PSO 9

WE ARE SO BACK (from Fall break)

How was the midterm?

Project due this thursday

Question 1

(Insertion and Deletion)

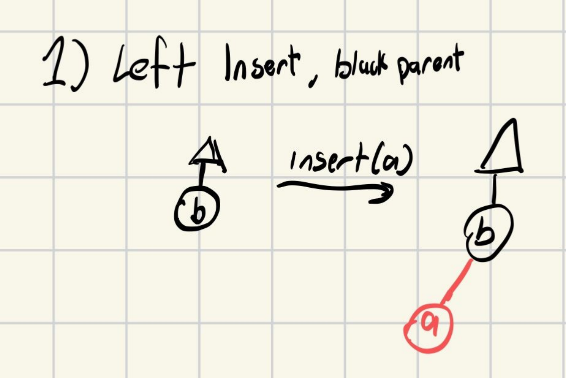
(1) Insert $\{15, 21, 7, 24, 0, 26, 3, 28, 29\}$ into an initially empty Left–Leaning Red–Black tree.

(2) Delete 7 in the final Left–Leaning Red–Black tree obtained in question (1).

LLRB Trees, what are they?

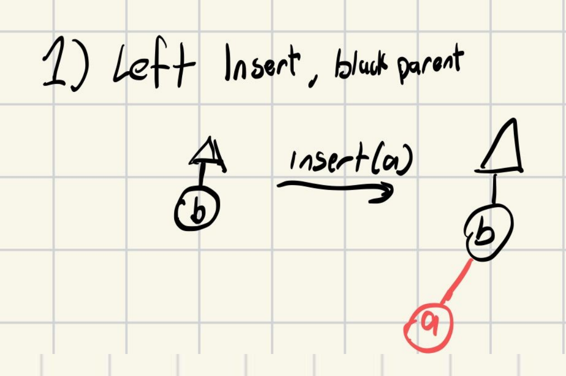
Types of LLRB inserts

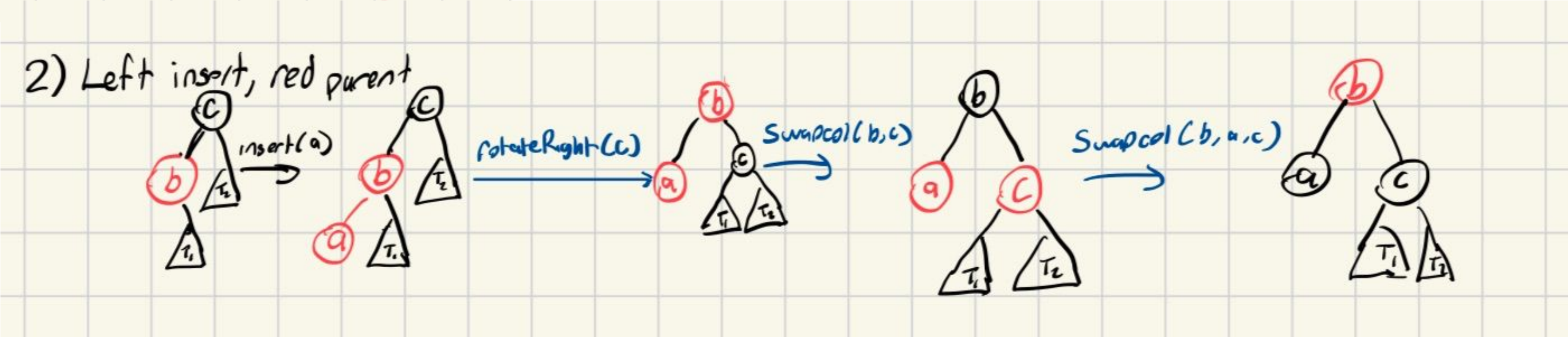
Always insert in a red node



Types of LLRB inserts

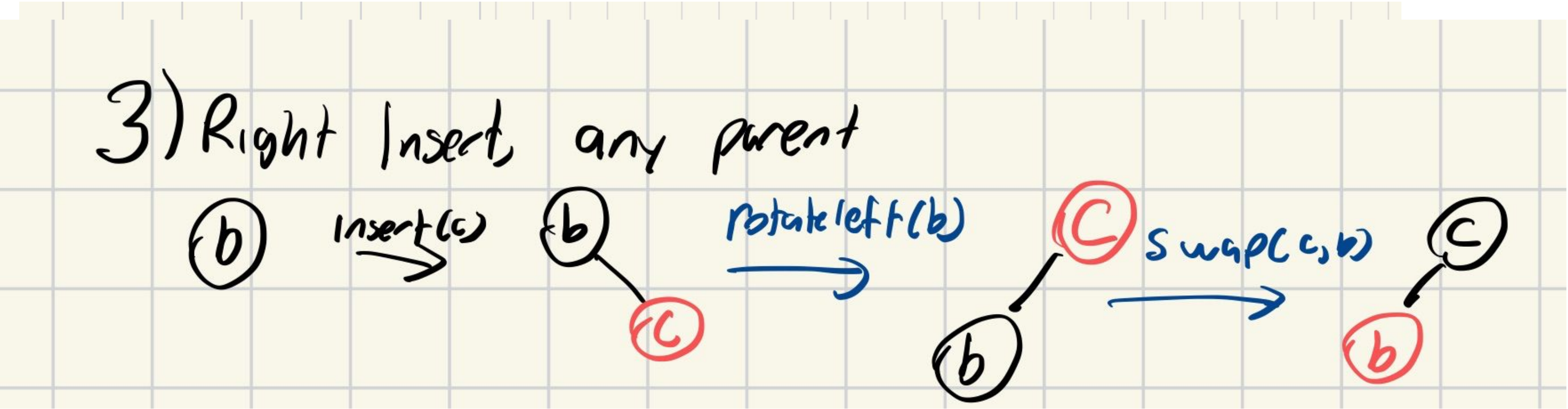
Always insert in a red node





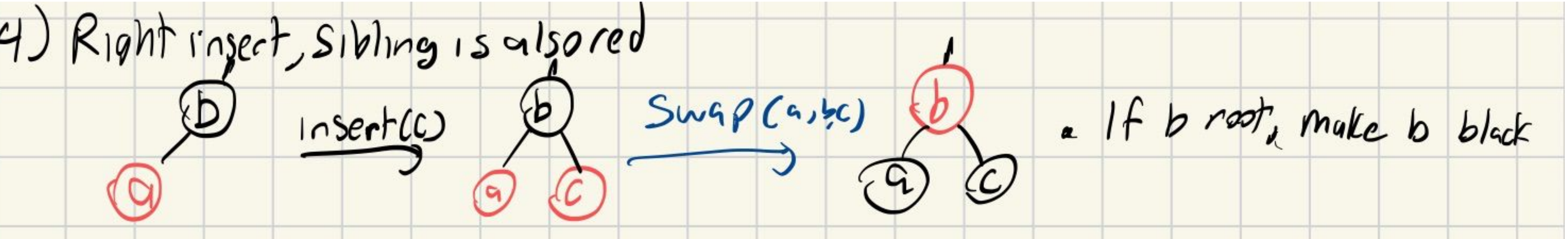
Types of LLRB inserts

Always insert in a red node



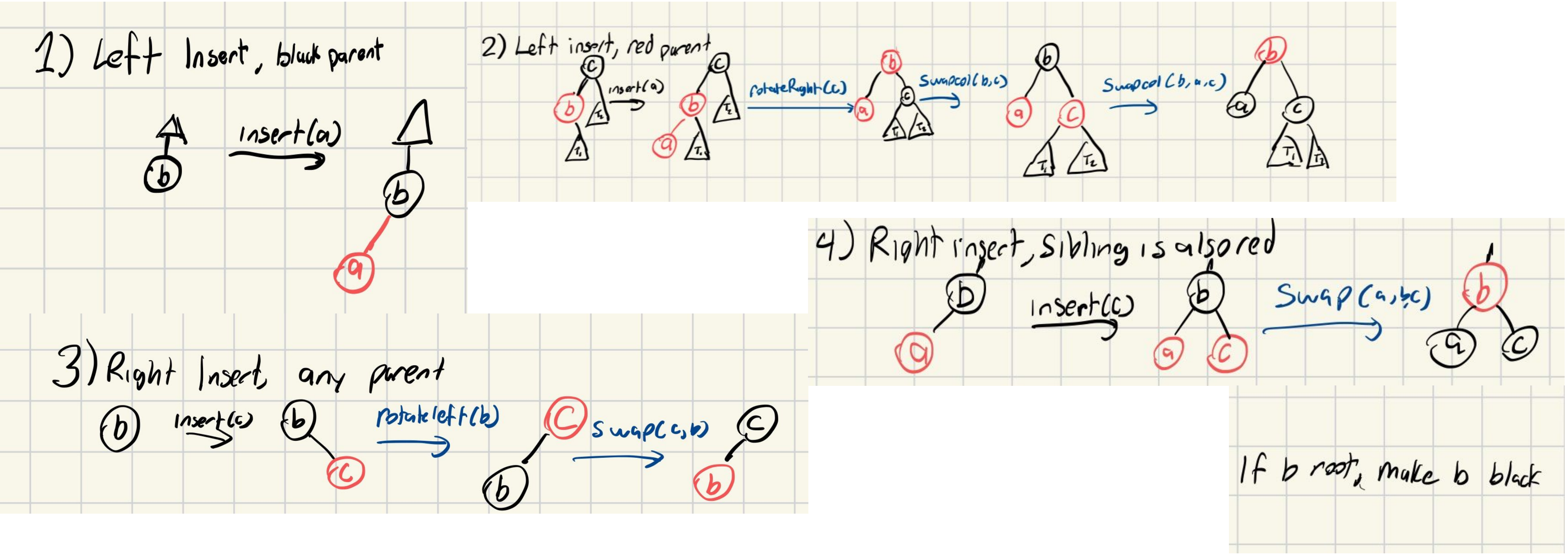
Types of LLRB inserts

Always insert in a red node



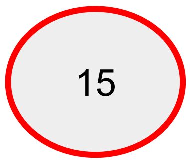
Types of LLRB inserts

Always insert in a red node



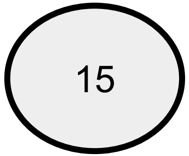
Insert: <u>15</u>,21,7,24,0,26,3,28,29

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



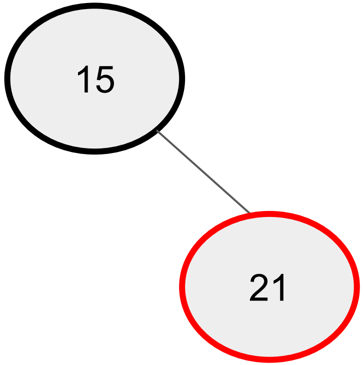
Insert: <u>15</u>,21,7,24,0,26,3,28,29

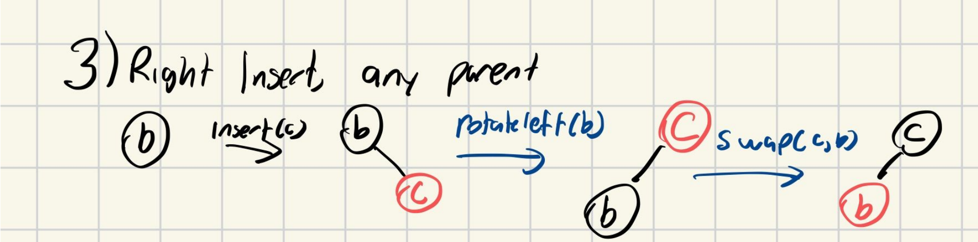
If root red, make it black



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

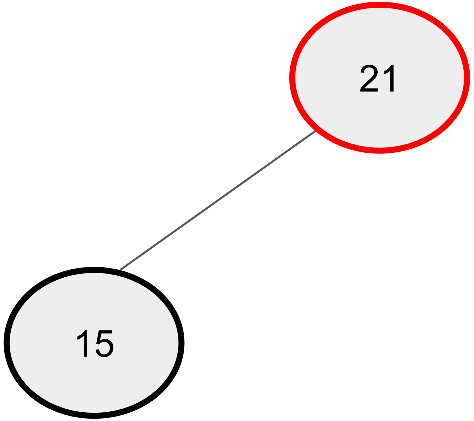
If root red, make it black

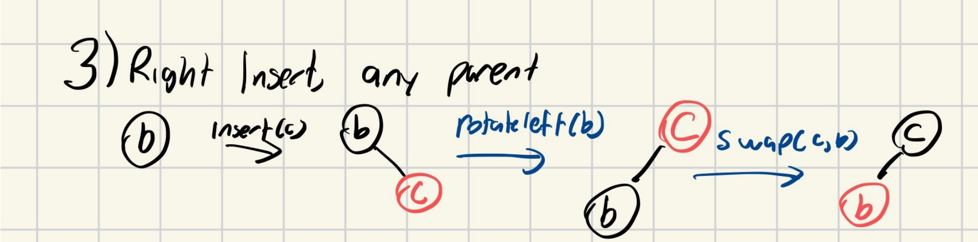




Insert: 15,21,7,24,0,26,3,28,29

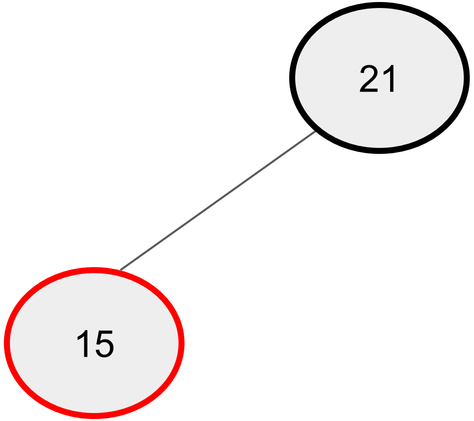
If root red, make it black

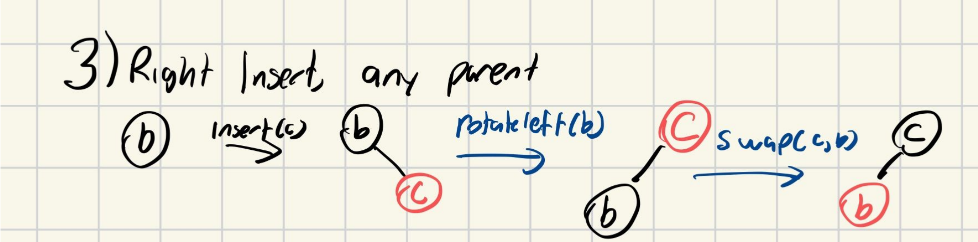




Insert: 15,21,7,24,0,26,3,28,29

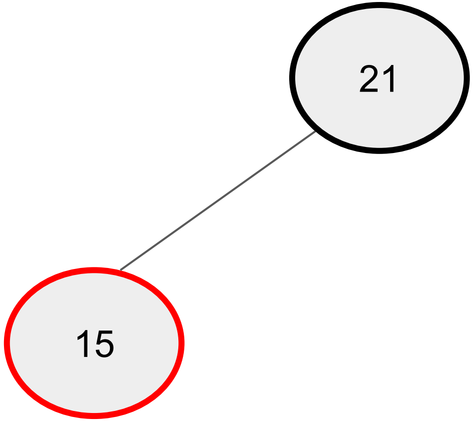
If root red, make it black





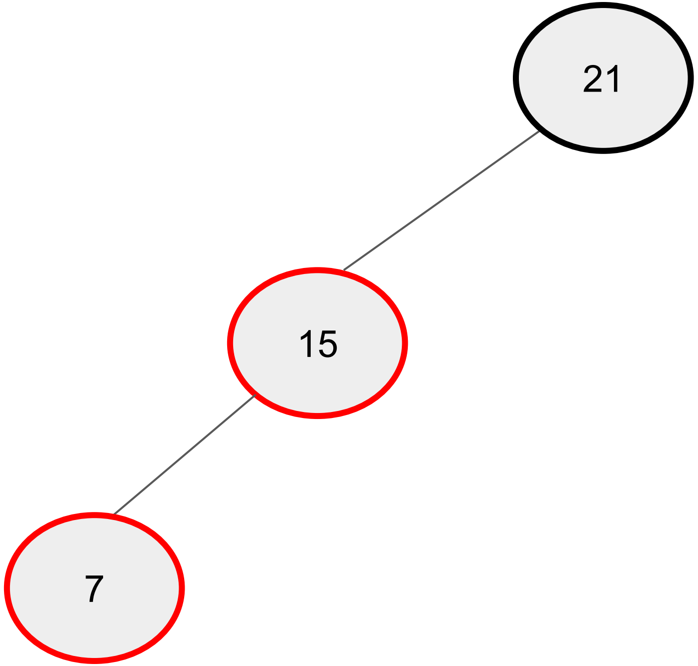
Insert: 15,21,7,24,0,26,3,28,29

If root red, make it black



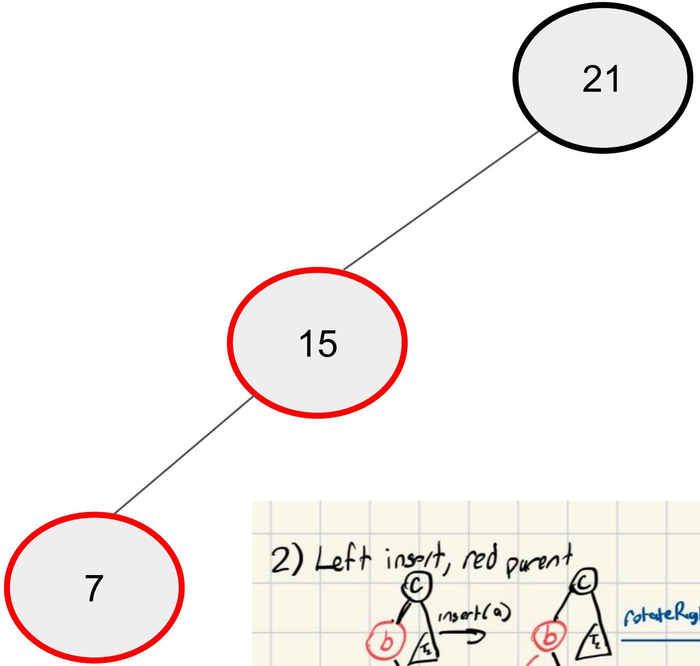
Insert: 15,21,<u>7</u>,24,0,26,3,28,29

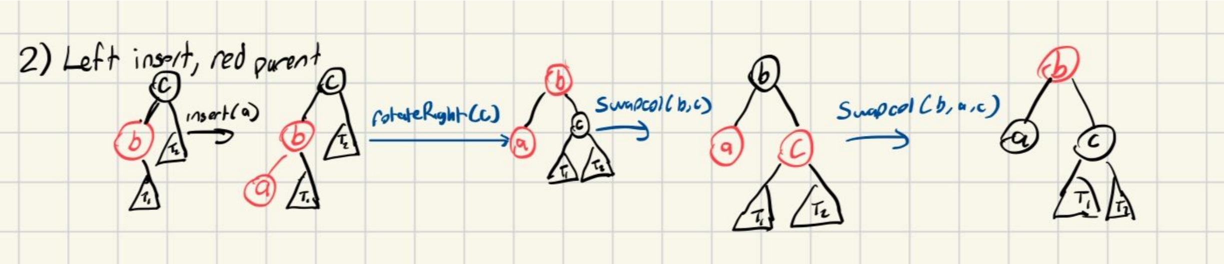
If root red, make it black



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

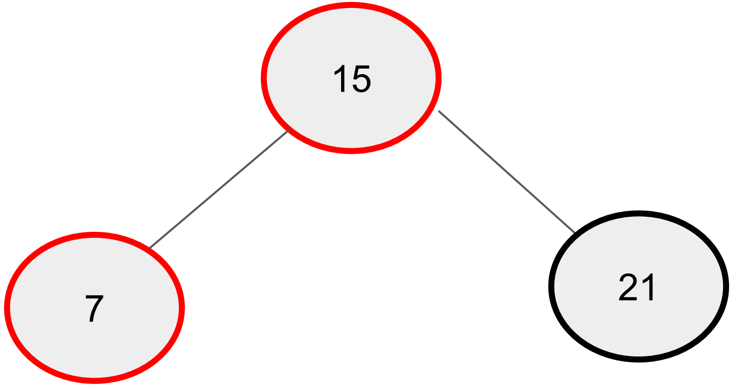
If root red, make it black

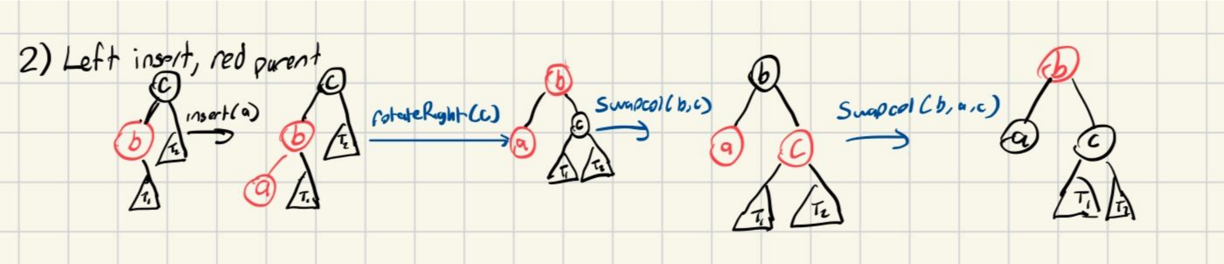




Insert: 15,21,7,24,0,26,3,28,29

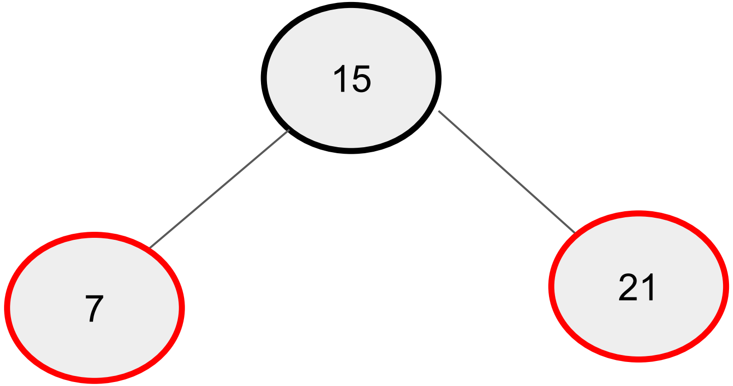
If root red, make it black

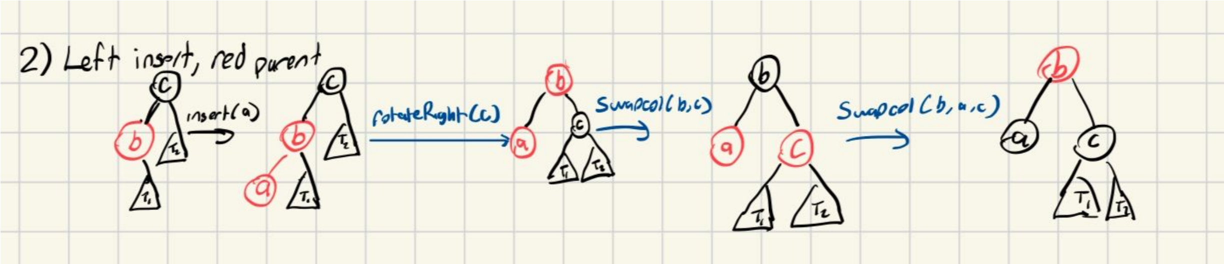




Insert: 15,21,7,24,0,26,3,28,29

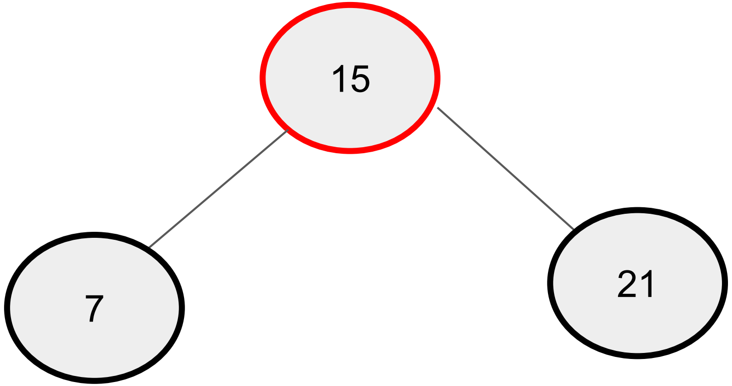
If root red, make it black

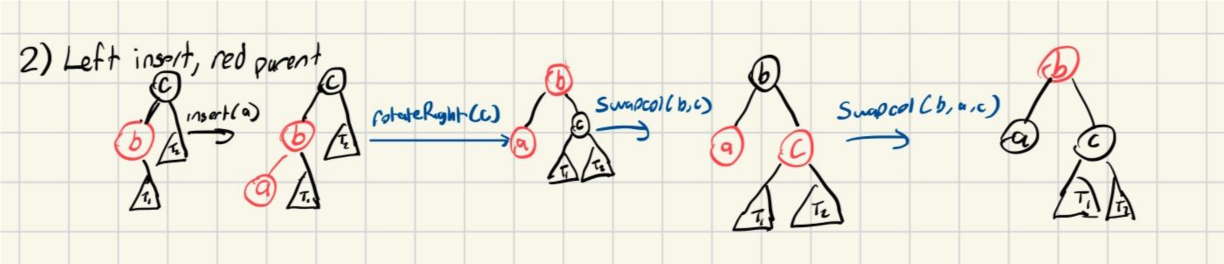




Insert: 15,21,<u>7</u>,24,0,26,3,28,29

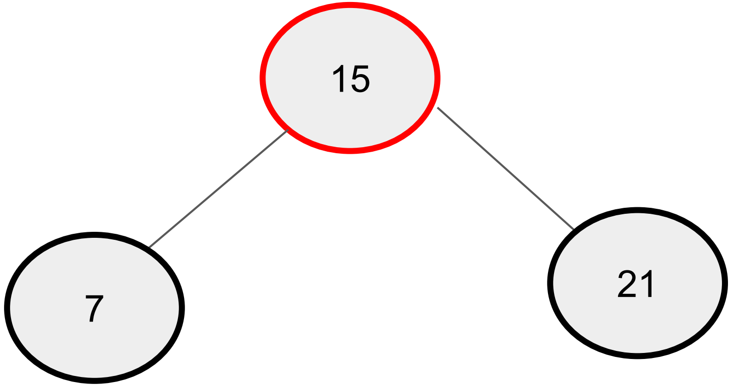
If root red, make it black

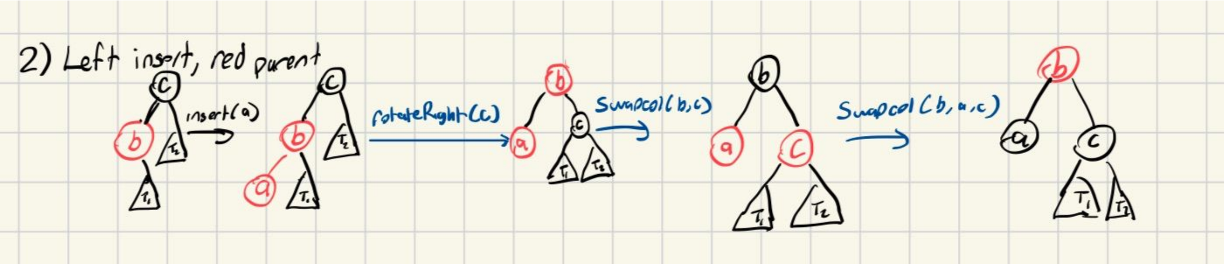




Insert: 15,21,7,24,0,26,3,28,29

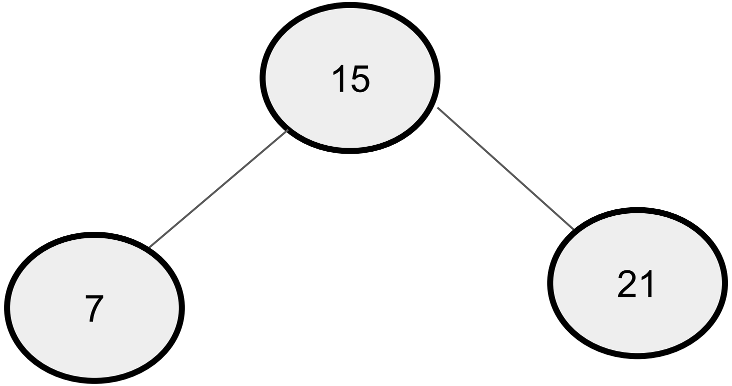
If root red, make it black

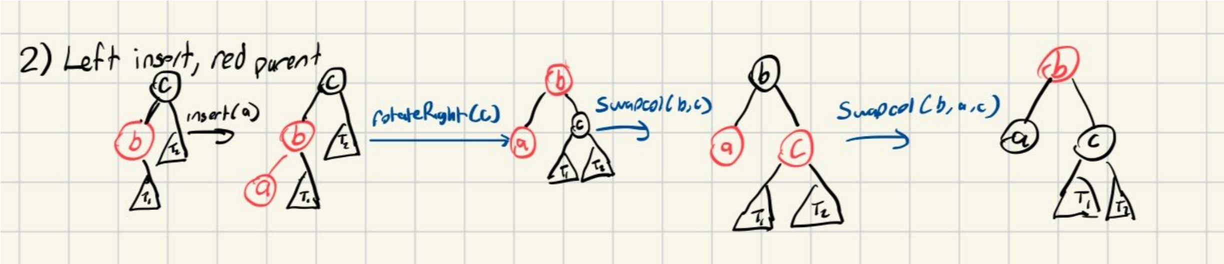




Insert: 15,21,<u>7</u>,24,0,26,3,28,29

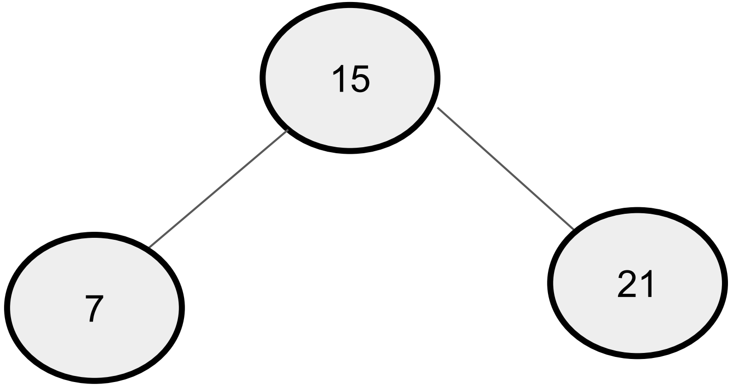
If root red, make it black





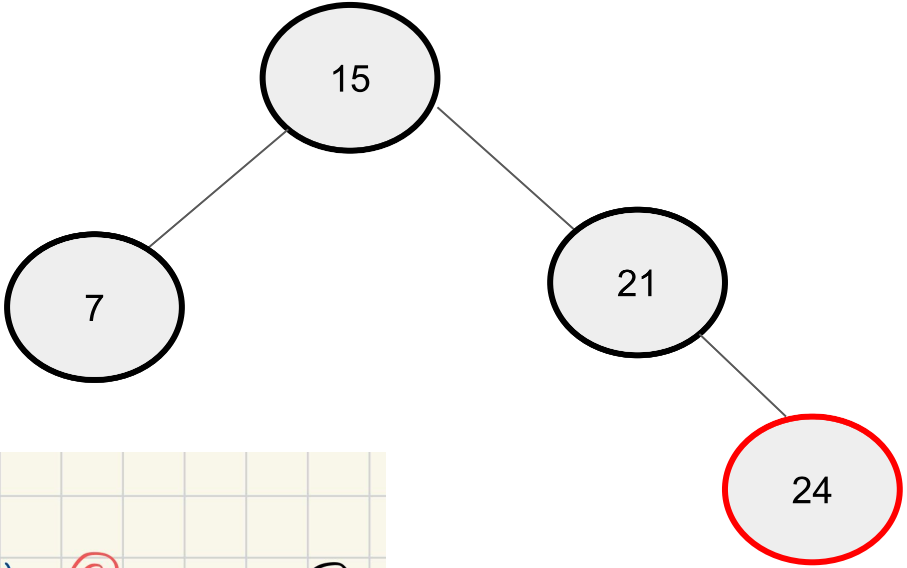
Insert: 15,21,7,24,0,26,3,28,29

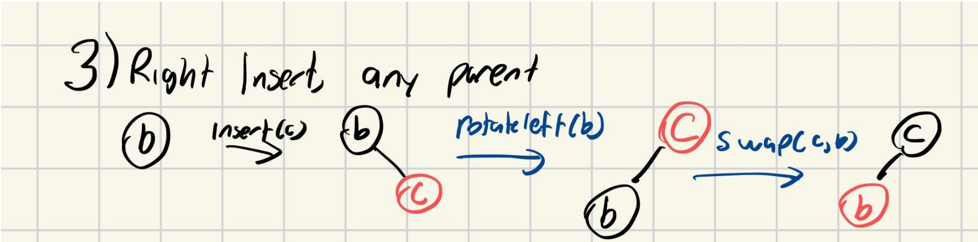
If root red, make it black



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

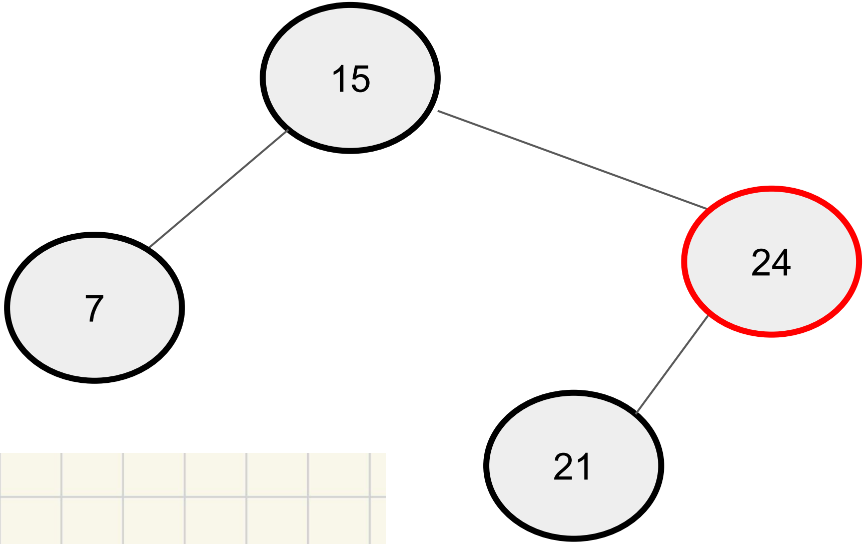
If root red, make it black

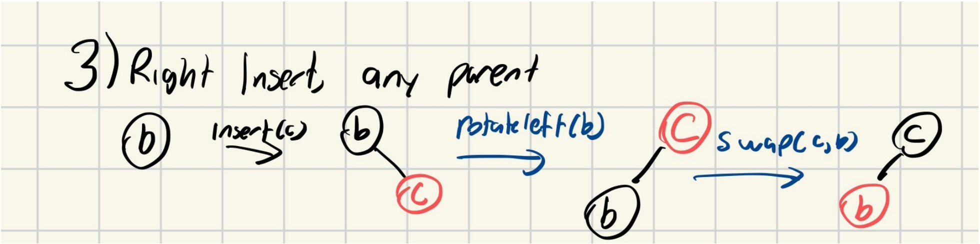




Insert: 15,21,7,24,0,26,3,28,29

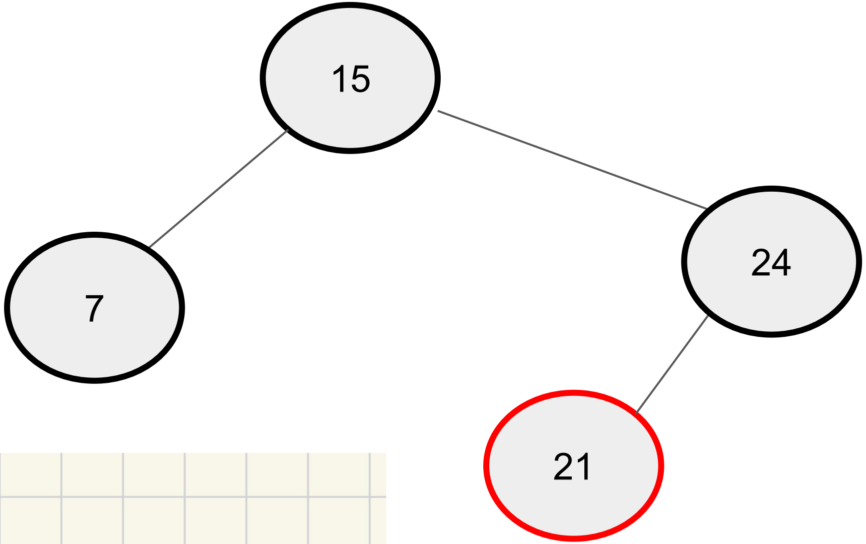
If root red, make it black

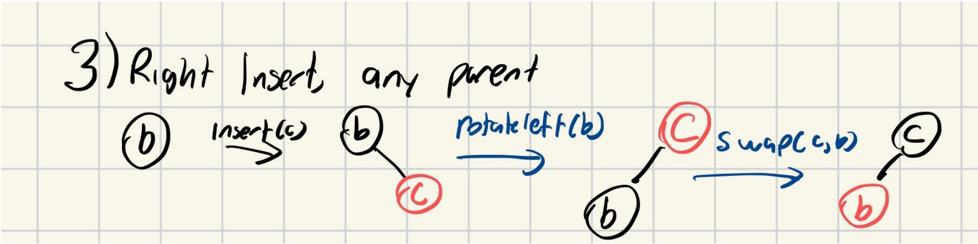




Insert: 15,21,7,24,0,26,3,28,29

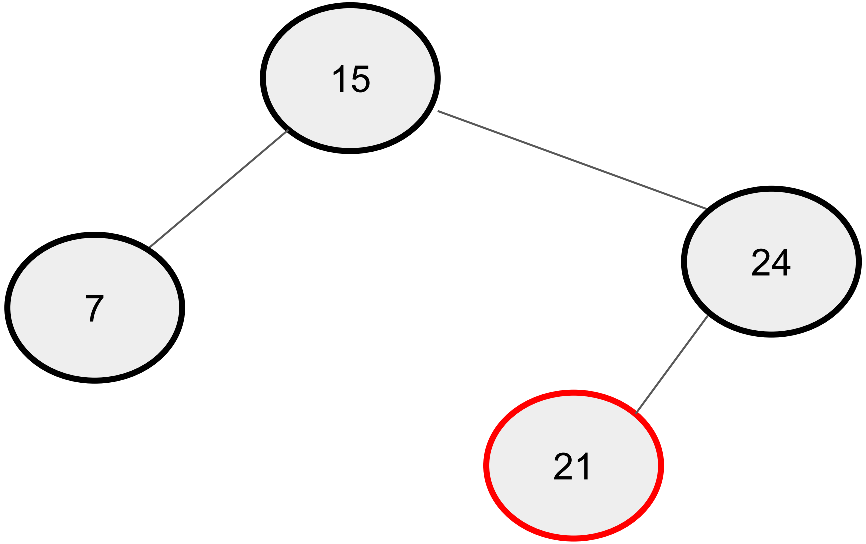
If root red, make it black





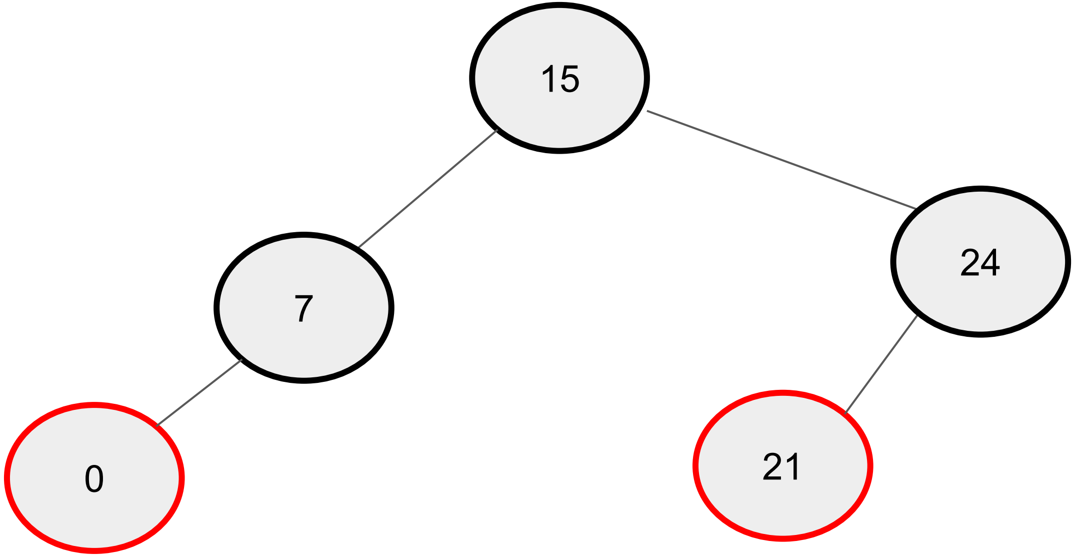
Insert: 15,21,7,24,<u>0</u>,26,3,28,29

If root red, make it black



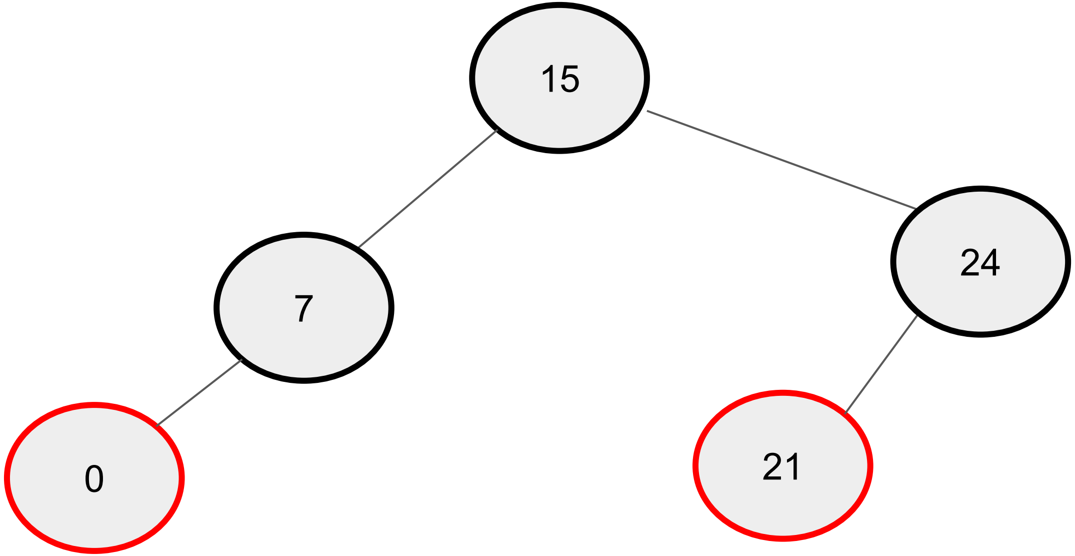
Insert: 15,21,7,24,<u>0</u>,26,3,28,29

If root red, make it black



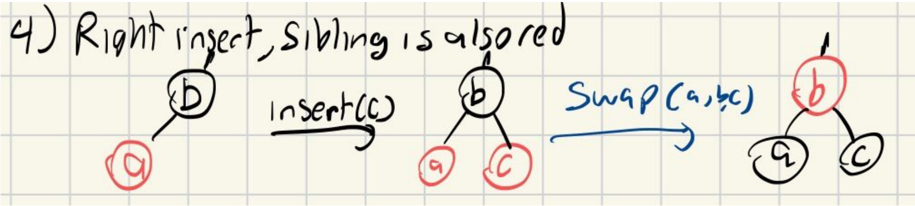
Insert: 15,21,7,24,0,<u>26</u>,3,28,29

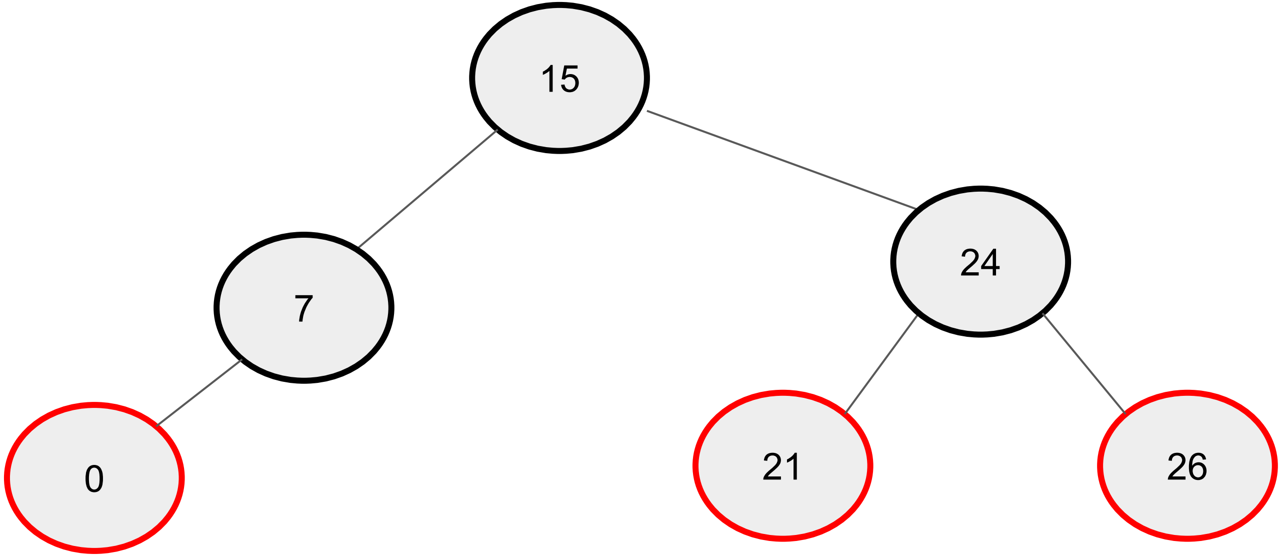
If root red, make it black



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

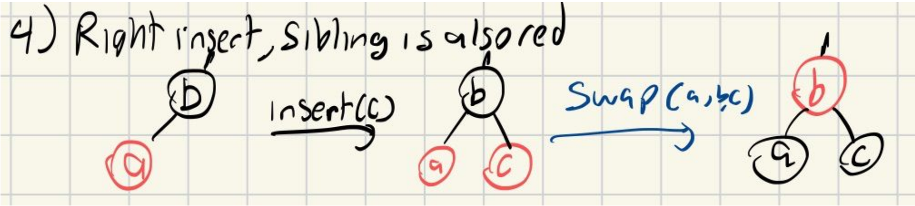
If root red, make it black

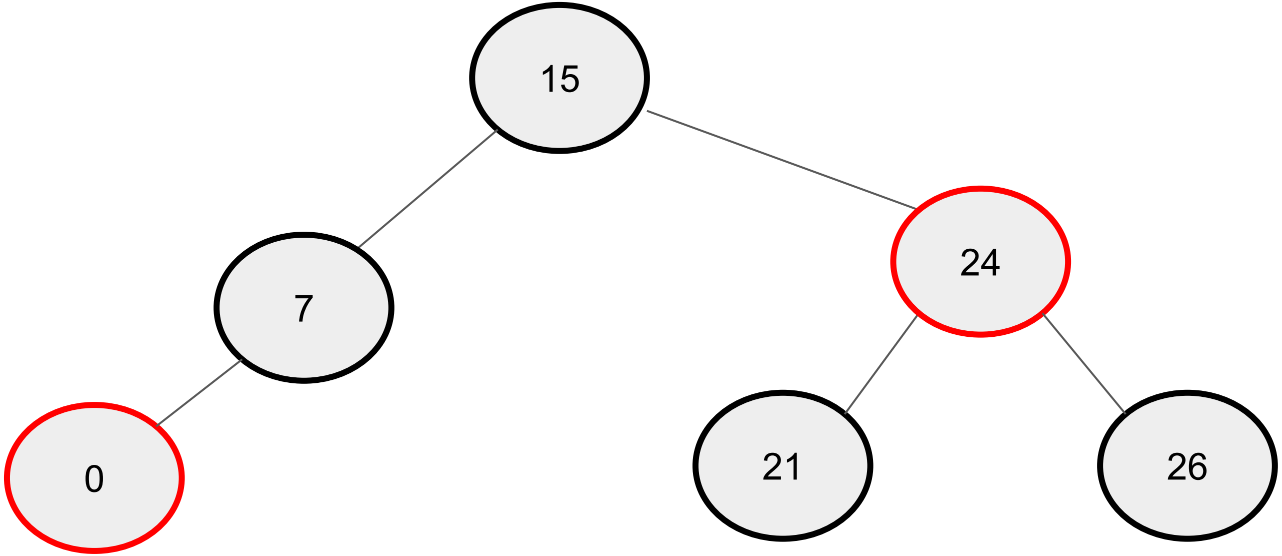




Insert: 15,21,7,24,0,26,3,28,29

If root red, make it black

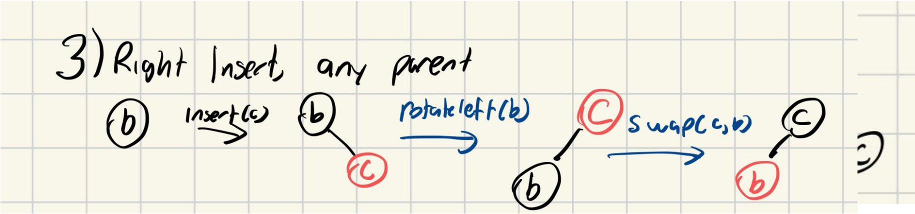


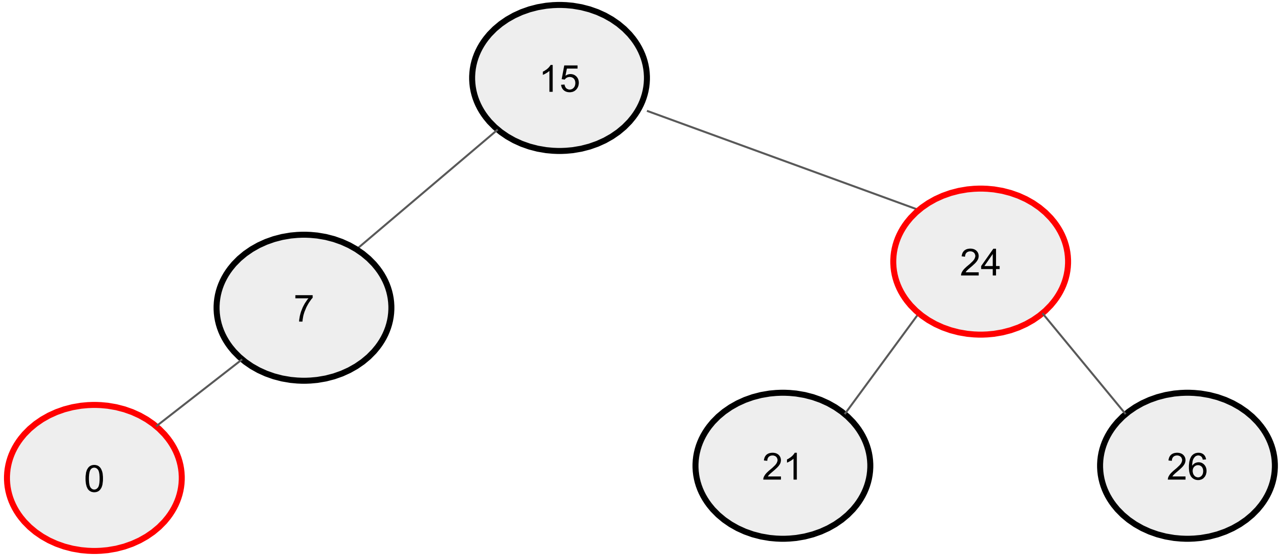


Oh no! Right red child, treat as red right insert

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

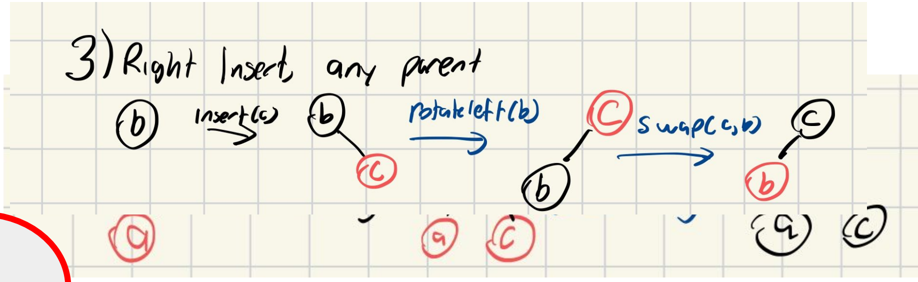
If root red, make it black



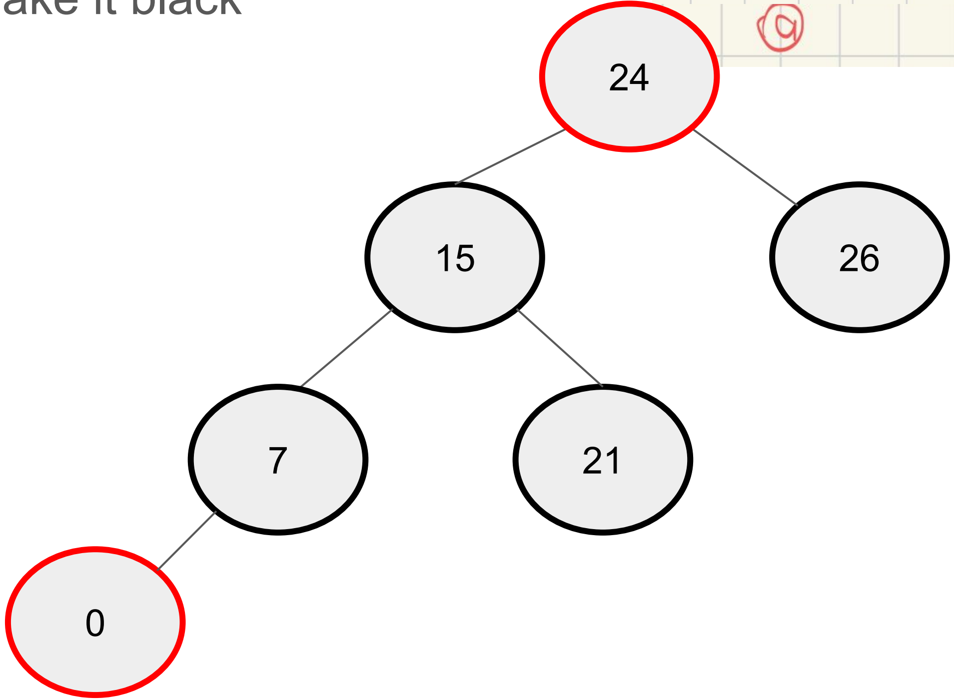


Oh no! Right red child, treat as red right insert

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

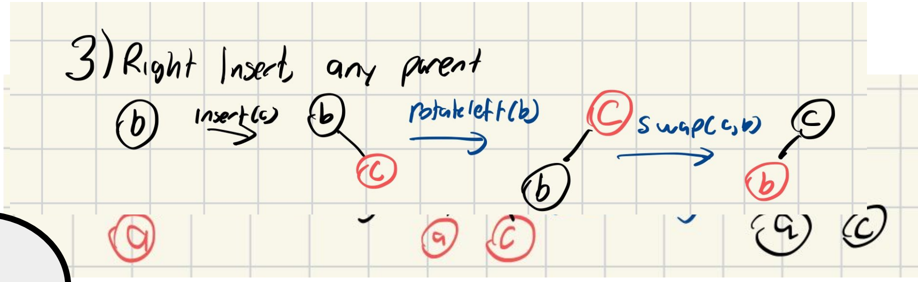


If root red, make it black

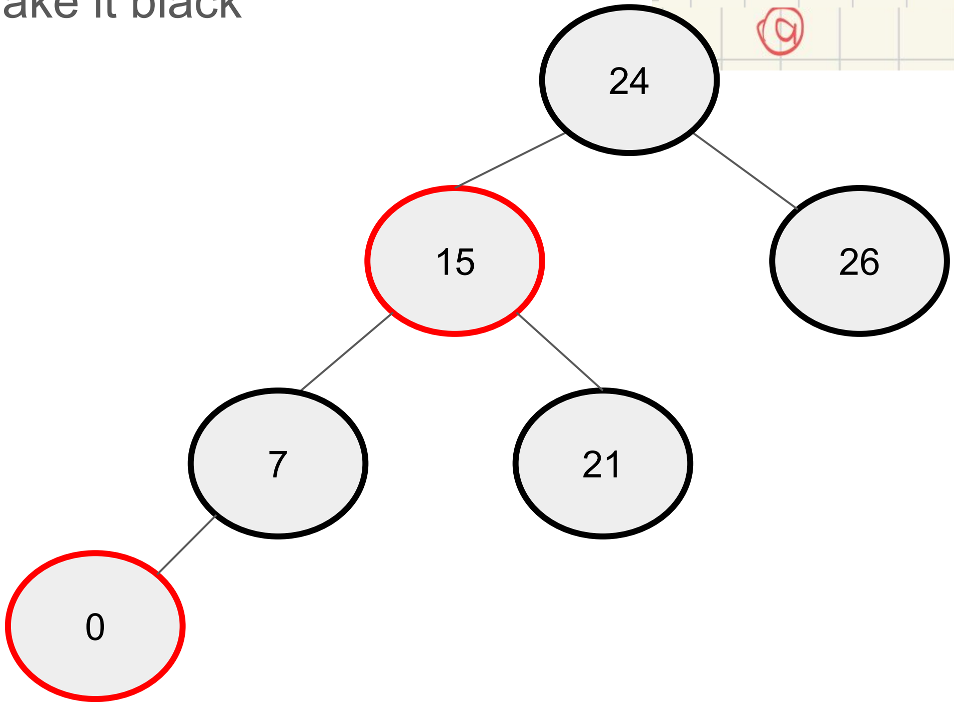


Oh no! Right red child, treat as red right insert

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

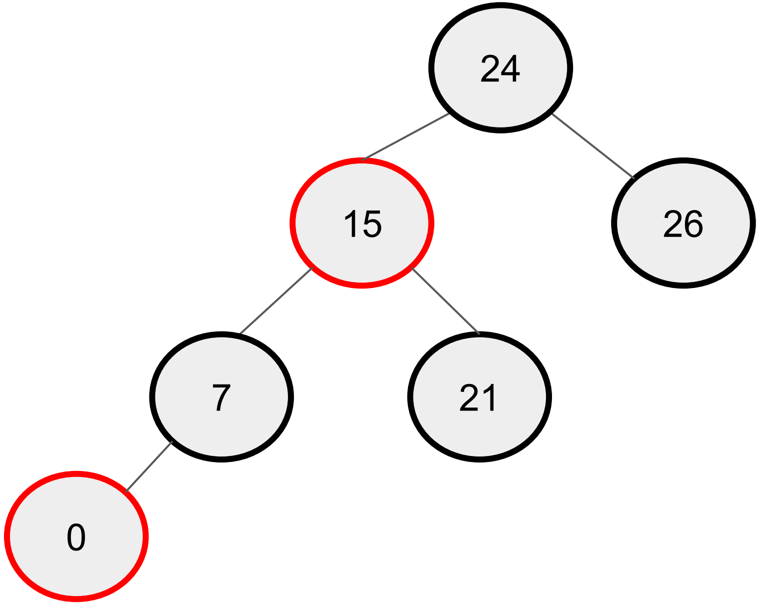


If root red, make it black

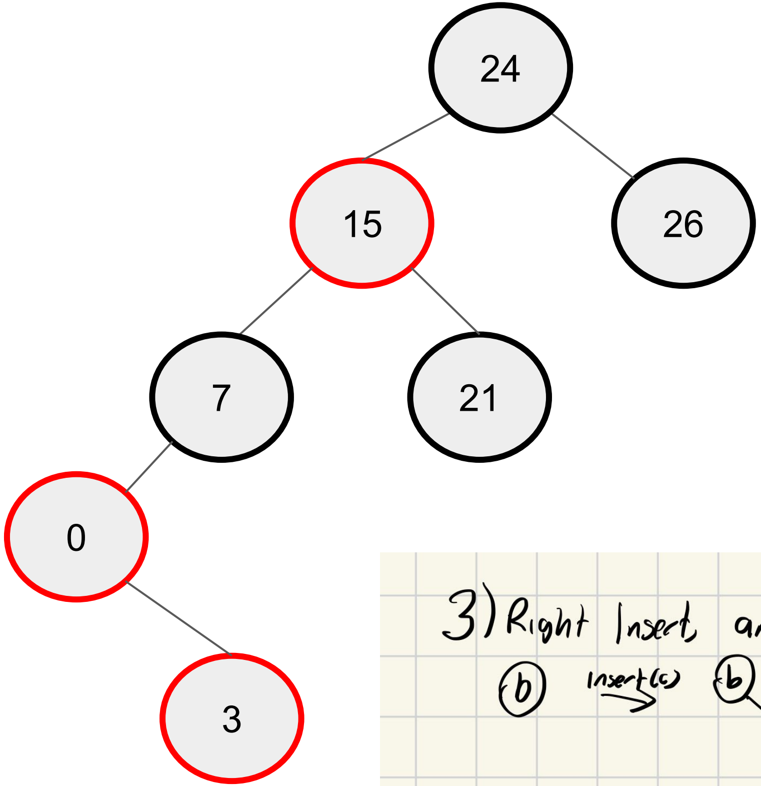


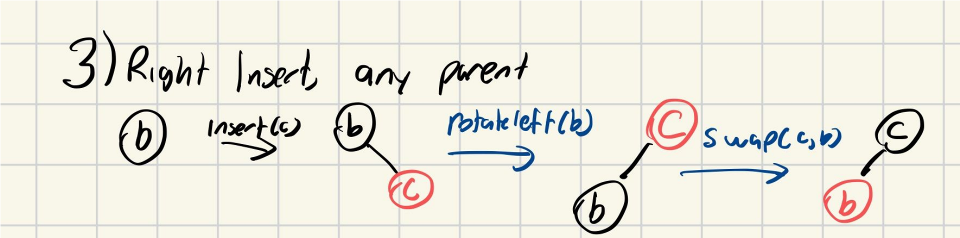
Oh no! Right red child, treat as red right insert

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



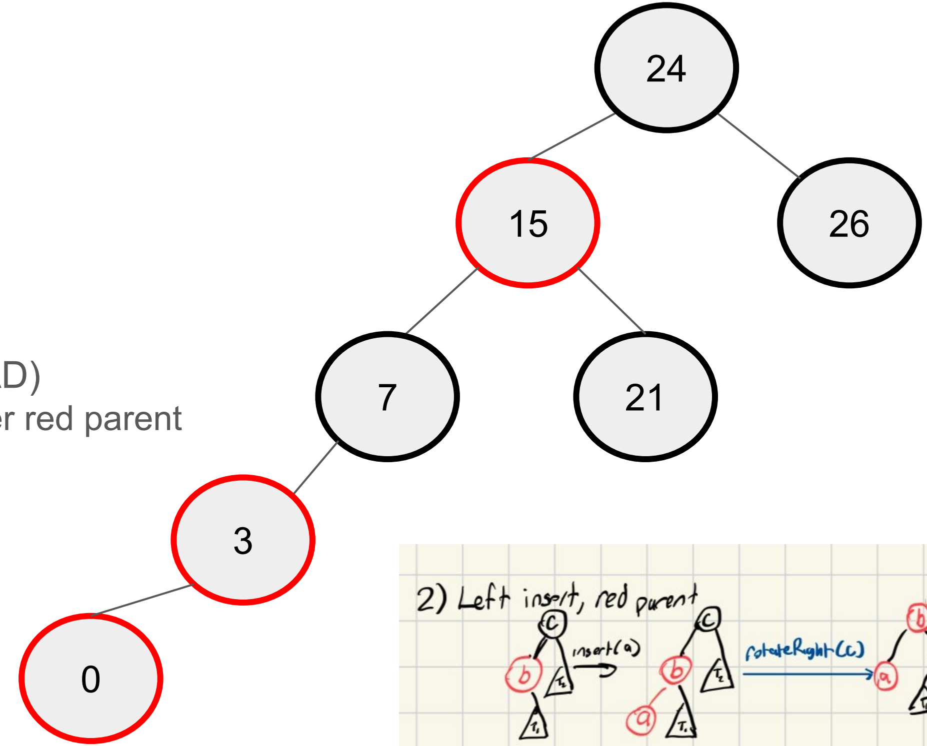
Insert: 15,21,7,24,0,26,3,28,29

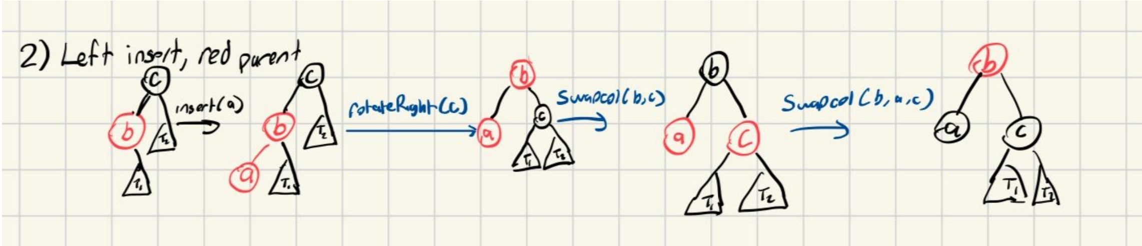




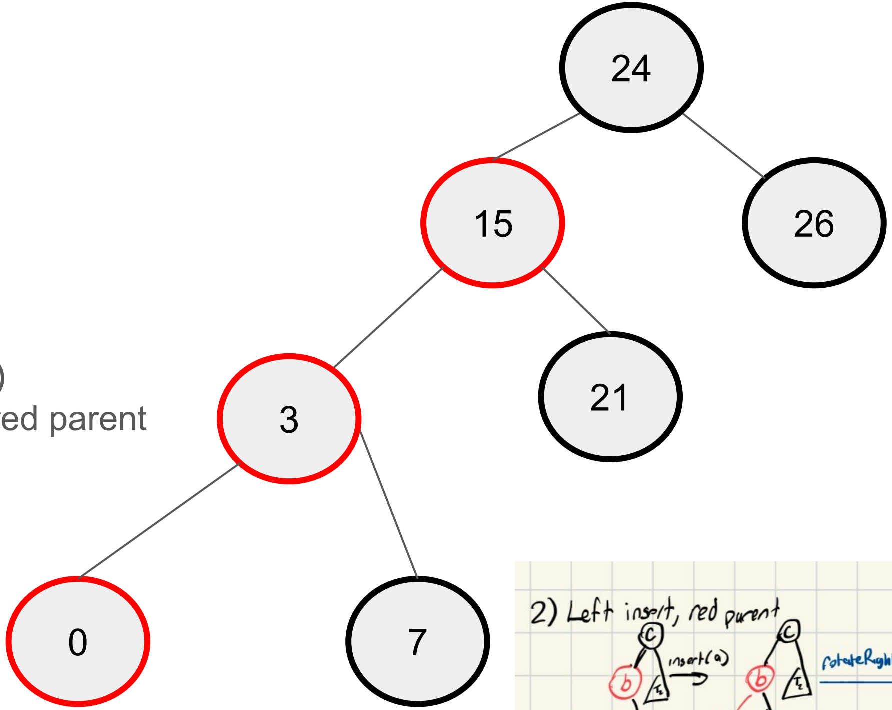
Insert: 15,21,7,24,0,26,3,28,29

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

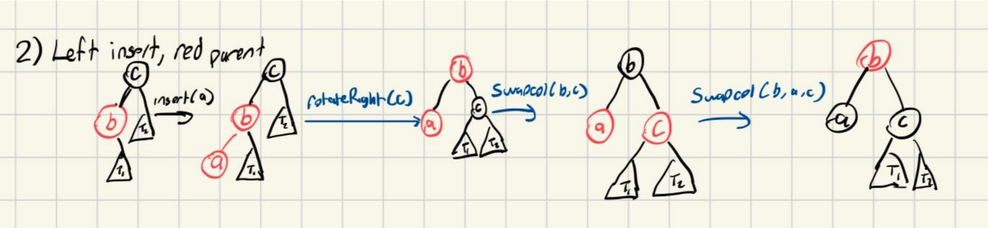




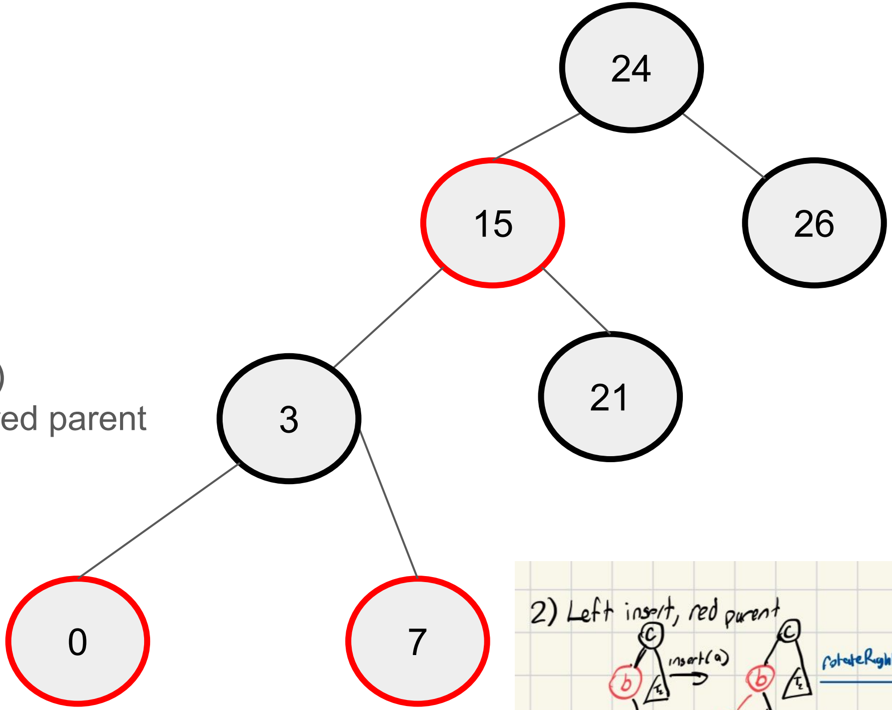
Insert: 15,21,7,24,0,26,3,28,29



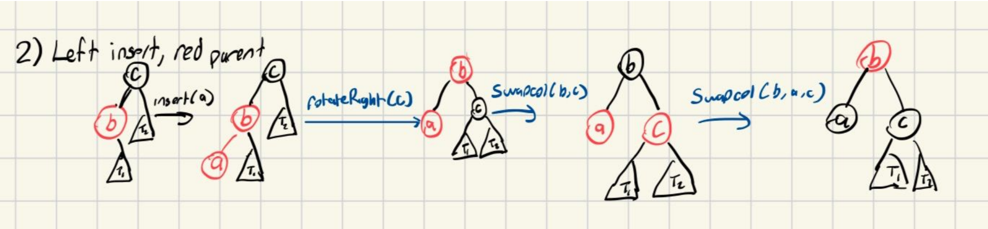
Two reds in a row (BAD)
Treat as red insert under red parent



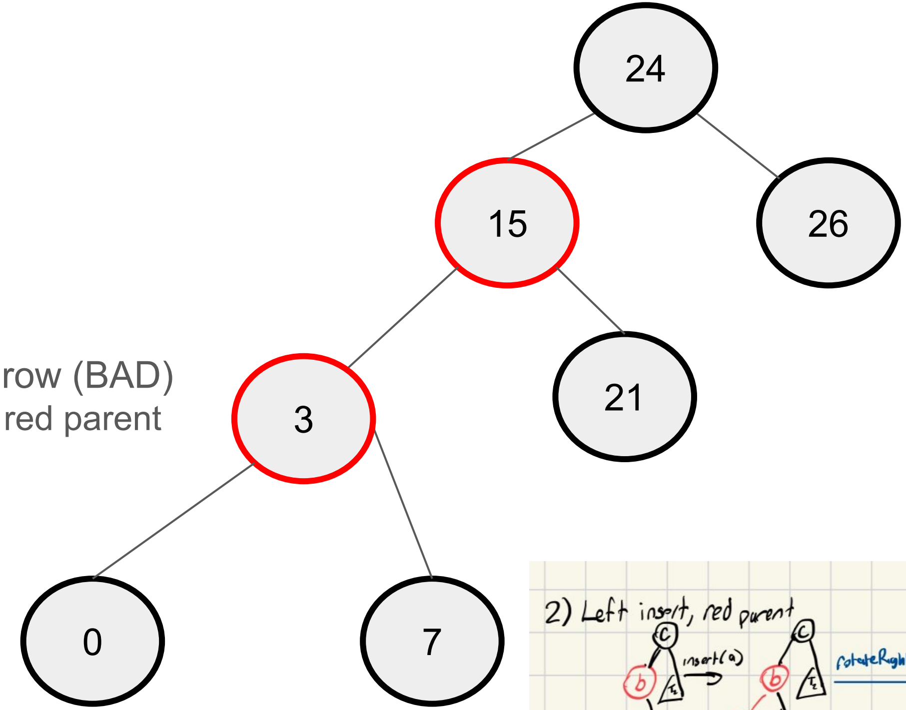
Insert: 15,21,7,24,0,26,<u>3</u>,28,29



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

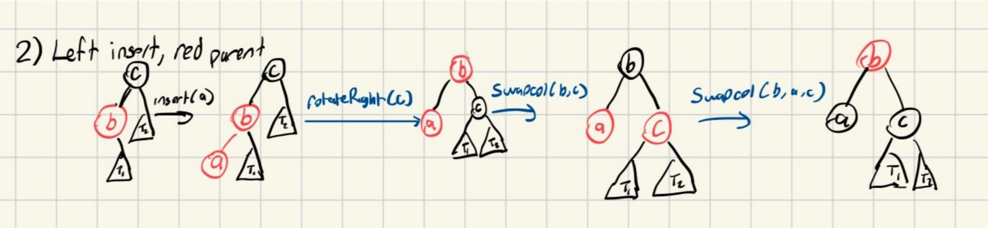


Insert: 15,21,7,24,0,26,<u>3</u>,28,29

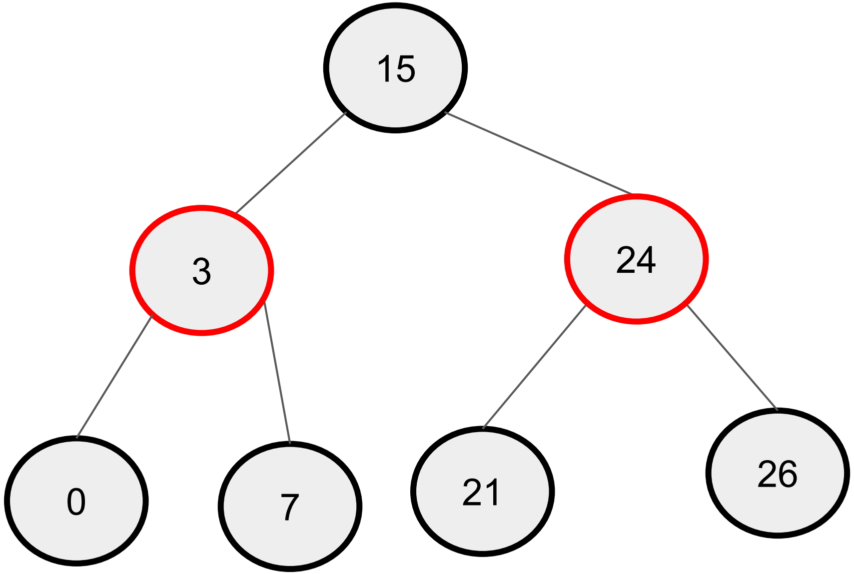


Rotate right at 24

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

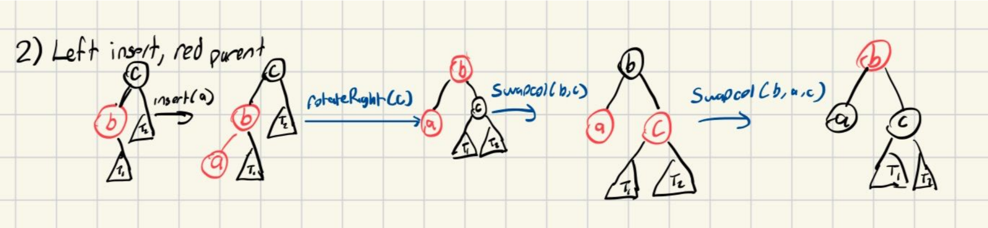


Insert: 15,21,7,24,0,26,3,28,29

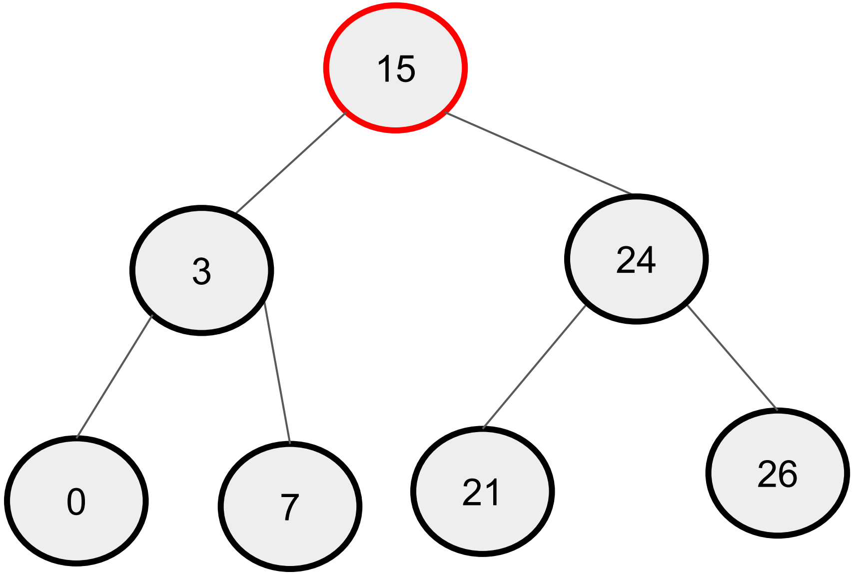


swap(15,24)

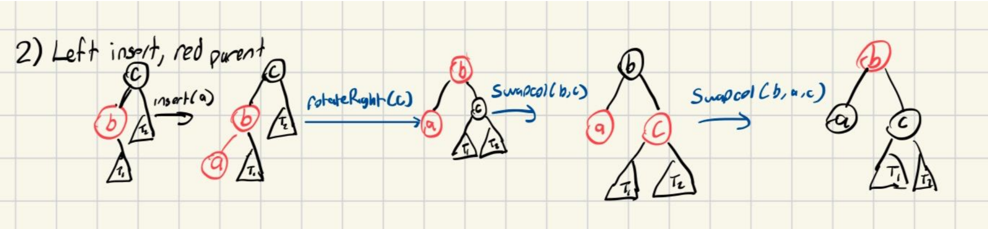
Another Two reds in a row (BAD)
Treat as red insert under red parent



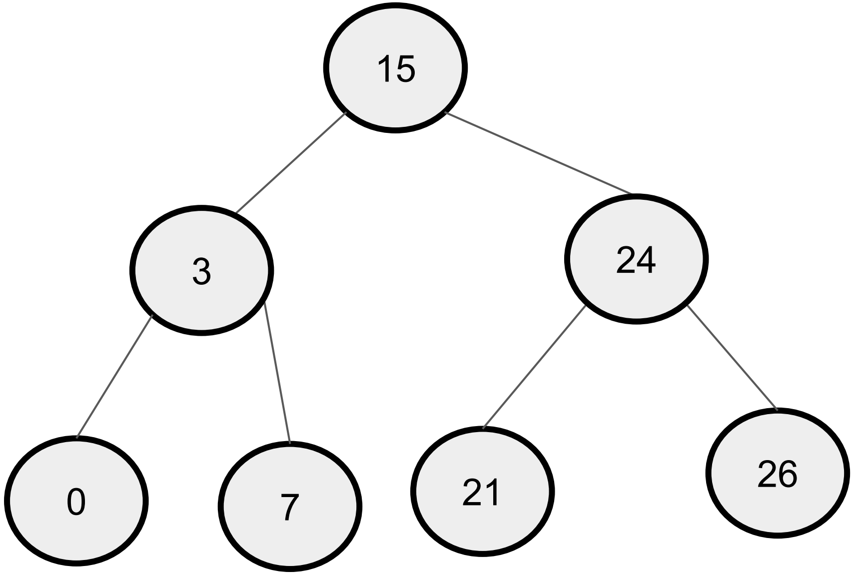
Insert: 15,21,7,24,0,26,3,28,29



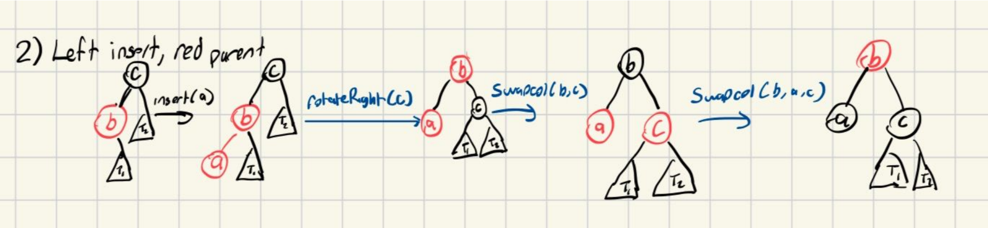
Another Two reds in a row (BAD)
Treat as red insert under red parent



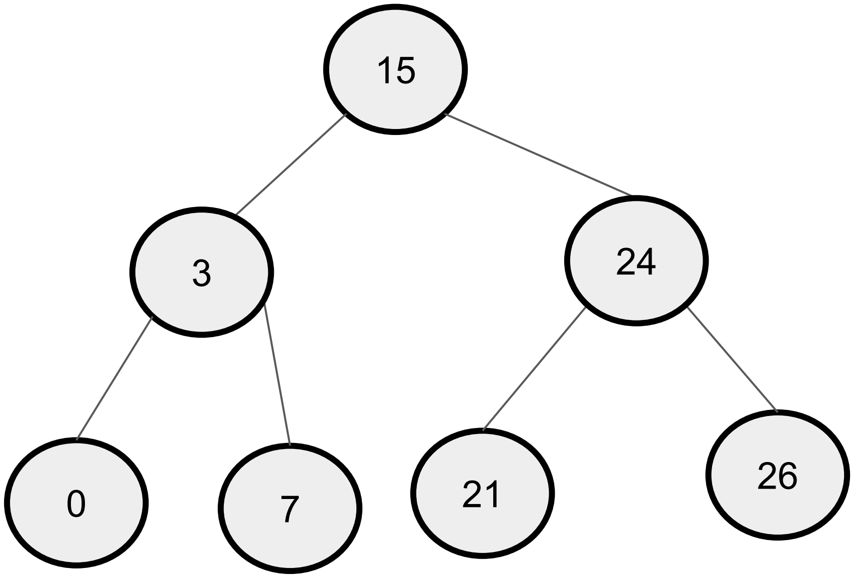
Insert: 15,21,7,24,0,26,3,28,29



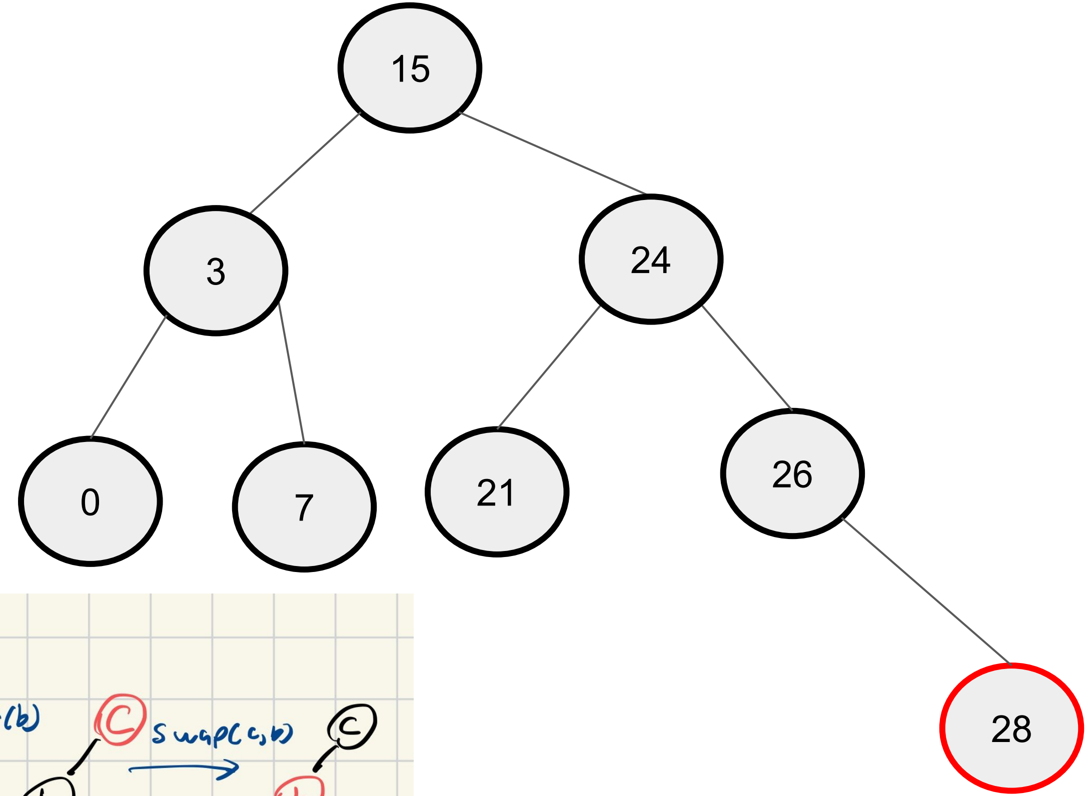
Another Two reds in a row (BAD)
Treat as red insert under red parent

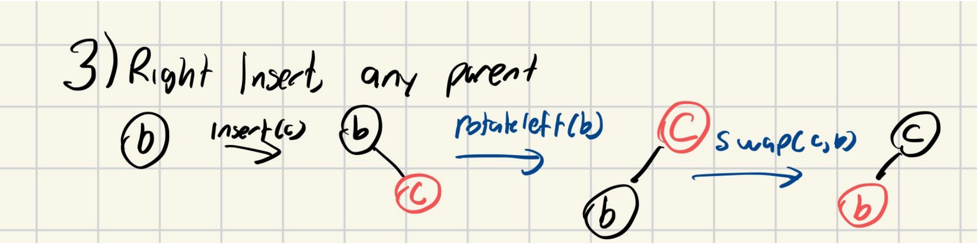


Insert: 15,21,7,24,0,26,3,<u>28</u>,29

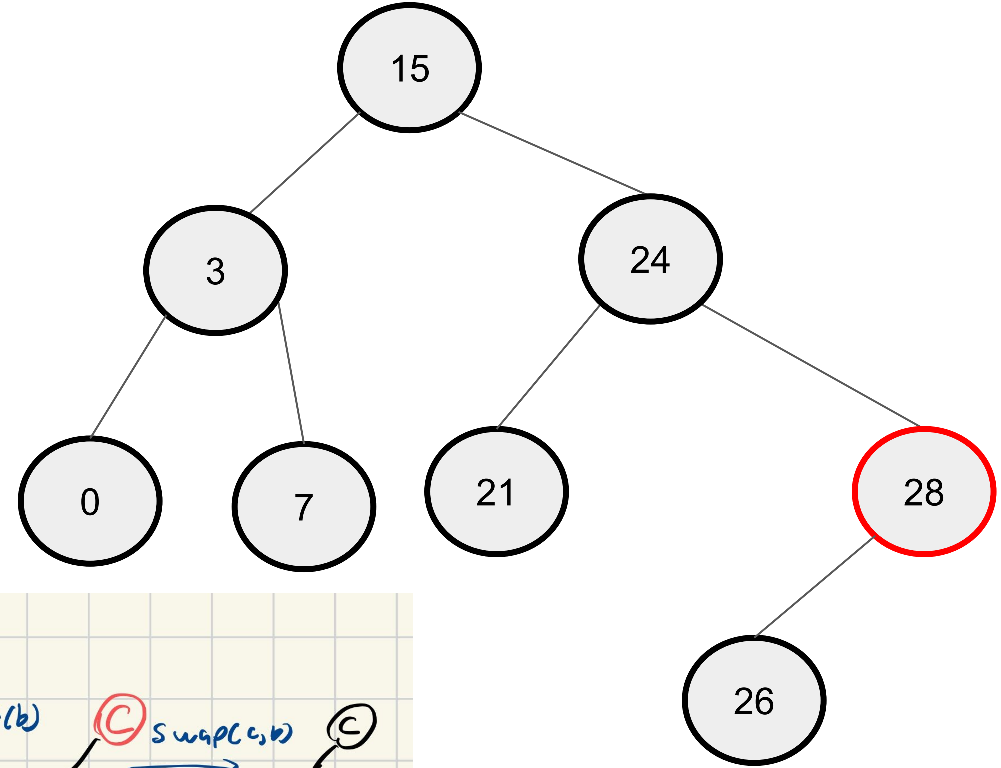


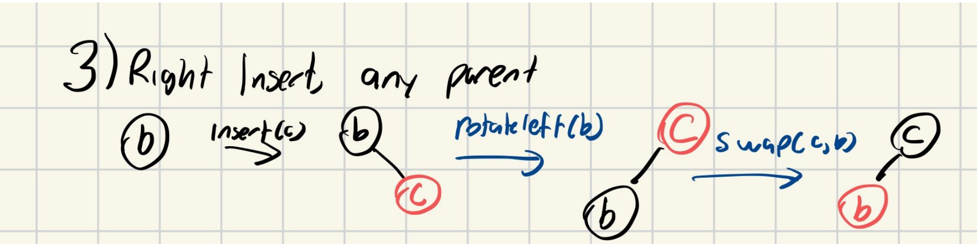
Insert: 15,21,7,24,0,26,3,<u>28</u>,29



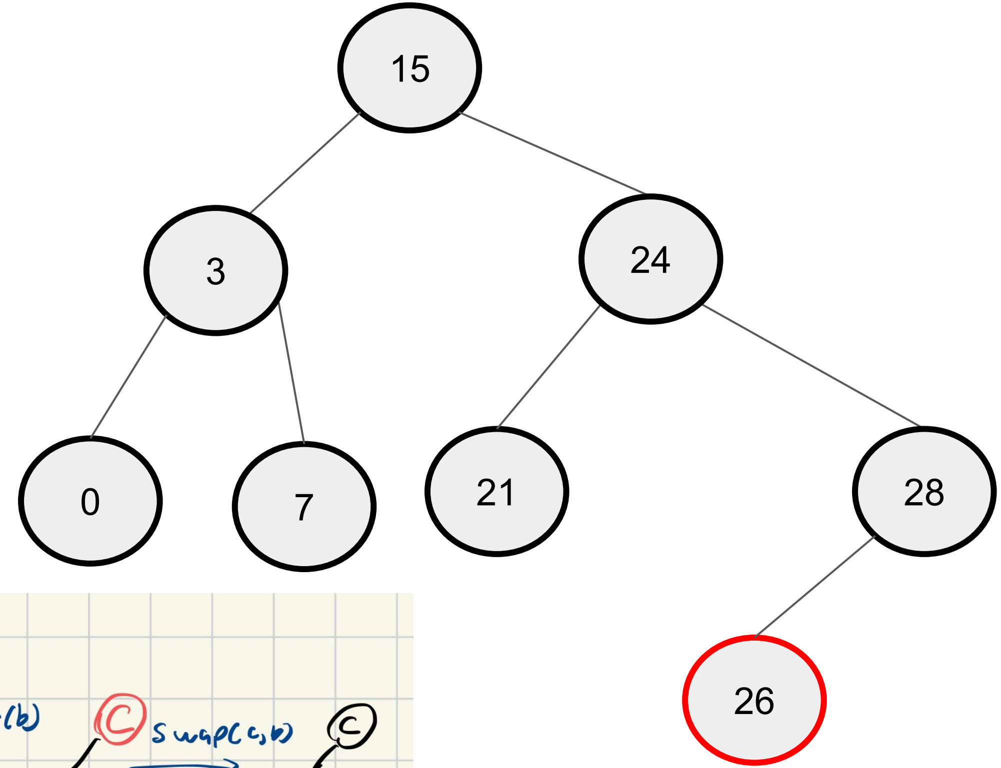


Insert: 15,21,7,24,0,26,3,28,29

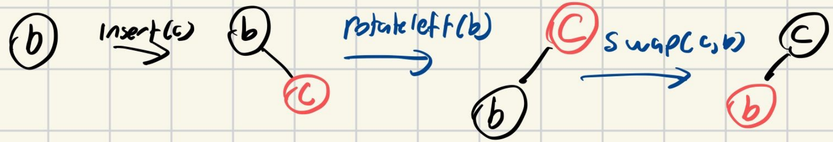




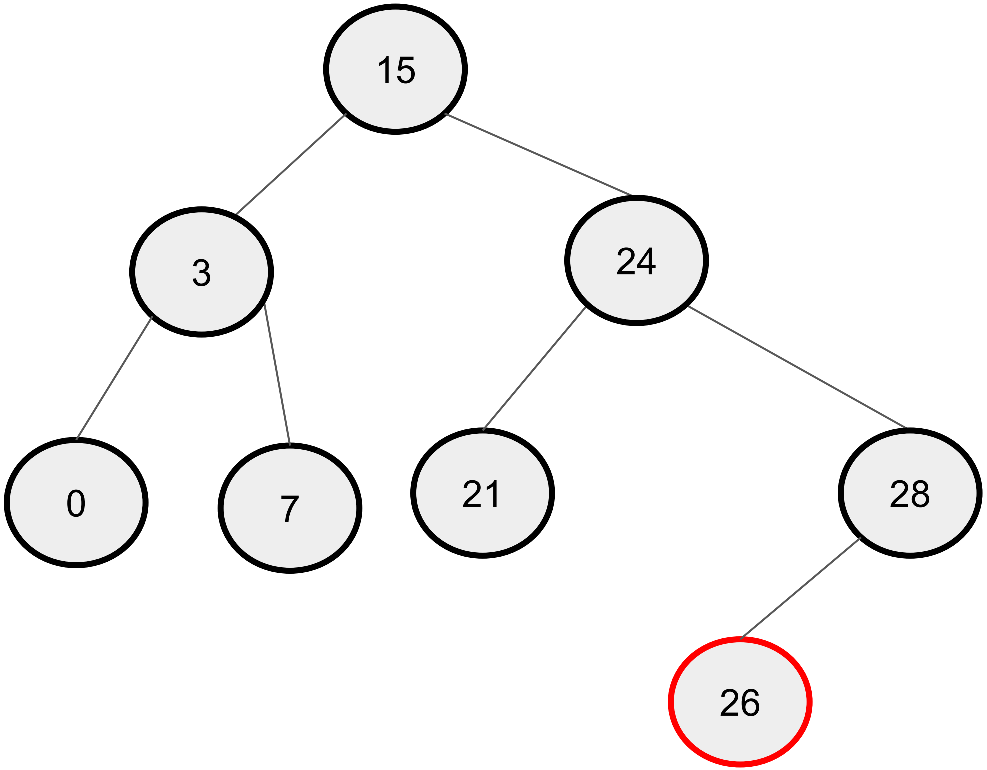
Insert: 15,21,7,24,0,26,3,<u>28</u>,29



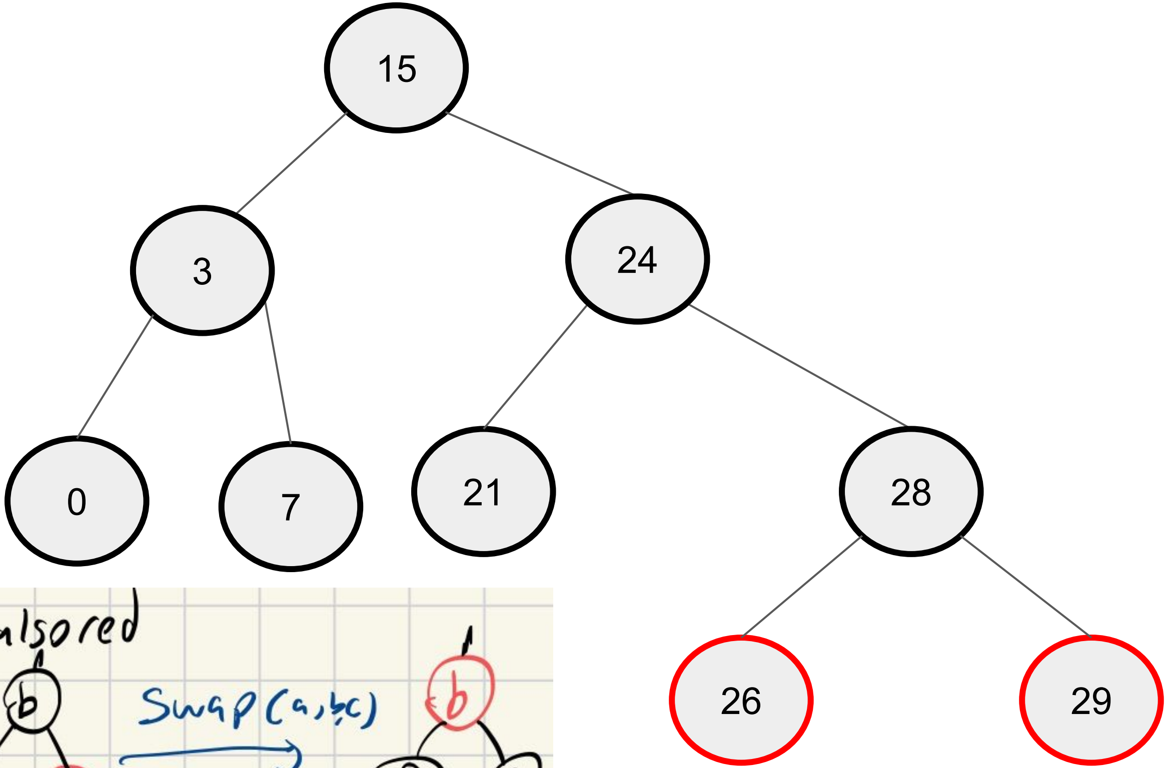
3) Right Insert, any parent

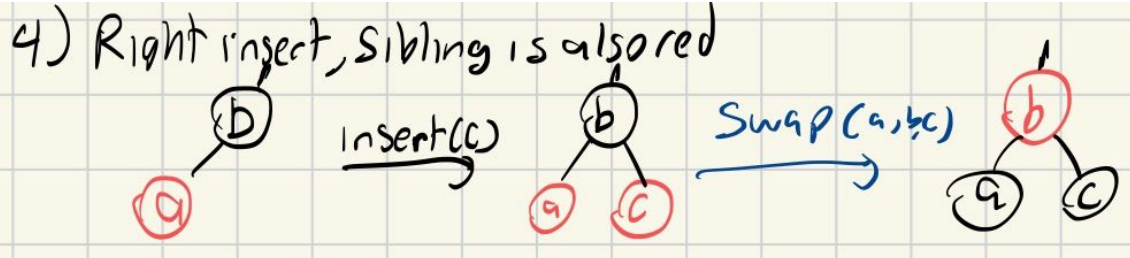


Insert: 15,21,7,24,0,26,3,28,29

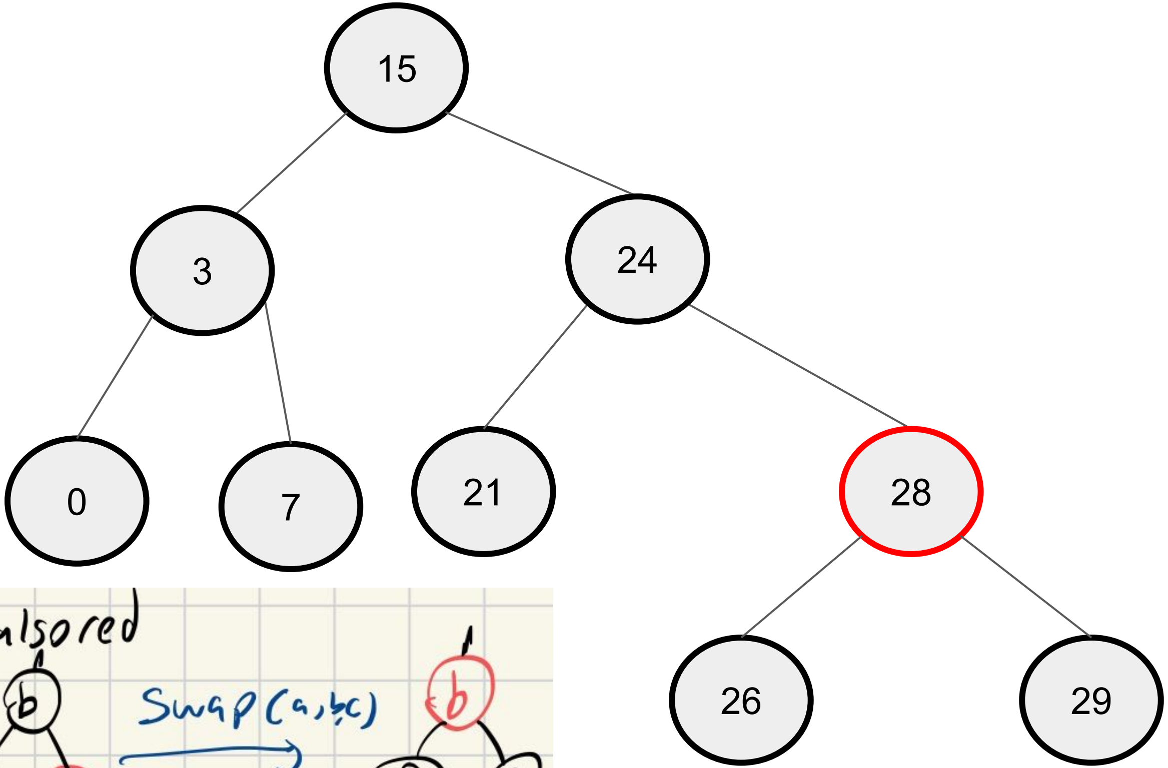


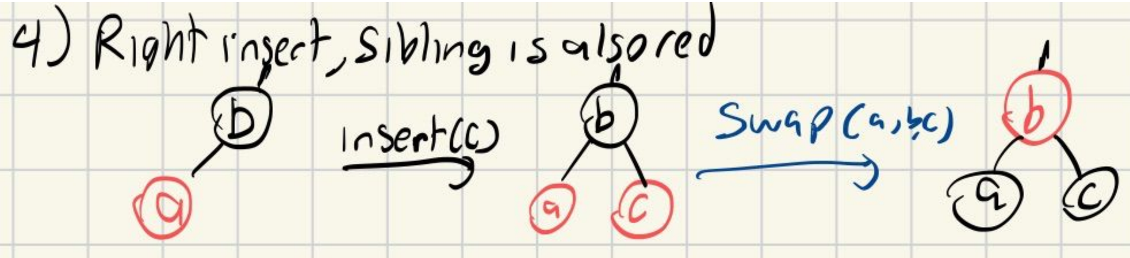
Insert: 15,21,7,24,0,26,3,28,29



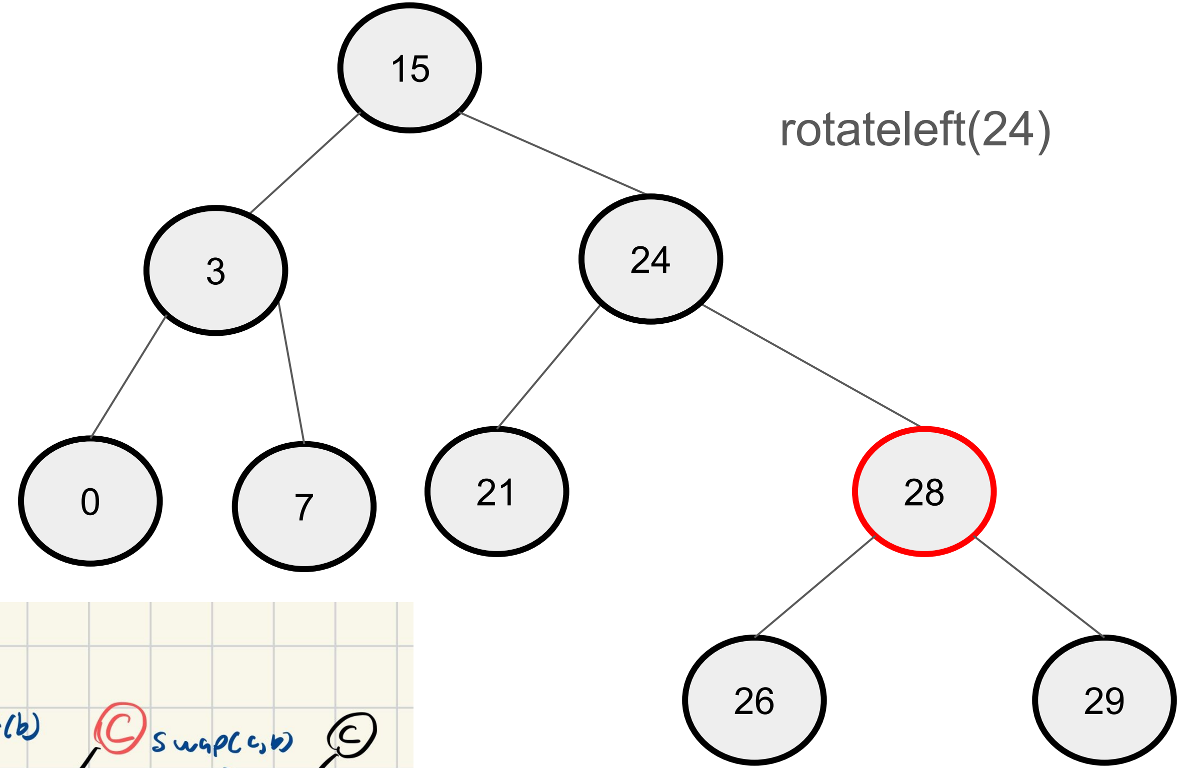


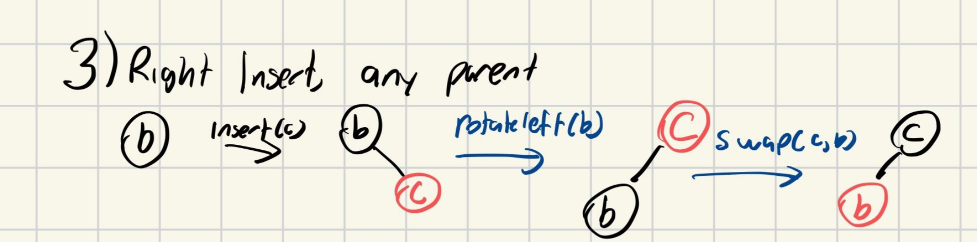
Insert: 15,21,7,24,0,26,3,28,29



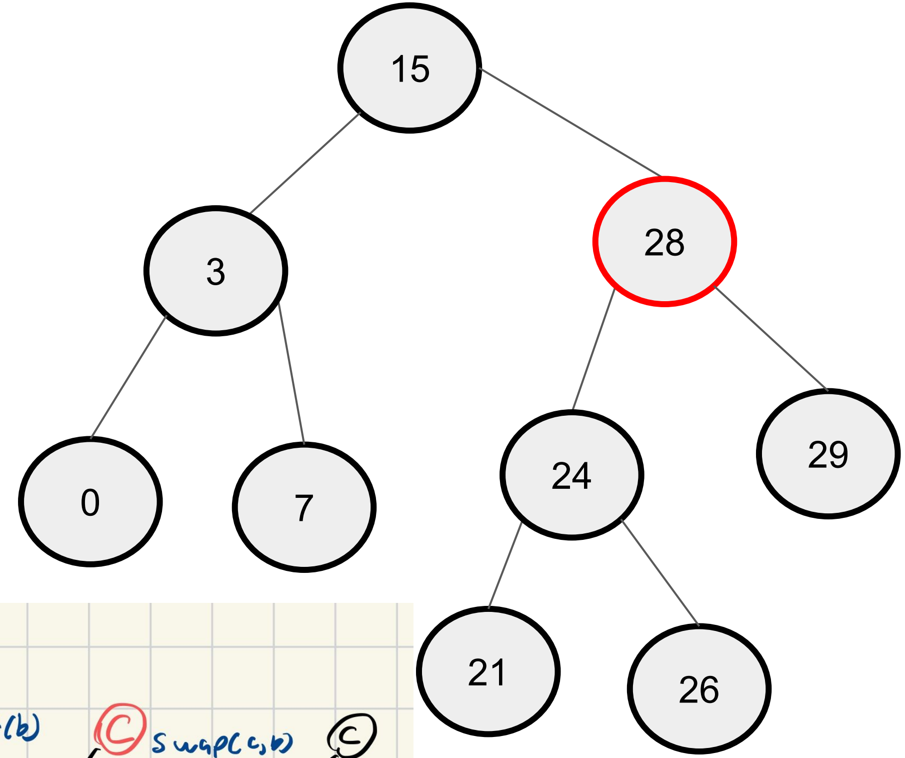


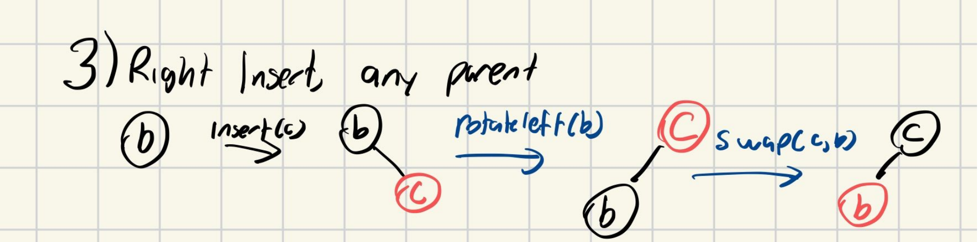
Insert: 15,21,7,24,0,26,3,28,<u>29</u>



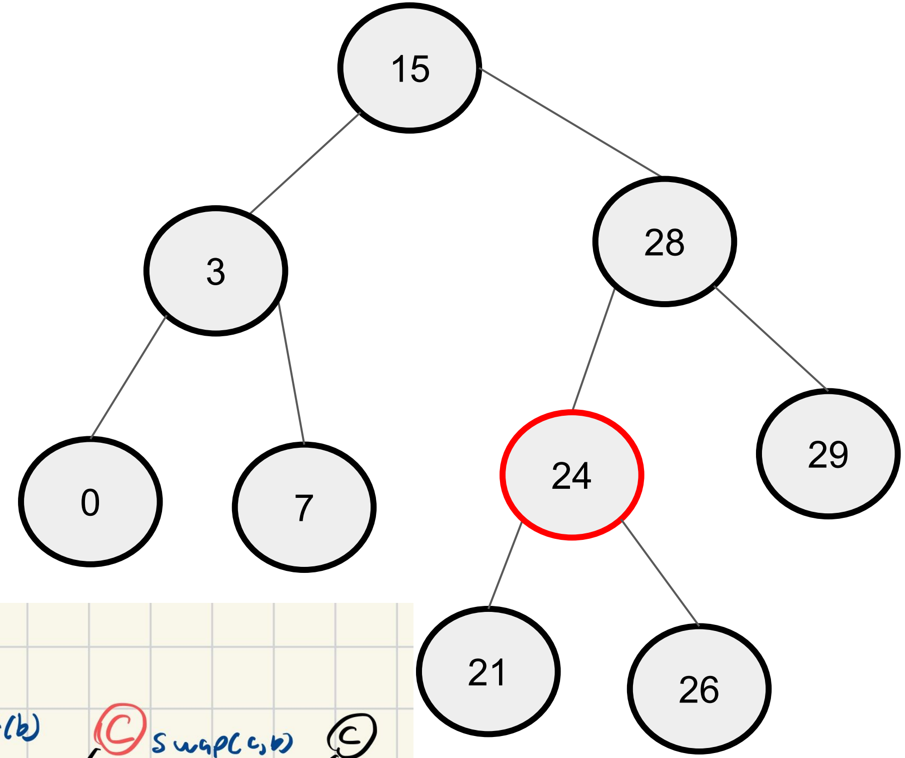


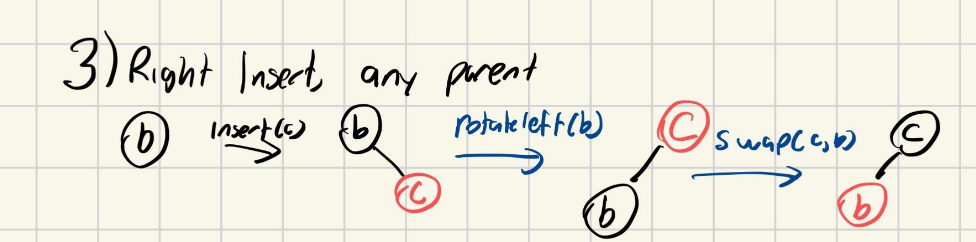
Insert: 15,21,7,24,0,26,3,28,<u>29</u>



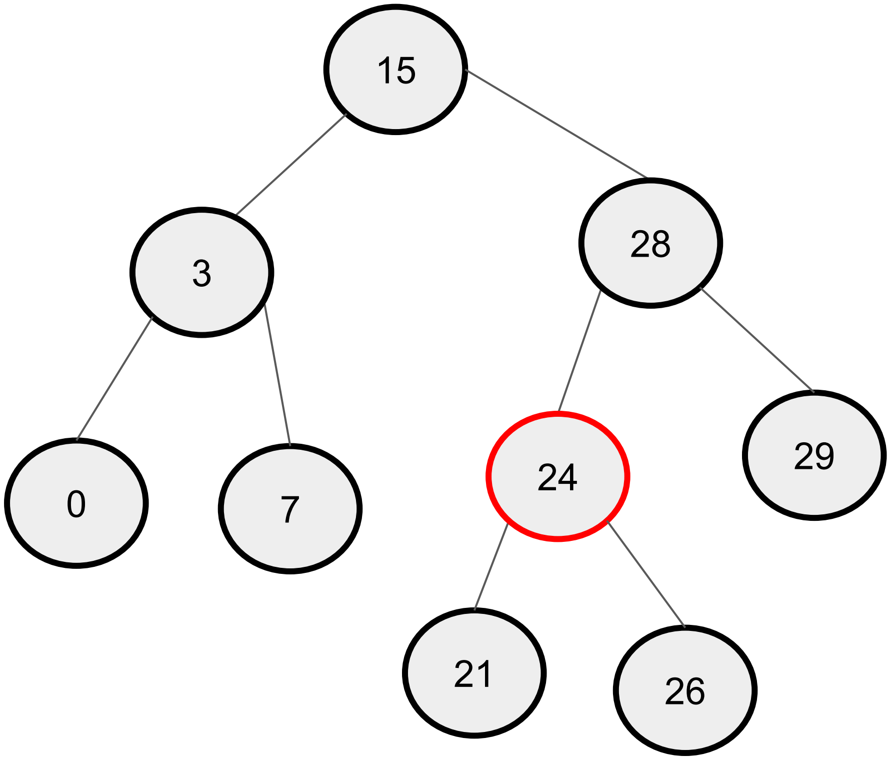


Insert: 15,21,7,24,0,26,3,28,<u>29</u>





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



Bonus: LLRB Hard to Understand

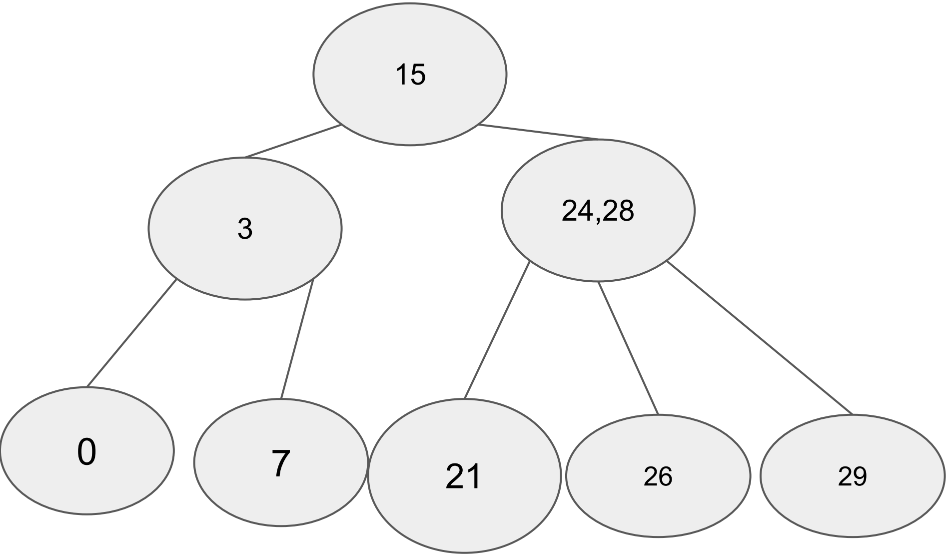
LLRB trees are “the same as” to 2-3 trees

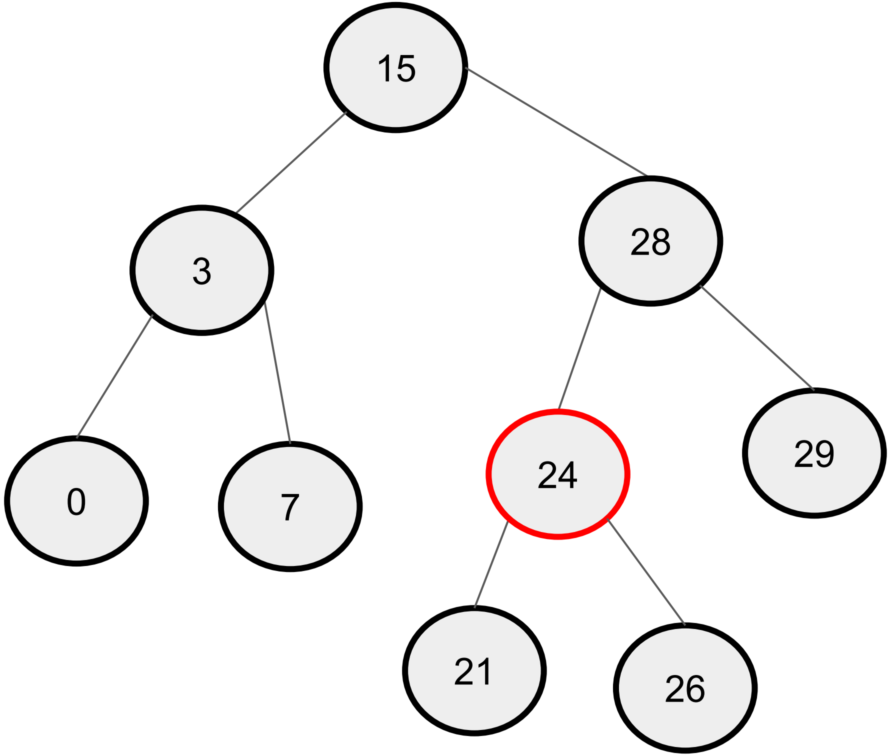
• binary tree equivalent of 2-3 trees

2-3 Tree	↔ 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!



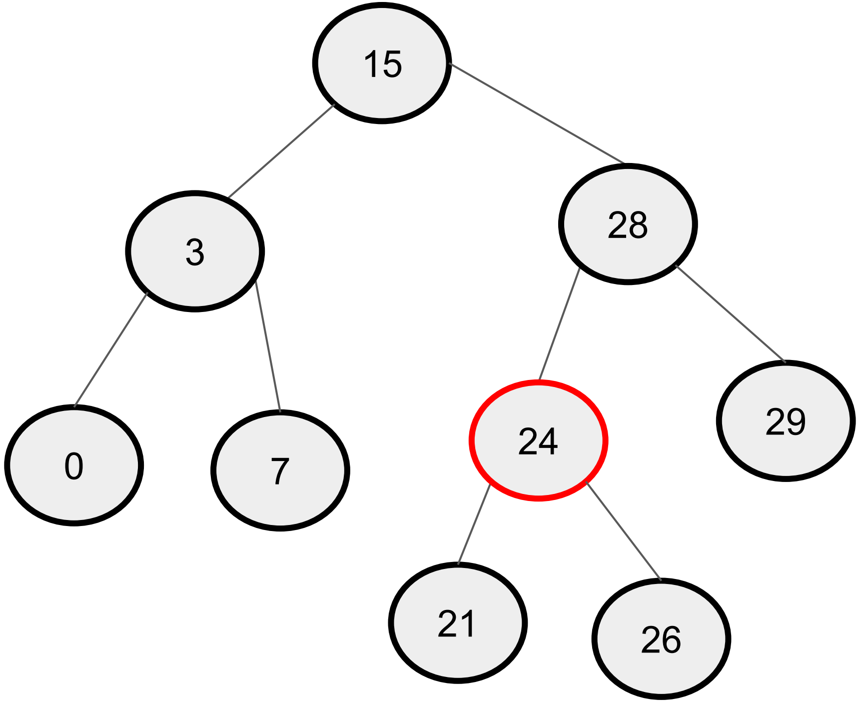


Question 3

(Deletion) Show intermediate steps of the following questions:

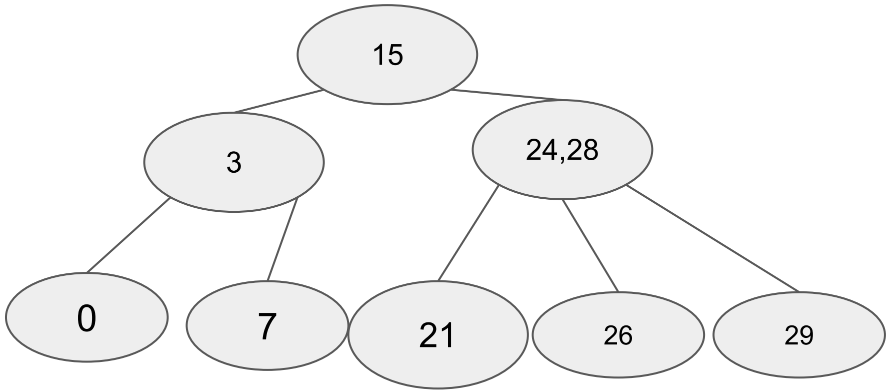
(1) How to delete 7 in the final 2-3 tree of Q1?

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

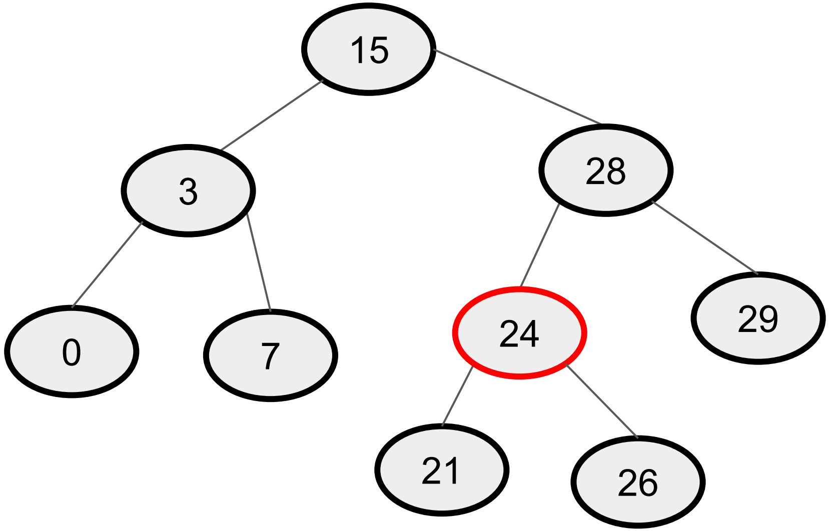


Deletion in LLRB is quite hard..

Idea: Use equivalence between LLRB and 2-3 trees



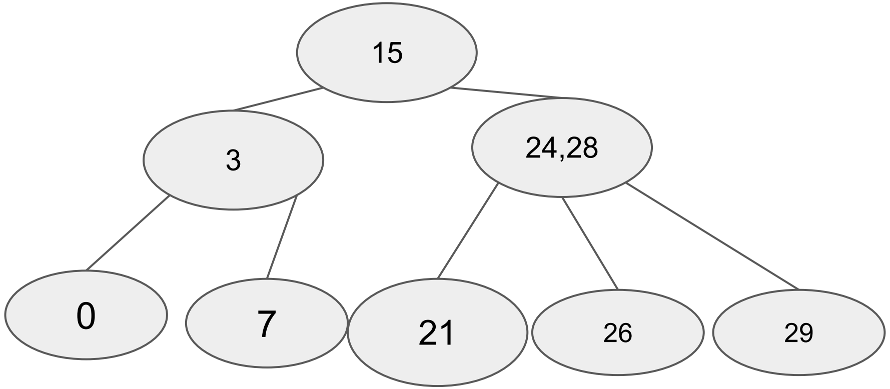
Idea: Use equivalence between LLRB and 2-3 trees



Take the LLRB

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

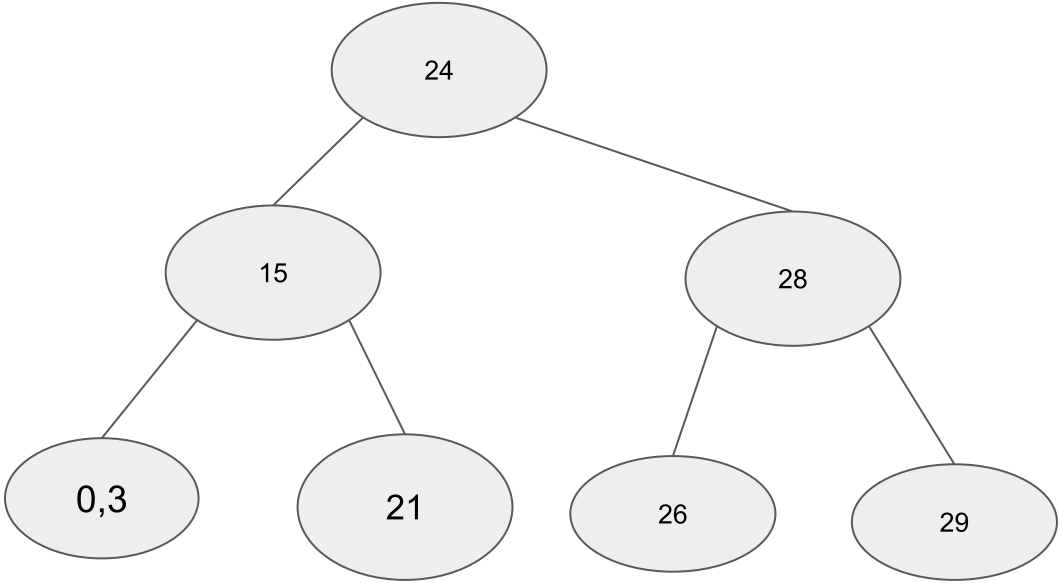
Idea: Use equivalence between LLRB and 2-3 trees



Take the LLRB

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

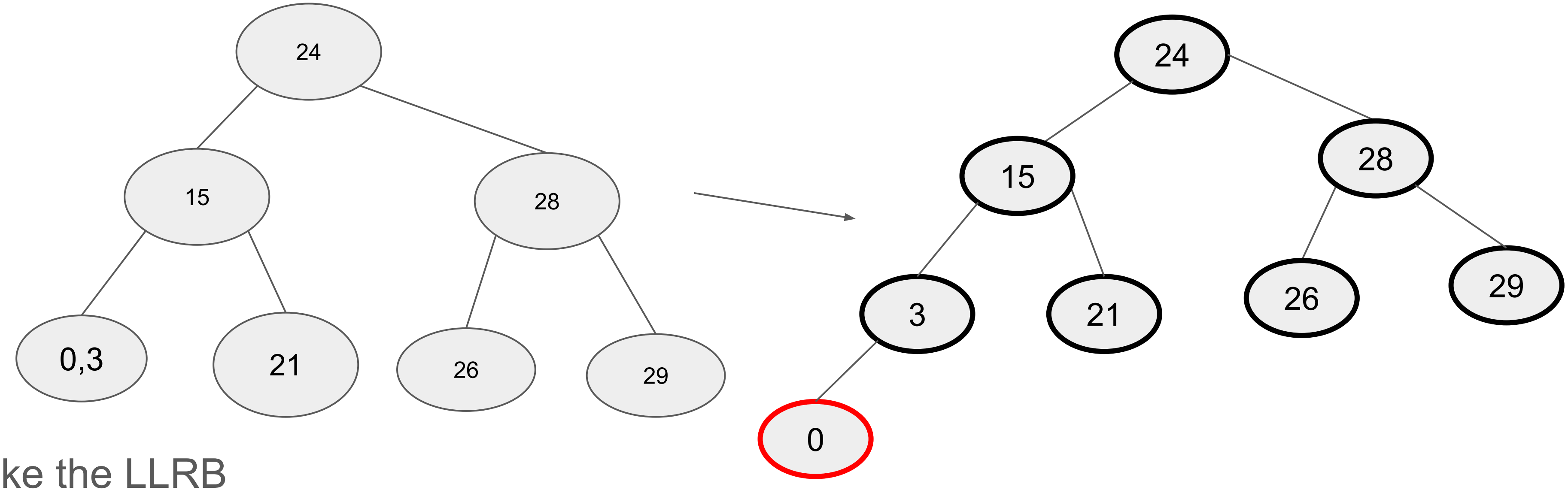
Idea: Use equivalence between LLRB and 2-3 trees



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 2

(Adjacency-matrix Representation)

1. Give an adjacency-matrix representation for a complete binary search tree on 7 vertices numbered from 1 to 7.

What in the world in an adjacency matrix?

Question 2

(Adjacency-matrix Representation)

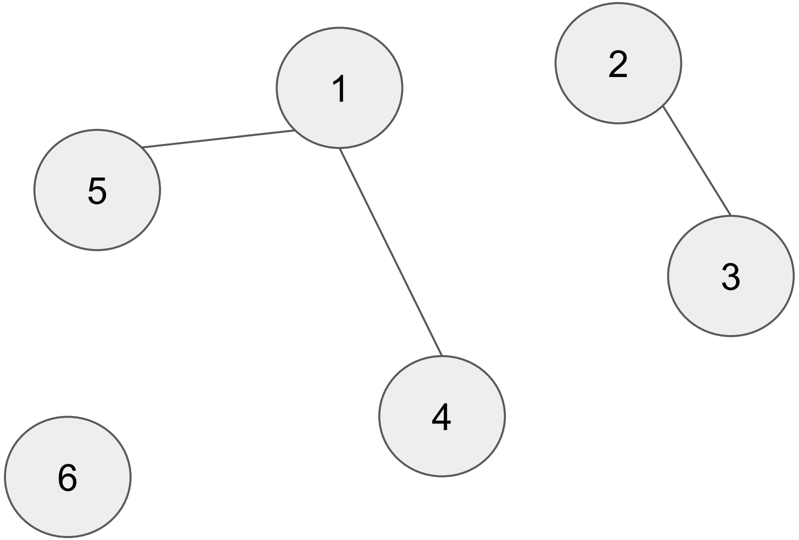
1. Give an adjacency-matrix representation for a complete binary search tree on 7 vertices numbered from 1 to 7.

What in the world in an adjacency matrix?

Adjacency Matrix

Edges represented in a |V| x |V| matrix

E.g. if undirected..



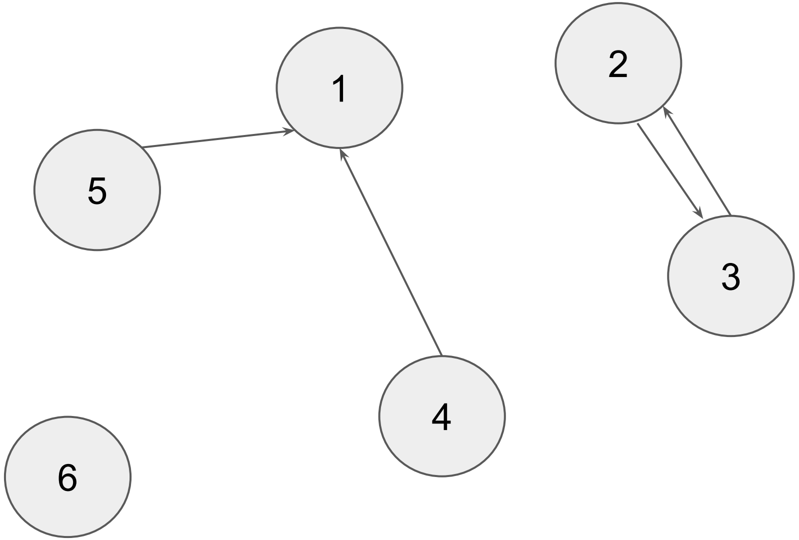
1 2 3 4 5 6

1
2
3
4
5
6

Adjacency Matrix

Edges represented in a |V| x |V| matrix

E.g. if **directed**..



"Row goes to column"

$$
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & & & & & & \\
2 & & & & & & \\
3 & & & & & & \\
4 & & & & & & \\
5 & & & & & & \\
6 & & & & & &
\end{array}
$$

Question 2

(Adjacency-matrix Representation)

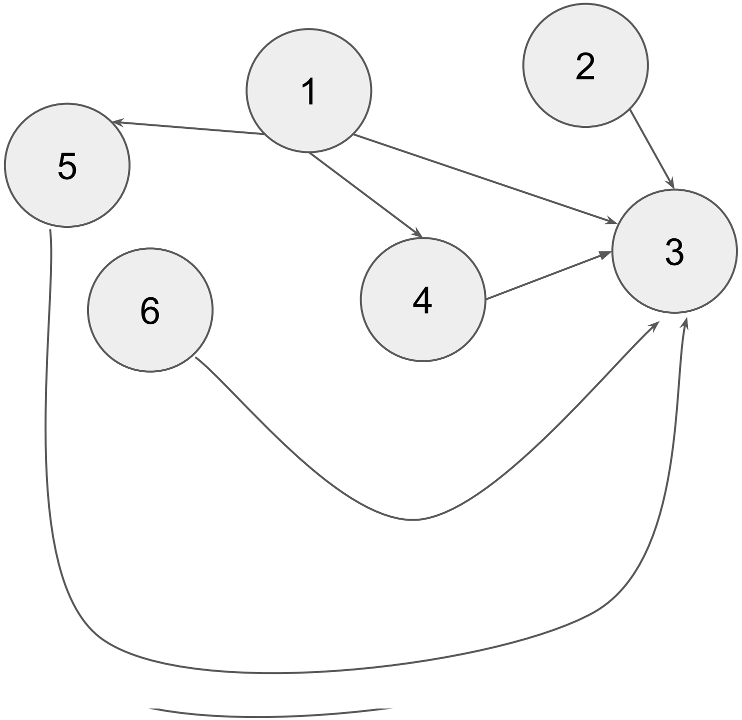
1. Give an adjacency-matrix representation for a complete binary search tree on 7 vertices numbered from 1 to 7.

Someone give me a complete binary search tree

Question 2

(Adjacency-matrix Representation)

1. Give an adjacency-matrix representation for a complete binary search tree on 7 vertices numbered from 1 to 7.



	1	2	3	4	5	6
1	0	0	1	1	1	0
2	0	0	1	0	0	0
3	0	0	0	0	0	0
4	0	0	1	0	0	0
5	0	0	1	0	0	0
6	0	0	1	0	0	0

Question 1

(Articulation point)

We define an *articulation point* as a vertex that when removed causes a connected graph to become disconnected. For this problem, we will try to find the articulation points in an undirected graph G.

(1) How can we efficiently check whether or not a graph is disconnected?

(2) How to determine if a node u is an articulation point or not?

(undirected)

(1) How can we efficiently check whether or not a graph is disconnected?

Two vertices u,v are connected if there is some way to get from u → v.

Graph is connected if for *all* u,v vertices, u and v are connected

Are there any algorithms that can help us here?

(undirected)
(1) How can we efficiently check whether or not a graph is disconnected?

Two vertices u,v are connected if there is some way to get from u → v.

Graph is connected if for *all* u,v vertices, u and v are connected

Use BFS/DFS, count the number of vertices visited

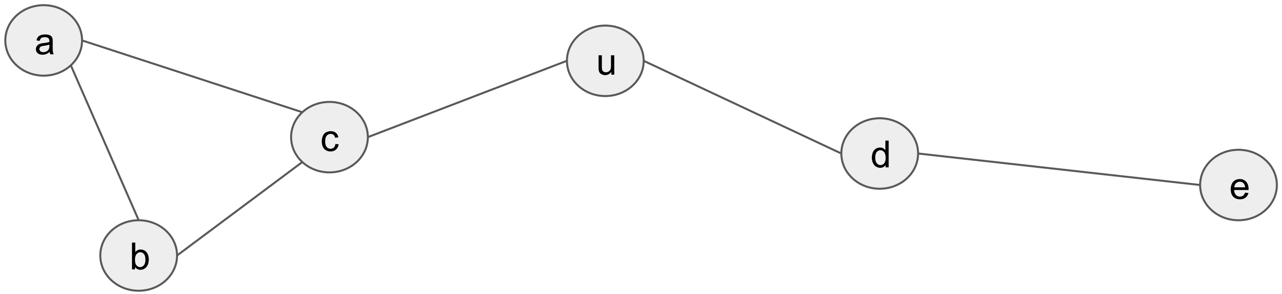
Question 1

(Articulation point)

We define an *articulation point* as a vertex that when removed causes a connected graph to become disconnected. For this problem, we will try to find the articulation points in an undirected graph G.

(2) How to determine if a node u is an articulation point or not?

Any idea from pt 1?



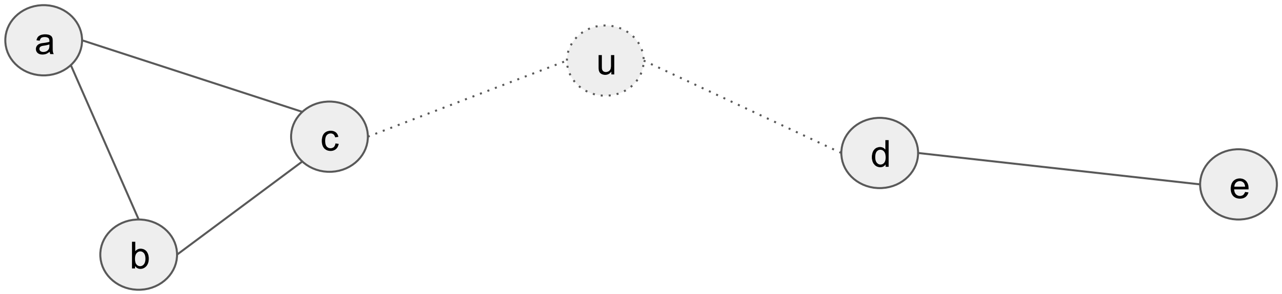
Question 1

(Articulation point)

We define an *articulation point* as a vertex that when removed causes a connected graph to become disconnected. For this problem, we will try to find the articulation points in an undirected graph G.

(2) How to determine if a node u is an articulation point or not?

Any idea from pt 1? Check connectivity when u is deleted



Exercise: Suppose you wanted to find all articulation points. Can you do so in O(|V| + |E|) time?
Hint: a point is an articulation point iff it is not in a cycle.