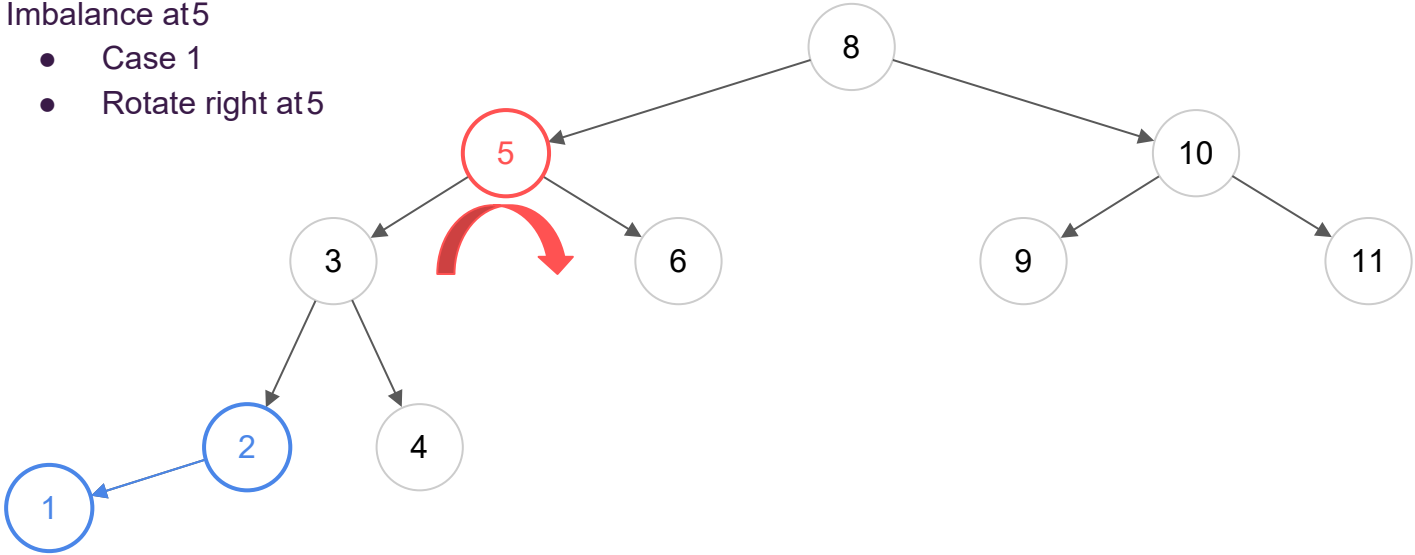
Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

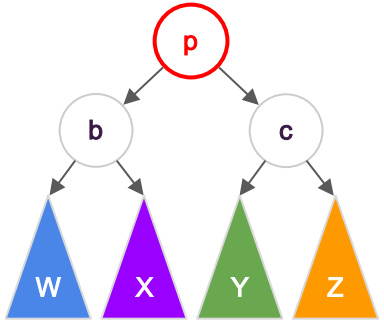
Imbalance at 5

- Case 1
- Rotate right at 5



P2 – Step 19

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation

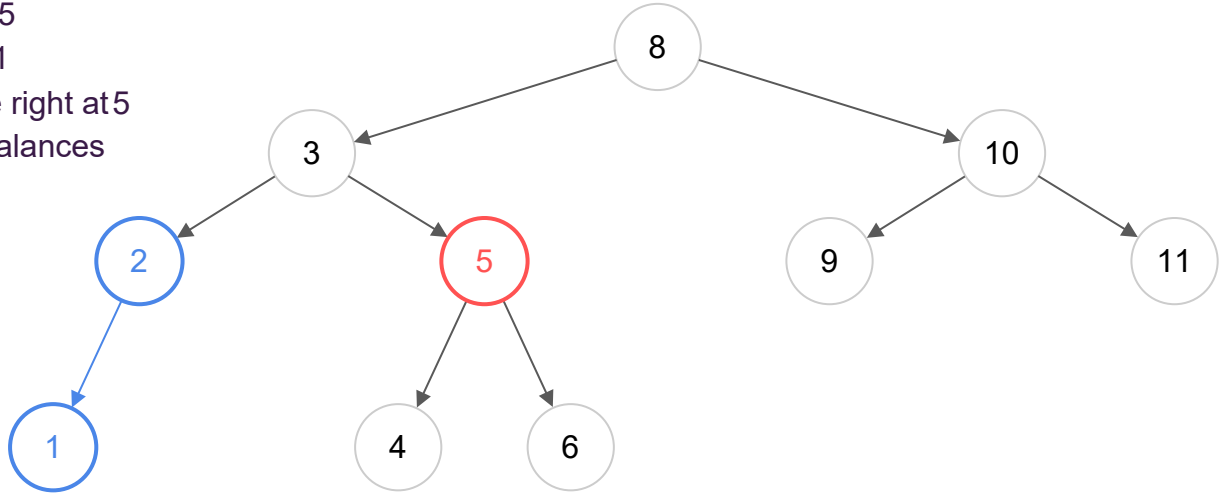


Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

Imbalance at 5

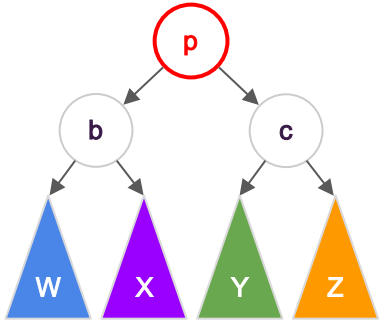
- Case 1
- Rotate right at 5

No more imbalances



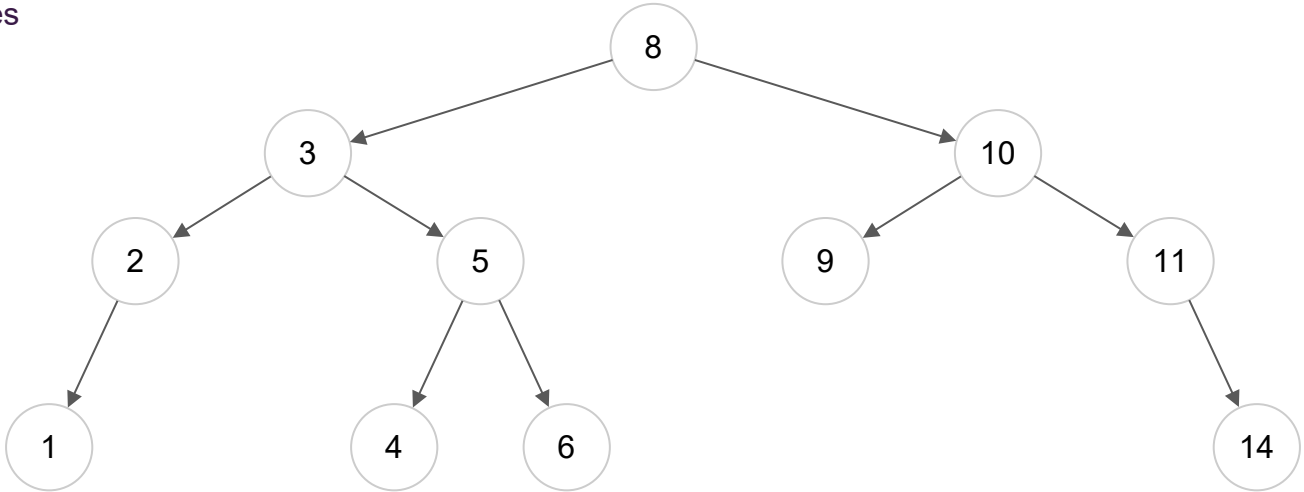
P2 – Step 20

Case	Insert Location	Tree Rotation(s)
1	Left of Left (W)	Single right rotation
2	Right of Left (X)	Double left-right rotation
3	Left of Right (Y)	Double right-left rotation
4	Right of Right (Z)	Single left rotation



Insert 10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14 into an initially empty AVL Tree

No imbalances



Problem 3

Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

Problem 3: Height = 0

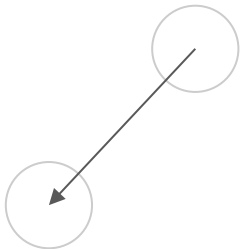
Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

Height = 0:

Problem 3: Height = 1

Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

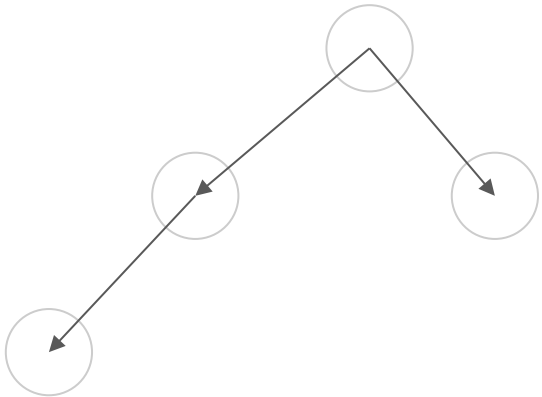
Height = 1:



Problem 3: Height = 2

Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

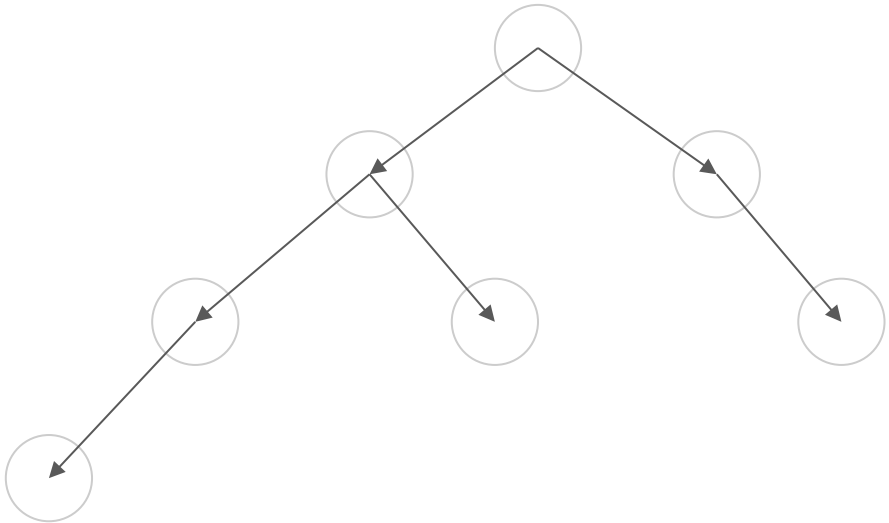
Height = 2:



Problem 3: Height = 3

Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

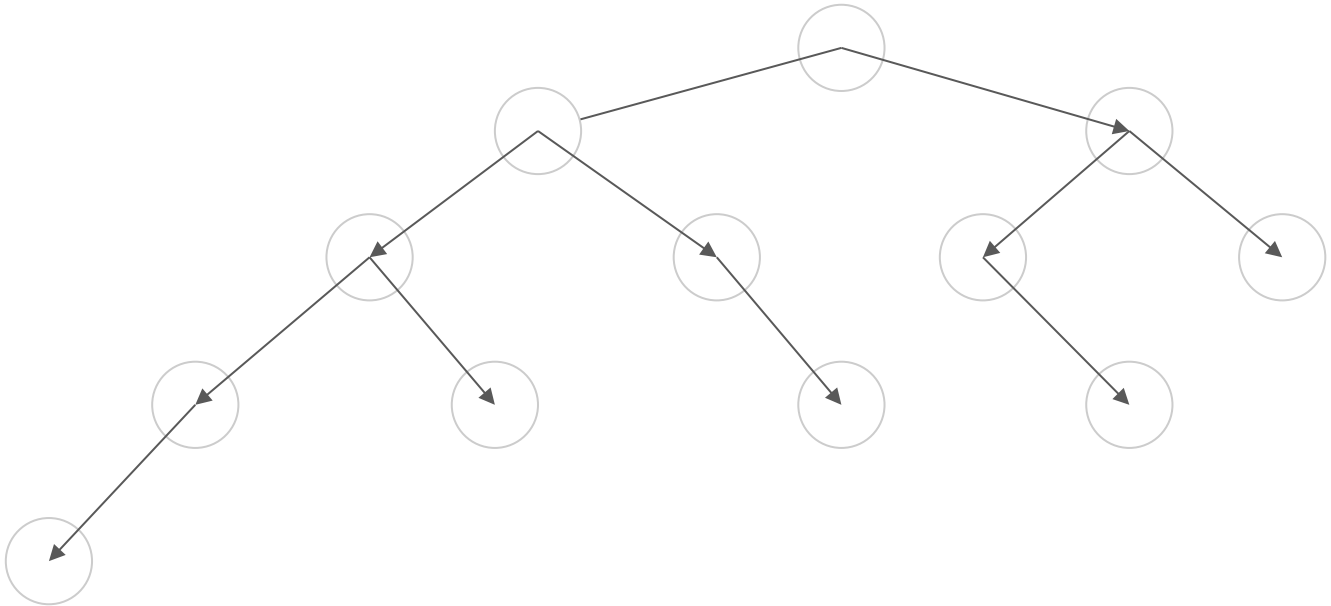
Height = 3:



Problem 3: Height = 4

Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

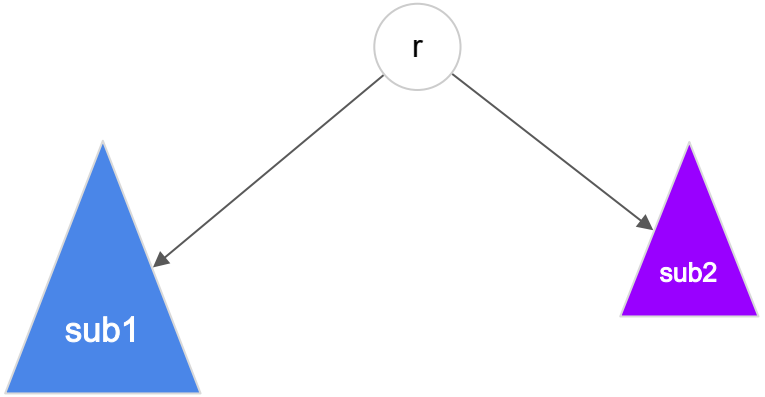
Height = 4:



Problem 3: General

More generally: What is the min number of nodes for a AVL tree with height h?

height(r) = h
height(sub1) = h-1
height(sub2) = h-2



minNum(h) = 1 + minNum(h-1) + minNum(h-2)