PSO 7

Trees, Trees, Trees, etc.

Midterm Tomorrow

Please make sure to go to your assigned room!

Get some sleep

Today on the agenda…

- 2-3 trees

Question 1

(2-3 tree)

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.

Today on the agenda…

- 2-3 trees
- LLRB Trees
  - insertion

Question 2

(Insertion)

(1) Insert $\{15, 21, 7, 24, 0, 26, 3, 28, 29\}$ (in the given order) into an initially empty 2-3 tree.

(2) Insert the same elements (in the same order) into an initially empty Left–Leaning Red–Black tree.

Today on the agenda…

- 2-3 trees
- LLRB Trees
    - Insertion
    - deletion

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?

Today on the agenda…

- 2-3 trees
- LLRB Trees
  - Insertion
  - Deletion
- B Trees

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

Today on the agenda…

- 2-3 trees
- LLRB Trees
  - Insertion
  - Deletion
- B Trees
- RB Trees facts

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Today on the agenda…

- 2-3 trees
- LLRB Trees
  - Insertion
  - Deletion
- B Trees
- RB Trees facts
- AVL Trees

Question 6

(AVL tree) An AVL tree is a binary search tree that is *height balanced*: for each node x, the heights of the left and right subtrees of x differ by at most 1.

Show that an AVL tree with n nodes has height $h = \mathcal{O}(\log n)$.

Hint1: show that an AVL tree of height h has at least $F_h - 1$ nodes, where F_h is the h-th Fibonacci number. Hint2: you may use the following fact of the Fibonacci number:

$$F_h = \left\lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \right\rfloor \quad \text{, with} \quad \phi = \frac{\sqrt{5}+1}{2}.$$

Question 1

(2-3 tree)

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.

First, what is a 2-3 tree?

Question 1

(2-3 tree)

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.

First, what is a 2-3 tree?

T is a 2-3 tree if:

- T empty
- T is a **2 node**
- T is a **3 node**

Question 1

(2-3 tree)

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.

First, what is a 2-3 tree?

T is a 2-3 tree if:

- <u>T empty</u>
- T is a **2 node**
- T is a **3 node**

T –>

Question 1

(2-3 tree)

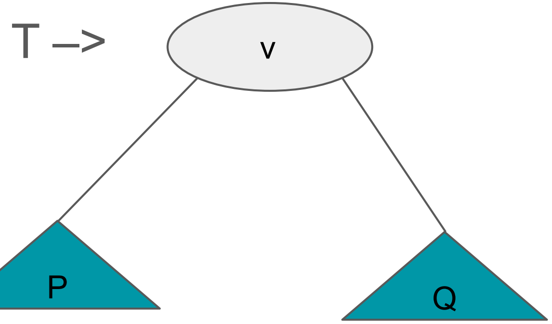
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.

First, what is a 2-3 tree?

T is a 2-3 tree if:

- T empty
- T is a **2 node**
- T is a **3 node**



P,Q are 2-3 subtrees where

- P,Q have the *same height*
- P < v < Q

Question 1

(2-3 tree)

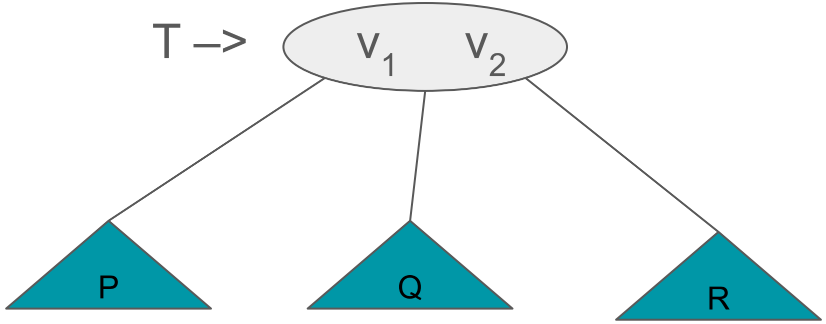
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.

First, what is a 2-3 tree?

T is a 2-3 tree if:

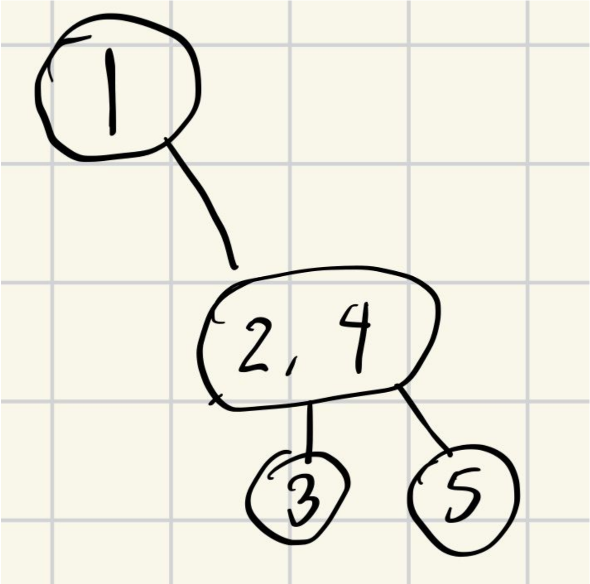
- T empty
- T is a **2 node**
- T is a **3 node**



P,Q,R are 2-3 subtrees where
- P,Q have the *same height*
- $P < v_1 < Q < v_2 < R$

Let's try to make some 2-3 trees

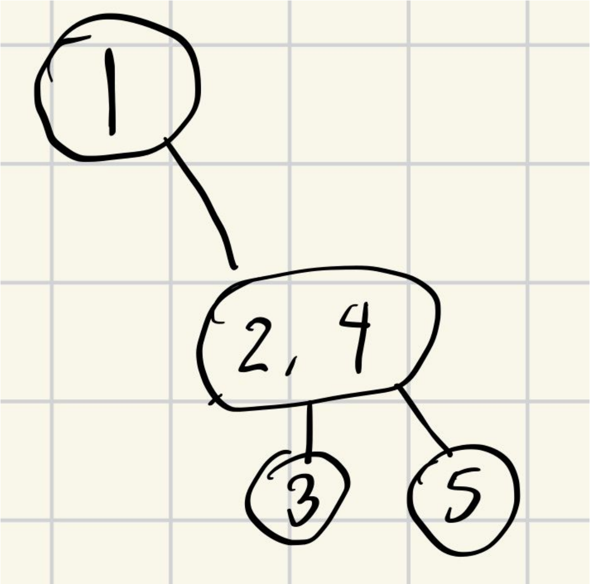
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



Is this a 2-3 tree?

Let’s try to make some 2-3 trees

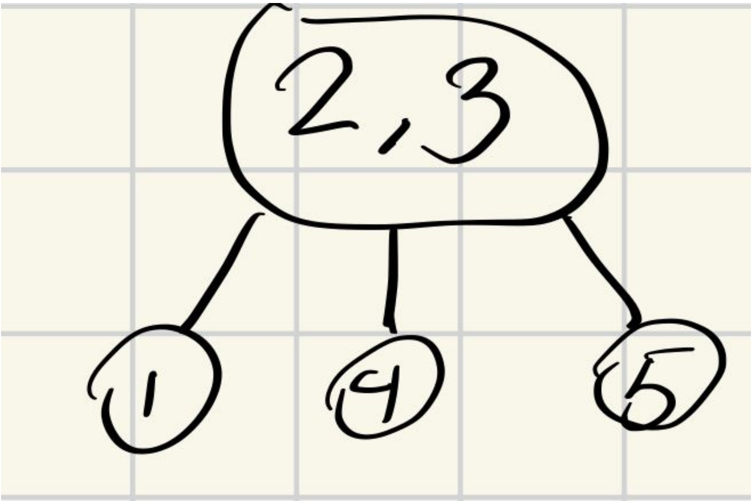
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



Is this a 2-3 tree?
No the root and inner node is missing its left child

Let’s try to make some 2-3 trees

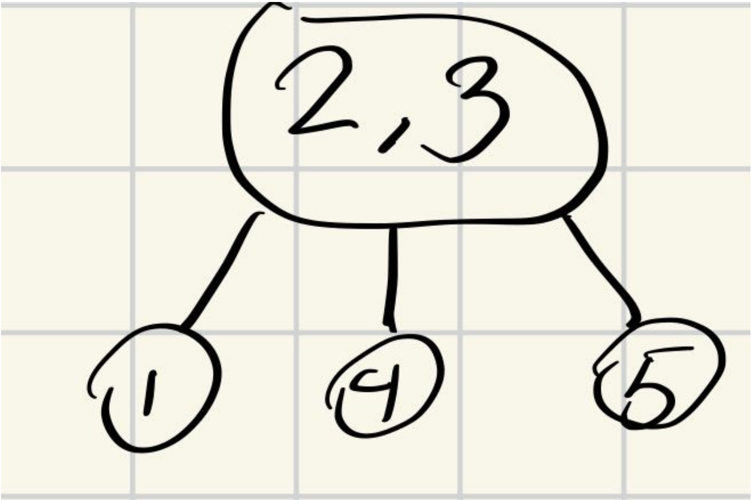
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



How about this one?

Let’s try to make some 2-3 trees

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

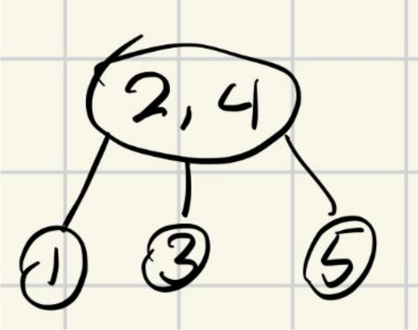


How about this one?
No,

$1 < 2 \not< 4 \not< 3 < 5$

Let’s try to make some 2-3 trees

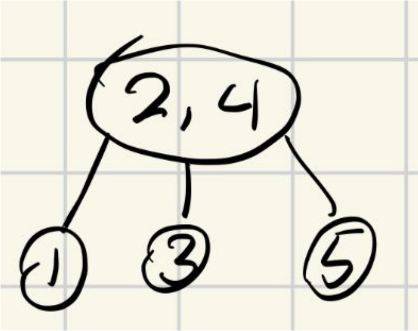
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



Surely this one is bad too, right?

Let's try to make some 2-3 trees

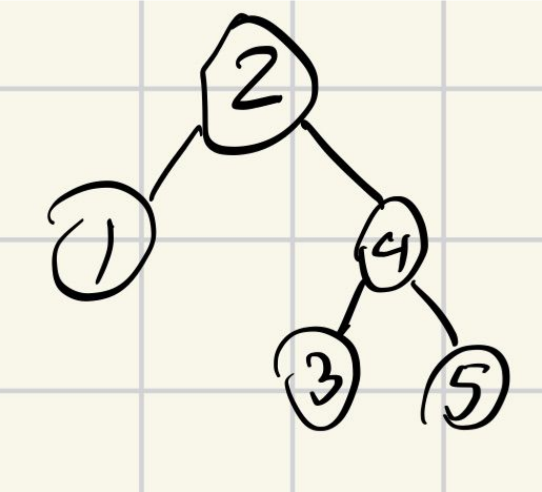
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



Surely this one is bad too, right?
This one is ok

Let’s try to make some 2-3 trees

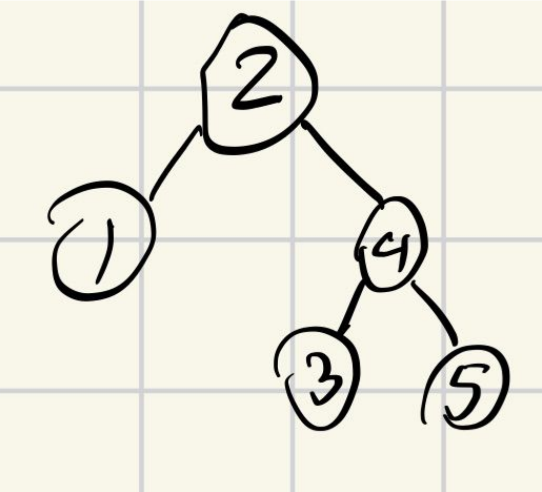
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



This one?

Let’s try to make some 2-3 trees

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

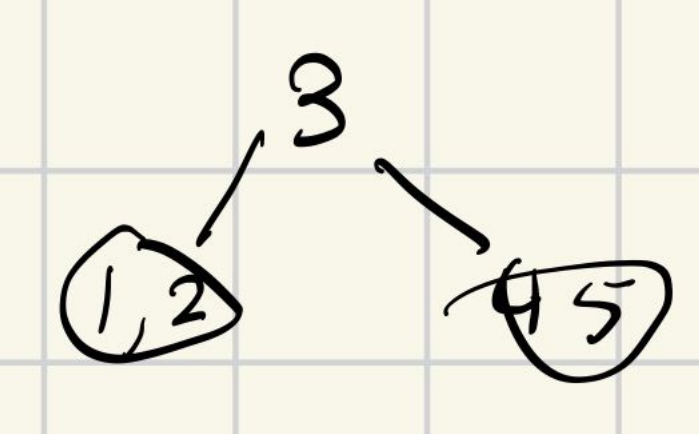


This one?
No, not all subtrees have the same height

(This *is* a BST though)

Let’s try to make some 2-3 trees

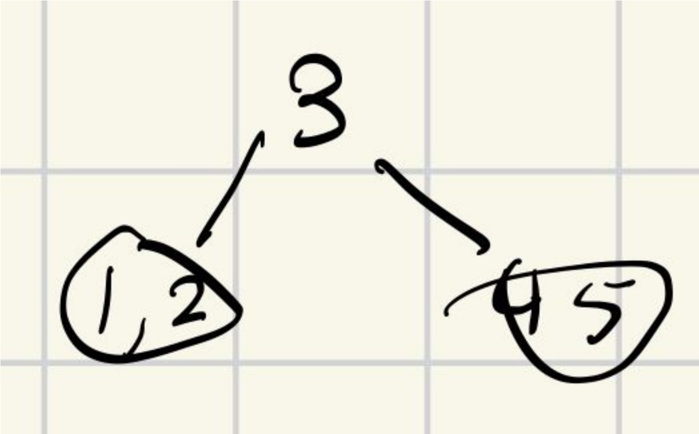
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



Last one I promise. Is this a 2-3 tree?

Let's try to make some 2-3 trees

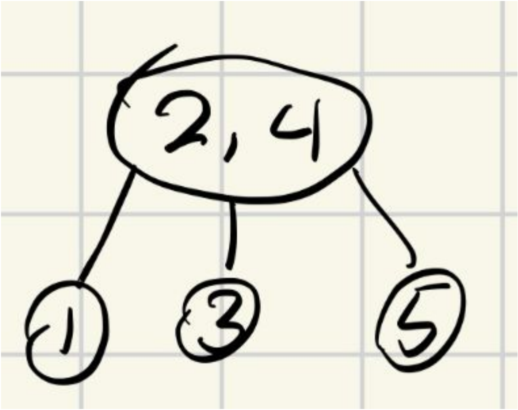
(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



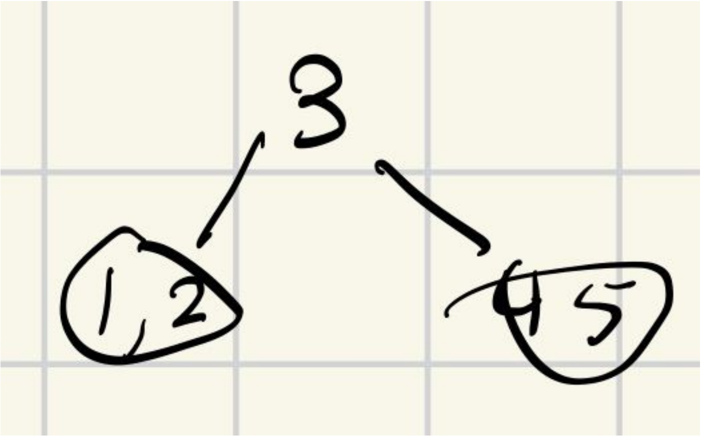
Last one I promise. Is this a 2-3 tree?
Yes nothing wrong here

The only 2-3 trees

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.



Only 2-3 tree with a 3-node root



Only 2-3 tree with a 2-node root

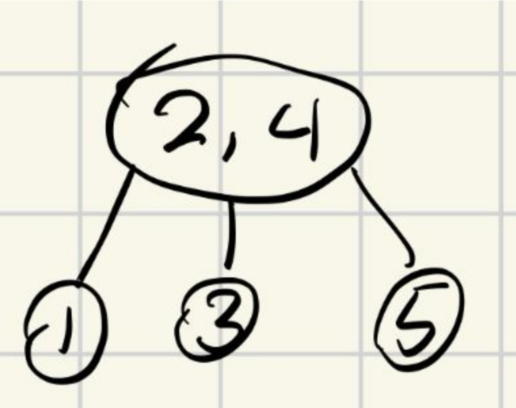
Can you see why?

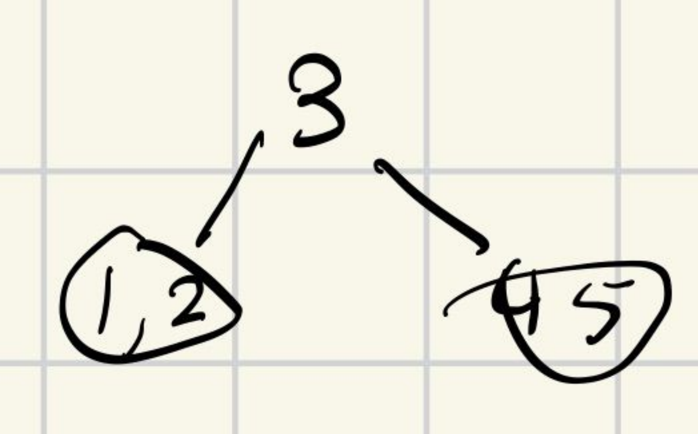
Question 1

(2-3 tree)

(1) How many 2–3 trees exist storing the keys $\{1, 2, 3, 4, 5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.





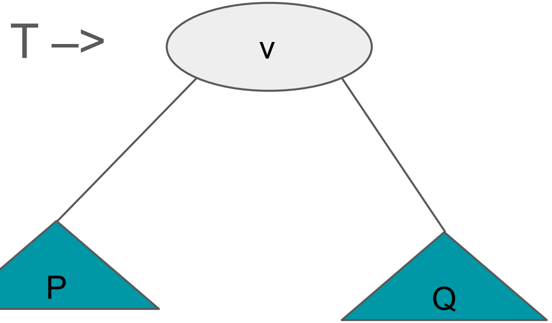
The 2-3 trees from part 1. Is Bruno right?

Question 1

(2-3 tree)

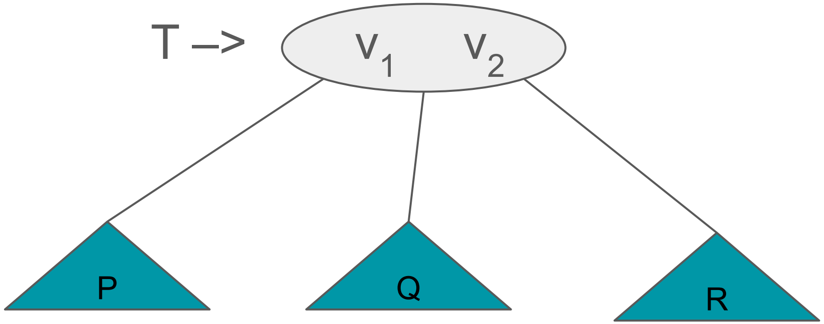
(1) How many 2–3 trees exist storing the keys $\{1,2,3,4,5\}$? Explain your answer.

(2) Bruno says the largest key in a 2–3 tree sometimes can be found in an interior node. Is Bruno right? Explain your answer.



P,Q are 2-3 subtrees where

- $P < \mathbf{v < Q}$



P,Q,R are 2-3 subtrees where

- $P < v_1 < Q < \mathbf{v_2 < R}$

The 2-3 trees from part 1. Is Bruno right?

Bruno is wrong. Recall the general structure of a 2-3 tree

Greatest element always in rightmost leaf

Question 2

(Insertion)

(1) Insert $\{15, 21, 7, 24, 0, 26, 3, 28, 29\}$ (in the given order) into an initially empty 2-3 tree.

(2) Insert the same elements (in the same order) into an initially empty Left–Leaning Red–Black tree.

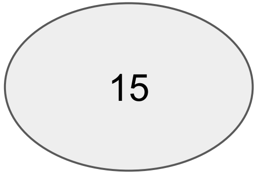
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

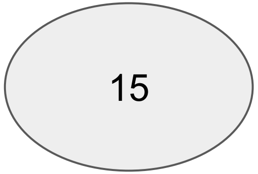
Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



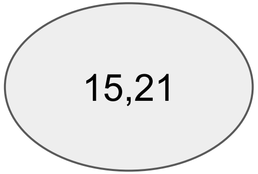
Insert: 15,<u>21</u>,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



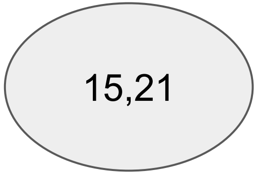
Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



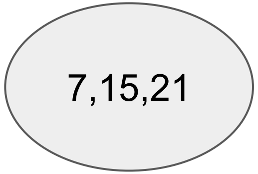
Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Insert: 15,21,<u>7</u>,24,0,26,3,28,29

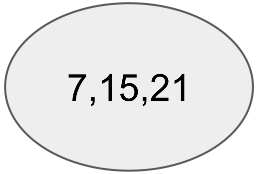
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



This is now a 4-node, need to fix.
Intuition: Pull up by the middle

Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



This is now a 4-node, need to fix.
Intuition: Pull up by the middle

Ur CPU core
fixing ur broken tree

Insert: 15,21,<u>7</u>,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

This is now a 4-node, need to fix.
Intuition: Pull up by the middle

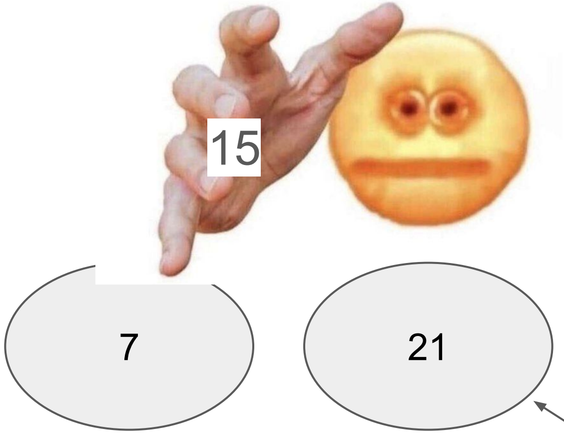
Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

This is now a 4-node, need to fix.
Intuition: Pull up by the middle

Insert: 15,21,<u>7</u>,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

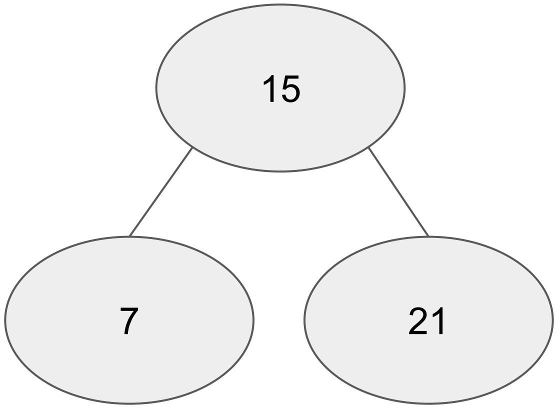


This is now a 4-node, need to fix.
Intuition: Pull up by the middle

The nodes split (out of fear)

Insert: 15,21,7,24,0,26,3,28,29

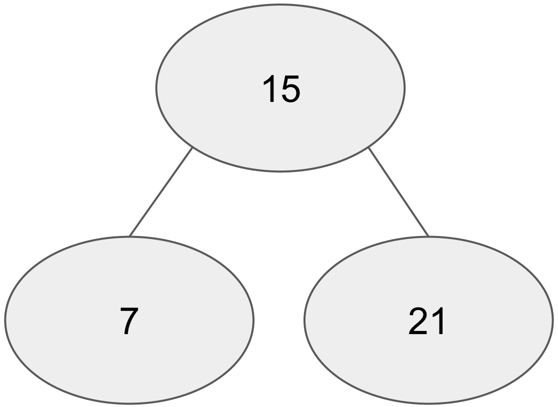
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



This is now a 4-node, need to fix.
Intuition: Pull up by the middle

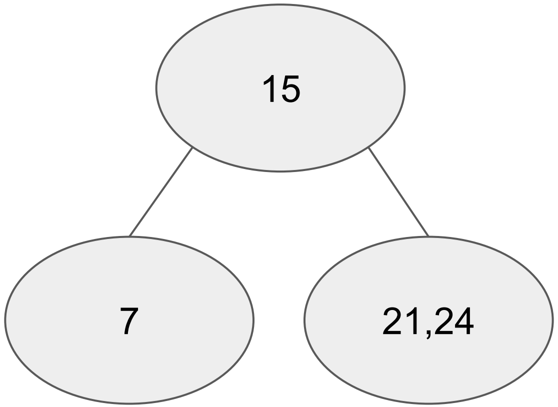
Insert: 15,21,7,<u>24</u>,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



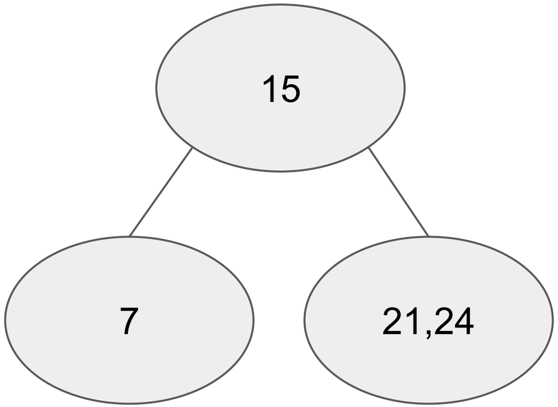
Insert: 15,21,7,<u>24</u>,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



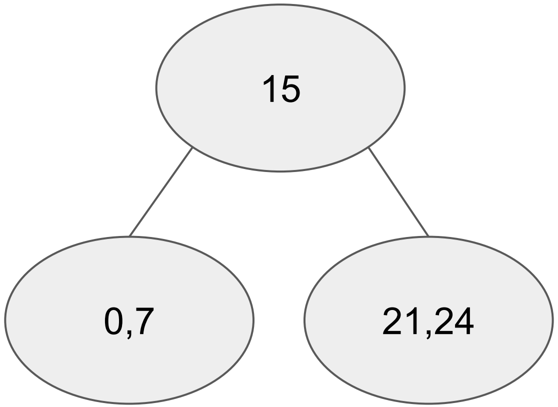
Insert: 15,21,7,24,<u>0</u>,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



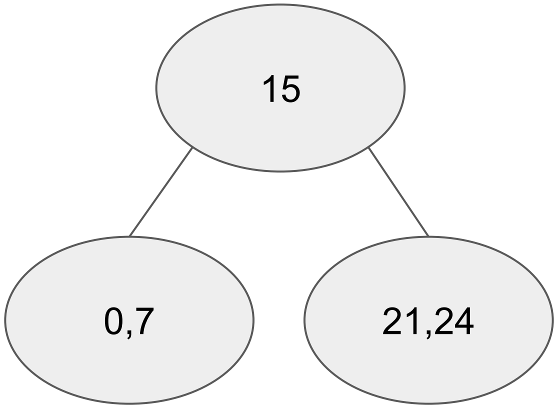
Insert: 15,21,7,24,<u>0</u>,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



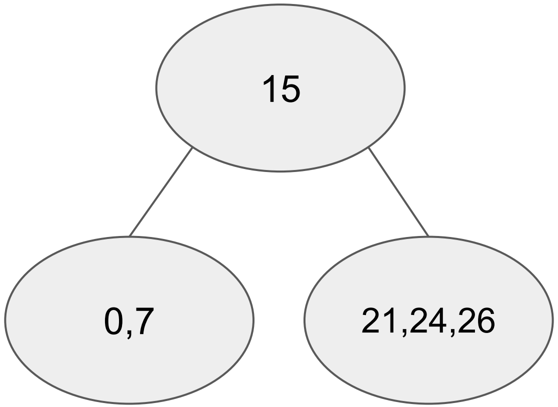
Insert: 15,21,7,24,0,<u>26</u>,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Insert: 15,21,7,24,0,<u>26</u>,3,28,29

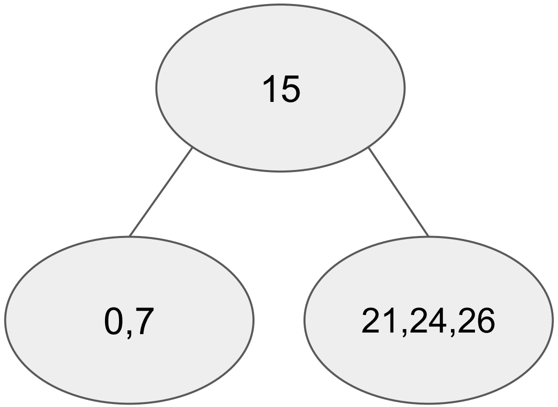
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



A 4-node…

Insert: 15,21,7,24,0,<u>26</u>,3,28,29

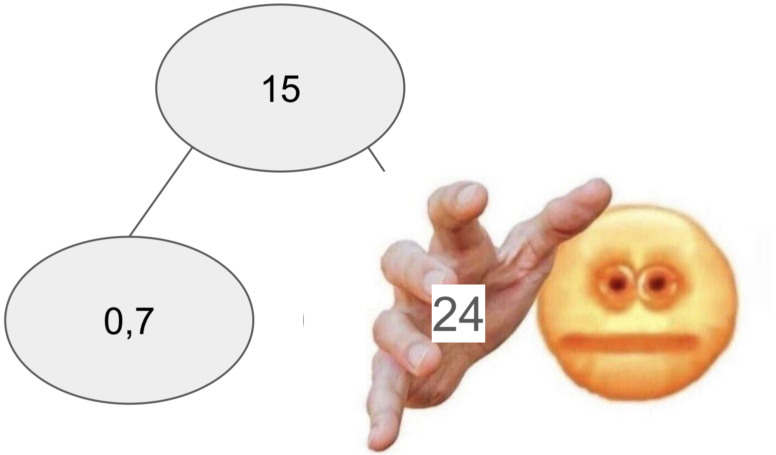
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



A 4-node…

Insert: 15,21,7,24,0,<u>26</u>,3,28,29

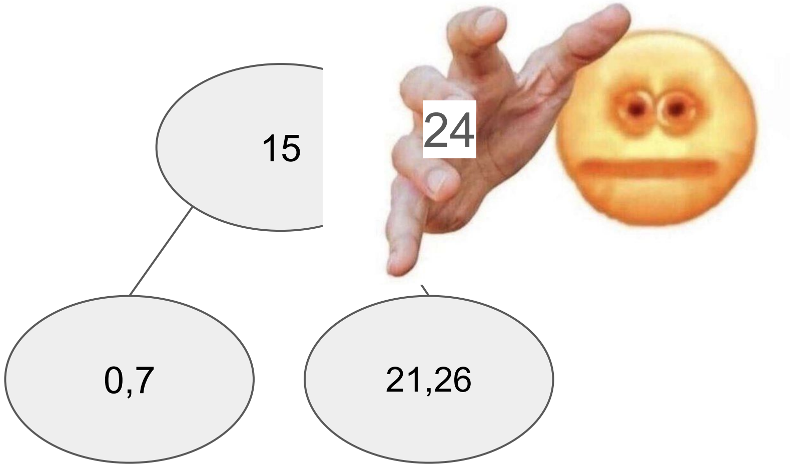
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



A 4-node… “pull it up”

Insert: 15,21,7,24,0,26,3,28,29

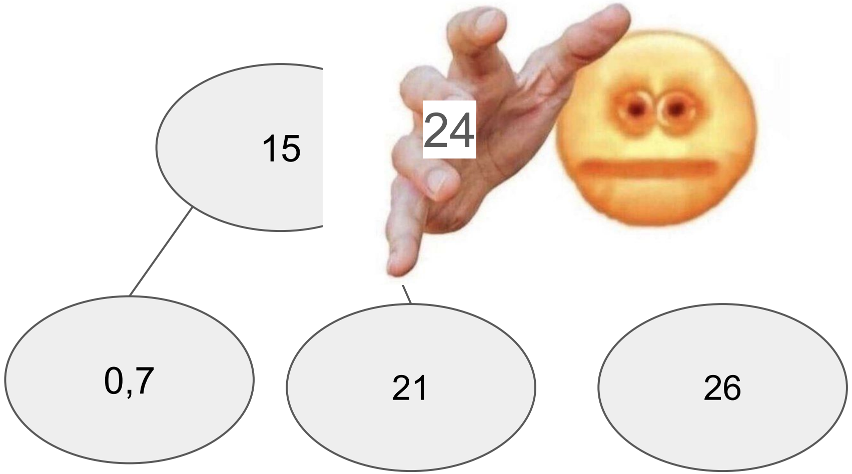
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



A 4-node… "pull it up"

Insert: 15,21,7,24,0,<u>26</u>,3,28,29

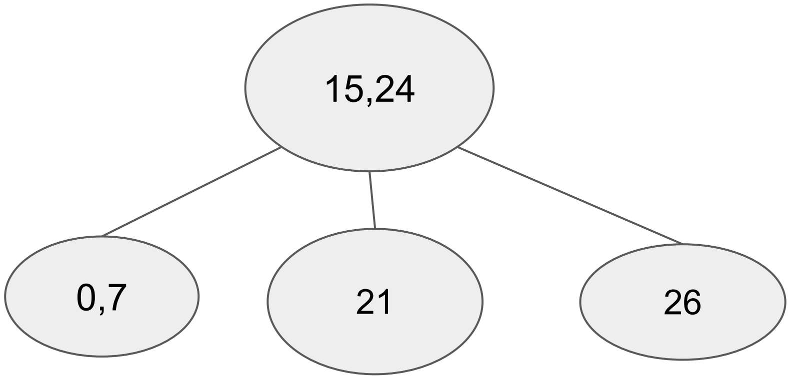
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



A 4-node… “pull it up”

Insert: 15,21,7,24,0,<u>26</u>,3,28,29

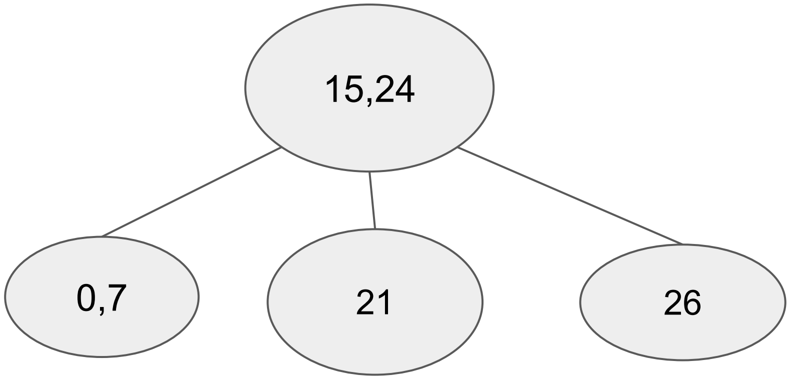
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



A 4-node… “pull it up”

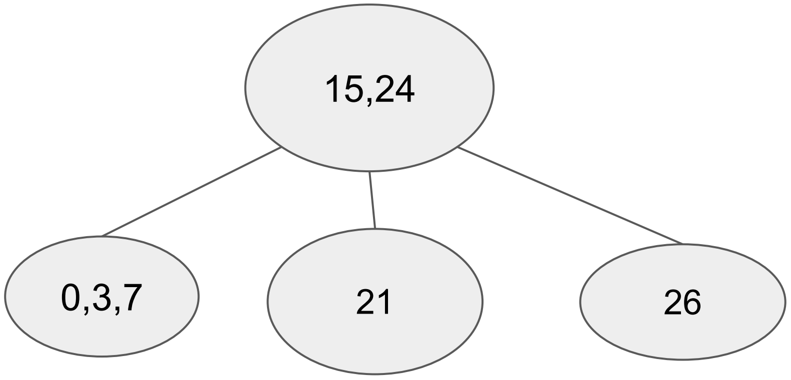
Insert: 15,21,7,24,0,26,<u>3</u>,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Insert: 15,21,7,24,0,26,3,28,29

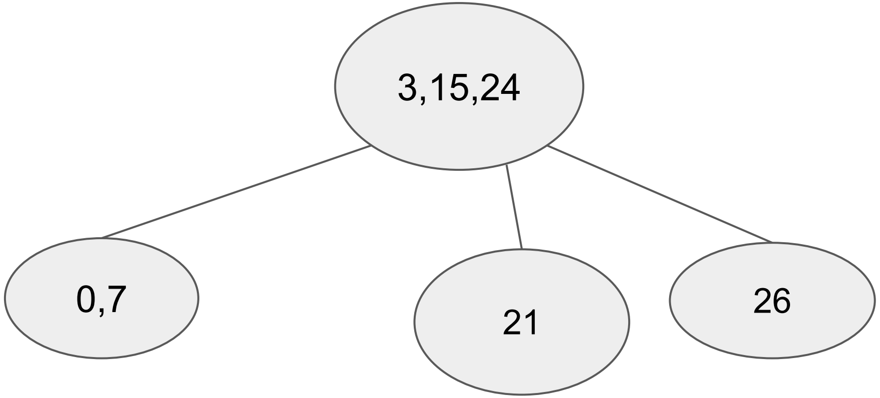
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Pull up the 3

Insert: 15,21,7,24,0,26,3,28,29

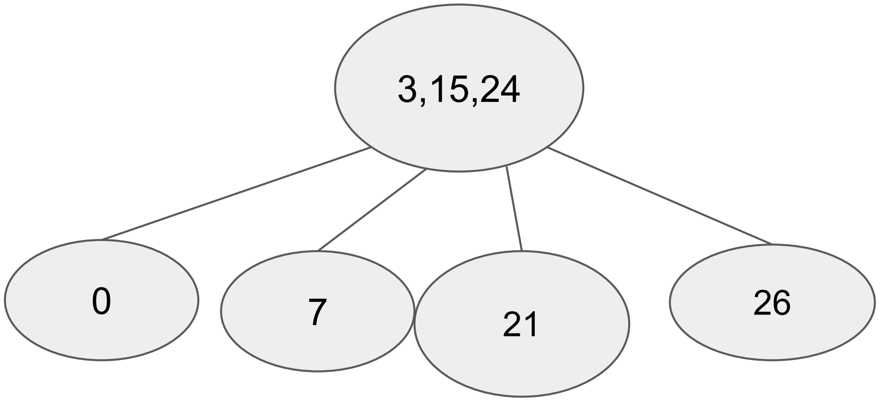
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Pull up the 3
Split the node

Insert: 15,21,7,24,0,26,<u>3</u>,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

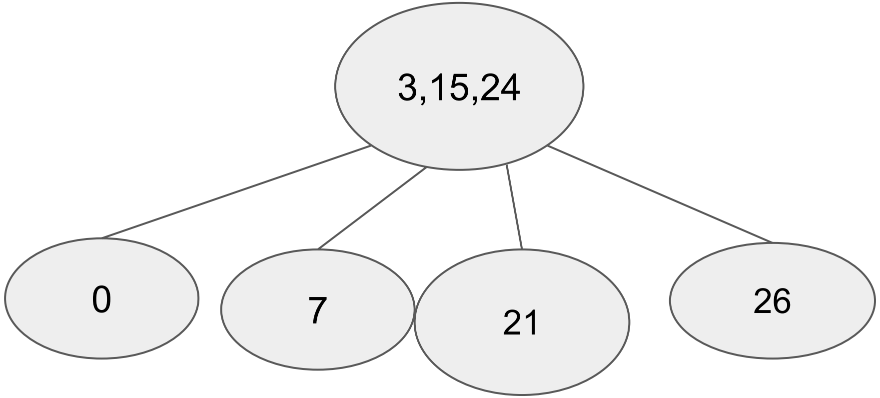


Pull up the 3
Split the node

Now the root is a 4-node, so
we need to keep pulling up

Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

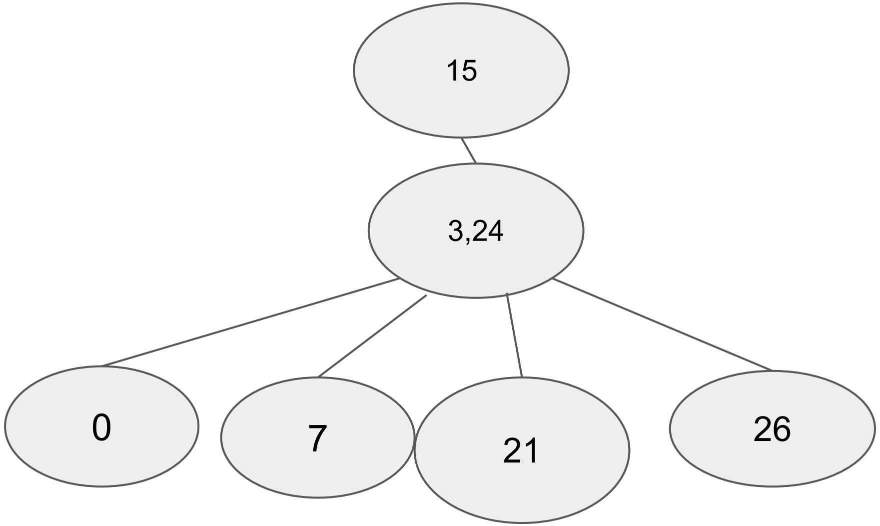


Pull up the 3
Split the node

Now the root is a 4-node, so
we need to keep pulling up

Insert: 15,21,7,24,0,26,3,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

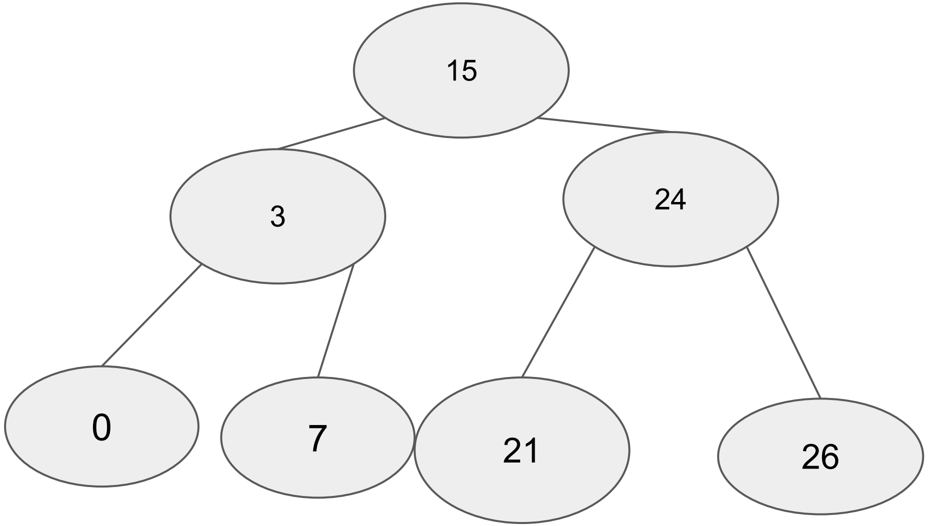


Pull up the 3
Split the node

Now the root is a 4-node, so
we need to keep pulling up

Insert: 15,21,7,24,0,26,<u>3</u>,28,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed

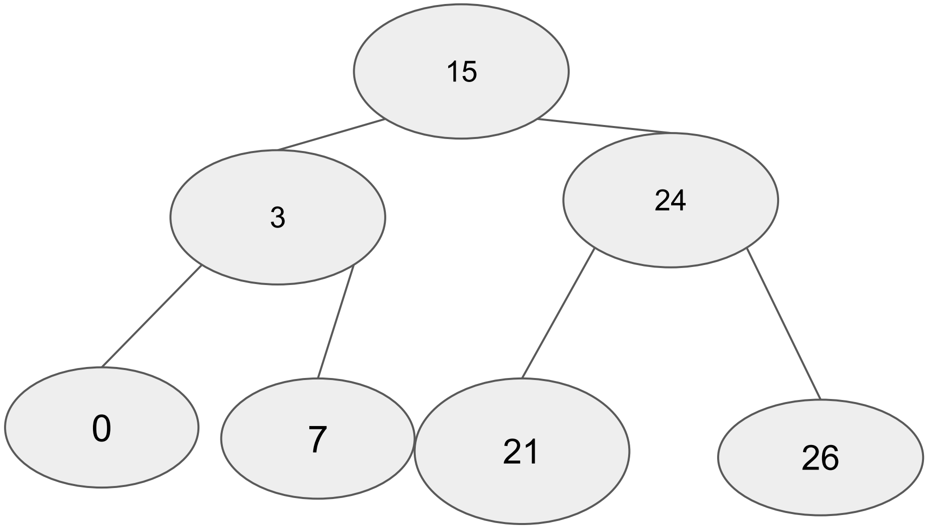


Pull up the 3
Split the node

Now the root is a 4-node, so
we need to keep pulling up

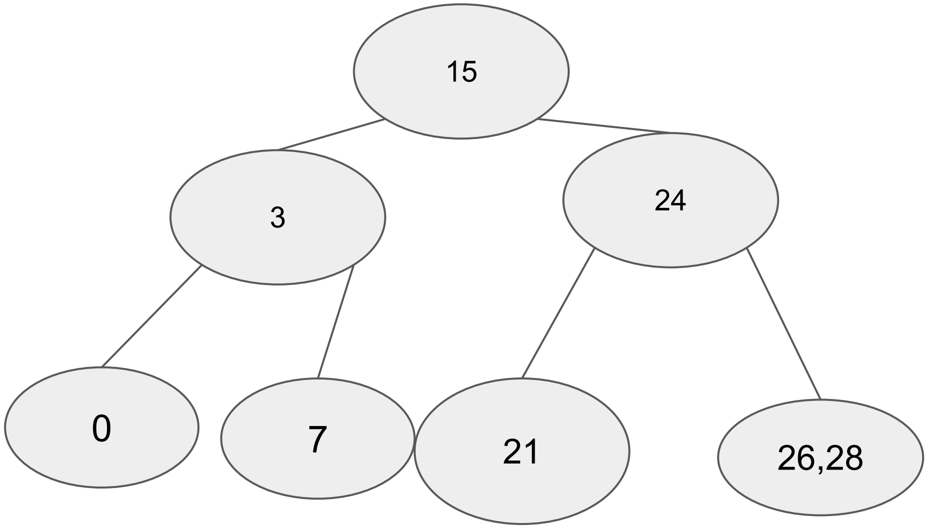
Insert: 15,21,7,24,0,26,3,<u>28</u>,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



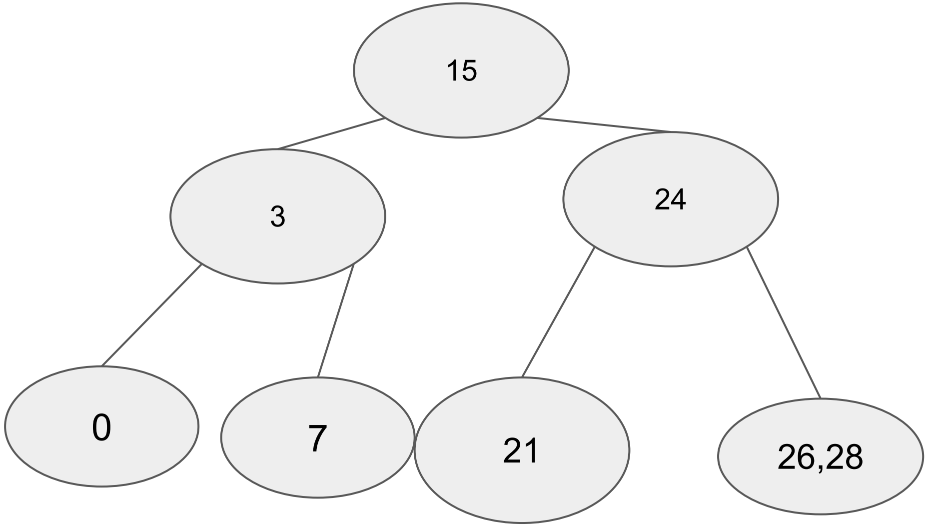
Insert: 15,21,7,24,0,26,3,<u>28</u>,29

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



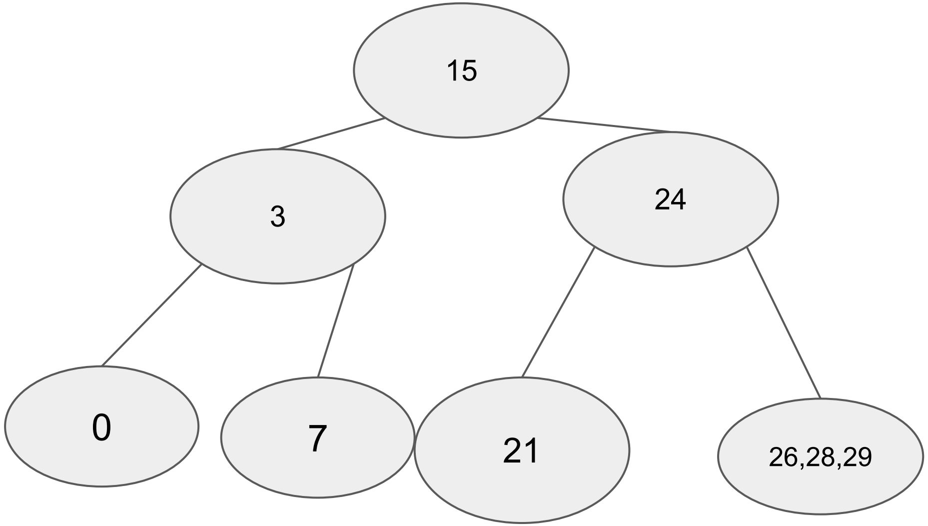
Insert: 15,21,7,24,0,26,3,28,<u>29</u>

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Insert: 15,21,7,24,0,26,3,28,<u>29</u>

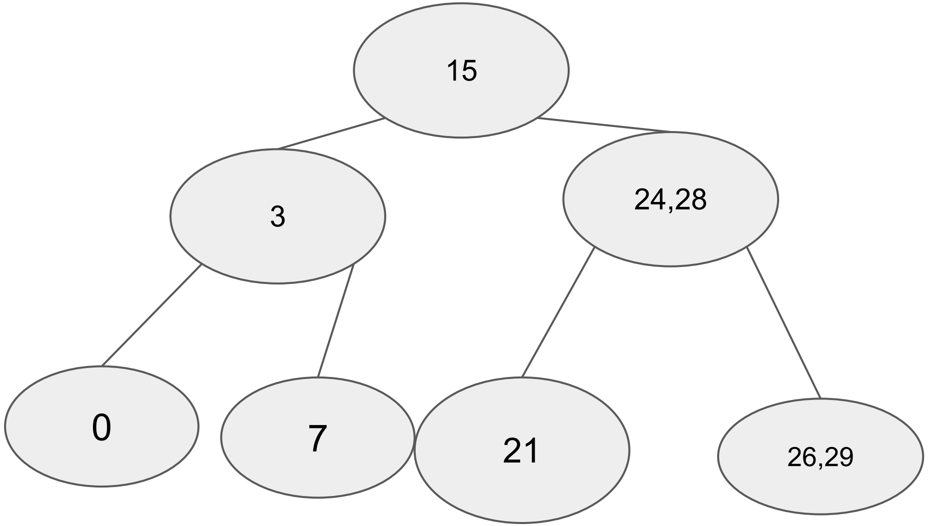
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Pull up by middle

Insert: 15,21,7,24,0,26,3,28,<u>29</u>

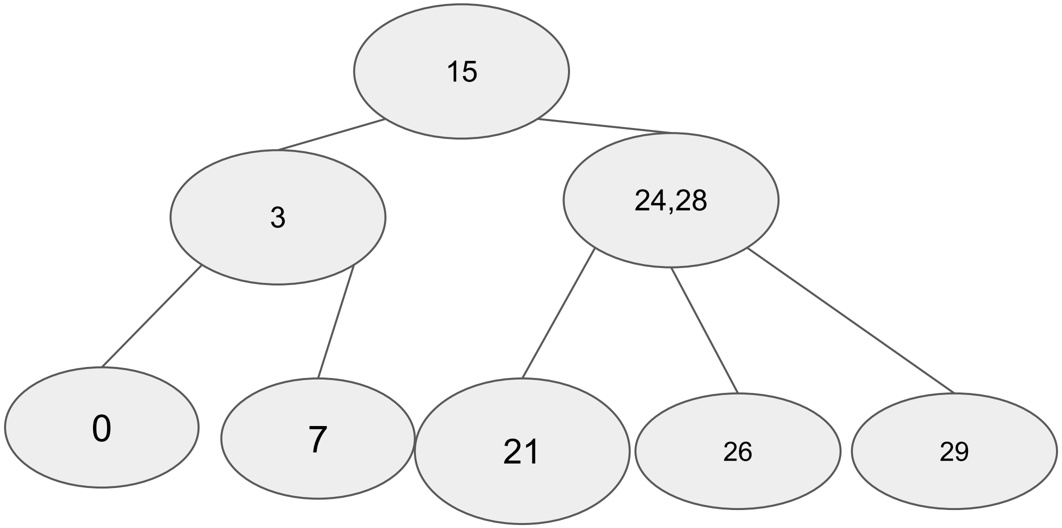
Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Pull up by middle
Split the node

Insert: 15,21,7,24,0,26,3,28,<u>29</u>

Inserting in 2-3: Find leaf the element would be in, add it and *split* if needed



Pull up by middle
Split the node

Question 2

(Insertion)

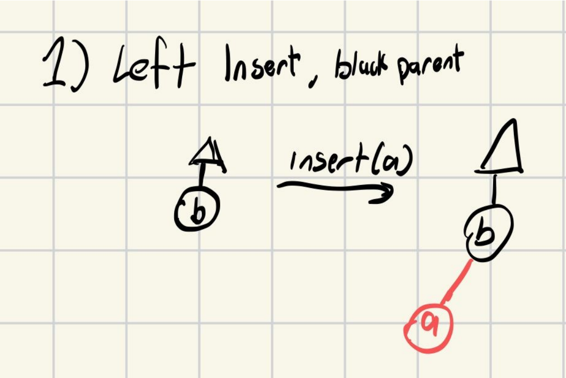
(1) Insert $\{15, 21, 7, 24, 0, 26, 3, 28, 29\}$ (in the given order) into an initially empty 2-3 tree.

(2) Insert the same elements (in the same order) into an initially empty Left–Leaning Red–Black tree.

LLRB Trees, what are they?

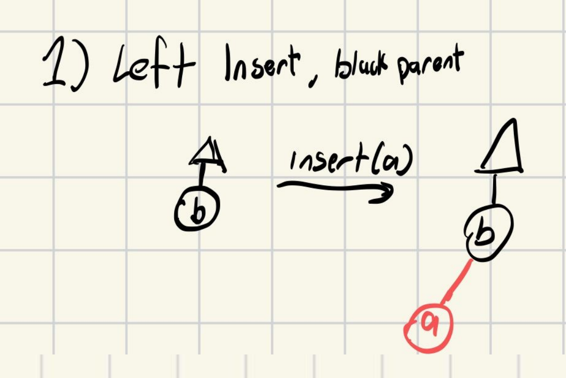
Types of LLRB inserts

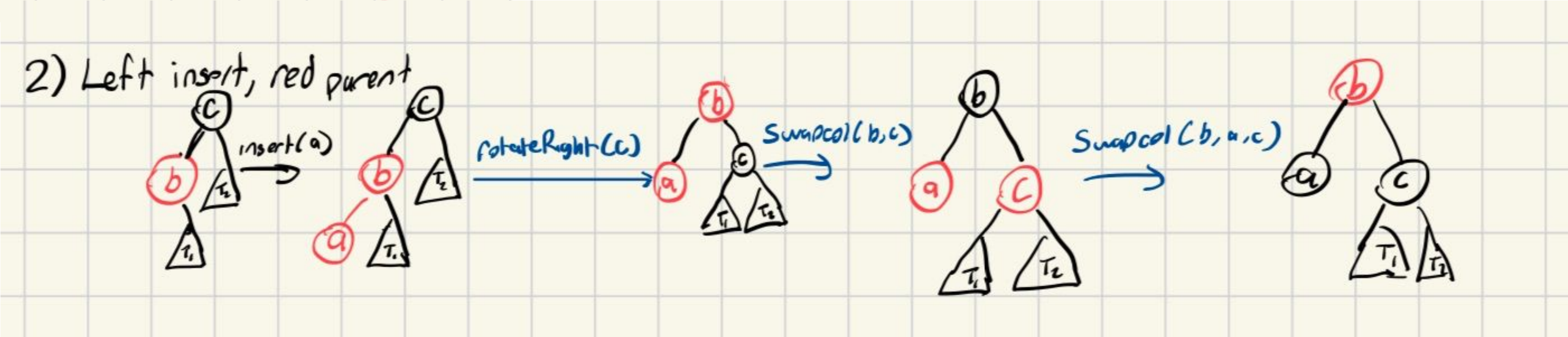
Always insert in a red node



Types of LLRB inserts

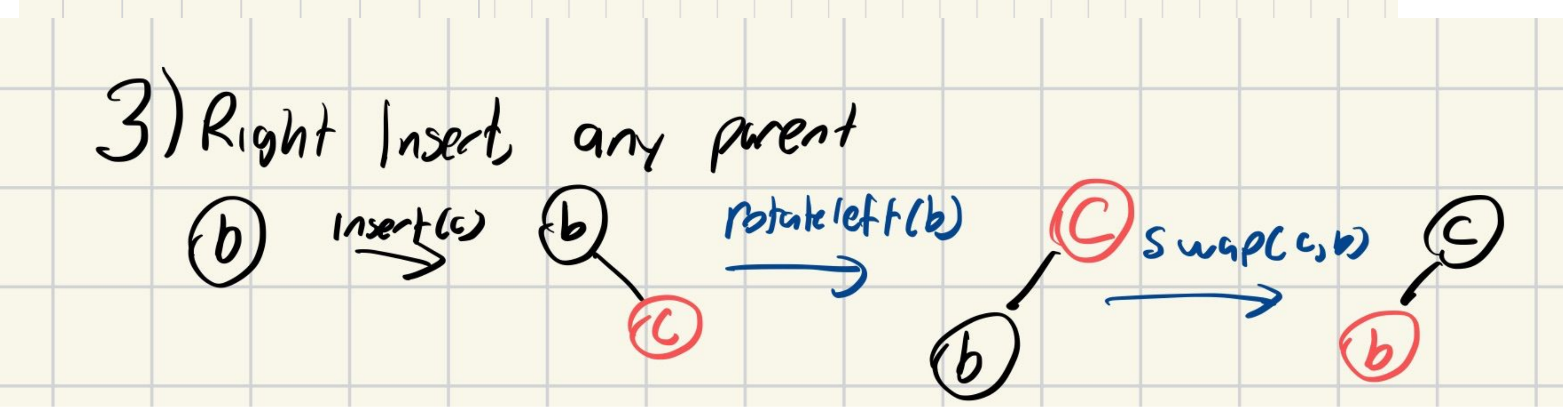
Always insert in a red node





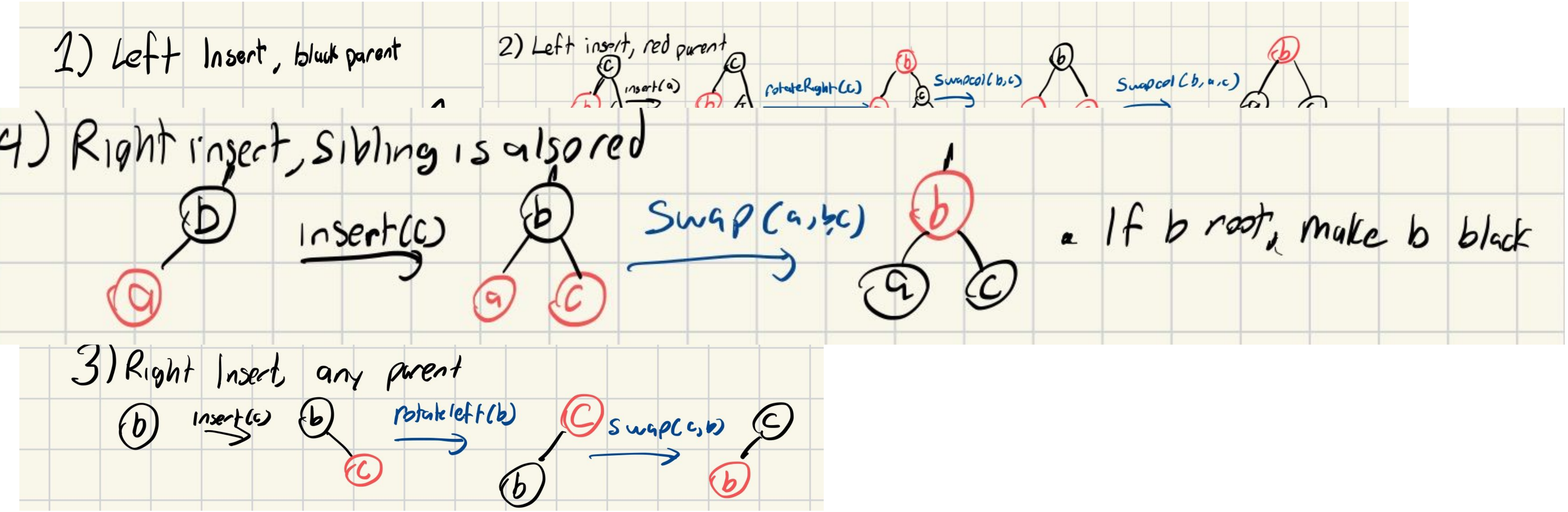
Types of LLRB inserts

Always insert in a red node



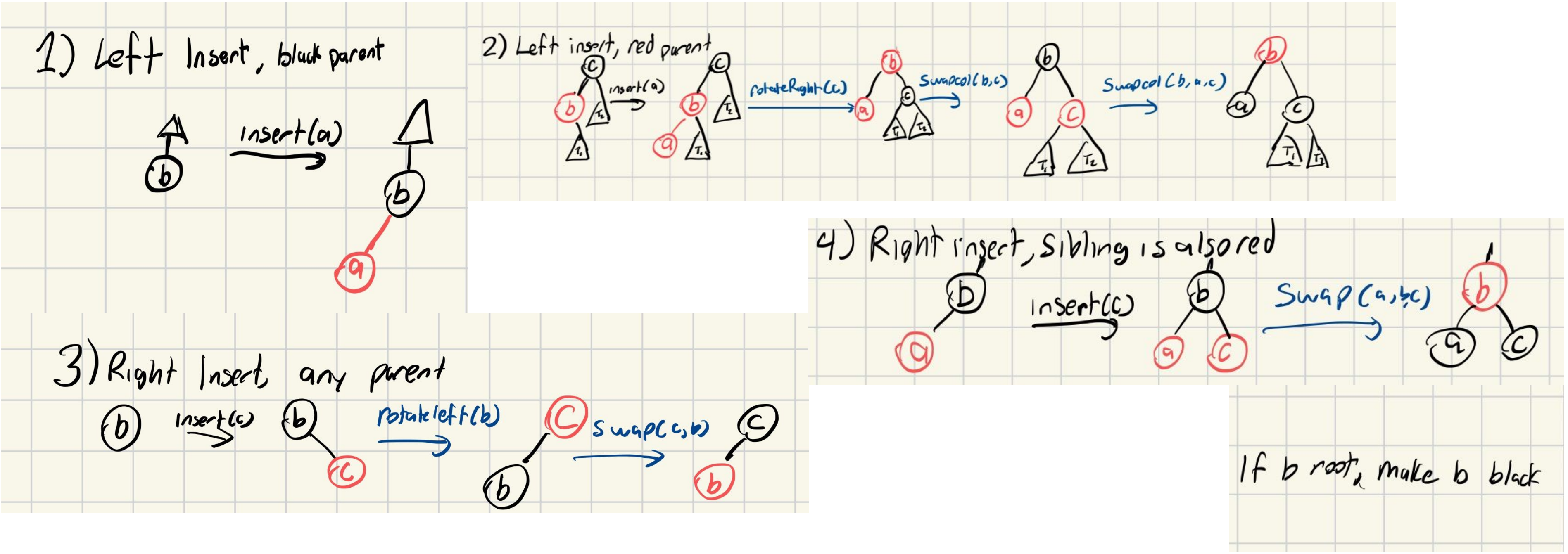
Types of LLRB inserts

Always insert in a red node



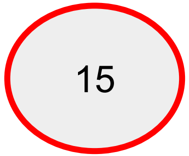
Types of LLRB inserts

Always insert in a red node



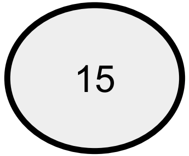
Insert: 15,21,7,24,0,26,3,28,29

Insert: 15,21,7,24,0,26,3,28,29



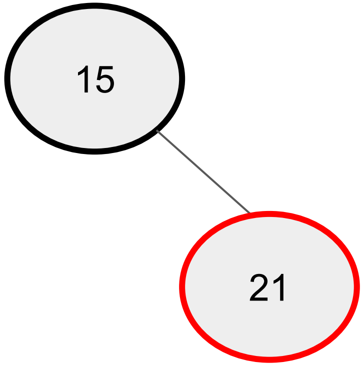
Insert: 15,21,7,24,0,26,3,28,29

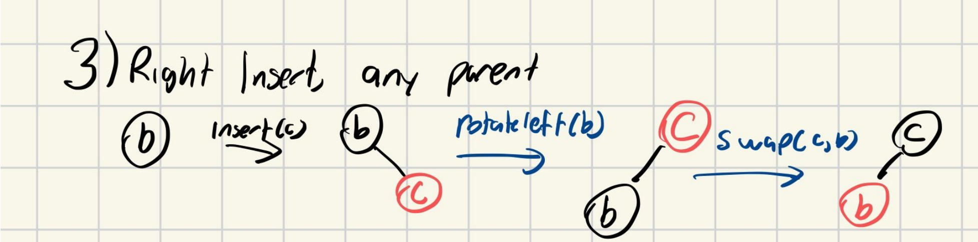
If root red, make it black



Insert: 15,21,7,24,0,26,3,28,29

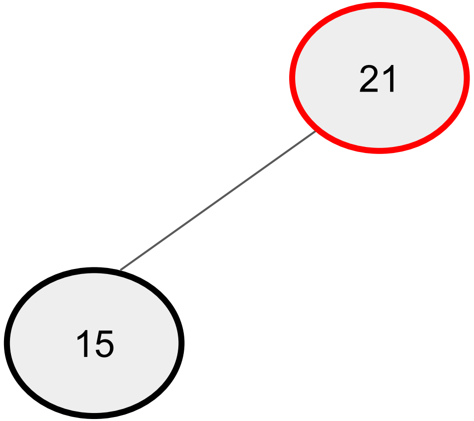
If root red, make it black

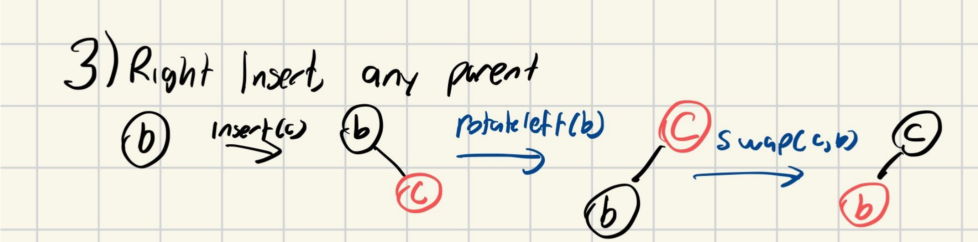




Insert: 15,21,7,24,0,26,3,28,29

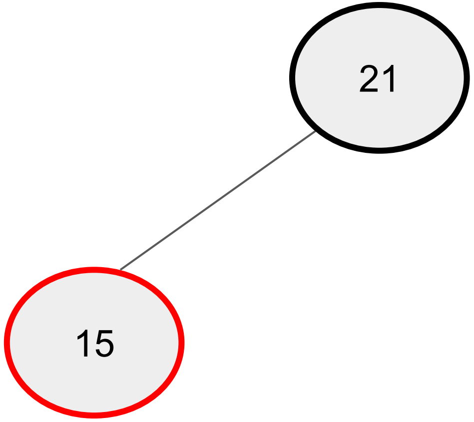
If root red, make it black

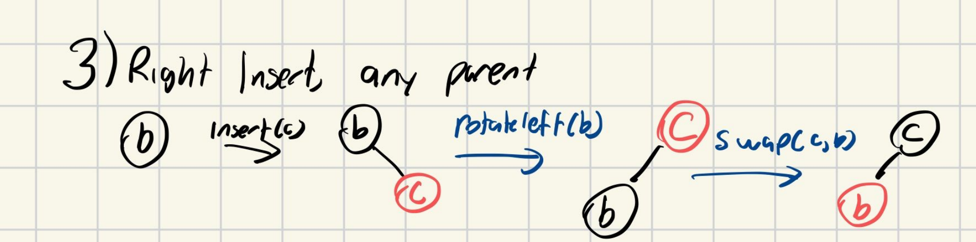




Insert: 15,21,7,24,0,26,3,28,29

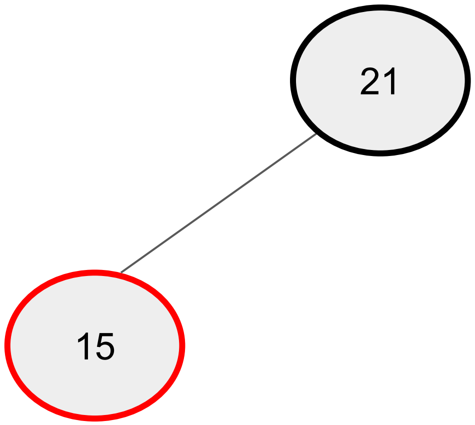
If root red, make it black





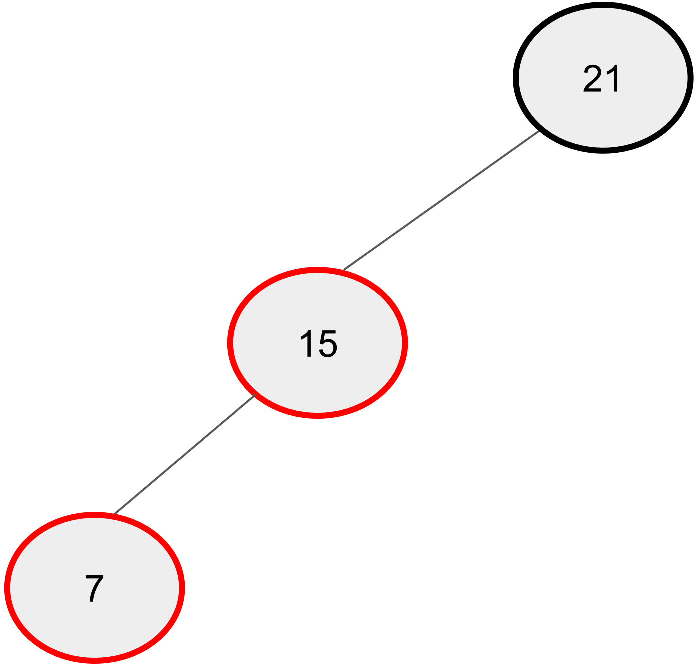
Insert: 15,21,<u>7</u>,24,0,26,3,28,29

If root red, make it black



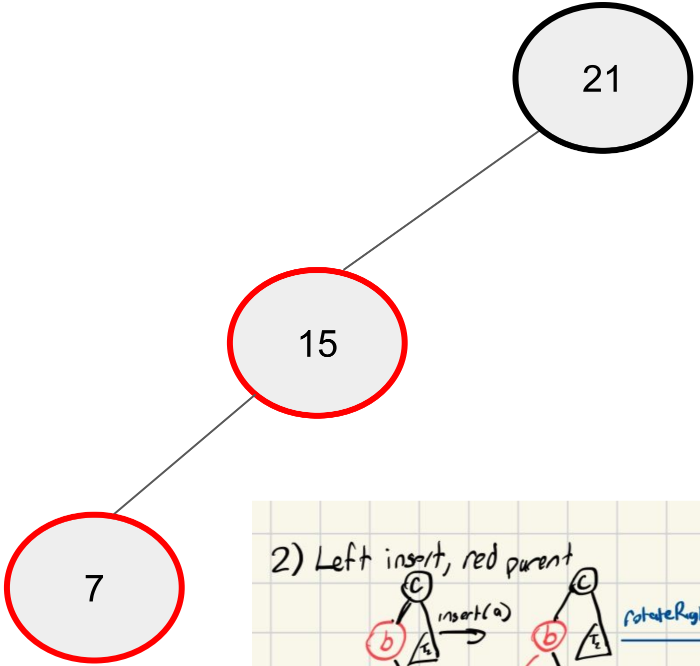
Insert: 15,21,7,24,0,26,3,28,29

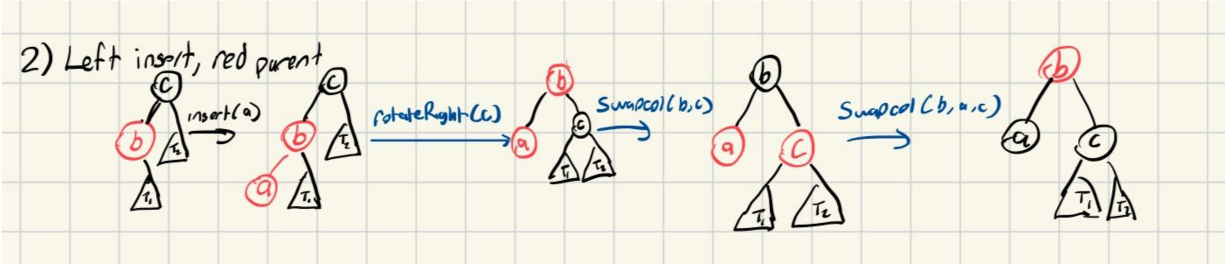
If root red, make it black



Insert: 15,21,7,24,0,26,3,28,29

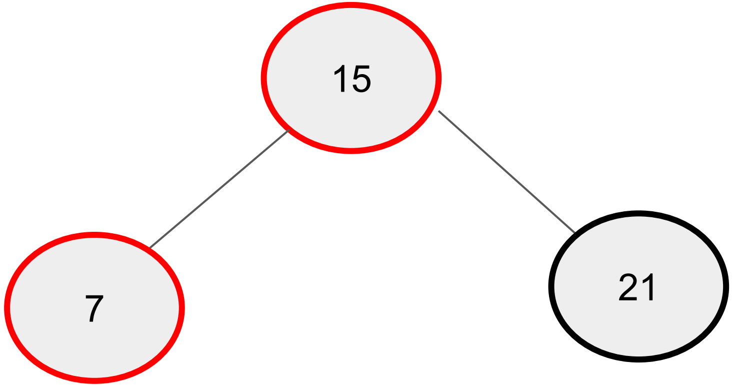
If root red, make it black

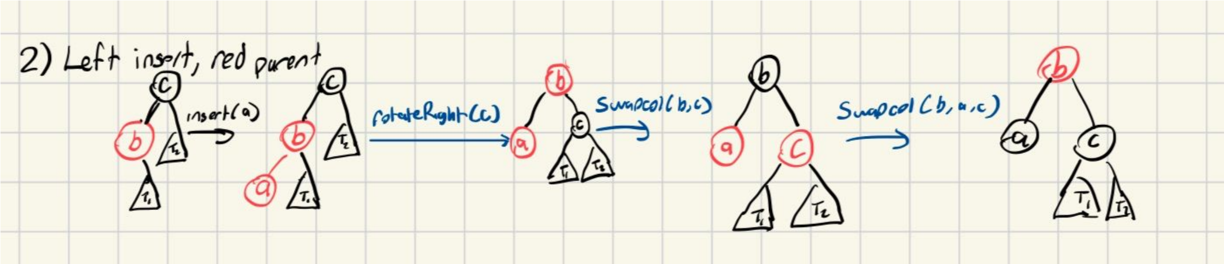




Insert: 15,21,7,24,0,26,3,28,29

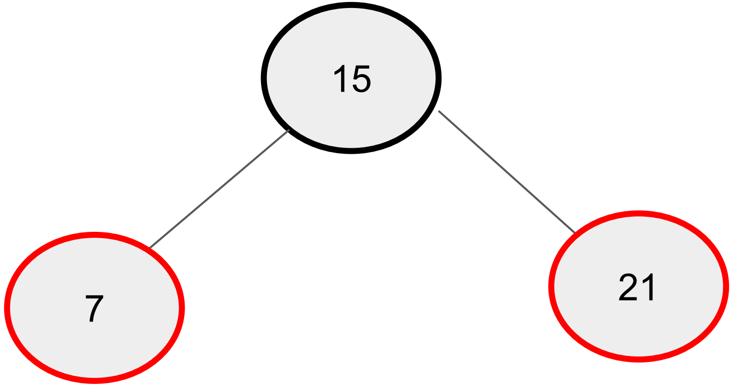
If root red, make it black

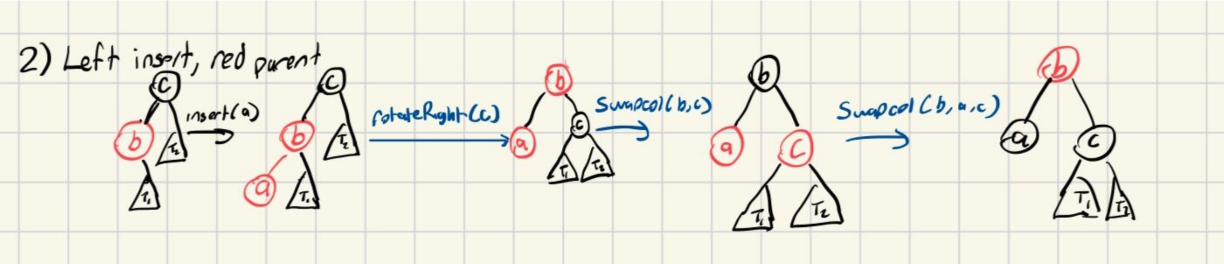




Insert: 15,21,<u>7</u>,24,0,26,3,28,29

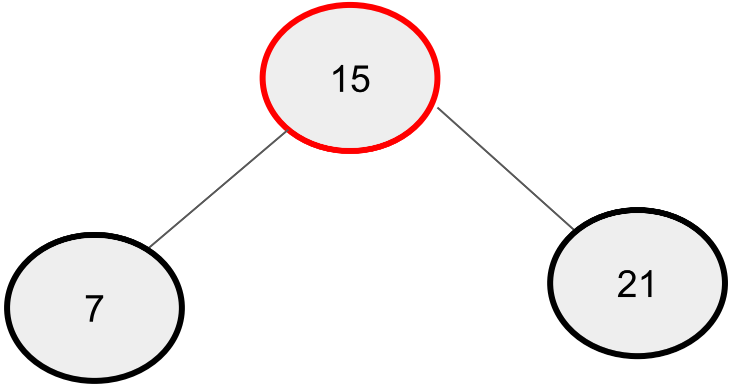
If root red, make it black

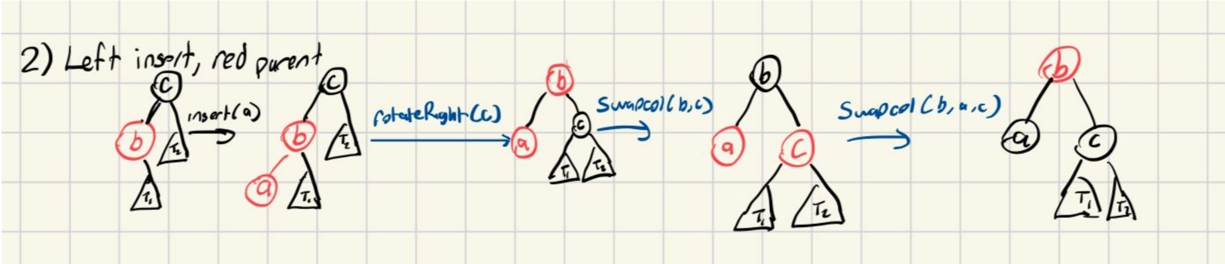




Insert: 15,21,7,24,0,26,3,28,29

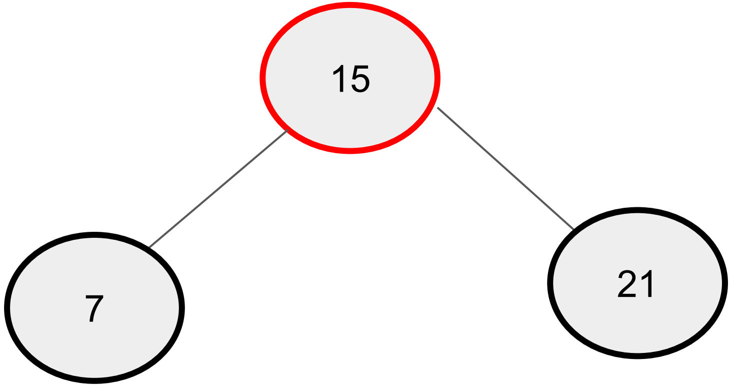
If root red, make it black

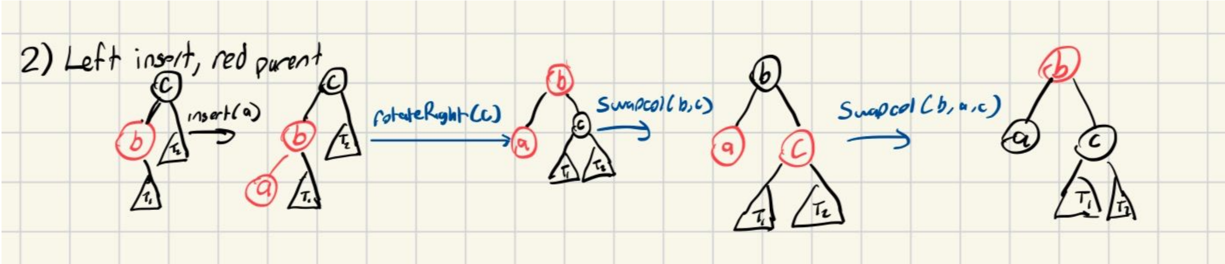




Insert: 15,21,<u>7</u>,24,0,26,3,28,29

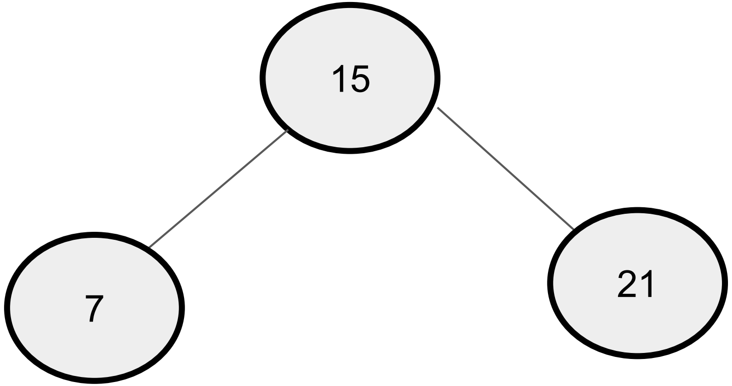
If root red, make it black

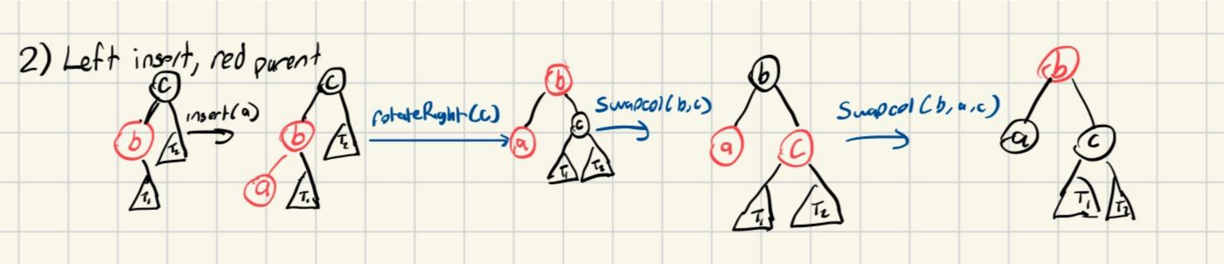




Insert: 15,21,7,24,0,26,3,28,29

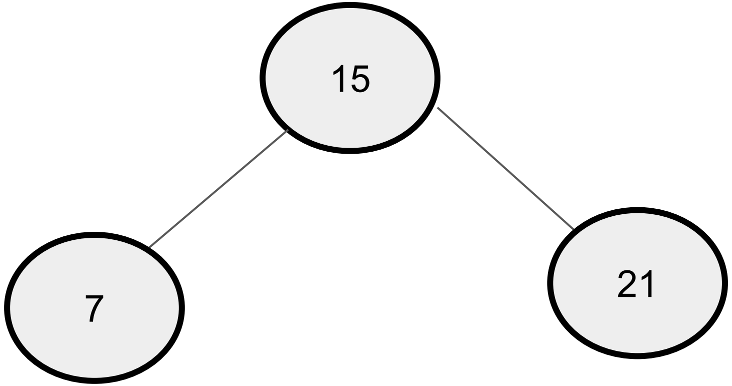
If root red, make it black





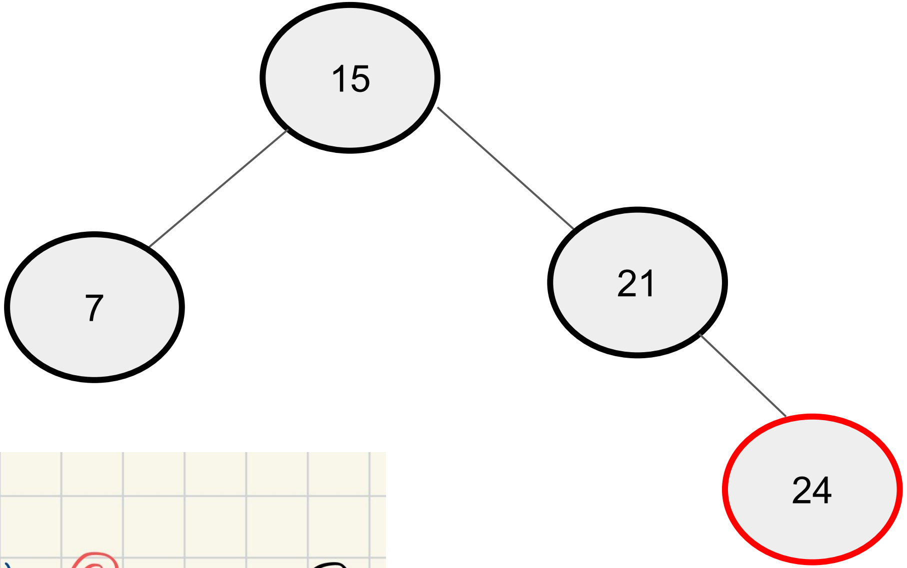
Insert: 15,21,7,<u>24</u>,0,26,3,28,29

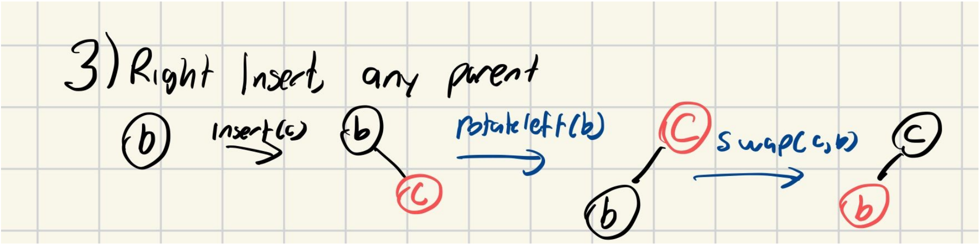
If root red, make it black



Insert: 15,21,7,24,0,26,3,28,29

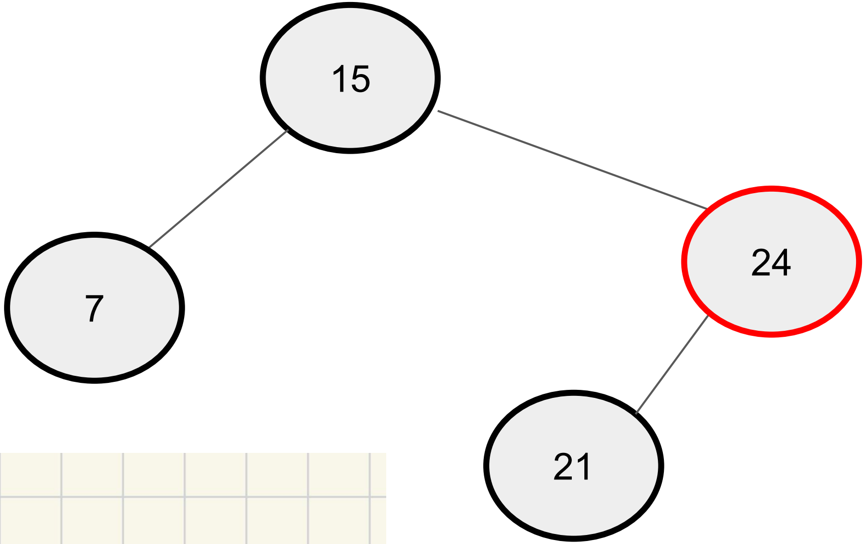
If root red, make it black

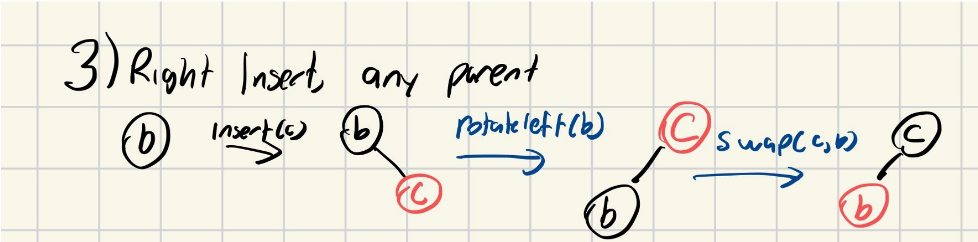




Insert: 15,21,7,24,0,26,3,28,29

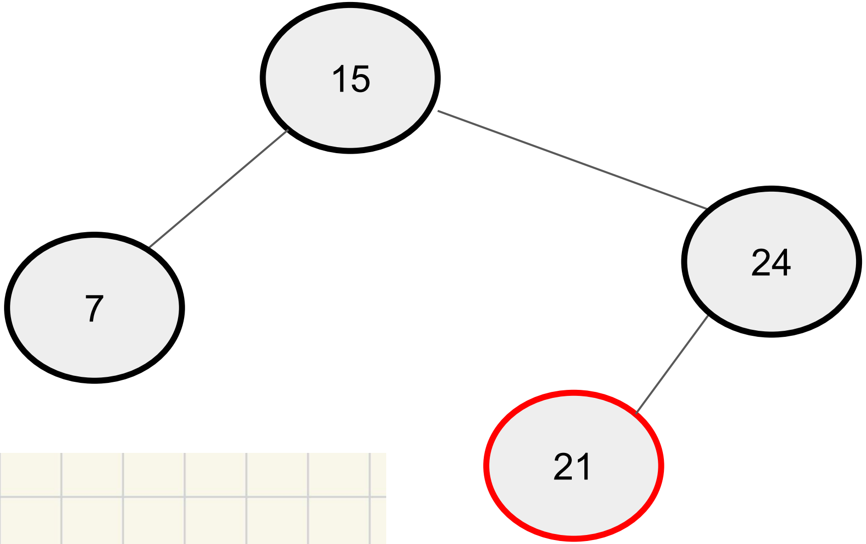
If root red, make it black

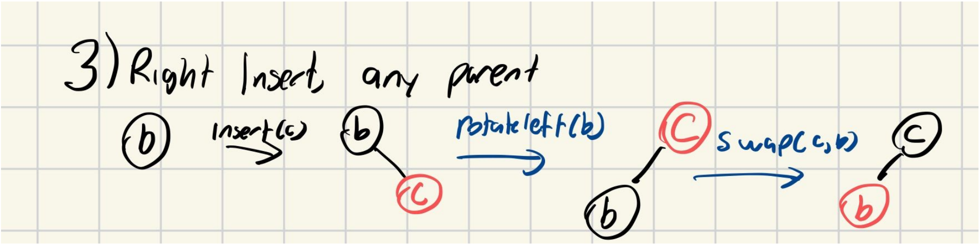




Insert: 15,21,7,24,0,26,3,28,29

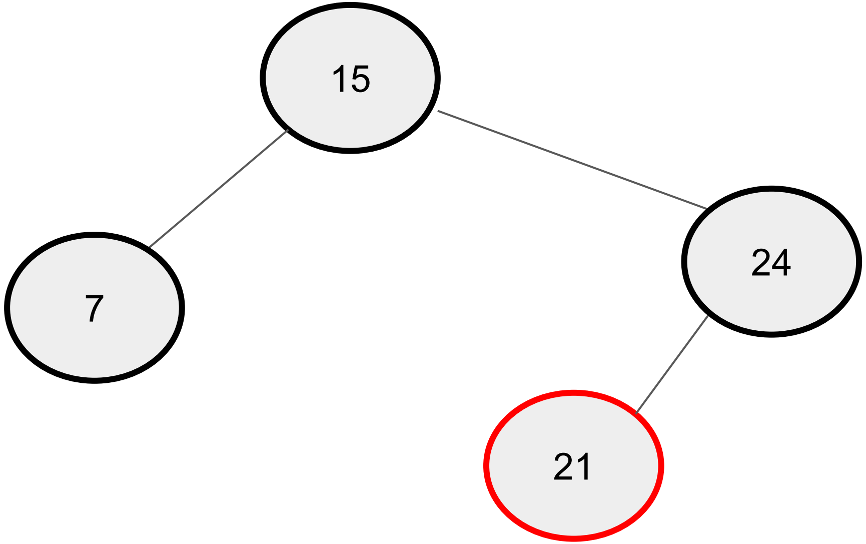
If root red, make it black





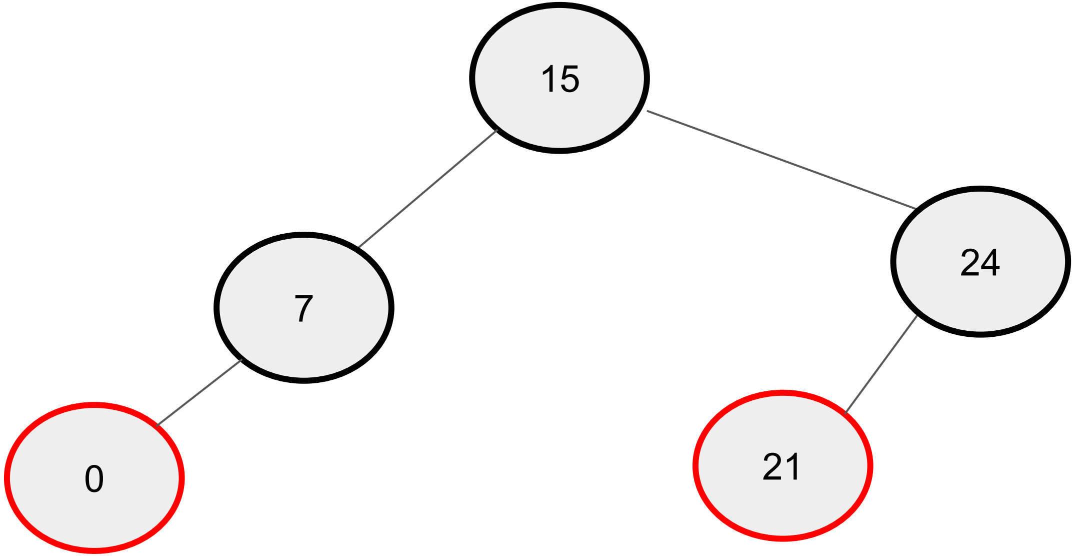
Insert: 15,21,7,24,<u>0</u>,26,3,28,29

If root red, make it black



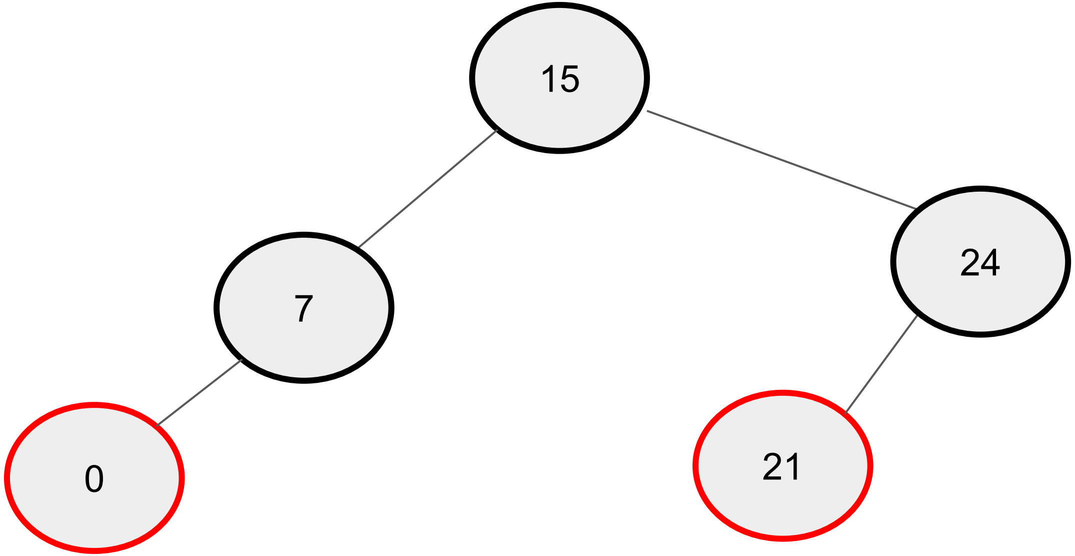
Insert: 15,21,7,24,<u>0</u>,26,3,28,29

If root red, make it black



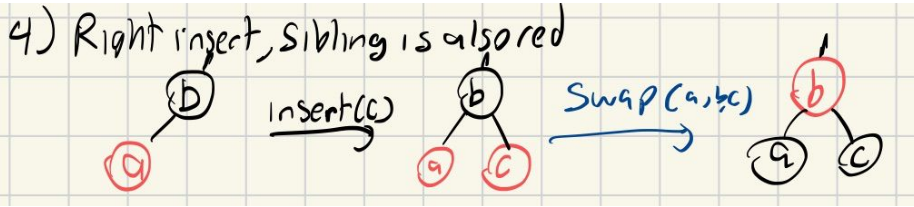
Insert: 15,21,7,24,0,<u>26</u>,3,28,29

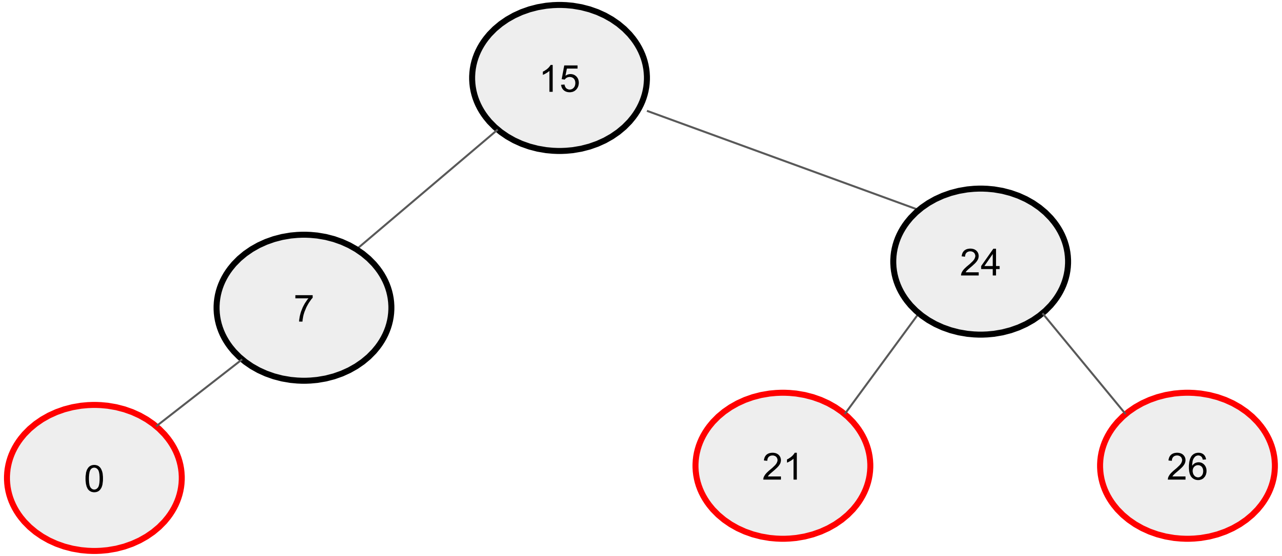
If root red, make it black



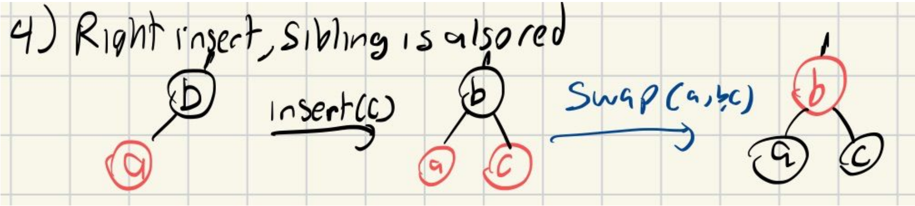
Insert: 15,21,7,24,0,26,3,28,29

If root red, make it black

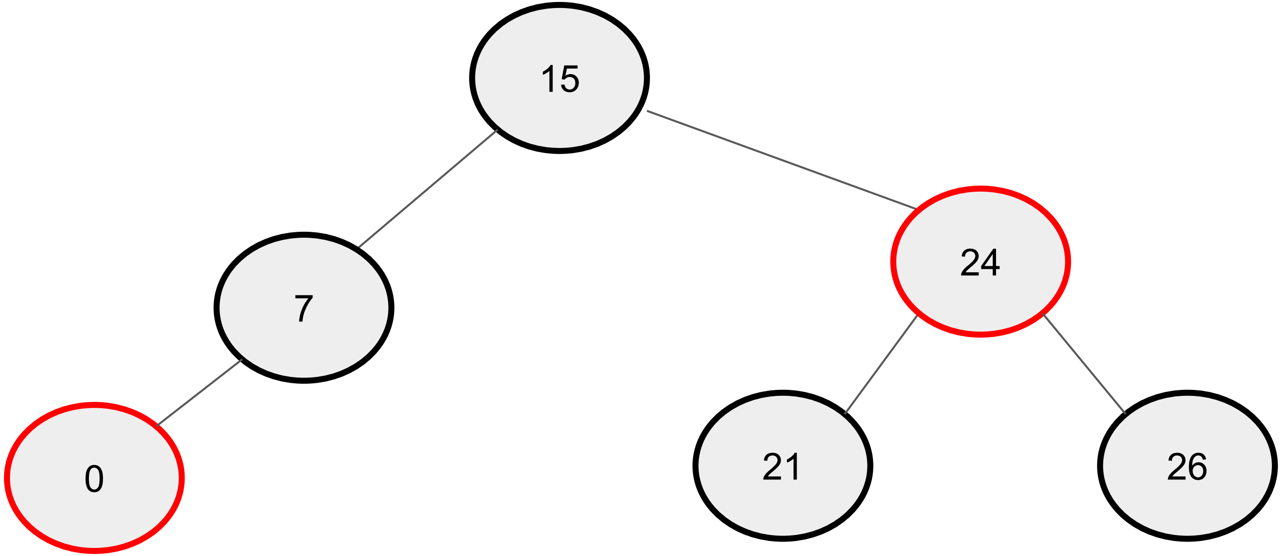




Insert: 15,21,7,24,0,<u>26</u>,3,28,29



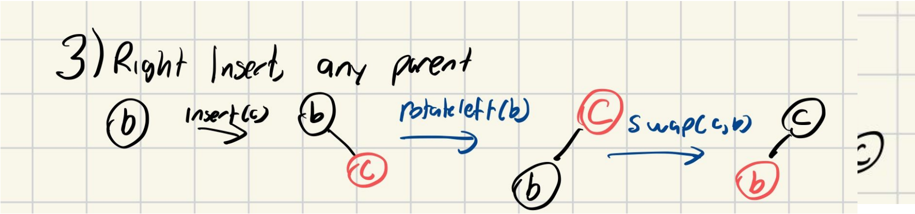
If root red, make it black

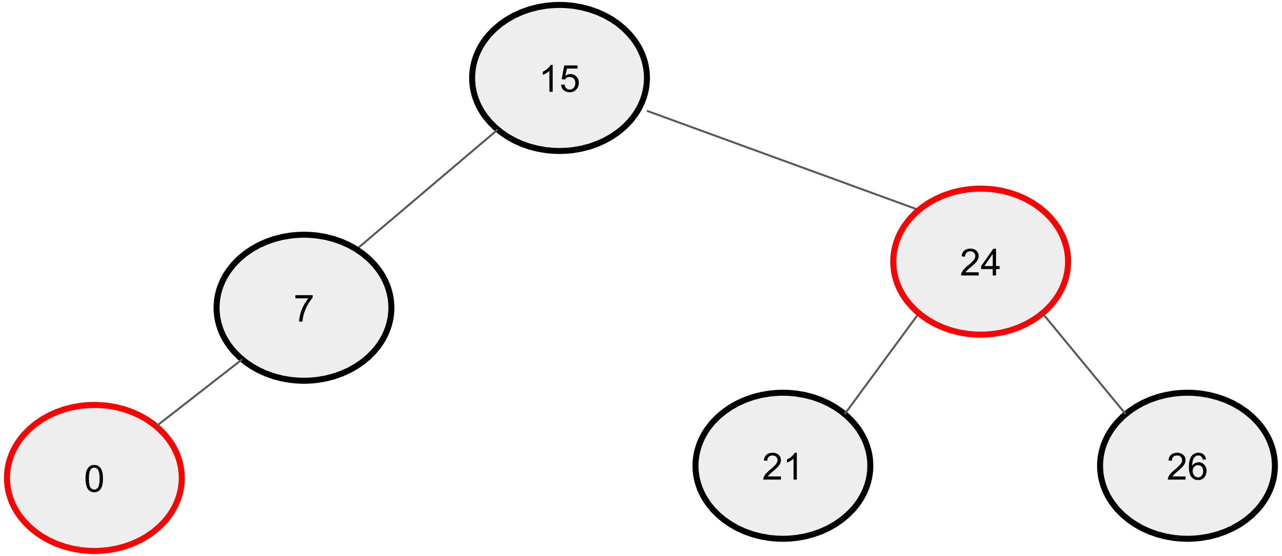


Oh no! Right red child, treat as red right insert

Insert: 15,21,7,24,0,<u>26</u>,3,28,29

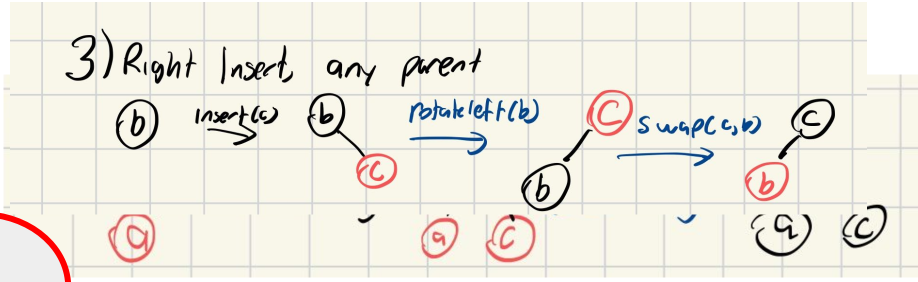
If root red, make it black



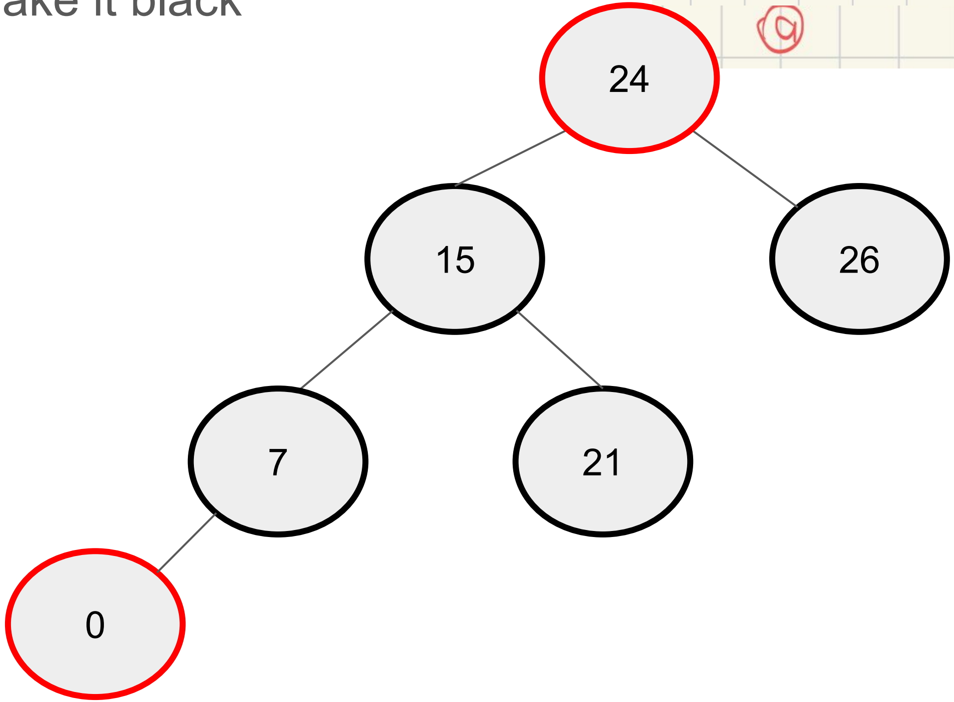


Oh no! Right red child, treat as red right insert

Insert: 15,21,7,24,0,<u>26</u>,3,28,29

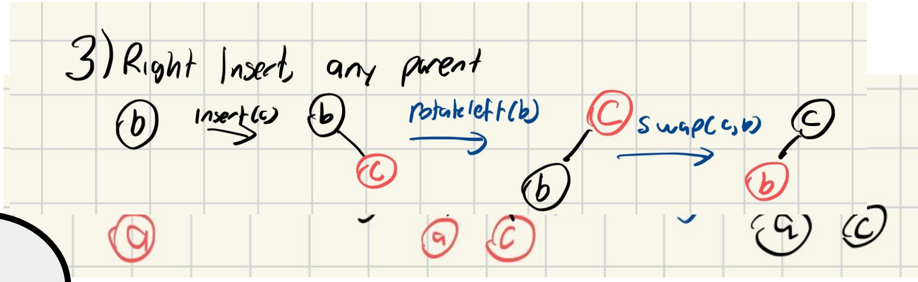


If root red, make it black

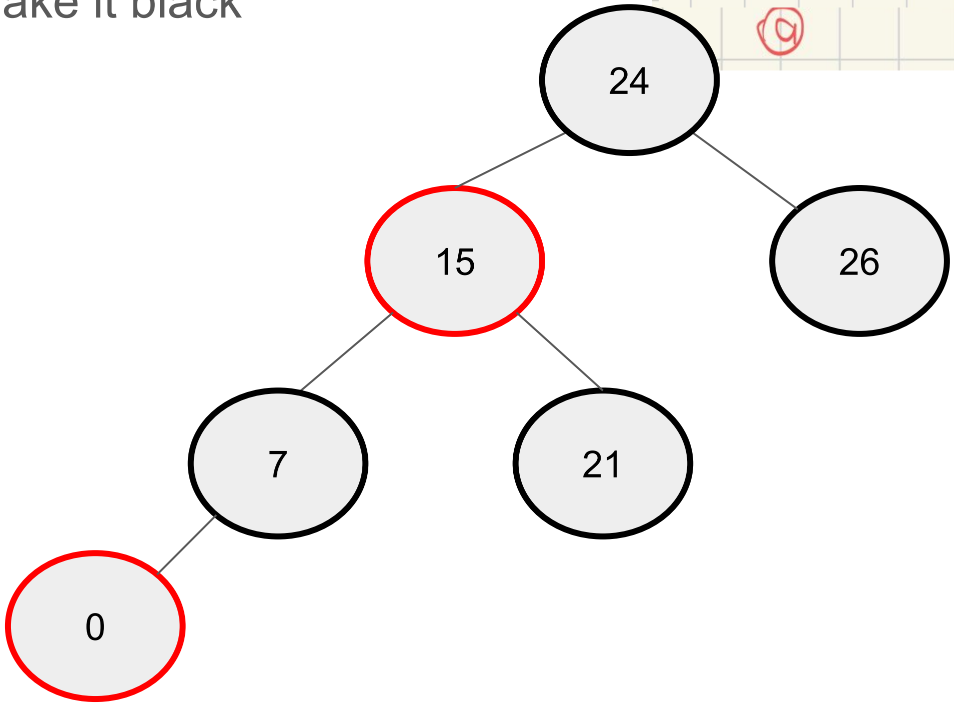


Oh no! Right red child, treat as red right insert

Insert: 15,21,7,24,0,26,3,28,29

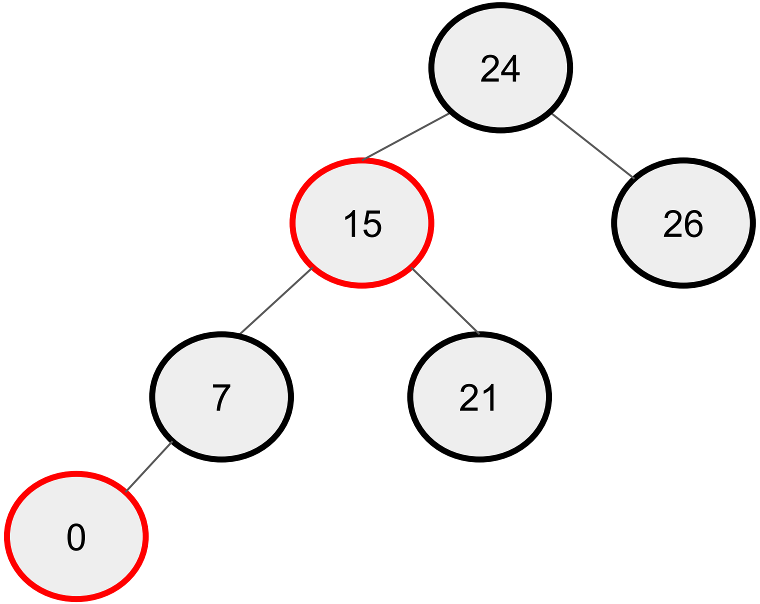


If root red, make it black

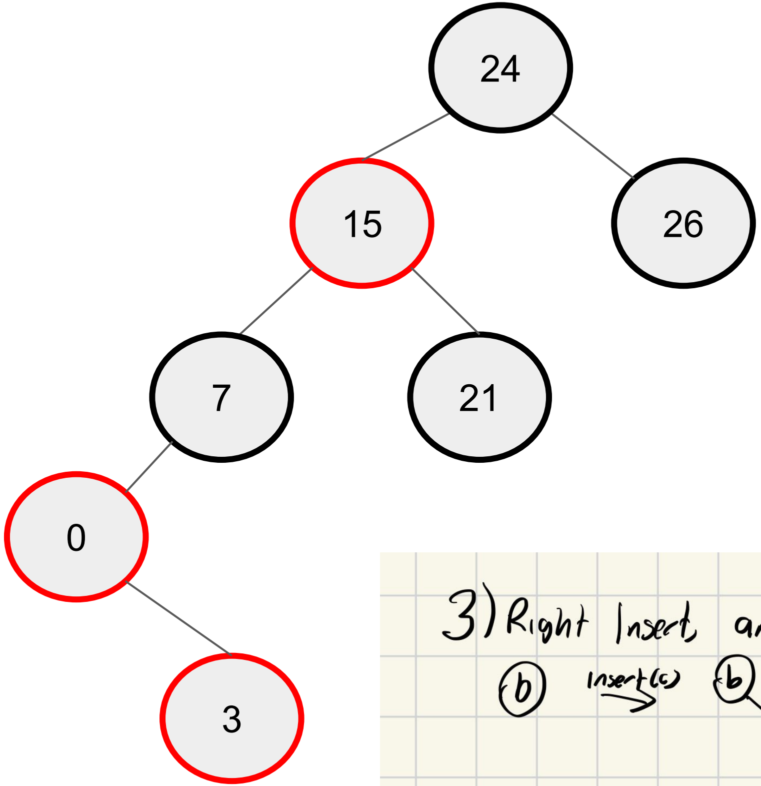


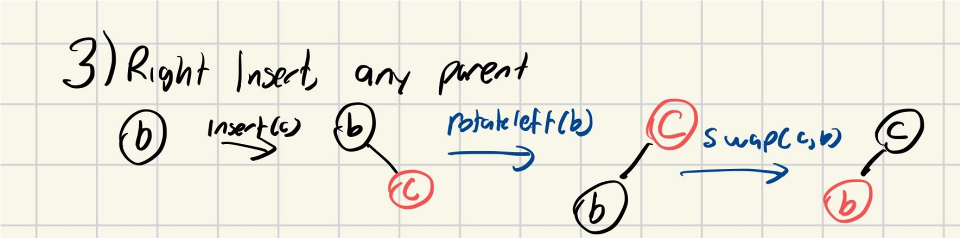
Oh no! Right red child, treat as red right insert

Insert: 15,21,7,24,0,26,<u>3</u>,28,29

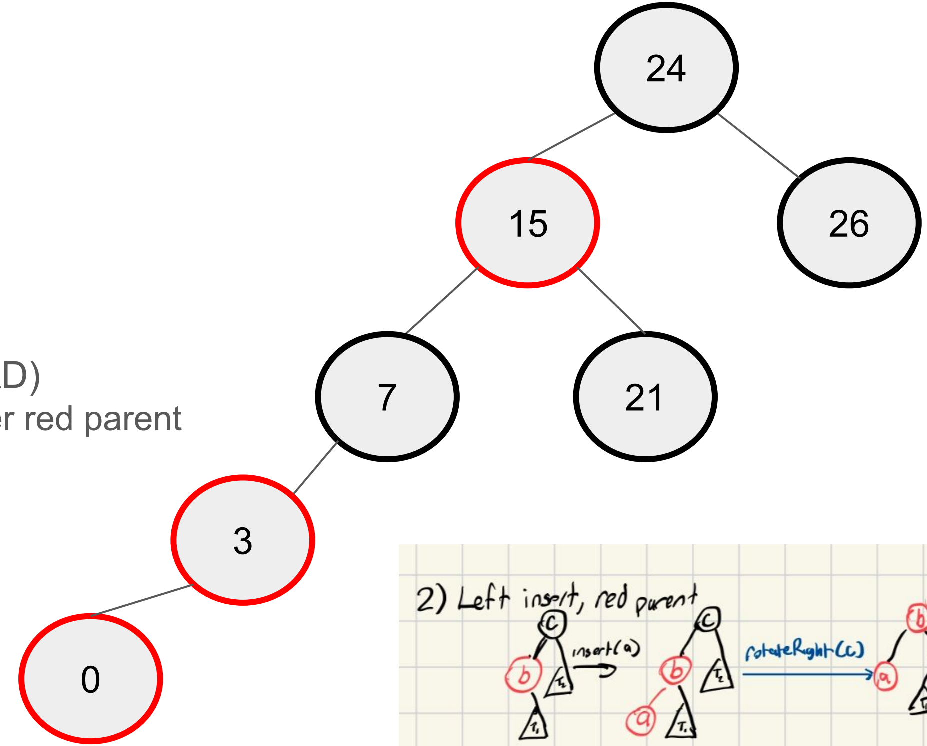


Insert: 15,21,7,24,0,26,3,28,29

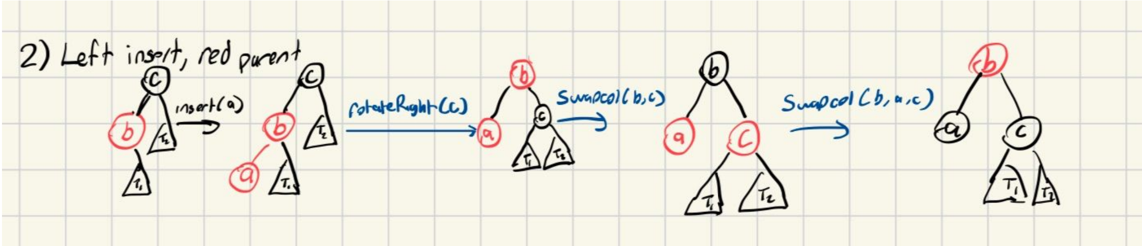




Insert: 15,21,7,24,0,26,<u>3</u>,28,29

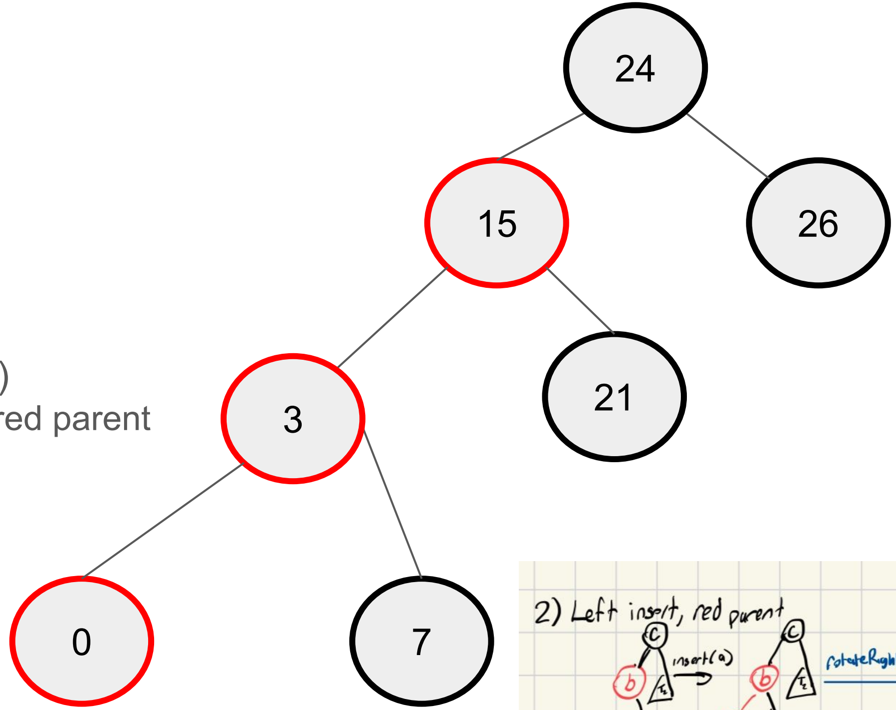


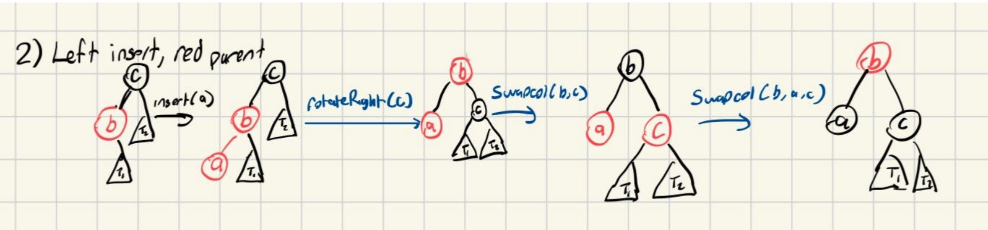
Two reds in a row (BAD)
Treat as red insert under red parent



Insert: 15,21,7,24,0,26,<u>3</u>,28,29

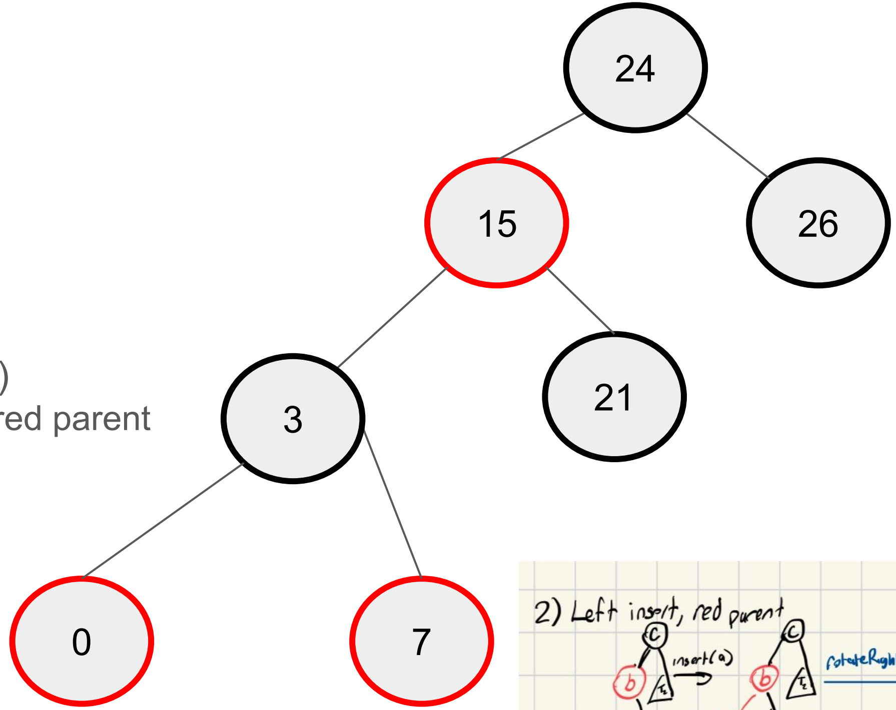
Two reds in a row (BAD)
Treat as red insert under red parent

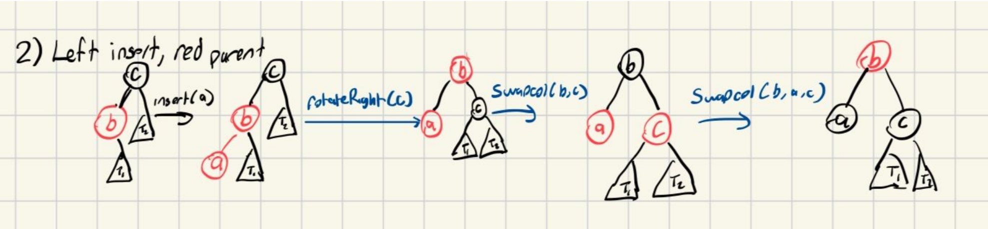




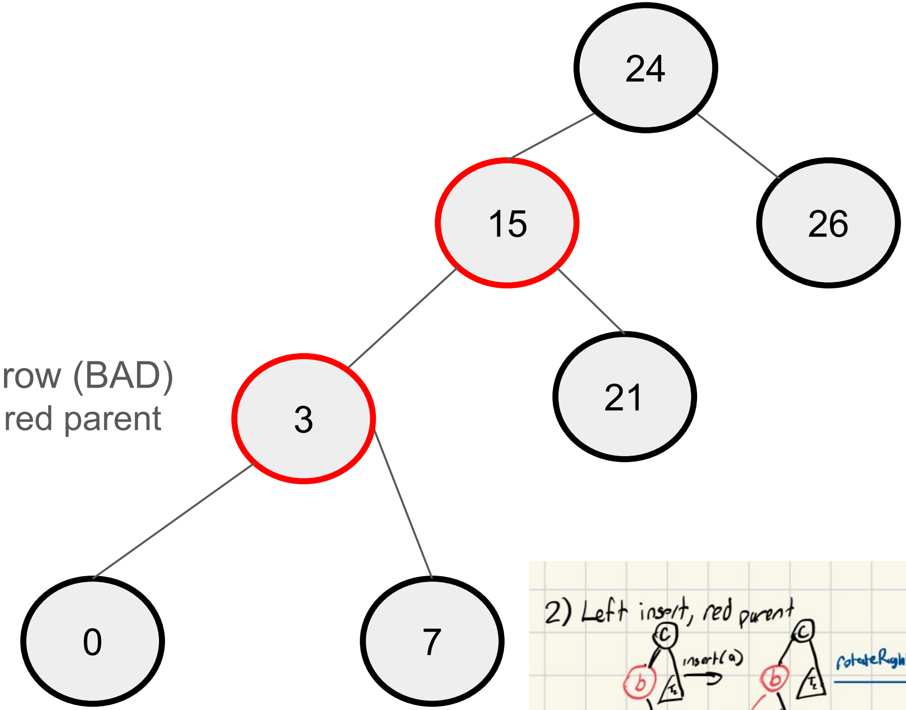
Insert: 15,21,7,24,0,26,<u>3</u>,28,29

Two reds in a row (BAD)
Treat as red insert under red parent



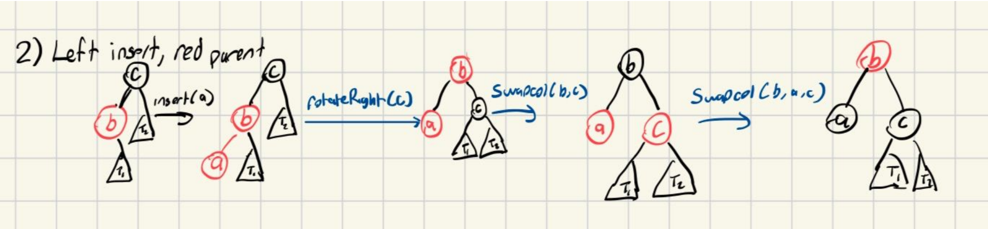


Insert: 15,21,7,24,0,26,3,28,29

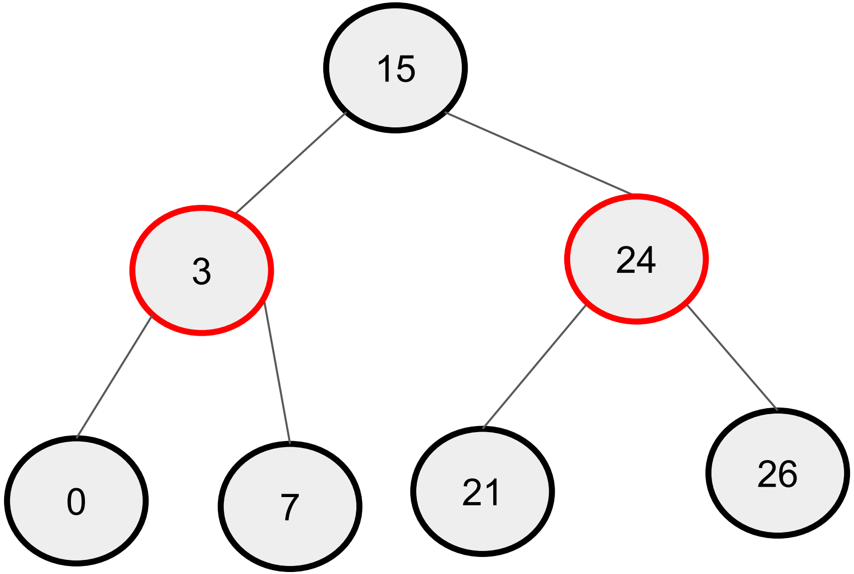


Rotate right at 24

Another Two reds in a row (BAD)
Treat as red insert under red parent

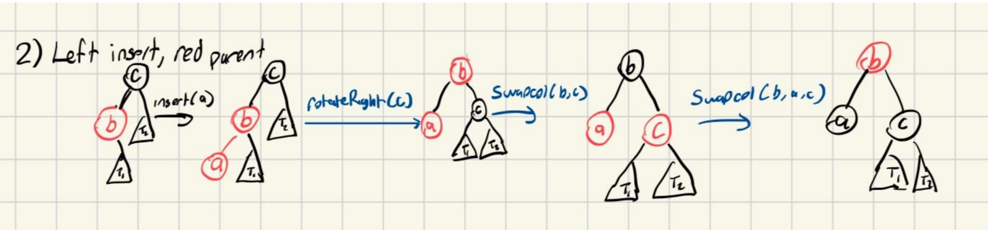


Insert: 15,21,7,24,0,26,3,28,29

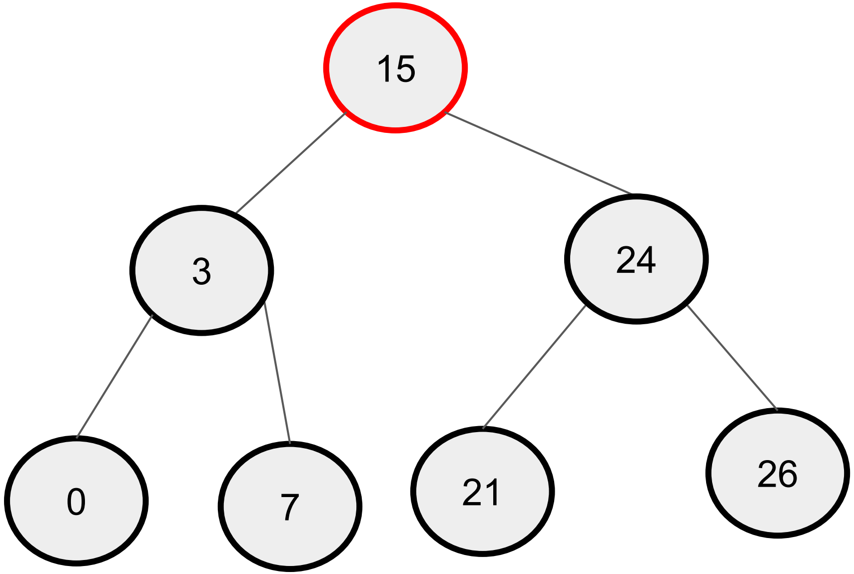


swap(15,24)

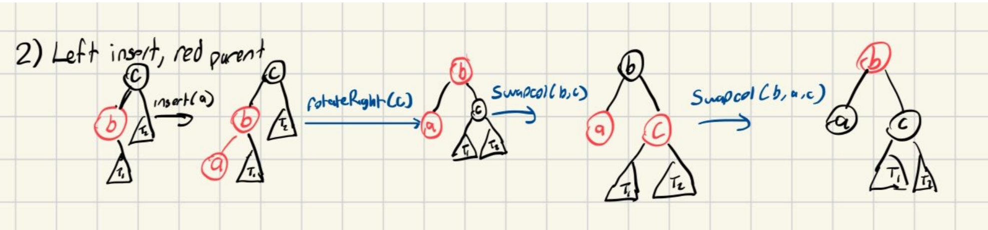
Another Two reds in a row (BAD)
Treat as red insert under red parent



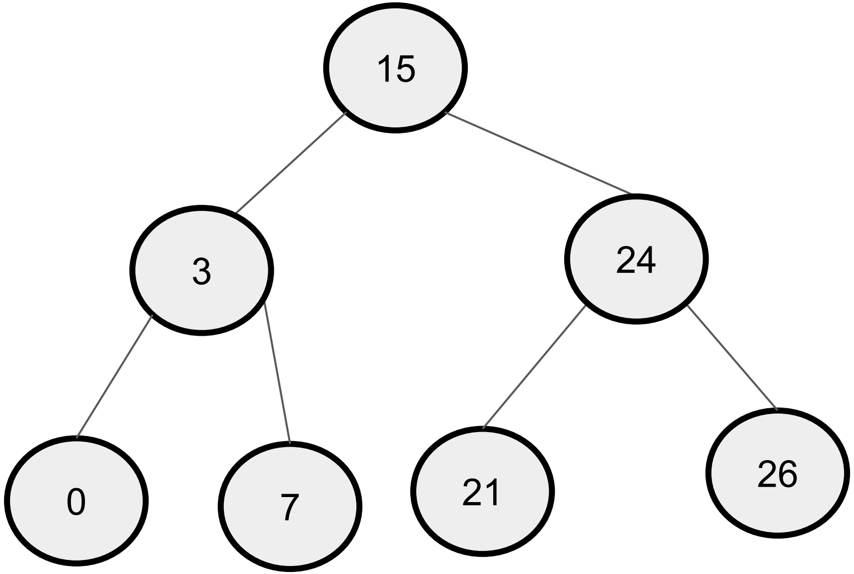
Insert: 15,21,7,24,0,26,3,28,29



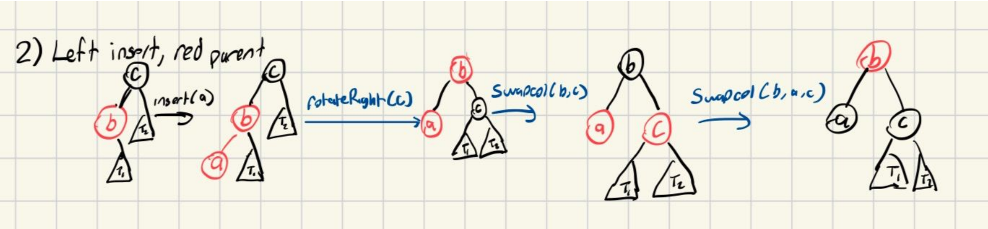
Another Two reds in a row (BAD)
Treat as red insert under red parent



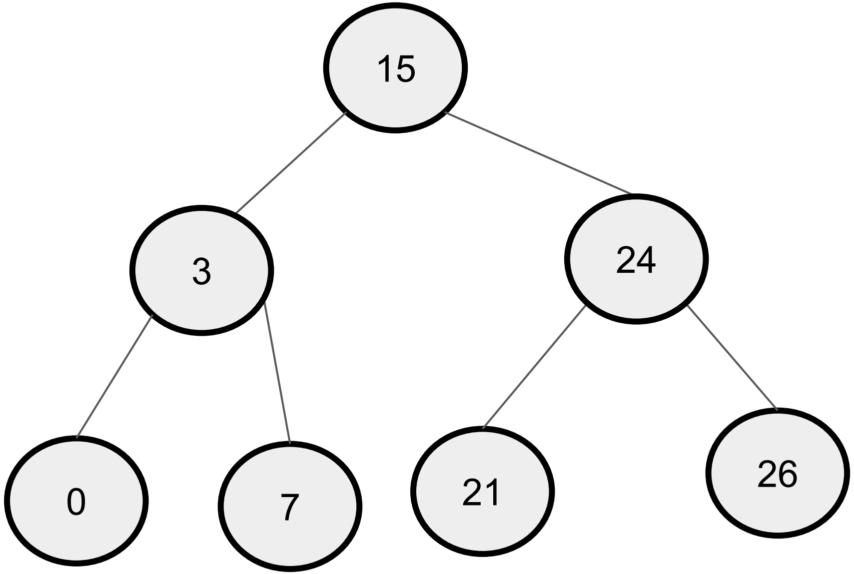
Insert: 15,21,7,24,0,26,3,28,29



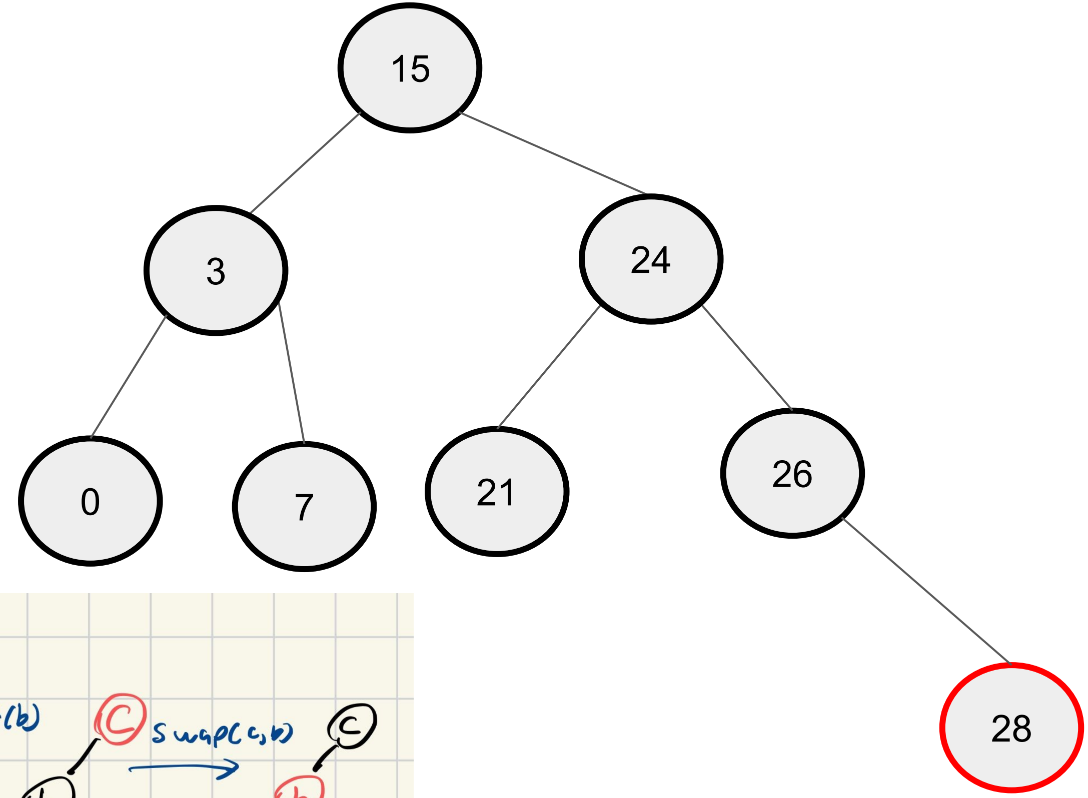
Another Two reds in a row (BAD)
Treat as red insert under red parent



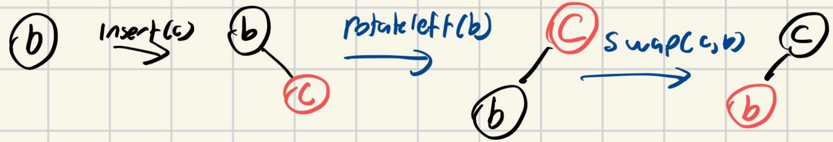
Insert: 15,21,7,24,0,26,3,<u>28</u>,29



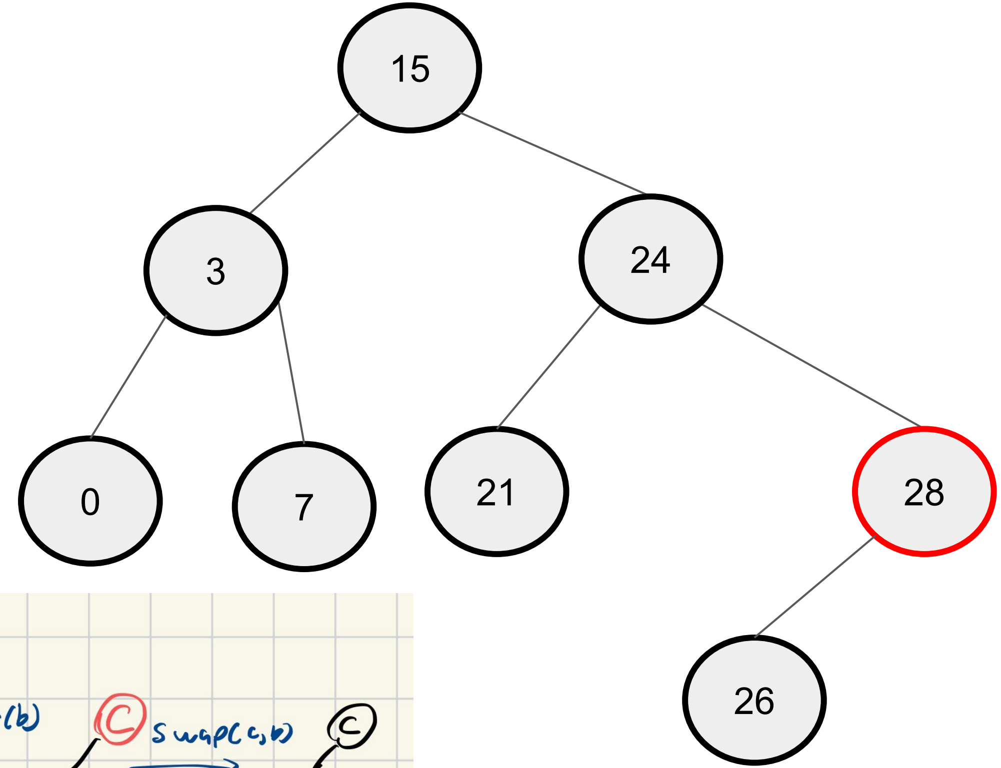
Insert: 15,21,7,24,0,26,3,28,29

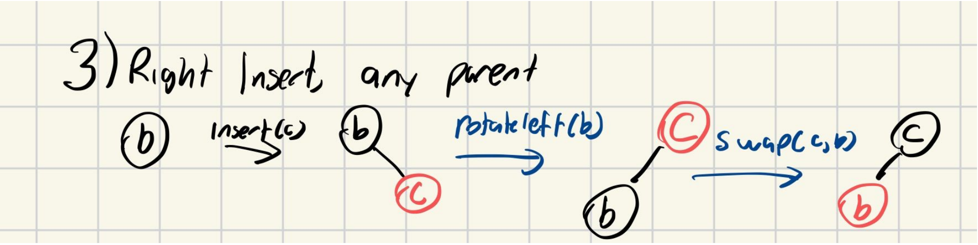


3) Right Insert, any parent

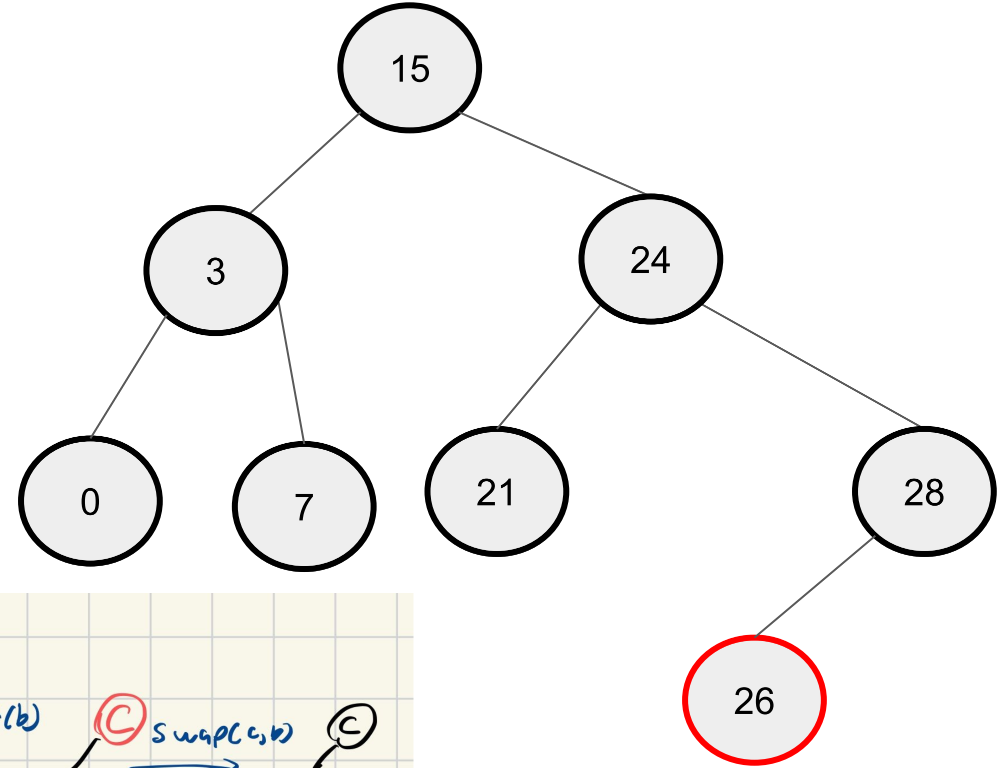


Insert: 15,21,7,24,0,26,3,<u>28</u>,29

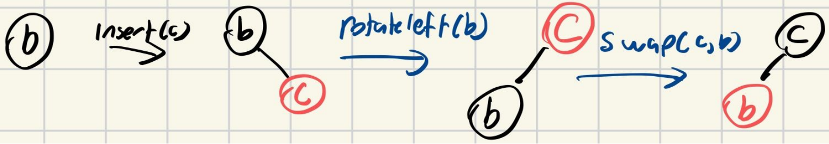




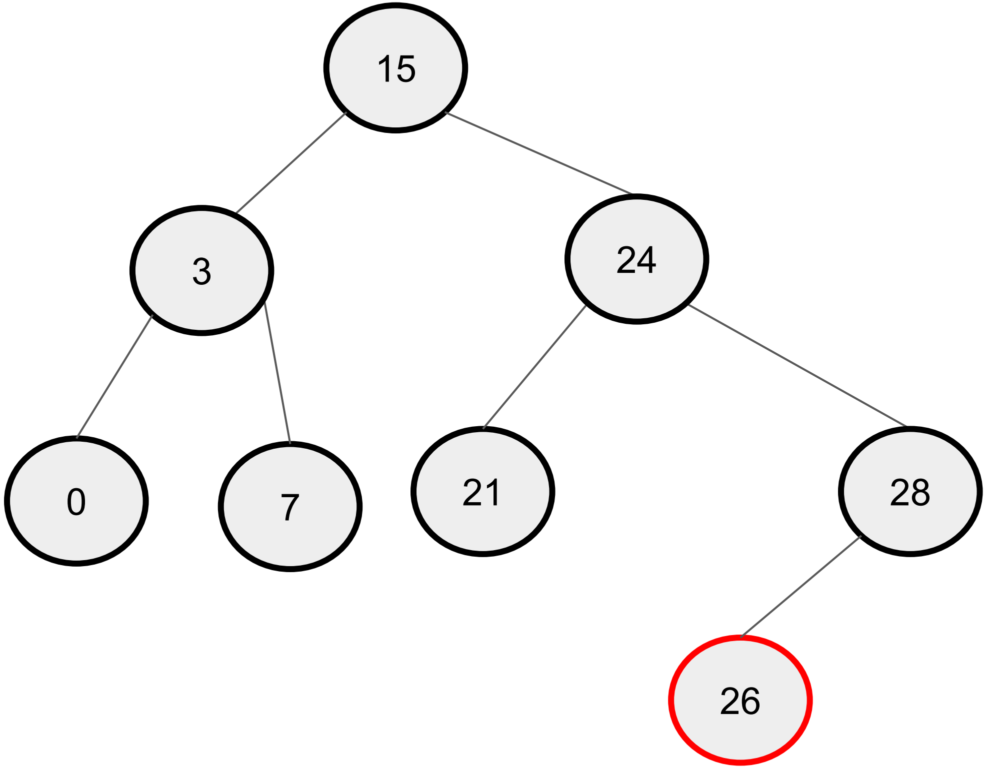
Insert: 15,21,7,24,0,26,3,28,29



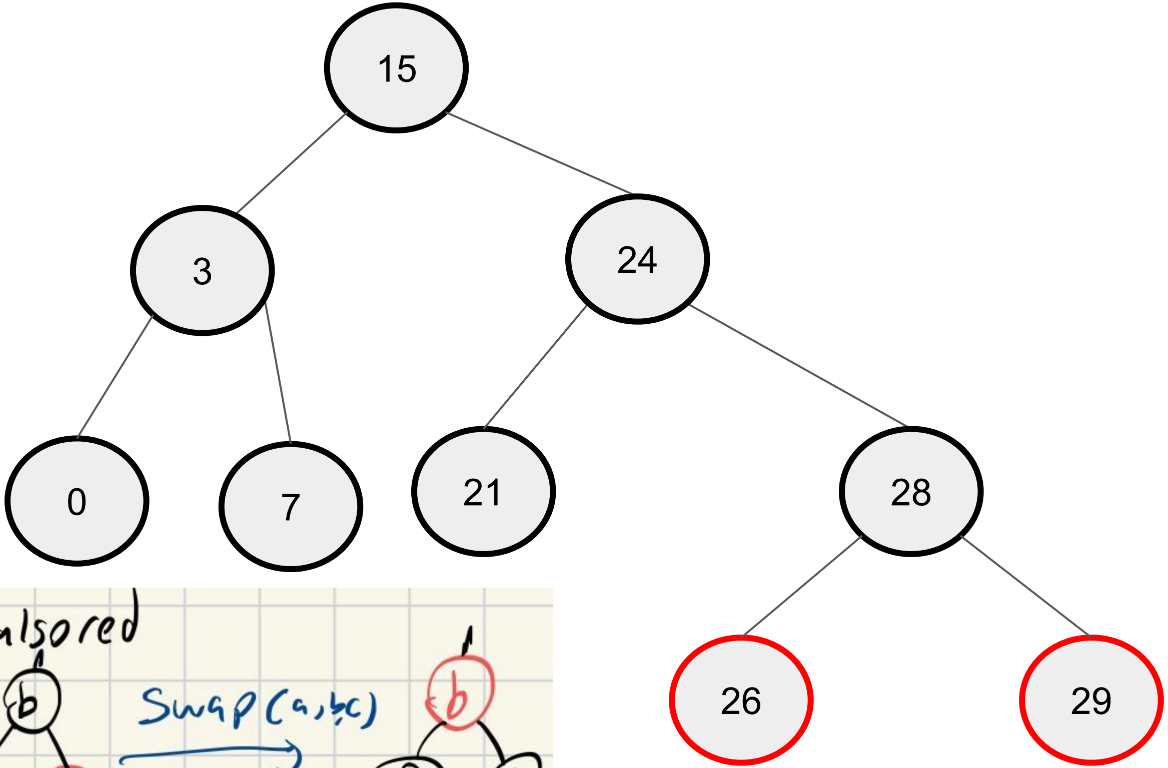
3) Right Insert, any parent

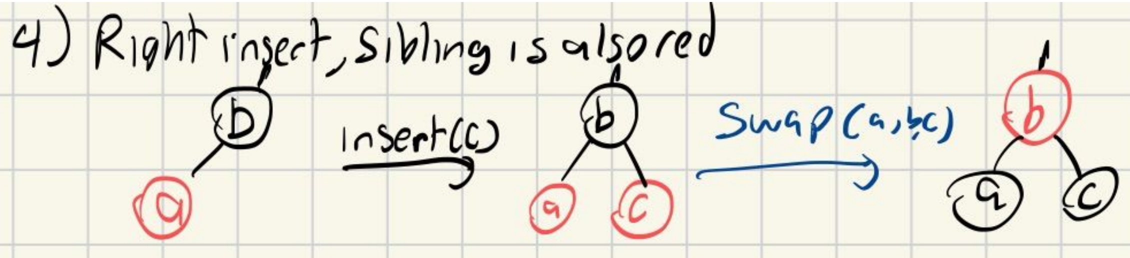


Insert: 15,21,7,24,0,26,3,28,<u>29</u>

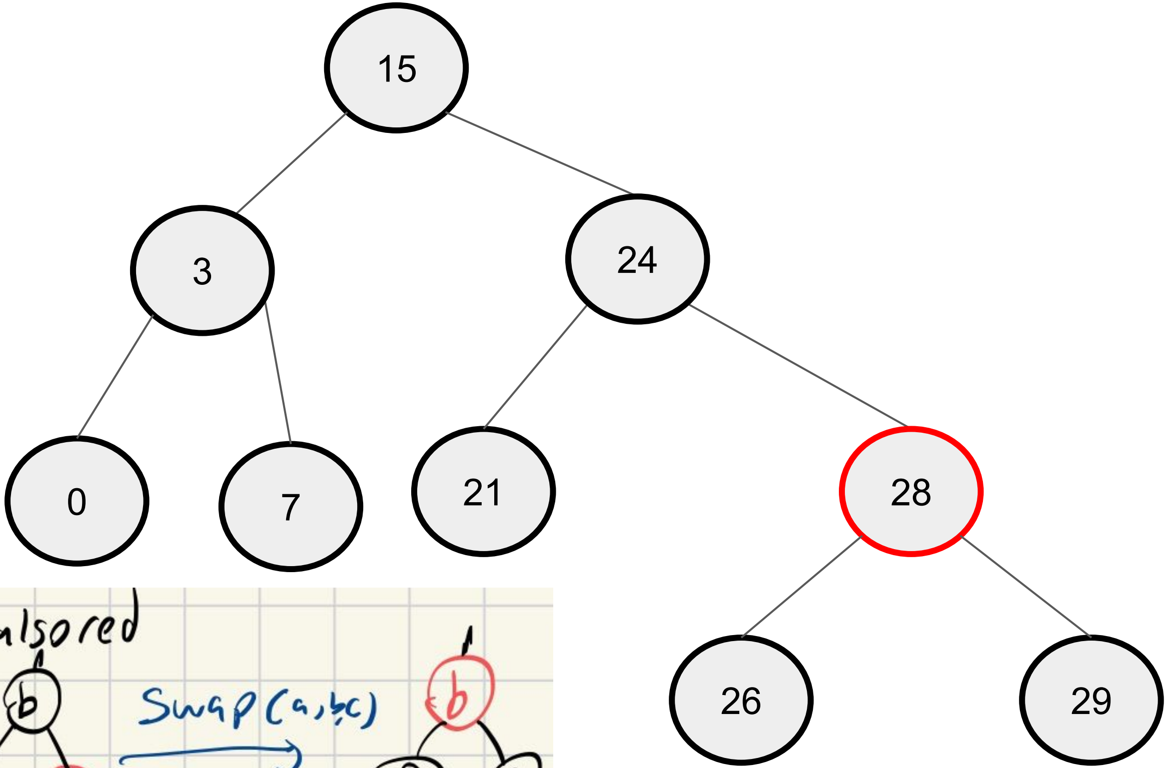


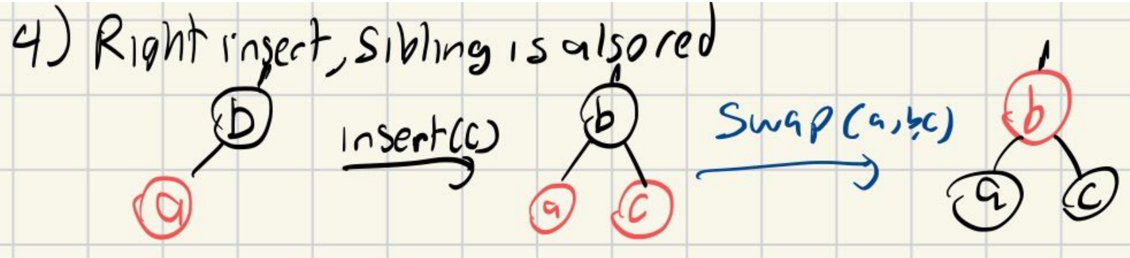
Insert: 15,21,7,24,0,26,3,28,<u>29</u>



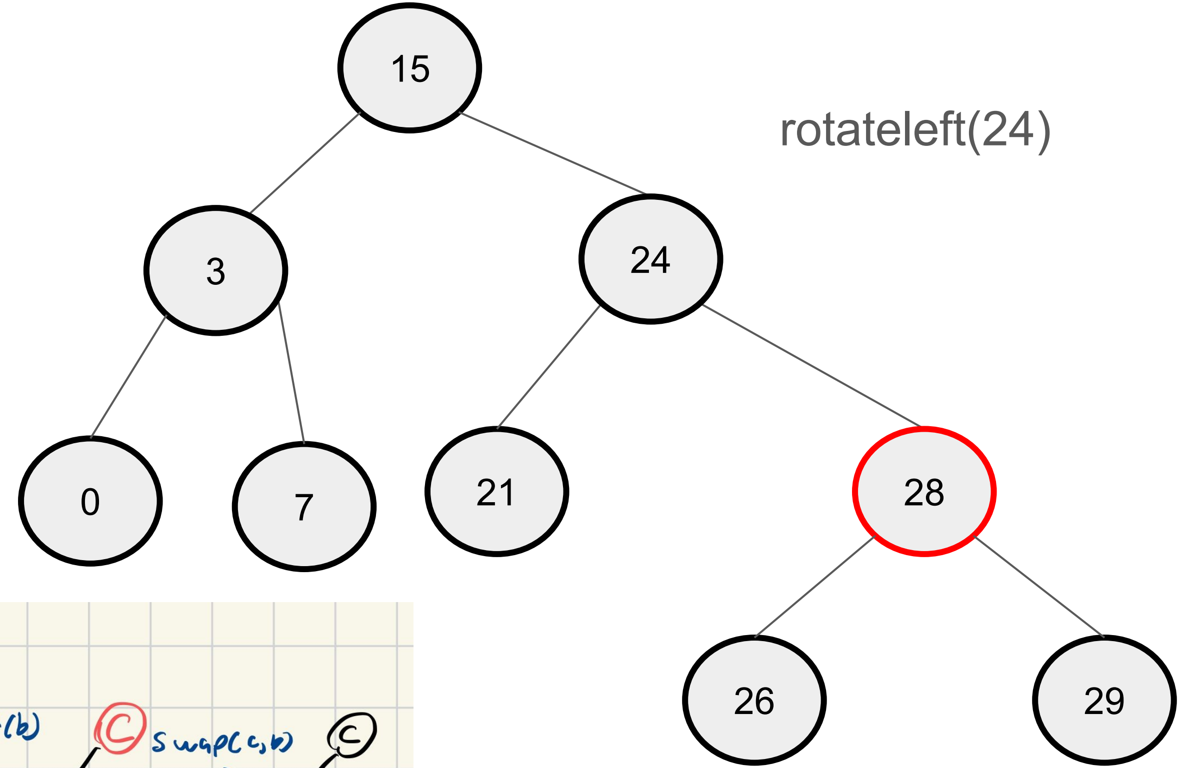


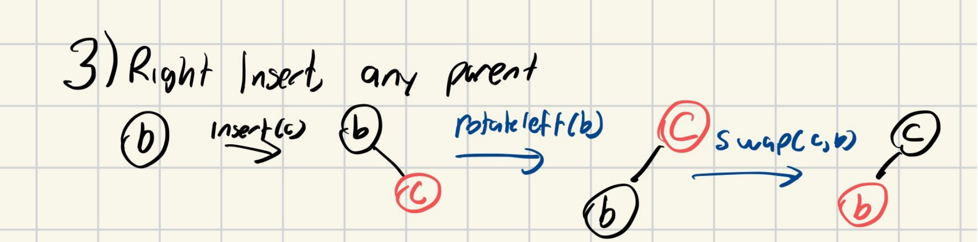
Insert: 15,21,7,24,0,26,3,28,29



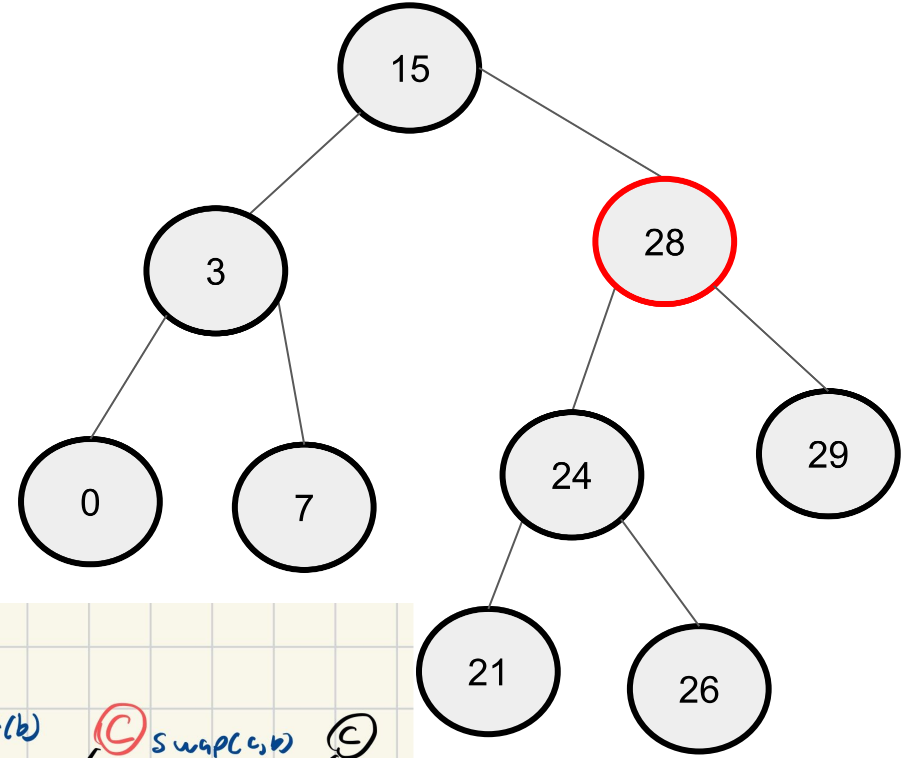


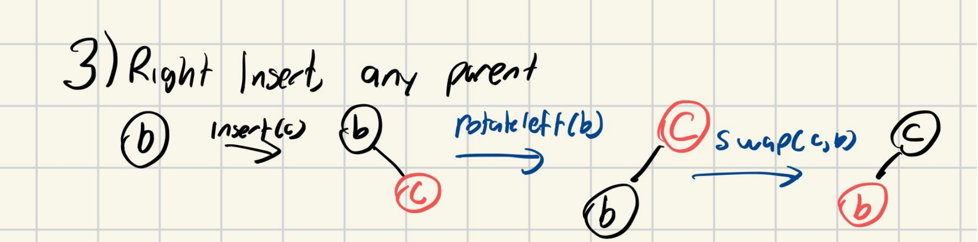
Insert: 15,21,7,24,0,26,3,28,<u>29</u>



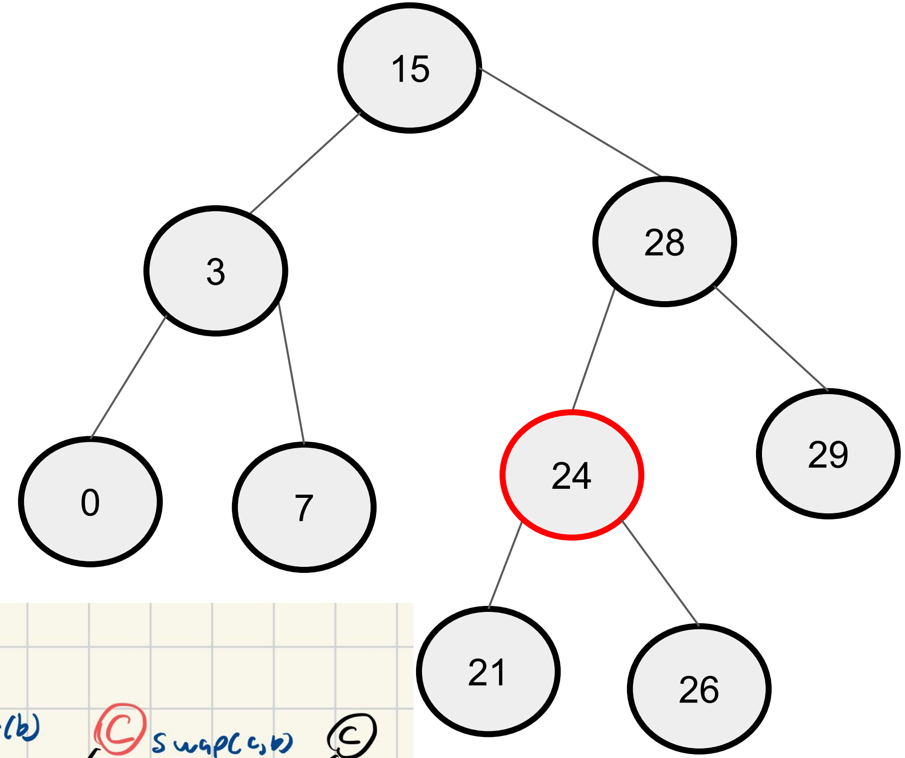


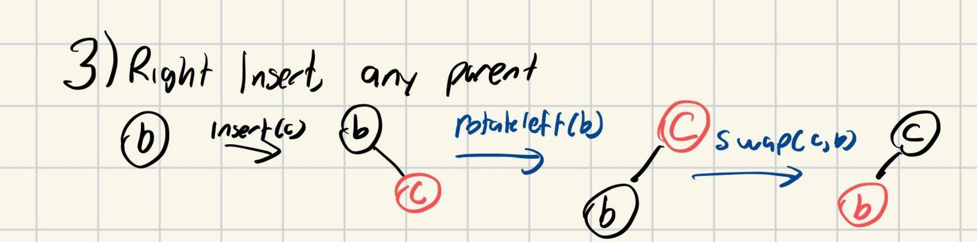
Insert: 15,21,7,24,0,26,3,28,<u>29</u>



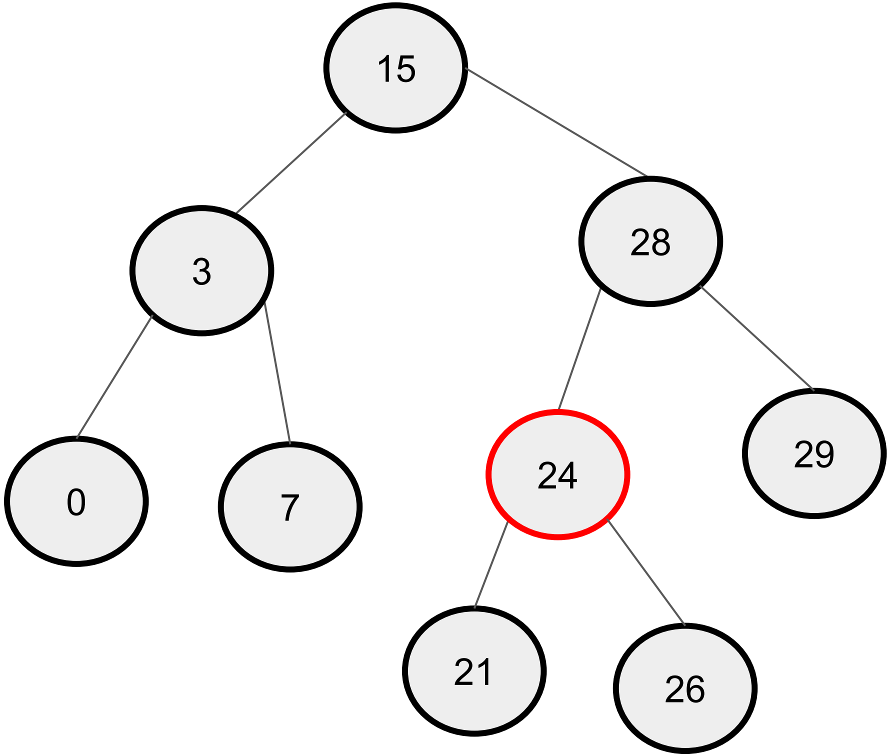


Insert: 15,21,7,24,0,26,3,28,<u>29</u>





Insert: 15,21,7,24,0,26,3,28,29



Bonus: LLRB Hard to Understand

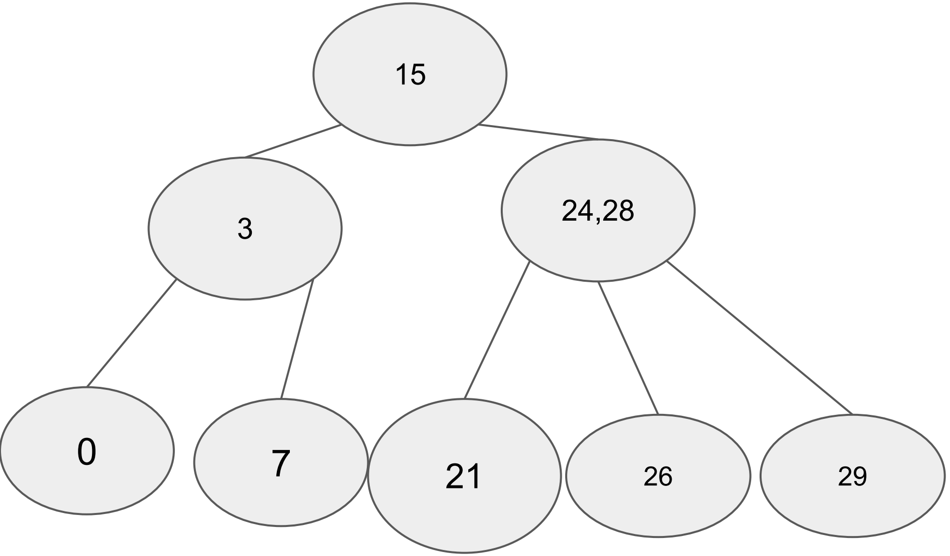
LLRB trees are “the same as” to 2-3 trees

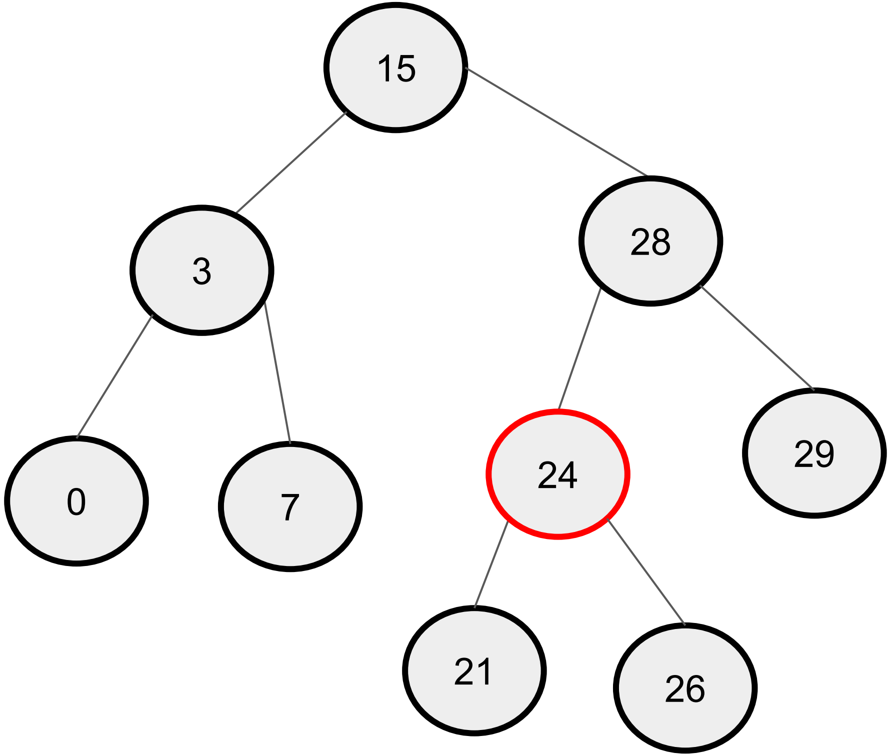
- binary tree equivalent of 2-3 trees

2-3 Tree	$\longleftrightarrow$ LLRB Tree
• all paths have same depth	• all paths have same # black nodes
• There are 3-nodes	- red nodes, left leaning
• Inserting 'overstuffed' nodes	• insert red nodes

Exercise: Compare the insert trace of the 2-3 tree vs the LLRB Tree

Notice how red nodes == 3-nodes!



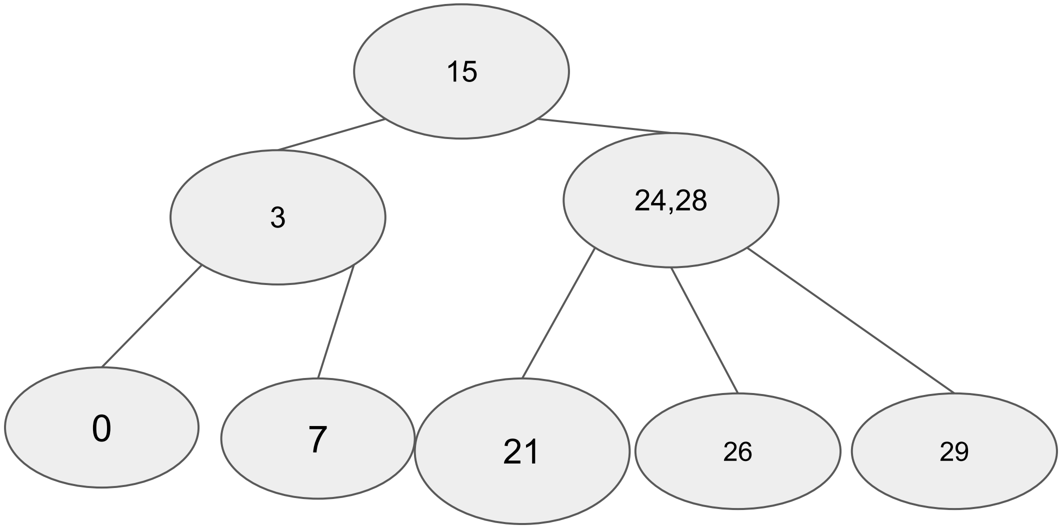


Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



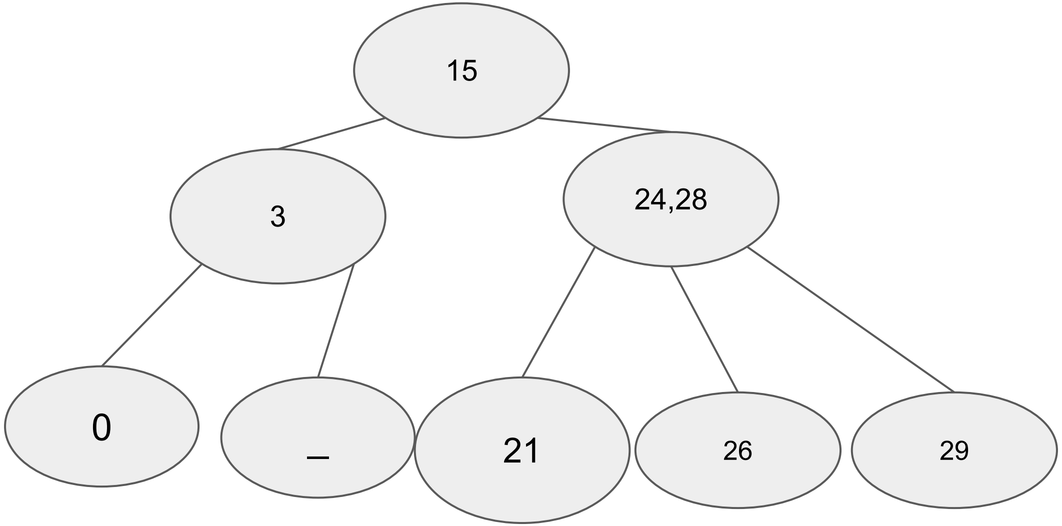
Intuition: I swap the deleted node **up to the root**

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



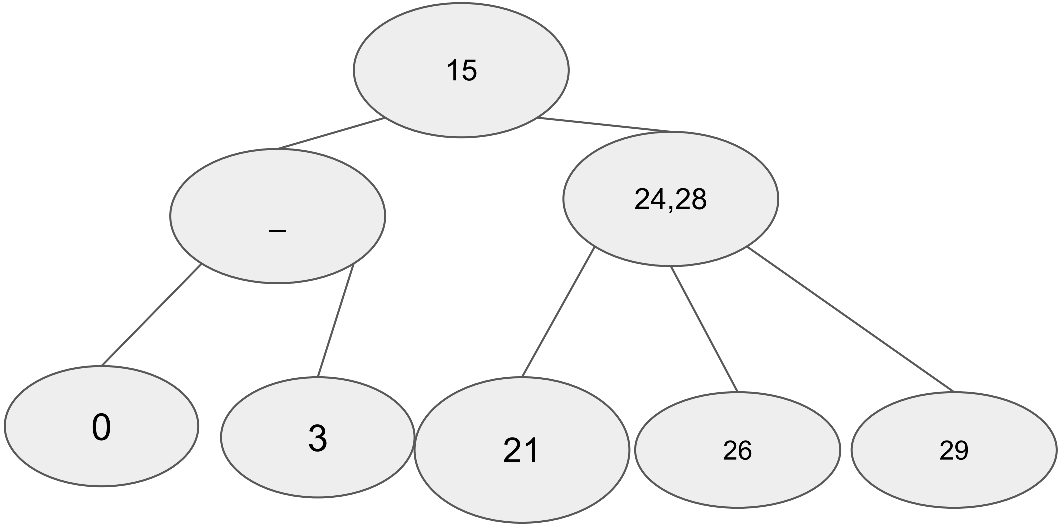
Intuition: I swap the deleted node **up to the root (let _ be deleted)**

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



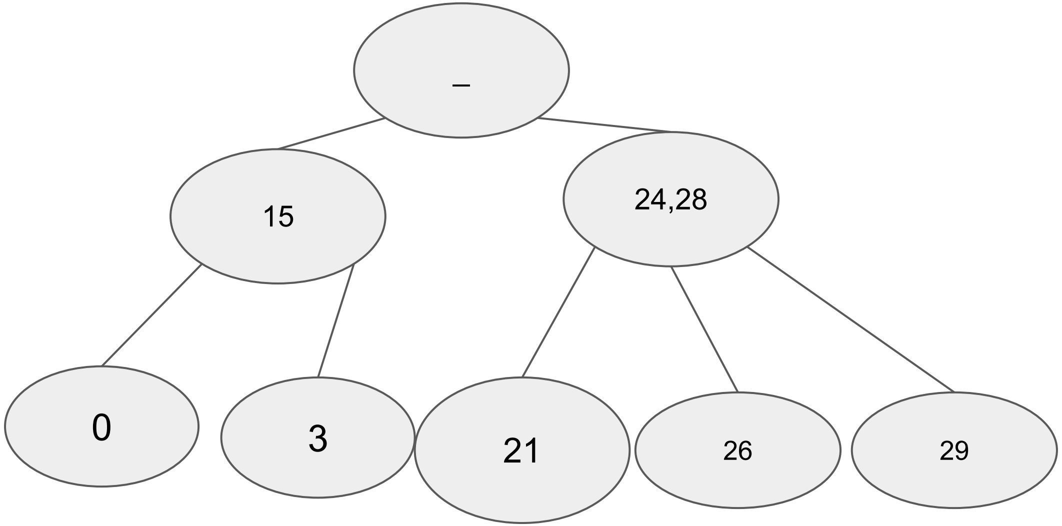
Intuition: I swap the deleted node **up to the root (let _ be deleted)**

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



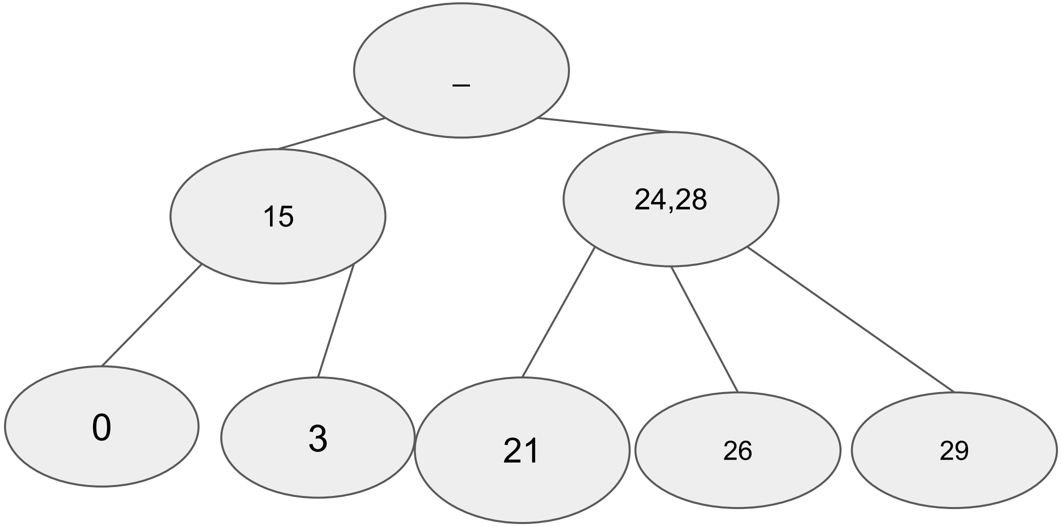
Intuition: I swap the deleted node **up to the root (let _ be deleted)**

Question 3

(Deletion) Show intermediate steps of the following questions:

$\boxed{\text{(1) How to delete 7 in the final 2-3 tree of Q1?}}$

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



Intuition: I swap the deleted node **up to the root (let _ be deleted)**

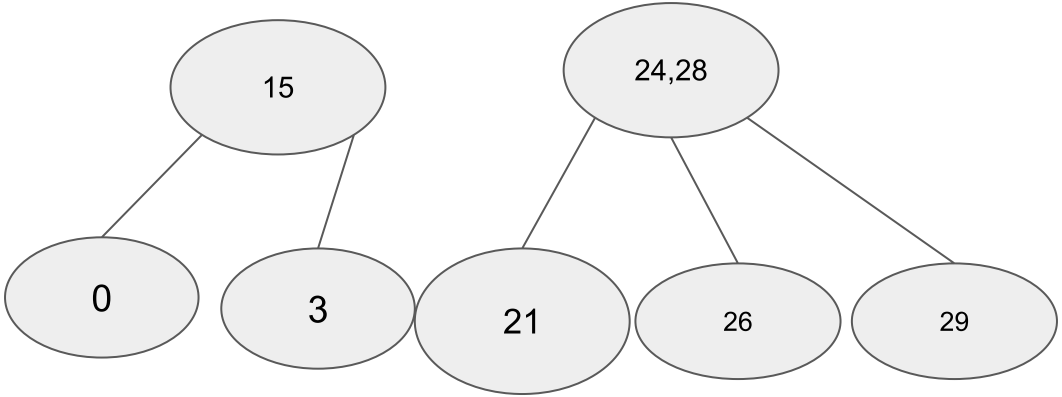
Delete root

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



Intuition: I swap the deleted node **up to the root (let _ be deleted)**

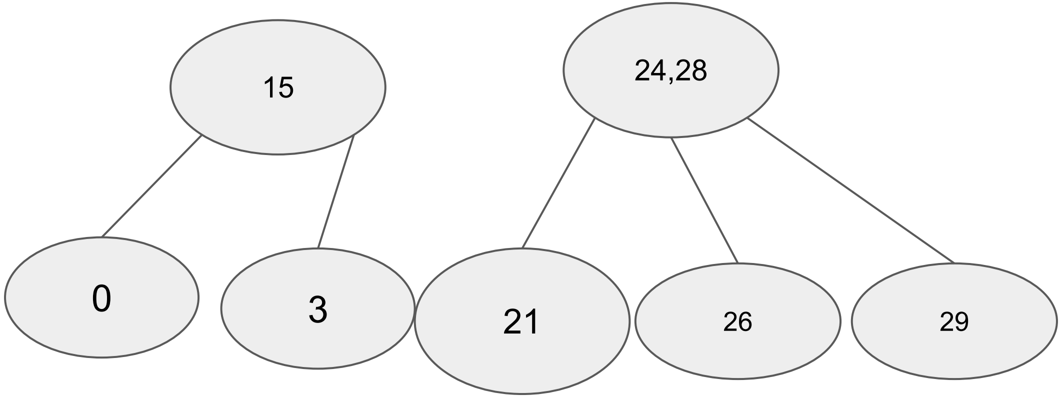
Delete root

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



Intuition: I swap the deleted node **up to the root (let _ be deleted)**

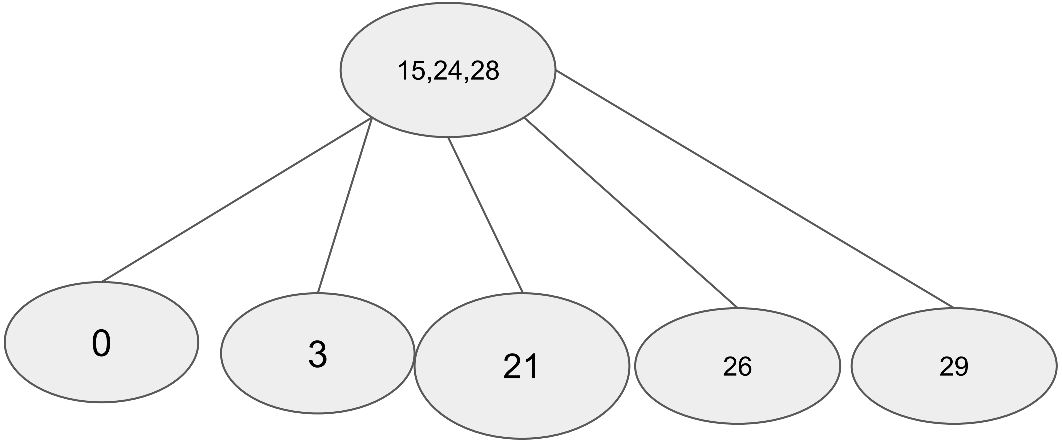
Delete root, merge children going downward

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



Intuition: I swap the deleted node **up to the root (let _ be deleted)**

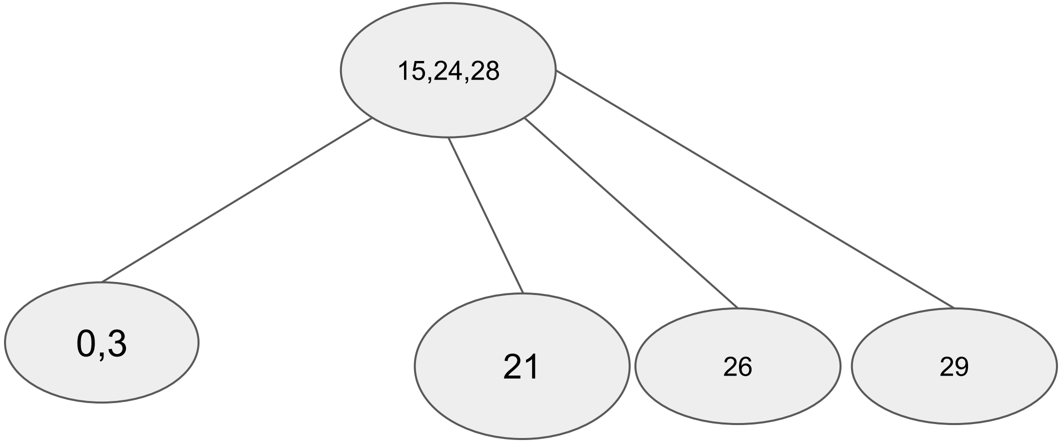
Delete root, merge children going downward

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



Intuition: I swap the deleted node **up to the root (let _ be deleted)**

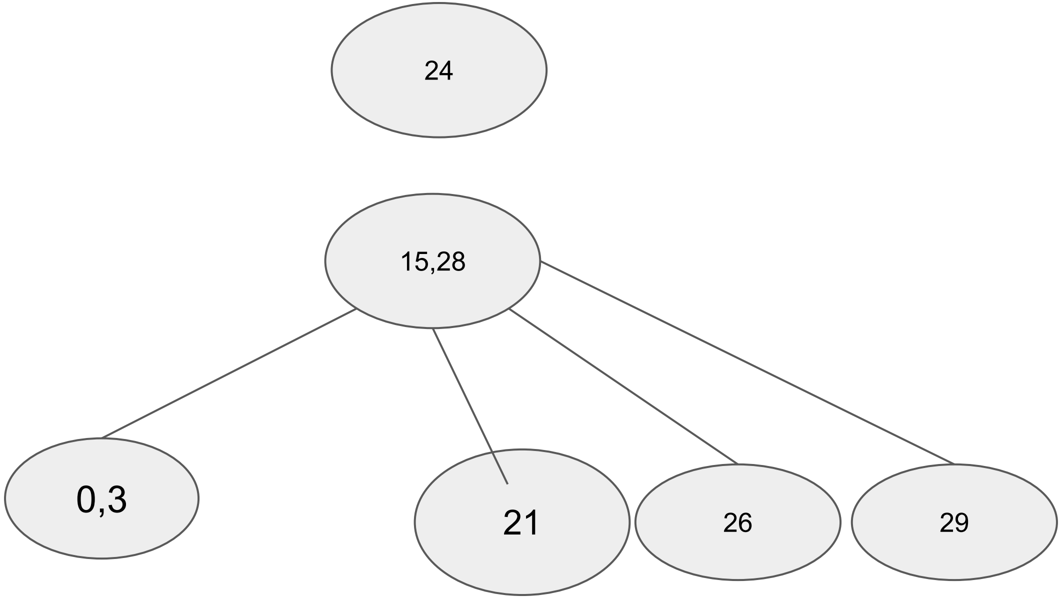
Delete root, merge children going downward, **lastly, fix any 4 node by pulling up**

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



Intuition: I swap the deleted node **up to the root (let _ be deleted)**

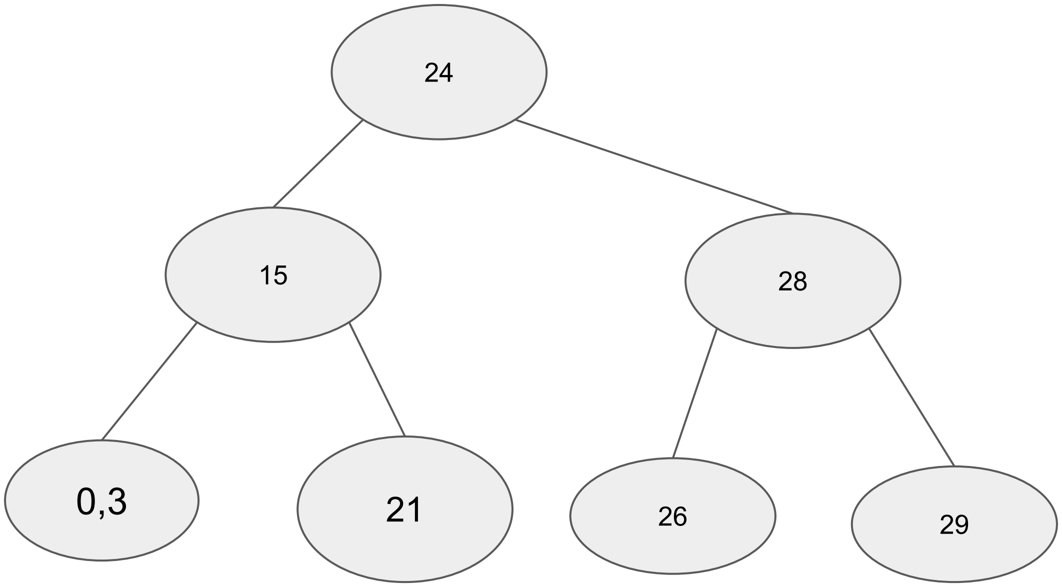
Delete root, merge children going downward, **lastly, fix any 4 node by pulling up**

Question 3

(Deletion) Show intermediate steps of the following questions:

(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?



Intuition: I swap the deleted node **up to the root (let _ be deleted)**

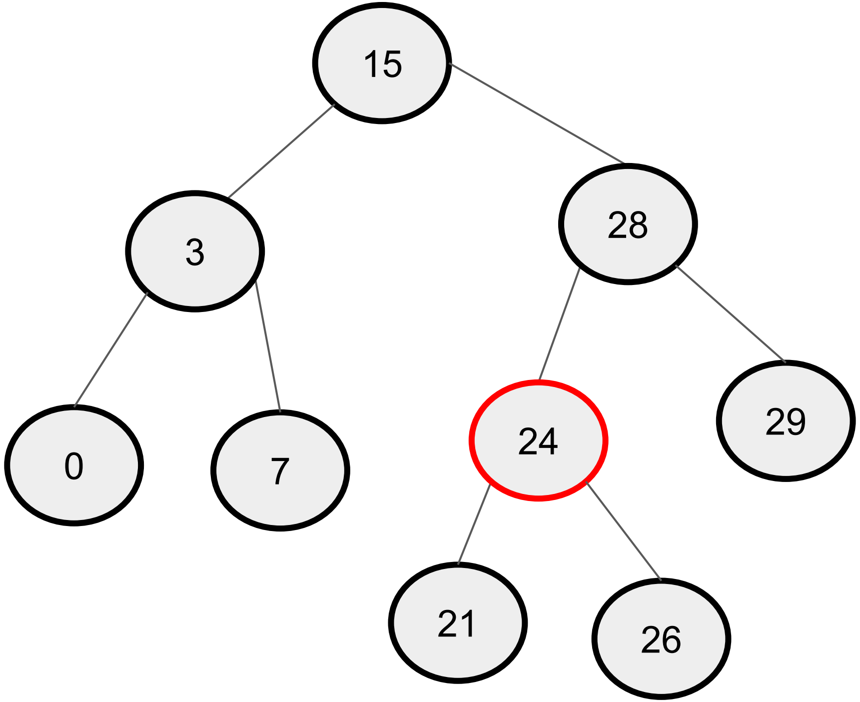
Delete root, merge children going downward, **lastly, fix any 4 node by pulling up**

Question 3

(Deletion) Show intermediate steps of the following questions:

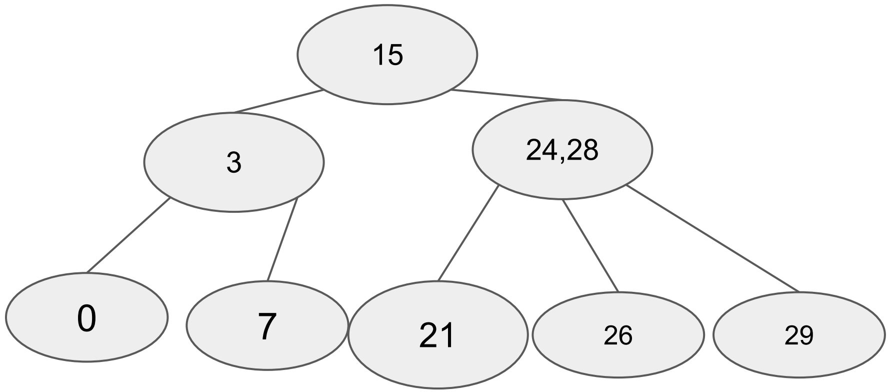
(1) How to delete 7 in the final 2-3 tree of Q1?

(2) How to delete 7 in the final Left–Leaning Red–Black tree of Q1?

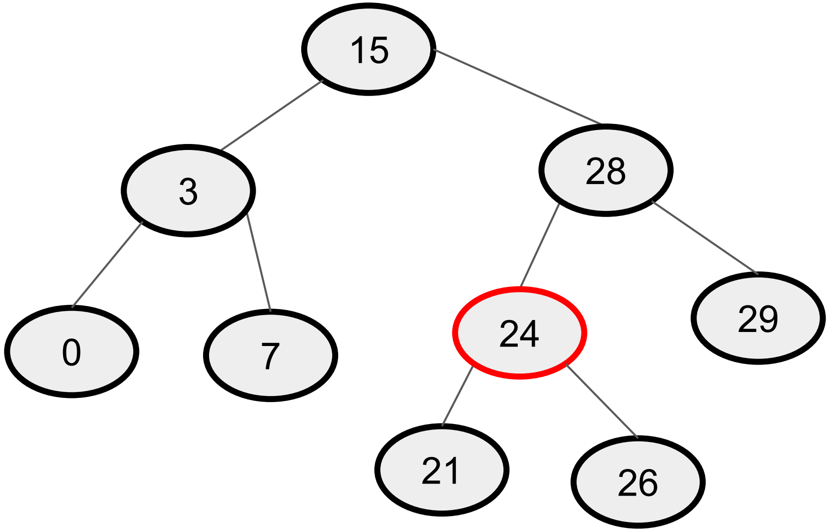


Deletion in LLRB is quite hard..

Idea: Use equivalence between LLRB and 2-3 trees



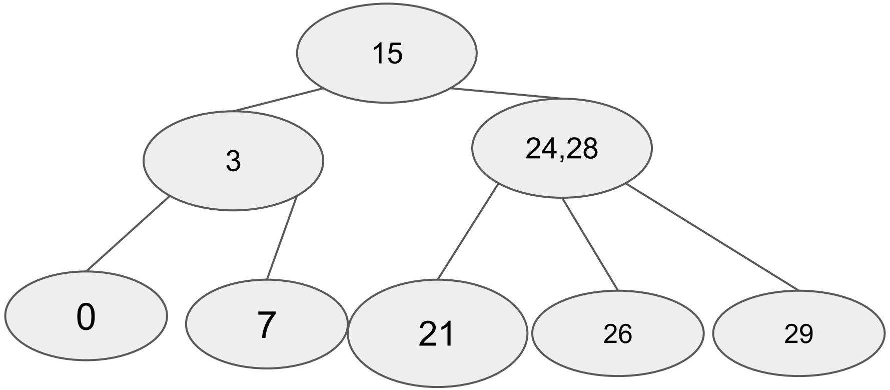
Idea: Use equivalence between LLRB and 2-3 trees



Take the LLRB

1) Turn it into a 2-3 tree
2) Run the delete algorithm
3) Turn it back into a LLRB tree

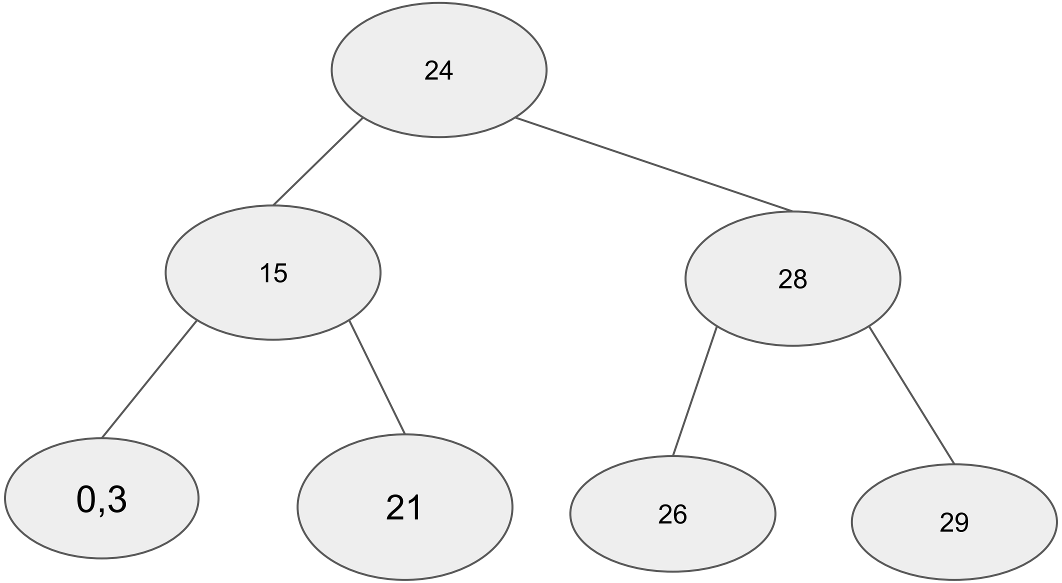
Idea: Use equivalence between LLRB and 2-3 trees



Take the LLRB

1) **Turn it into a 2-3 tree**
2) Run the delete algorithm
3) Turn it back into a LLRB tree

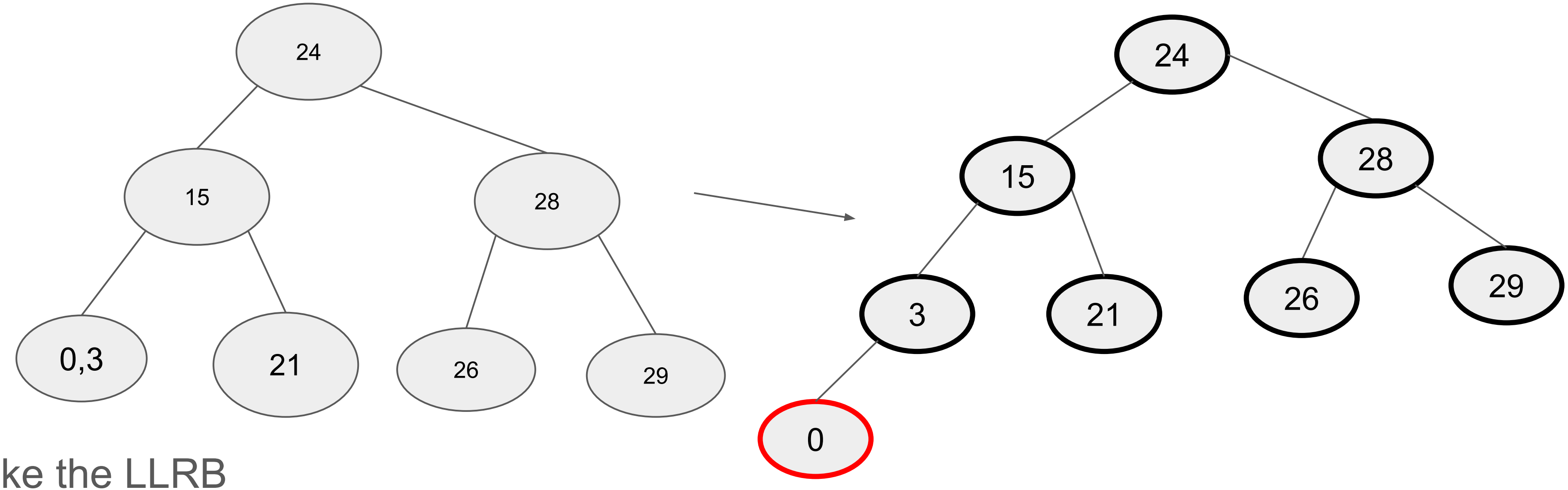
Idea: Use equivalence between LLRB and 2-3 trees



Take the LLRB

1) Turn it into a 2-3 tree
2) **Run the delete algorithm**
3) Turn it back into a LLRB tree

Idea: Use equivalence between LLRB and 2-3 trees



Take the LLRB

1) Turn it into a 2-3 tree
2) Run the delete algorithm
3) **Turn it back into a LLRB tree**

Keep things simple :)

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

The definition with t = 1

- Every node other than the root has at least 1 key. Every internal node has at least 1 children. The root must have at least 1 key
- Every node contains at most 2 values and 3 children

This is a 2-3 tree

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

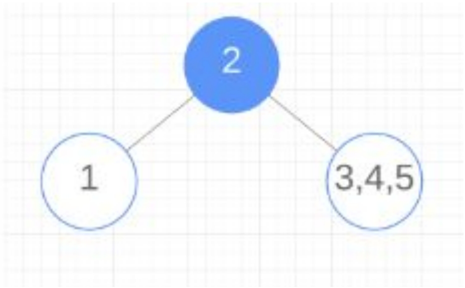
(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

The definition with t = 2

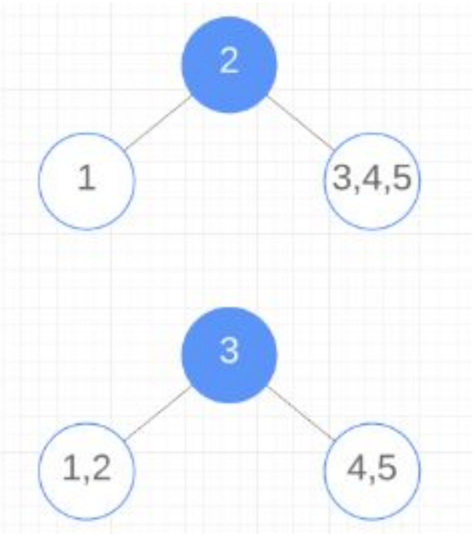
- Every node other than the root has at least 2 key. Every internal node has at least 2 children. The root must have at least 1 key
- Every node contains at most 3 values and 4 children

This is a 2-3-4 tree! How many 2-3-4 trees for {1,2,3,4,5}?

2-3-4 trees of {1,2,3,4,5}

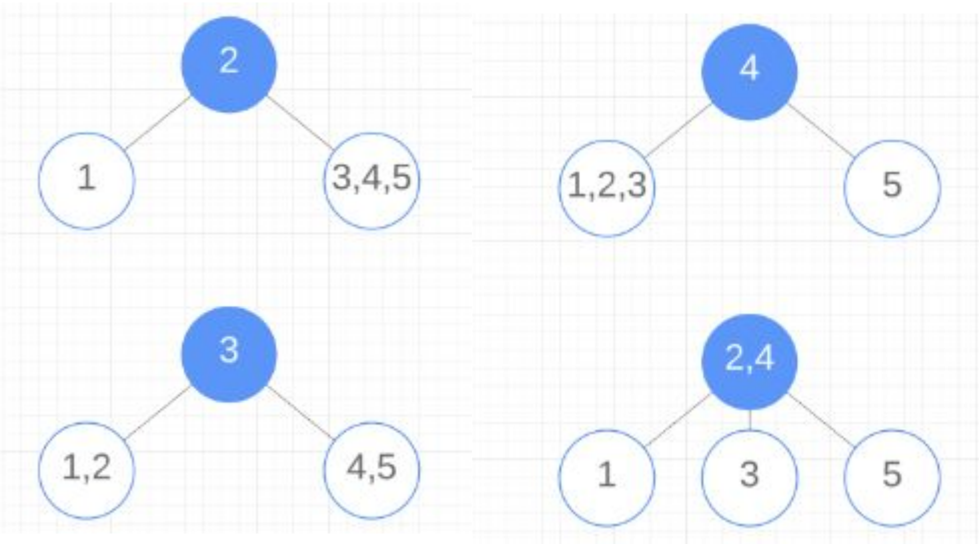


2-3-4 trees of {1,2,3,4,5}



2-3-4 trees of {1,2,3,4,5}

2-3-4 trees of {1,2,3,4,5}



Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find max # keys, first find max # nodes.

Let T_h be be max # nodes in B-tree with degree t and height h

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find max # keys, first find max # nodes.

Let T_h be be max # nodes in B-tree with degree t and height h

T_n = (1 root) + (max # of children) T_{h-1}

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find max # keys, first find max # nodes.

Let T_h be be max # nodes in B-tree with degree t and height h

T_n = (1 root) + 2t T_{h-1}

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find max # keys, first find max # nodes.

Let T_h be be max # nodes in B-tree with degree t and height h

T_n = (1 root) + 2t T_{h-1}

Solving for this we get $T_h = 1 + 2t + (2t)^2 + \ldots + (2t)^h$

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find max # keys, first find max # nodes.

Let T_h be be max # nodes in B-tree with degree t and height h

T_n = (1 root) + 2t T_{h-1}

Solving for this we get $T_h = 1 + 2t + (2t)^2 + \ldots + (2t)^h$

And since each node has at most (2t - 1) keys..

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find max # keys, first find max # nodes.

Let T_h be be max # nodes in B-tree with degree t and height h

T_n = (1 root) + 2t T_{h-1}

Solving for this we get $T_h = 1 + 2t + (2t)^2 + \ldots + (2t)^h$

And since each node has at most (2t - 1) keys..

Max keys = $(2t - 1)\, T_h = (2t)^{h+1} - 1$ **Exercise: Confirm this**

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find **min** # keys, root can have at the very least, **1** key

Min= 1 +

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find **min** # keys, root can have at the very least, **1** key

Then, it has 2 children.

Min= 1 + 2(..)

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find **min** # keys, root can have at the very least, **1** key

Then, it has 2 children.

Those children are internal, so they have at the very least t children.

Min= 1 + 2(t + grandchildren + grand-grandchildren + …)

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find **min** # keys, root can have at the very least, **1** key

Then, it has 2 children.

Those children are internal, so they have at the very least t children.

Min= 1 + 2(t + t^2 + grand-grandchildren + …)

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t - 1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t - 1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find **min** # keys, root can have at the very least, **1** key

Then, it has 2 children.

Those children are internal, so they have at the very least t children.

Each of those children have at the very least (t - 1) keys

$$\text{Min} = 1 + 2(t + t^2 + t^3 + \ldots + t^h)(t - 1)$$

Question 4

(B-tree) The notion of minimum degree appears in a more general definition of B-trees. Specifically, a B-tree is defined with a minimum degree t, that is saying

- Every node other than the root must have at least $t-1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.
- Every node may contain at most $2t-1$ keys. Therefore, an internal node may have at most $2t$ children.

(1) What is a B-tree with minimum degree 2? Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

(2) As a function of the minimum degree t, what is the maximum and minimum number of keys that can be stored in a B-tree of height h?

To find **min** # keys, root can have at the very least, **1** key

Then, it has 2 children.

Those children are internal, so they have at the very least t children.

Each of those children have at the very least (t - 1) keys

$$\text{Min} = 1 + 2(t + t^2 + t^3 + \ldots + t^h)(t - 1) = 2t^h - 1$$

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Red black tree facts

- Any two paths have the same number of black nodes
- There are never consecutive red nodes

Suppose we kno

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Red black tree facts

- Any two paths have the same number of black nodes
- There are never consecutive red nodes

nodes in a path = (# black nodes) + (# red nodes)

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Red black tree facts

- Any two paths have the same number of black nodes
- There are never consecutive red nodes

l_{max} = (# black nodes) + (MAX # red nodes)

l_{min} = (# black nodes) + (MIN # red nodes)

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Red black tree facts

- Any two paths have the same number of black nodes
- There are never consecutive red nodes

l_{max} = (# black nodes) + (MAX # red nodes)

l_{min} = (# black nodes) + (MIN # red nodes)

≥ (# black nodes)

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Red black tree facts

- Any two paths have the same number of black nodes
- There are never consecutive red nodes

l_{max} = (# black nodes) + (MAX # red nodes)

l_{min} = (# black nodes) + (MIN # red nodes)

$\geq$ (# black nodes)

So l_{max} / l_{min} $\leq$ 1 + (MAX # red nodes) / (# black nodes)

$l_{max} / l_{min} \leq 1 + (\text{MAX \# red nodes}) / (\text{\# black nodes})$

Suppose l_B is # black nodes, l_{RMAX} is MAX # red nodes

$$l_{max} / l_{min} \leq 1 + (\text{MAX \# red nodes}) / (\text{\# black nodes})$$

Suppose l_B is # black nodes, l_{RMAX} is MAX # red nodes

Since there are never consecutive red nodes,

$$l_{RMAX} \leq \lfloor (l_{max} - 1)/2 \rfloor$$

$l_{max} / l_{min} \leq 1 + (\text{MAX \# red nodes}) / (\text{\# black nodes})$

Suppose l_B is # black nodes, l_{RMAX} is MAX # red nodes

Since there are never consecutive red nodes,

$$l_{RMAX} \leq \lfloor (l_{max} - 1)/2 \rfloor$$

Then, $l_B \geq \lceil (l_{max} / 2) \rceil$

$l_{max} / l_{min} \leq 1 + (\text{MAX \# red nodes}) / (\text{\# black nodes})$

Suppose l_B is # black nodes, l_{RMAX} is MAX # red nodes

Since there are never consecutive red nodes,

$$l_{RMAX} \leq \lfloor (l_{max} - 1)/2 \rfloor$$

Then, $l_B \geq \lceil (l_{max} / 2) \rceil$

So,

$$l_{max} / l_{min} \leq 1 + l_{RMAX} / l_B \leq$$

$l_{max} / l_{min} \leq 1 + (\text{MAX \# red nodes}) / (\text{\# black nodes})$

Suppose l_B is # black nodes, l_{RMAX} is MAX # red nodes

Since there are never consecutive red nodes,

$$l_{RMAX} \leq \lfloor (l_{max} - 1)/2 \rfloor$$

Then, $l_B \geq \lceil (l_{max} / 2) \rceil$

So,

$$l_{max} / l_{min} \leq 1 + l_{RMAX} / l_B \leq 1 + \lfloor (l_{max} - 1)/2 \rfloor \; / \; \lceil (l_{max} / 2) \rceil$$

$l_{max} / l_{min} \leq 1 + (\text{MAX \# red nodes}) / (\text{\# black nodes})$

Suppose l_B is # black nodes, l_{RMAX} is MAX # red nodes

Since there are never consecutive red nodes,

$$l_{RMAX} \leq \lfloor (l_{max} - 1)/2 \rfloor$$

Then, $l_B \geq \lceil (l_{max} / 2) \rceil$

So,

$$l_{max} / l_{min} \leq 1 + l_{RMAX} / l_B \leq 1 + \lfloor (l_{max} - 1)/2 \rfloor \;/\; \lceil (l_{max} / 2) \rceil \leq 1 + 1 = \mathbf{2}$$

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Red black tree facts

- Any two paths have the same number of black nodes
- There are never consecutive red nodes

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

(2) Given a red-black tree with n keys, what is the largest possible ratio of red internal nodes to black internal nodes?

Red black tree facts

- Any two paths have the same number of black nodes
- **There are never consecutive red nodes**

Every red node has two black children, so this a ratio of ⅓

How about black nodes?

Question 5

(Red-Black tree)

(1) Given any red black tree, let the length of the shortest path from root to a leaf be ℓ_{min} and the length of the longest path from root to a leaf be ℓ_{max}. How large can ℓ_{max}/ℓ_{min} be?

$\boxed{\text{(2) Given a red-black tree with } n \text{ keys, what is the largest possible ratio of red internal nodes to black internal nodes?}}$

Red black tree facts

- Any two paths have the same number of black nodes
- **There are never consecutive red nodes**

Every red node has two black children, so this a ratio of ⅓

A black node *can* have two red children, so at best our ratio is ⅔

Question 6

(AVL tree) An AVL tree is a binary search tree that is *height balanced*: for each node x, the heights of the left and right subtrees of x differ by at most 1.

Show that an AVL tree with n nodes has height $h = \mathcal{O}(\log n)$.

Hint1: show that an AVL tree of height h has at least $F_h - 1$ nodes, where F_h is the h-th Fibonacci number. Hint2: you may use the following fact of the Fibonacci number:

$$F_h = \left\lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \right\rfloor \quad \text{, with } \phi = \frac{\sqrt{5}+1}{2}.$$

Let T_h denote the minimum size of an AVL tree of height h

What is the recurrence?

Question 6

(AVL tree) An AVL tree is a binary search tree that is *height balanced*: for each node x, the heights of the left and right subtrees of x differ by at most 1.

Show that an AVL tree with n nodes has height $h = \mathcal{O}(\log n)$.

Hint1: show that an AVL tree of height h has at least $F_h - 1$ nodes, where F_h is the h-th Fibonacci number. Hint2: you may use the following fact of the Fibonacci number:

$$F_h = \left\lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \right\rfloor \quad \text{, with } \phi = \frac{\sqrt{5}+1}{2}.$$

Let T_h denote the minimum size of an AVL tree of height h

What is the recurrence?

$T_h = T_{h-1} + T_{h-2}$ + (1 root node)

(**height of left/right trees differs by at most 1)**

Question 6

(AVL tree) An AVL tree is a binary search tree that is *height balanced*: for each node x, the heights of the left and right subtrees of x differ by at most 1.

Show that an AVL tree with n nodes has height $h = \mathcal{O}(\log n)$.

Hint1: show that an AVL tree of height h has at least $F_h - 1$ nodes, where F_h is the h-th Fibonacci number. Hint2: you may use the following fact of the Fibonacci number:

$$F_h = \left\lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \right\rfloor \quad \text{, with } \phi = \frac{\sqrt{5}+1}{2}.$$

Let T_h denote the minimum size of an AVL tree of height h

What is the recurrence?

$T_h = T_{h-1} + T_{h-2}$ + (1 root node)

(**height of left/right trees differs by at most 1)**

This is the fibonacci number!

Question 6

(AVL tree) An AVL tree is a binary search tree that is *height balanced*: for each node x, the heights of the left and right subtrees of x differ by at most 1.

Show that an AVL tree with n nodes has height $h = \mathcal{O}(\log n)$.

Hint1: show that an AVL tree of height h has at least $F_h - 1$ nodes, where F_h is the h-th Fibonacci number. Hint2: you may use the following fact of the Fibonacci number:

$$F_h = \left\lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \right\rfloor \text{, with } \phi = \frac{\sqrt{5}+1}{2}.$$

Let T_h denote the minimum size of an AVL tree of height h

What is the recurrence?

$T_h = T_{h-1} + T_{h-2}$ + (1 root node)

(**height of left/right trees differs by at most 1)**

This is the fibonacci number! $T_h = F_h - 1$

Question 6

(AVL tree) An AVL tree is a binary search tree that is *height balanced*: for each node x, the heights of the left and right subtrees of x differ by at most 1.

Show that an AVL tree with n nodes has height $h = \mathcal{O}(\log n)$.

Hint1: show that an AVL tree of height h has at least $F_h - 1$ nodes, where F_h is the h-th Fibonacci number. Hint2: you may use the following fact of the Fibonacci number:

$$F_h = \left\lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \right\rfloor \quad \text{, with } \phi = \frac{\sqrt{5}+1}{2}.$$

Let T_h denote the minimum size of an AVL tree of height h

What is the recurrence?

$T_h = T_{h-1} + T_{h-2} + (1 \text{ root node})$

(**height of left/right trees differs by at most 1)**

This is the fibonacci number! $T_h = F_h - 1 < c^h$ for some constant c

Question 6

(AVL tree) An AVL tree is a binary search tree that is *height balanced*: for each node x, the heights of the left and right subtrees of x differ by at most 1.

Show that an AVL tree with n nodes has height $h = \mathcal{O}(\log n)$.

Hint1: show that an AVL tree of height h has at least $F_h - 1$ nodes, where F_h is the h-th Fibonacci number. Hint2: you may use the following fact of the Fibonacci number:

$$F_h = \left\lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \right\rfloor \text{ , with } \phi = \frac{\sqrt{5}+1}{2}.$$

Let T_h denote the minimum size of an AVL tree of height h

What is the recurrence?

$T_h = T_{h-1} + T_{h-2}$ + (1 root node)

(**height of left/right trees differs by at most 1)**

This is the fibonacci number! $T_h = F_h - 1 < c^h$ for some constant c

Since, $T_h = n$, we have $c^h < n$ so $h = O(\log n)$