Red-Black Tree
2-3 Tree

AVL Tree Review

- Balance factors are always maintained
  - by rotation, rotation and more rotation
- When insertion/deletion is frequent, the overhead increases
- Implementation is difficult
- Do we really need that strict balance factor?
  - maybe, maybe not

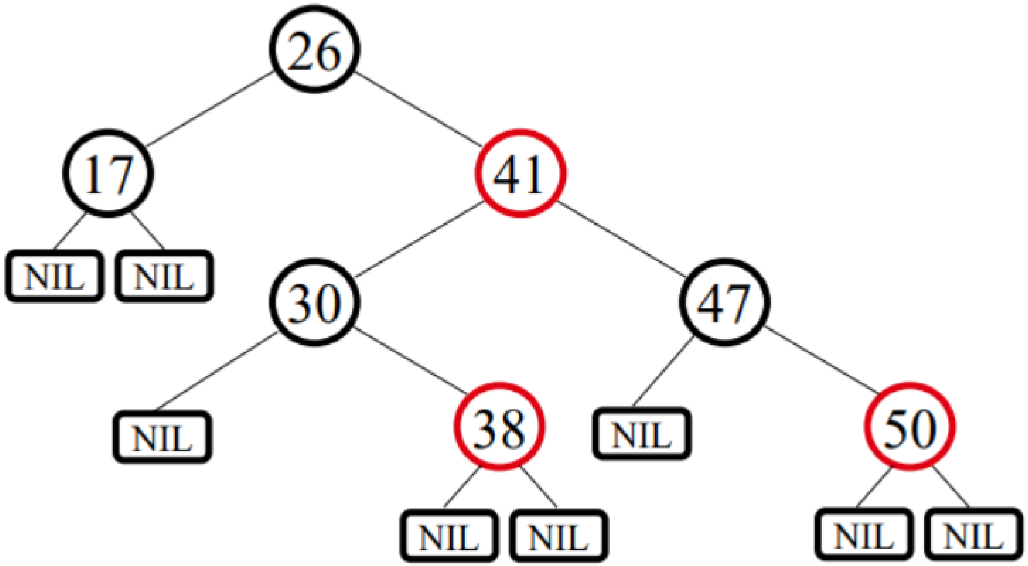
Red-Black

- Definition
  - **P1.** Every node is either **red** or **black**.
  - **P2.** Each **NULL pointer** is considered to point a black node called NIL
  - **P3.** The **root** is **black**.
  - **P4.** If a node is **red**, then both of its children are **black**.
  - P5. Every path from the root to any leaf node contains the same number of black nodes.

- The **black-height** of the red-black tree
  - The number of black nodes on any paths from the root to a leaf node.

- A height of a leaf cannot be larger than 2 times of a height of any leaf(P4, P5)

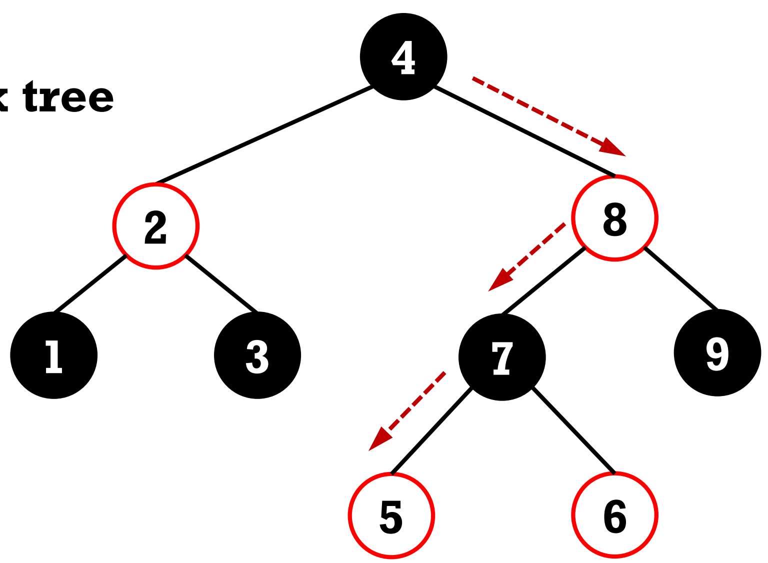
What is Red-Black Tree?



Searching in Red-Black Tree

- Description: This is the same way in the BST.
  - Compare the key of the node with the element.
    - If it is equal to the key, the element is found.
    - If it is less than the key, **search a left subtree**.
    - If it is greater than the key, **search a right subtree**.
  - Repeat until **the element is found** or **the node is NULL**.

Searching 5 in the red-black tree



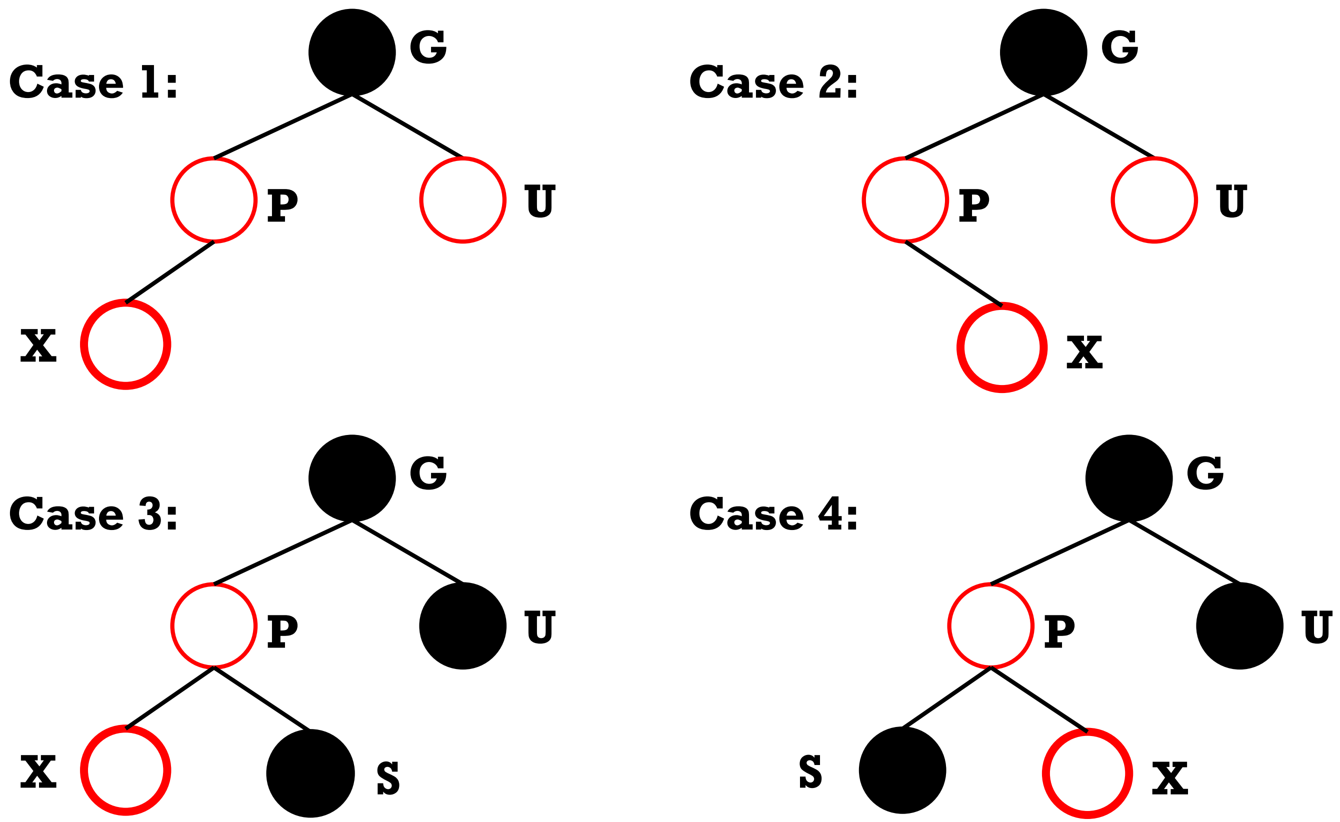
Insertion in Red-Black Tree

- How to perform insertions?
  - Insert an element as usual in the BST. (replace NIL node)
  - Color the node **RED**. (black causes P5 violation)
  - Check if the properties of the red-black tree is violated.
  - If violated, modify the red-black tree.
    - Color promotion, single rotation, and double rotation

- Which properties are violated?
  - **P3.** The **root** is **black**.
  - **P4.** If a node is **red**, then both of its children are **black**.
  - **P5.** Every **path from the root to any leaf node** contains **the same number of black nodes**.

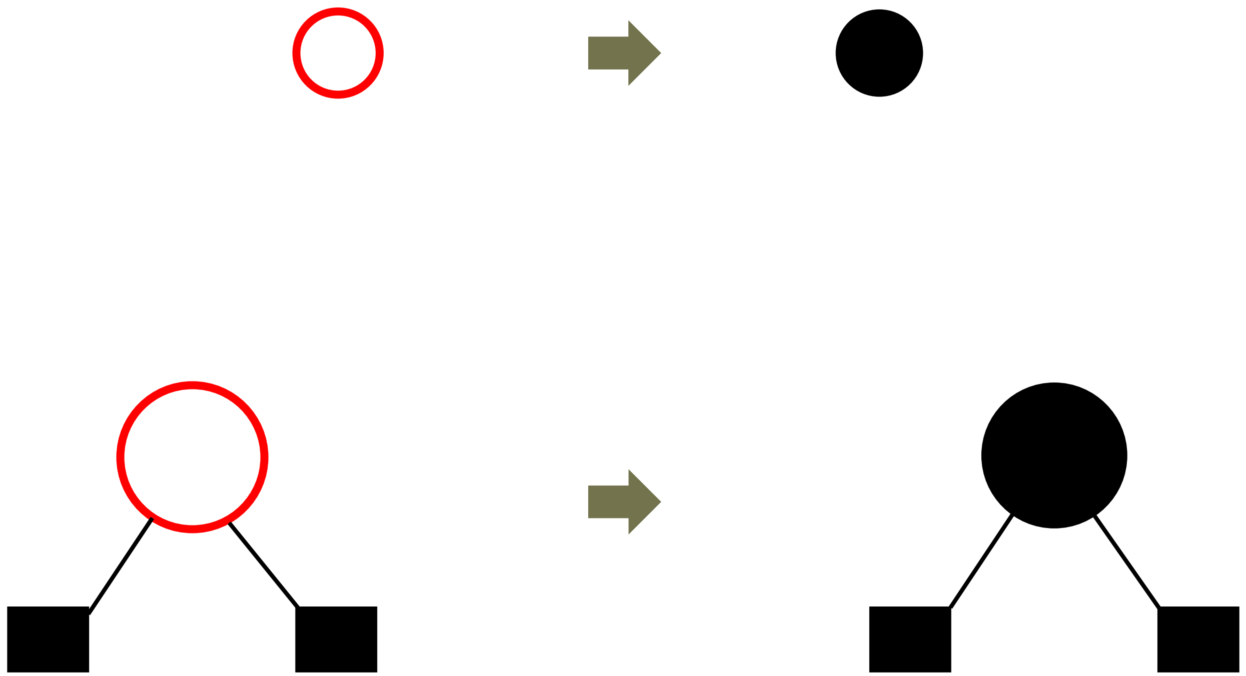
Insertion in Red-Black Tree

- When node ***X*** is inserted, there are five cases violating the properties:
  - Case 0: ***X*** is the root.
  - Case 1~4: The position of ***X*** and the color of the uncle.



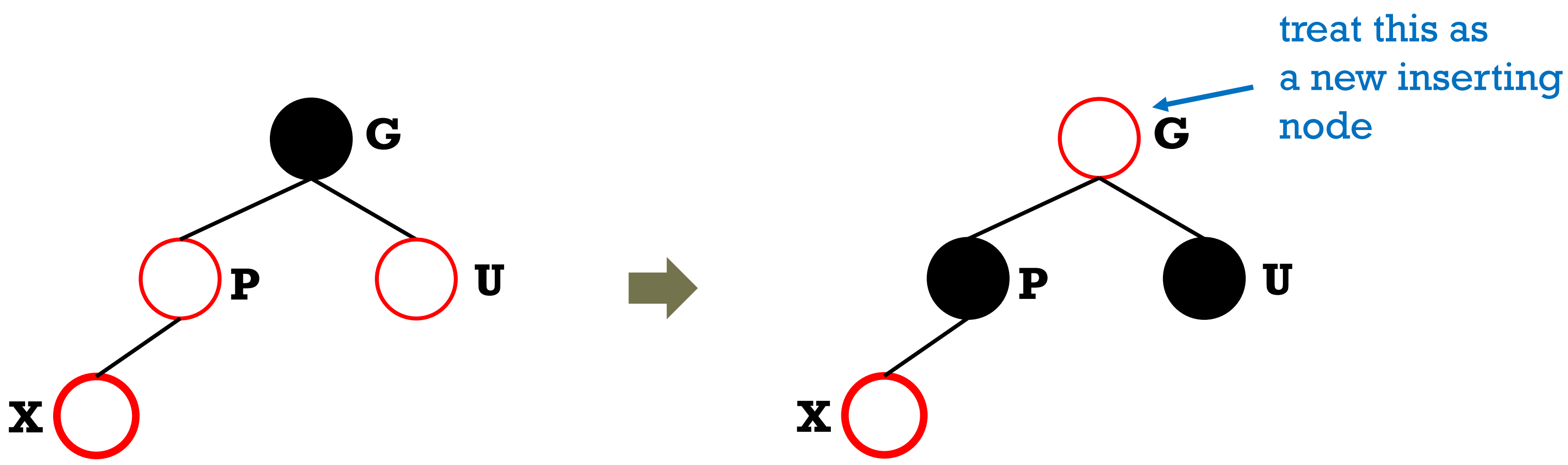
Insertion in Red-Black Tree

- Case 0: ***X*** is the Root
  - Because it violates P3, change the color X as black.



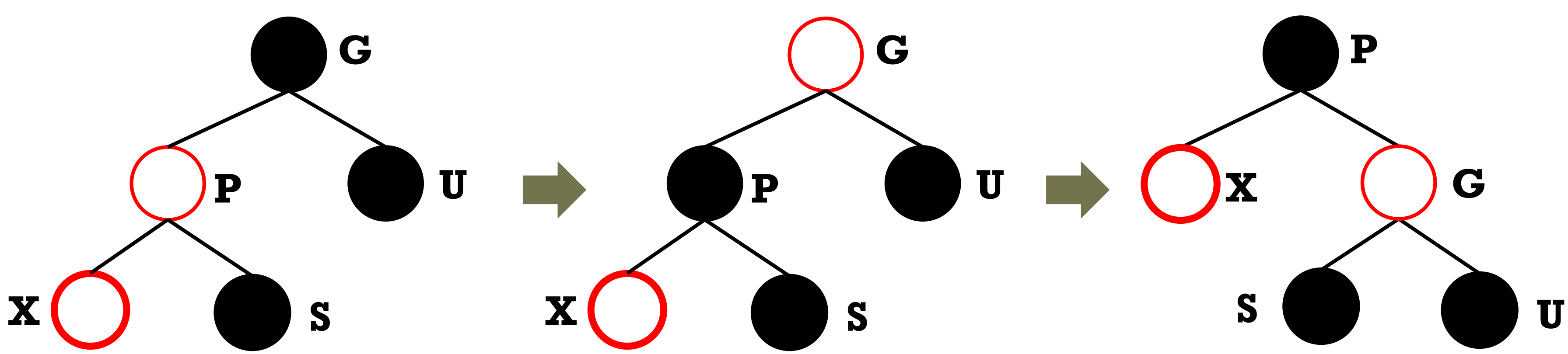
Insertion in Red-Black Tree

- Cases 1 and 2: The uncle of X is red
  - Because it violates P4 and P5, change the colors of G, P and U.
    - Change the colors of its parent and uncle as black. (color promotion)
    - Change the color of grandparent as red. (P5)
    - For grandparent G, check new situation in a **recursive** manner.



Insertion in Red-Black Tree

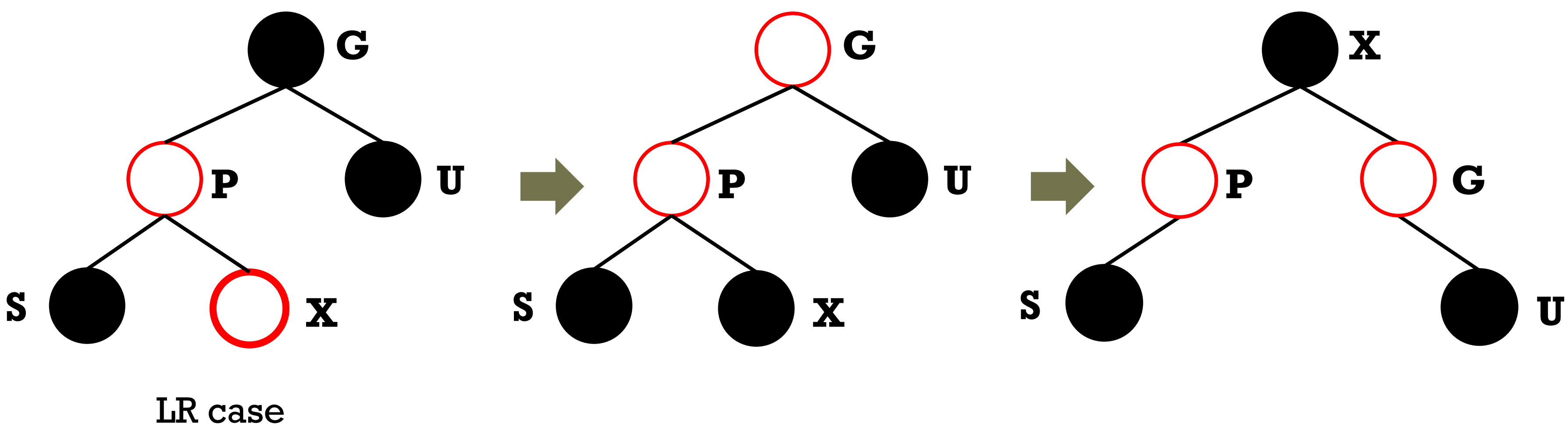
- Case 3: X is on the left and its uncle is black.
  - Because it violates P4 and P5, change the colors of G and P.
    - Change the color of its grandparent as red.
    - Change the color of its parent as black.
  - Still, because it violates P5(path with U lost one black node), try any restructuring…..RL LR RR LL.
    - similar to LL case of AVL, so RR wins



LL case

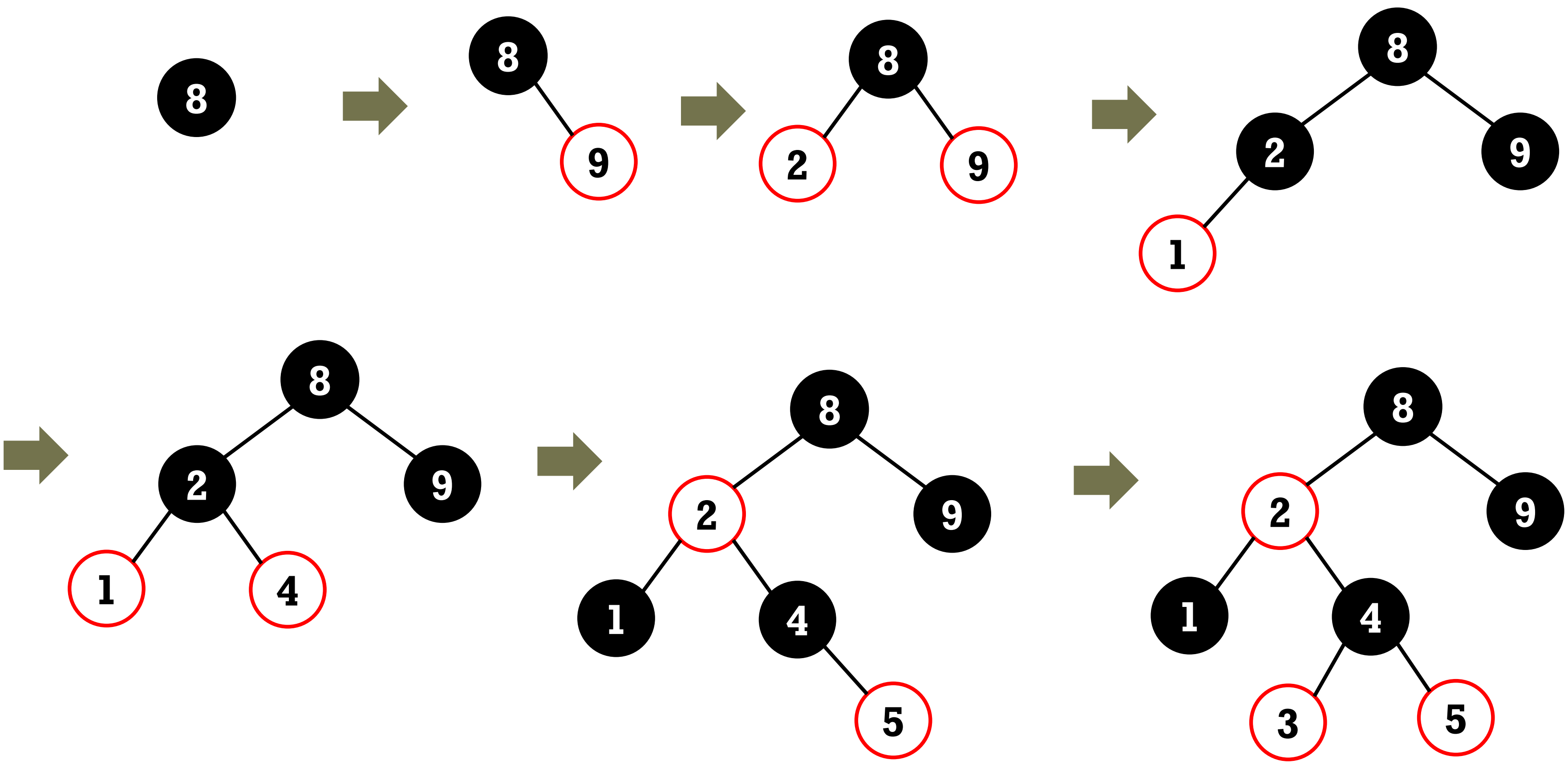
Insertion in Red-Black Tree

- Case 4: X is on the right and its uncle is black.
  - Because it violates P4 and P5, change the colors of G and X.
    - Change the color of its grandparent as red.
    - Change the color of X as black.
  - Still, because it violates P4 (S and U lost one black), RL rotate.



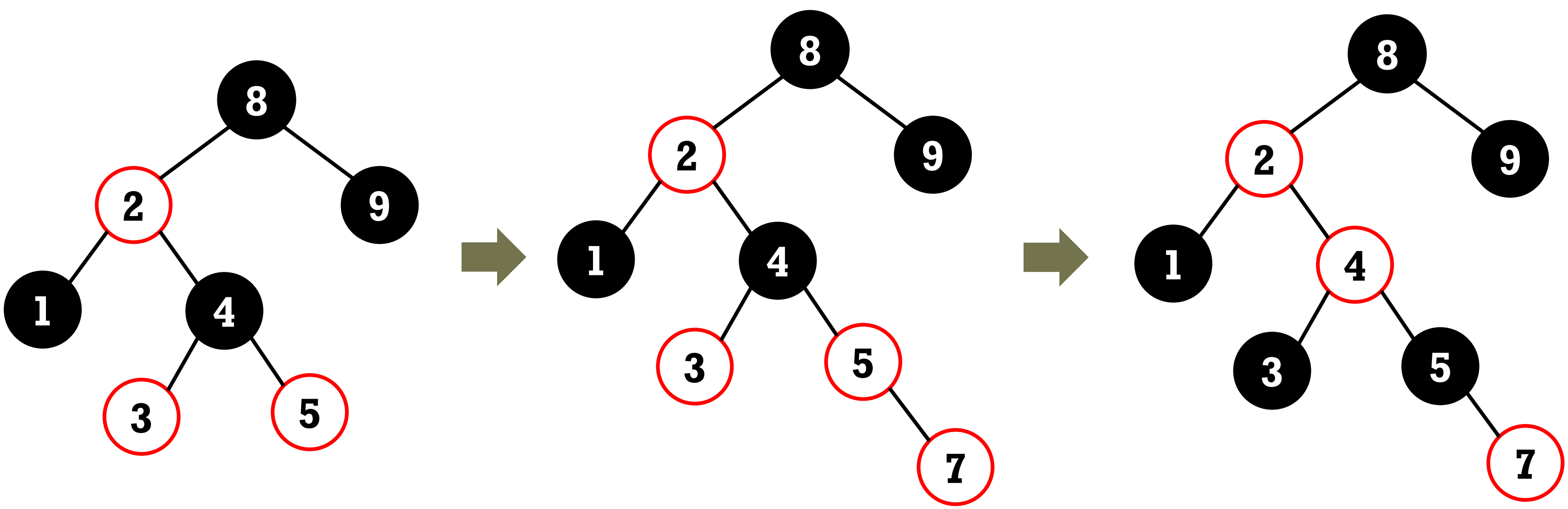
Insertion Example in Red-Black Tree

- Inserting 8, 9, 2, 1, 4, 5, 3, 7, and 6
  - https://www.cs.usfca.edu/~galles/visualization/RedBlack.html



Insertion Example in Red-Black Tree

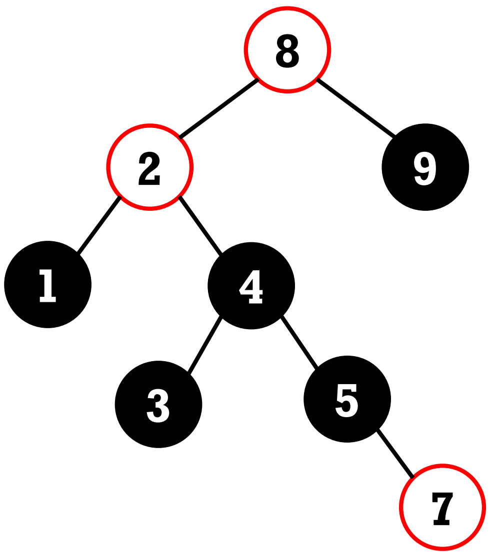
- Inserting 7
  - Because the uncle of 7 is red, perform color promotion.
  - Then, because the uncle of 4 is black, perform LR rotation.



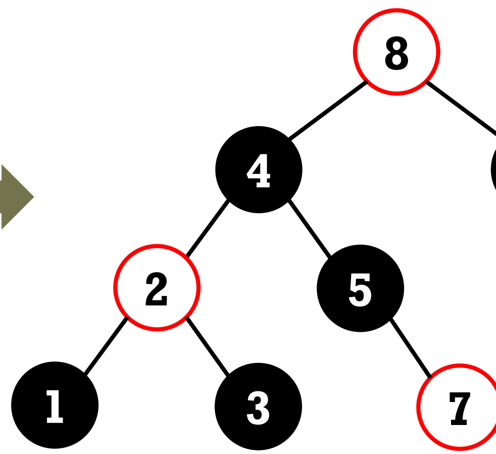
Color promotion

Insertion Example in Red-Black Tree

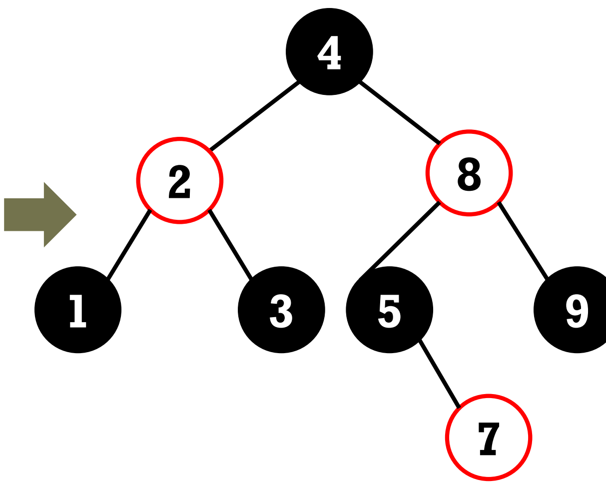
- Inserting 7
  - Because the uncle of 7 is red, perform color promotion.
  - Then, because the uncle of 4 is black, perform LR rotation.



Changing color
for 4 and 8



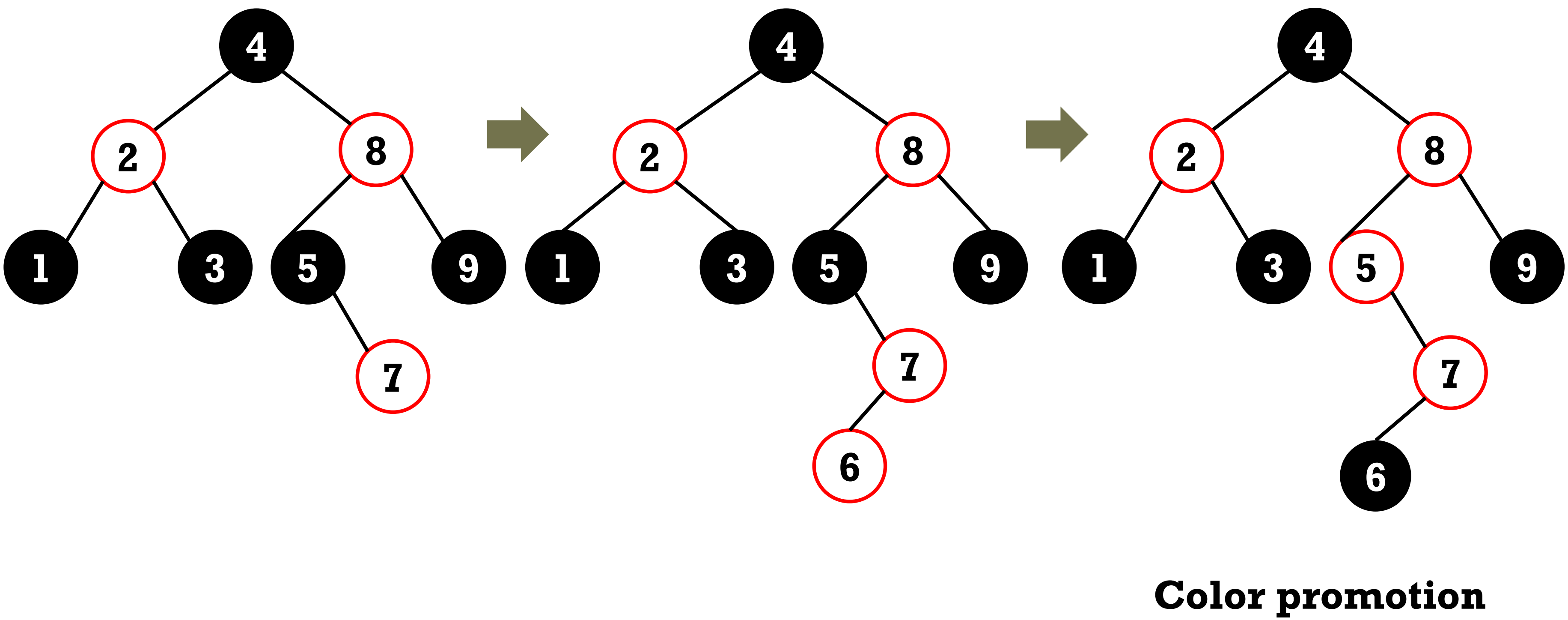
LL rotation for 2 and 4



RR rotation for 4 and 8

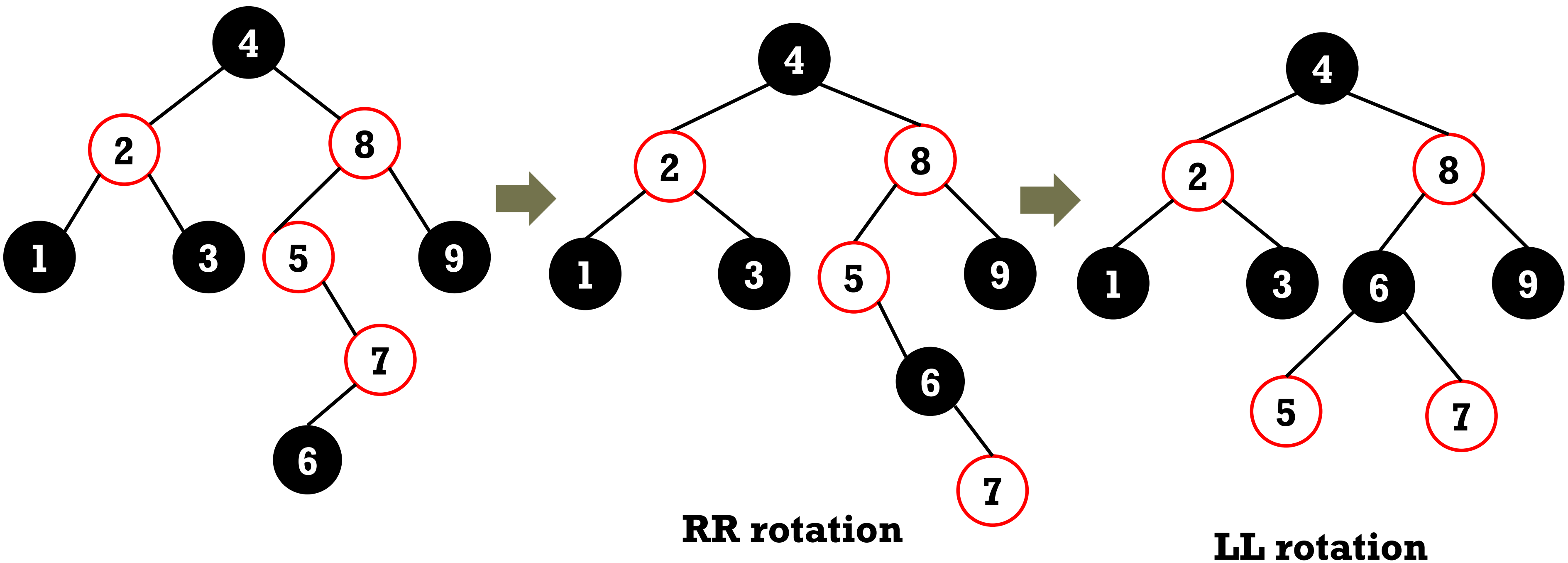
Insertion Example in Red-Black Tree

- Inserting 6
  - Because the uncle of 6 is black, do RL rotation for 5, 6, and 7.



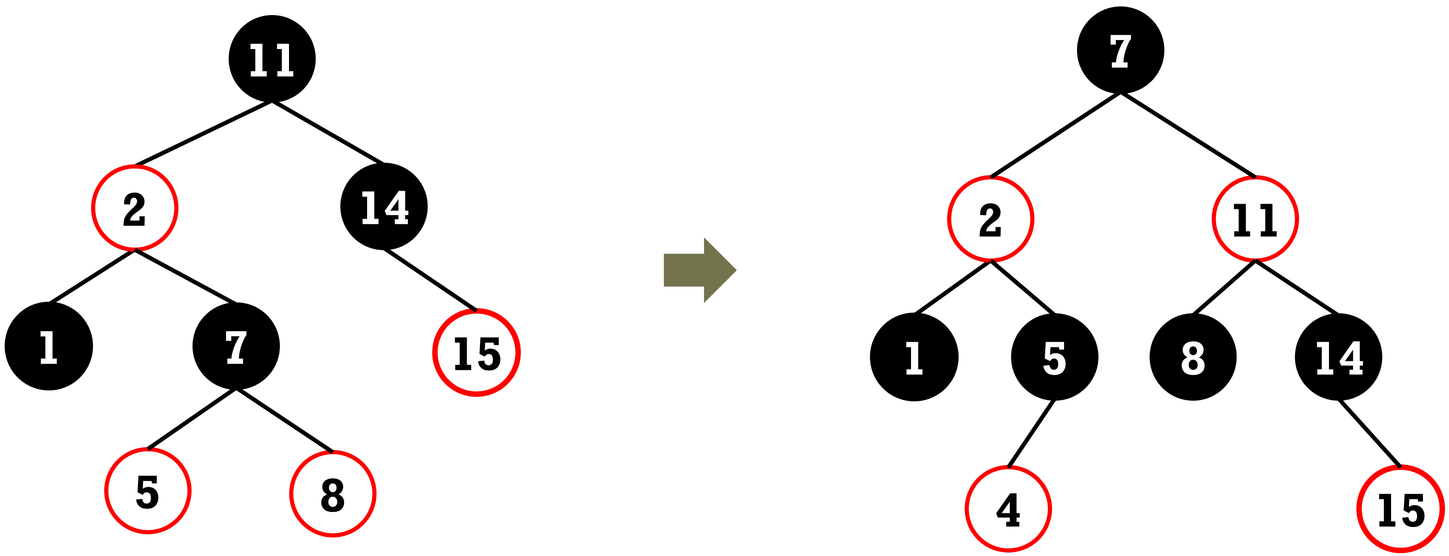
Insertion Example in Red-Black Tree

- Inserting 6
  - Because the uncle of 6 is black, do RL rotation for 5, 6, and 7.



Exercise: Insertion in Red-Black Tree

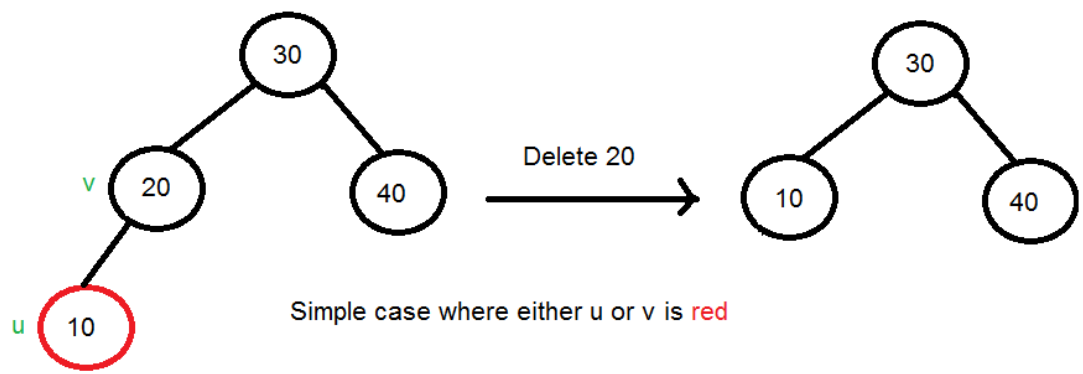
- What if inserting 4 to this red-black tree?



- More examples
  - https://www.youtube.com/watch?v=1IqZT54bhz8

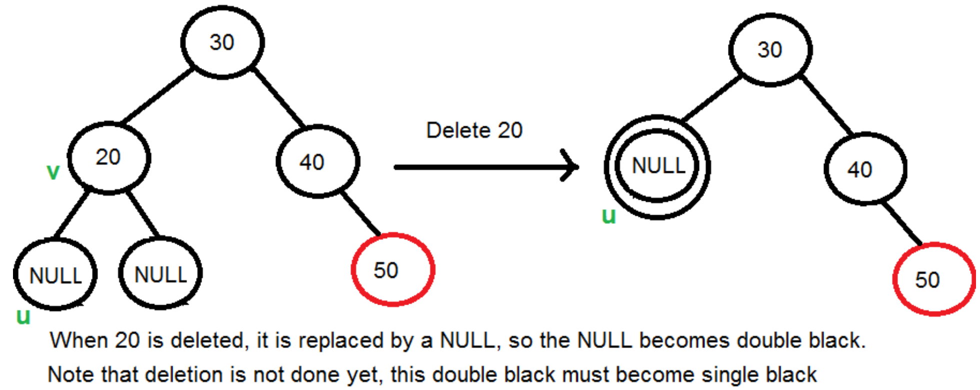
Deletion in RB Tree

- Perform BST deletion
  - copy the leftmost(smallest) value of right subtree to the node to be deleted
  - delete the leftmost node (leaf node or a node with one child)
- Let v be the node to be deleted
- Let u be the child that replaces v
- If either u or v is red

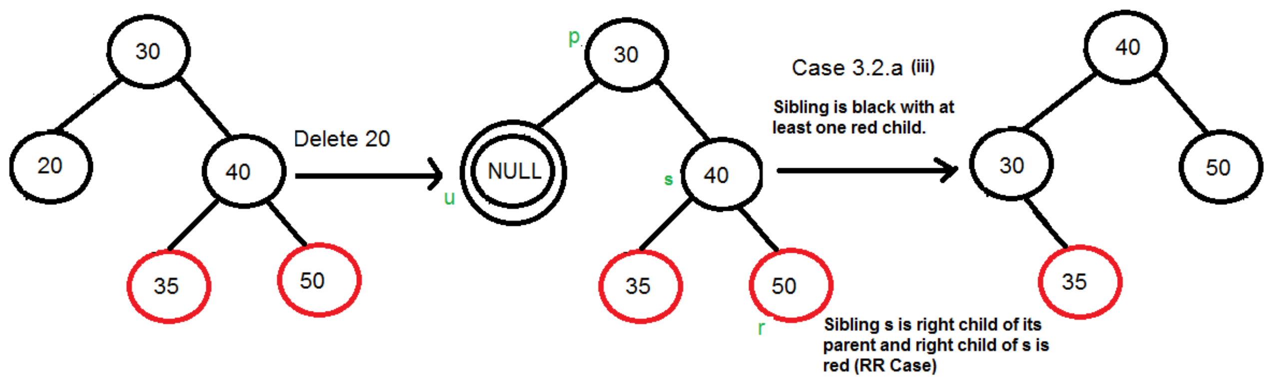


Deletion in RB Tree

- Both u and v are black
  - color u as double black meaning black height is decreased
  - let's remove double black

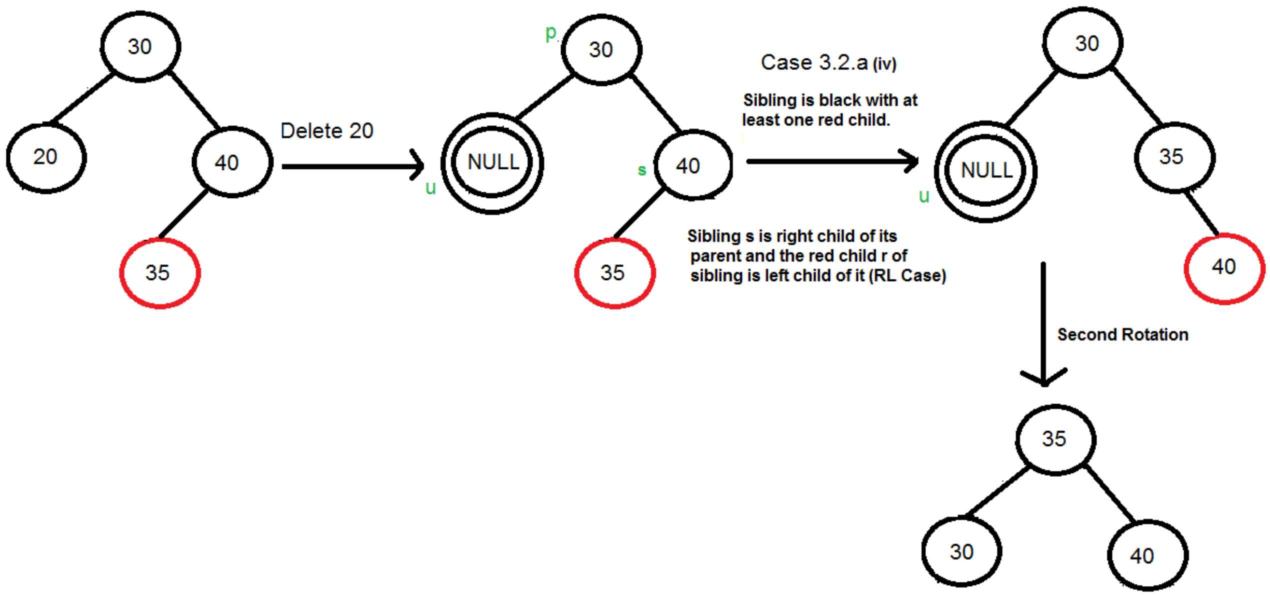


- sibling s is black and at least one child of s is red



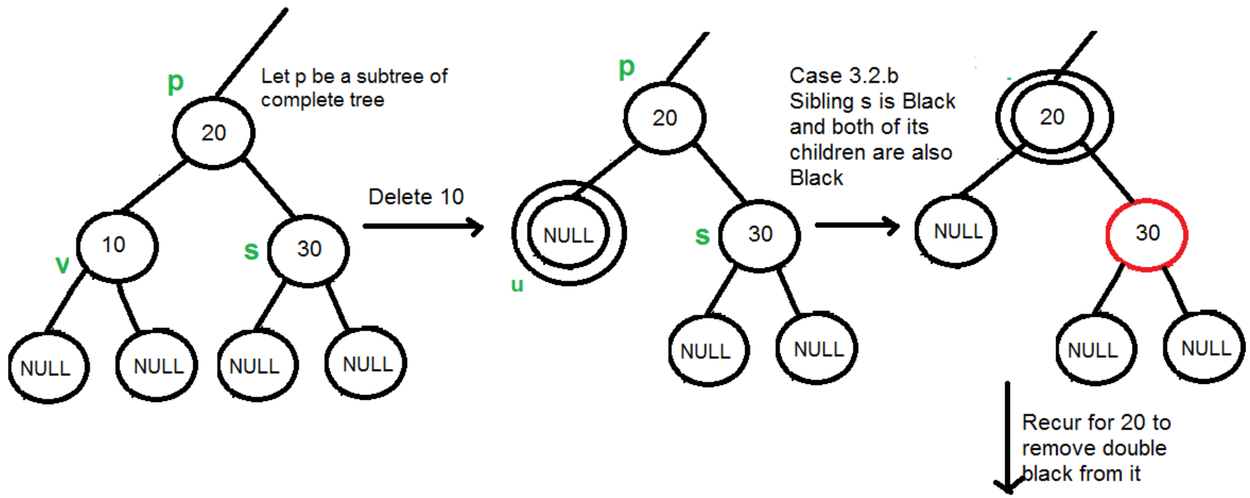
Deletion in RB Tree

- RL case
  - s is a right child of p and has a left red child

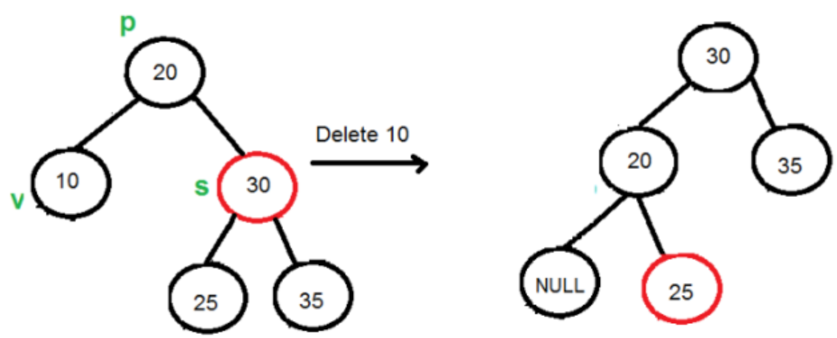


Deletion in RB Tree

- all black



- if S is red, (S cannot be a NIL and P should be black)



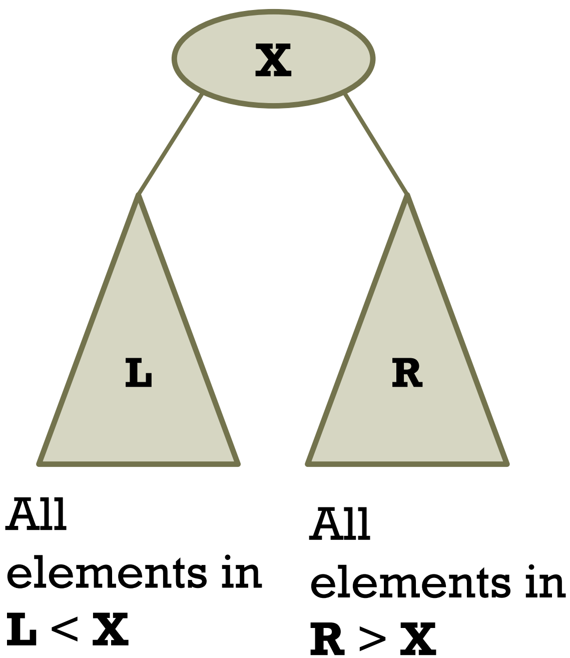
2-3 Tree

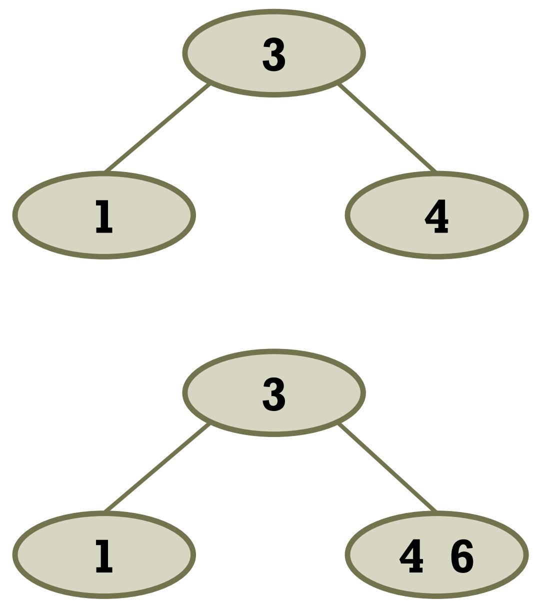
- Description
  - Extension of a binary search tree
    - The number of elements for each node is at most two.
    - The number of children for each node is at most three.
  - Invented by **John Hopcroft** in 1970
  - “A B-tree of order 3 is a 2-3 tree” by Donald Knuth

- Definition
  - Every internal node has either a 2-node or a 3-node.
    - **2-node**: it has **one element** with **two children**.
    - **3-node**: it has **two elements** with **three children**.
  - **Self-balanced tree**: All leaf nodes are at the same heights.

What is 2-3 Tree?

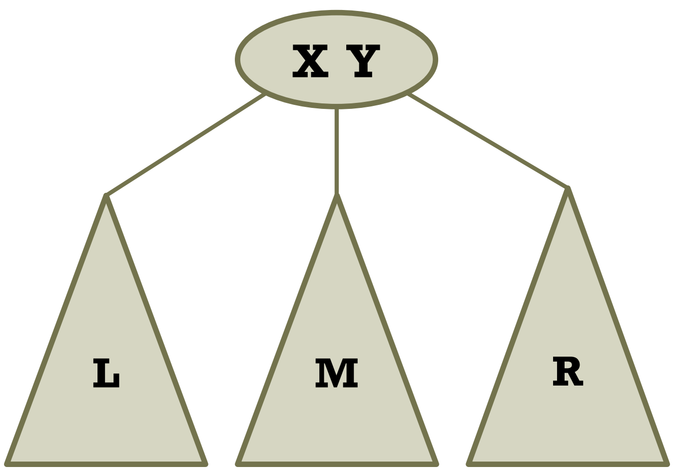
- The 2-node has one element X with two children.
  - Let **L** and **R** be a left child node and a right child node.
    - **X** is greater than all elements in **L**.
    - **X** is less than all elements in **R**.
  - **L** and **R** are non-empty 2–3 trees of the **same heights**.



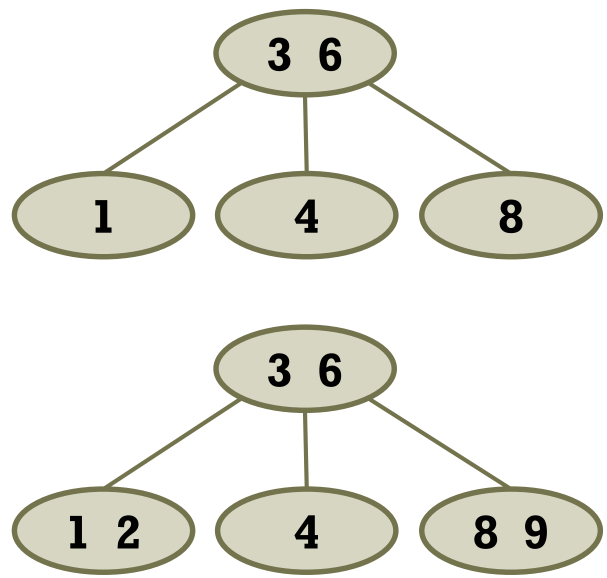


What is 2-3 Tree?

- The 3-node has two elements X and Y, where X < Y.
  - Let **L**, **M**, and **R** be a left child node, a middle child node, and a right child node, respectively.
    - **X** is greater than each data element in **L** and less than each data element in **M**.
    - **Y** is greater than each data element in **M** and less than each data element in **R**.
  - **L**, **M**, and **R** are non-empty 2–3 trees of **same heights**.

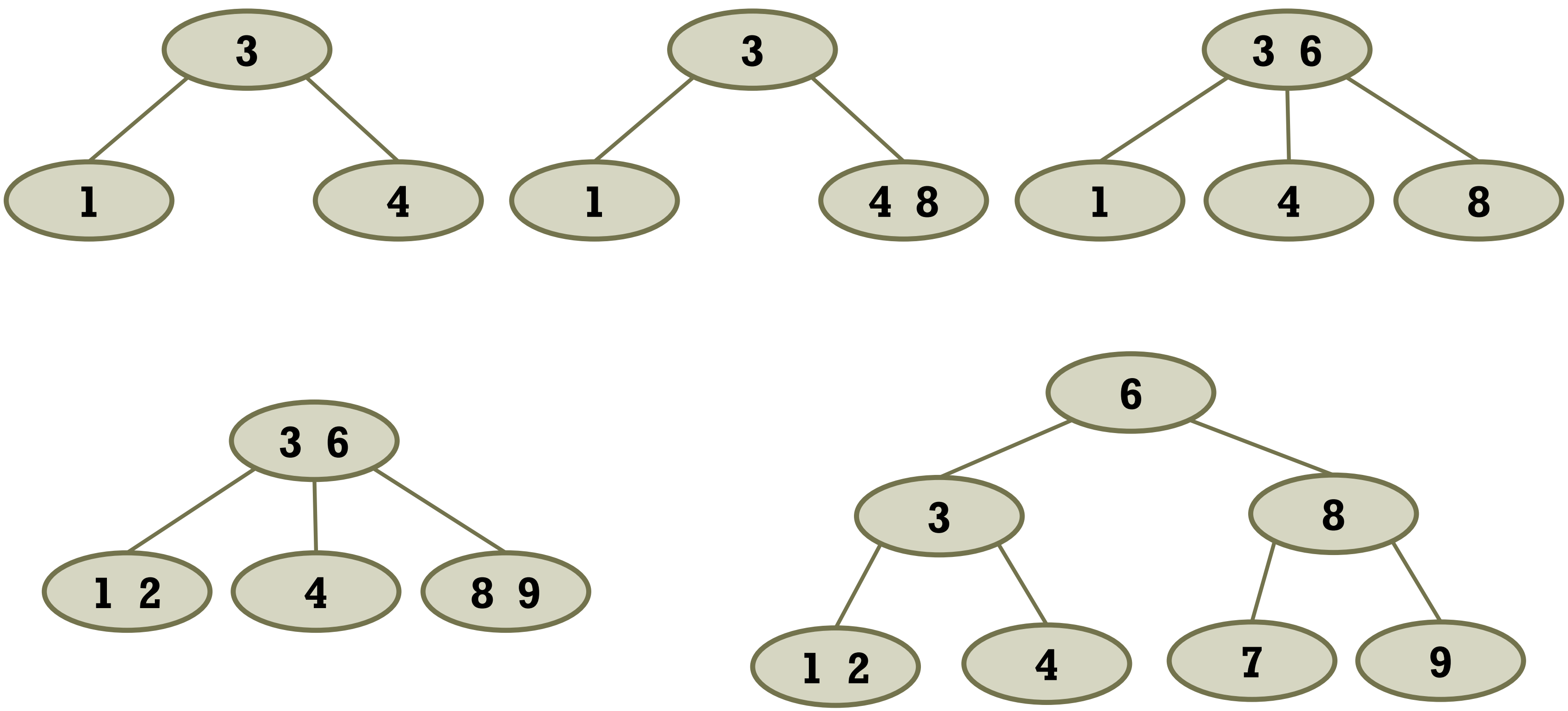


X < All elements in **M** < **Y**



What is 2-3 Tree?

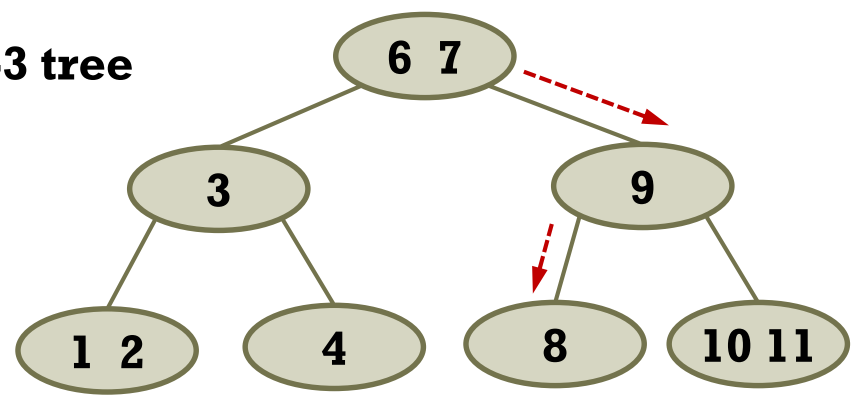
- Examples



Searching in 2-3 Tree

- Description: This is similar to the BST.
  - Compare the key of the node with the element.
    - If it is equal to the first key, the element is found.
    - If it is less than the first key, **search a left subtree**.
    - If it is less than the second key, **search a middle subtree**.
    - If it is greater than the second key, **search a right subtree**.
  - Repeat until **the element is found** or **the node is NULL**.

Searching 8 in the 2-3 tree

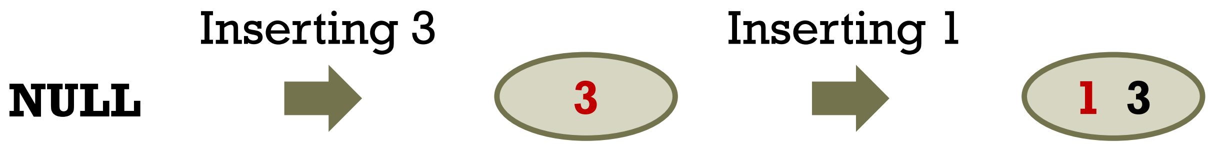


Insertion in 2-3 Tree

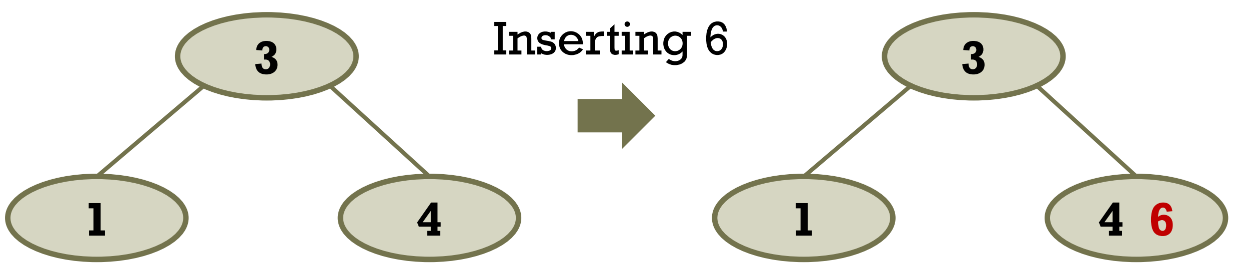
- There are three possible cases for insertions.
  - **Case 1**: If the tree is **empty**, **create a node** and put a value into the node.
  - **Case 2**: If the leaf node has **only one value**, **simply put a new value** into the node.
  - **Case 3**: If the leaf node has **two values**, **split the node** and **promote the median** of the three values to parent.
    - If **the parent then has three values**, **continue to split** and **promote**, forming a new root node if necessary.

Insertion in 2-3 Tree

- Case 1: Initializing a 2-3 tree
  - If the tree is empty, create a node and put a value into the node.

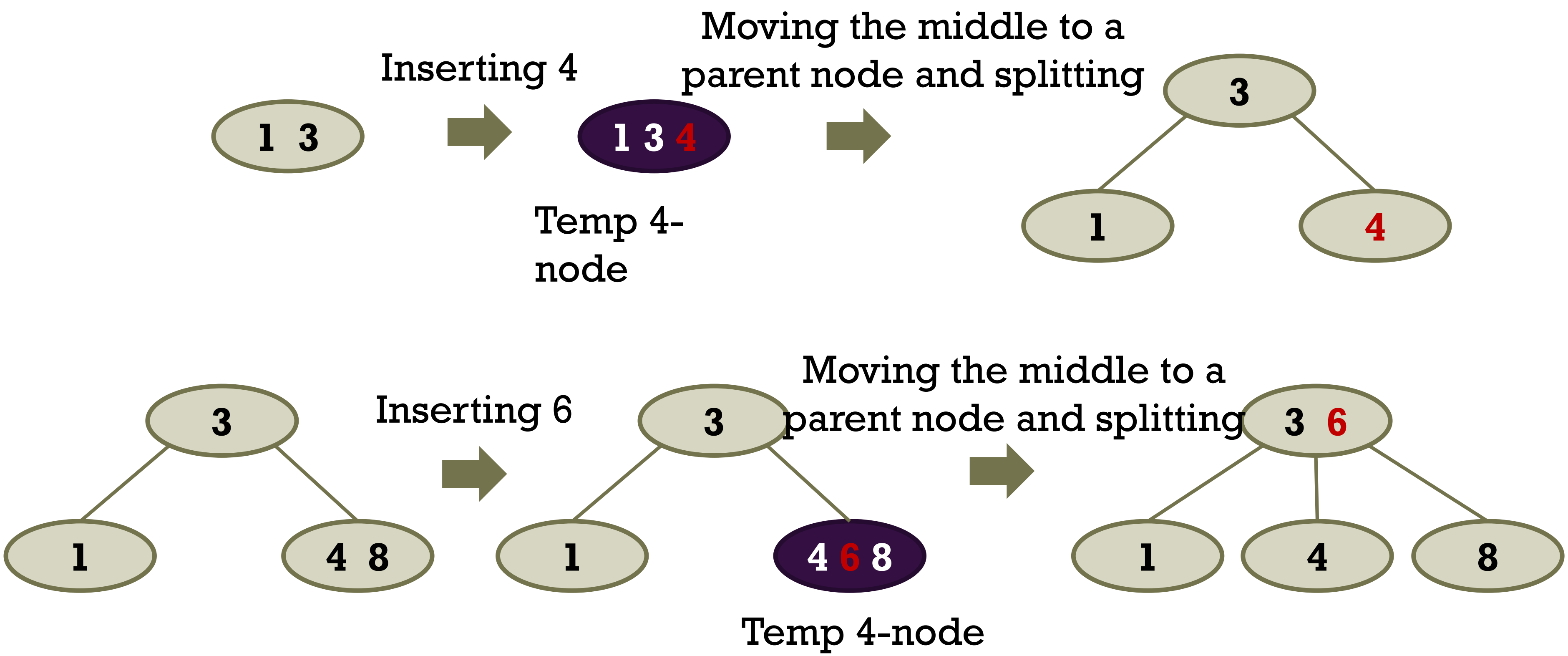


- Case 2: Inserting an element to a 2-node
  - If the leaf node has only one value, simply put the new value into the node.



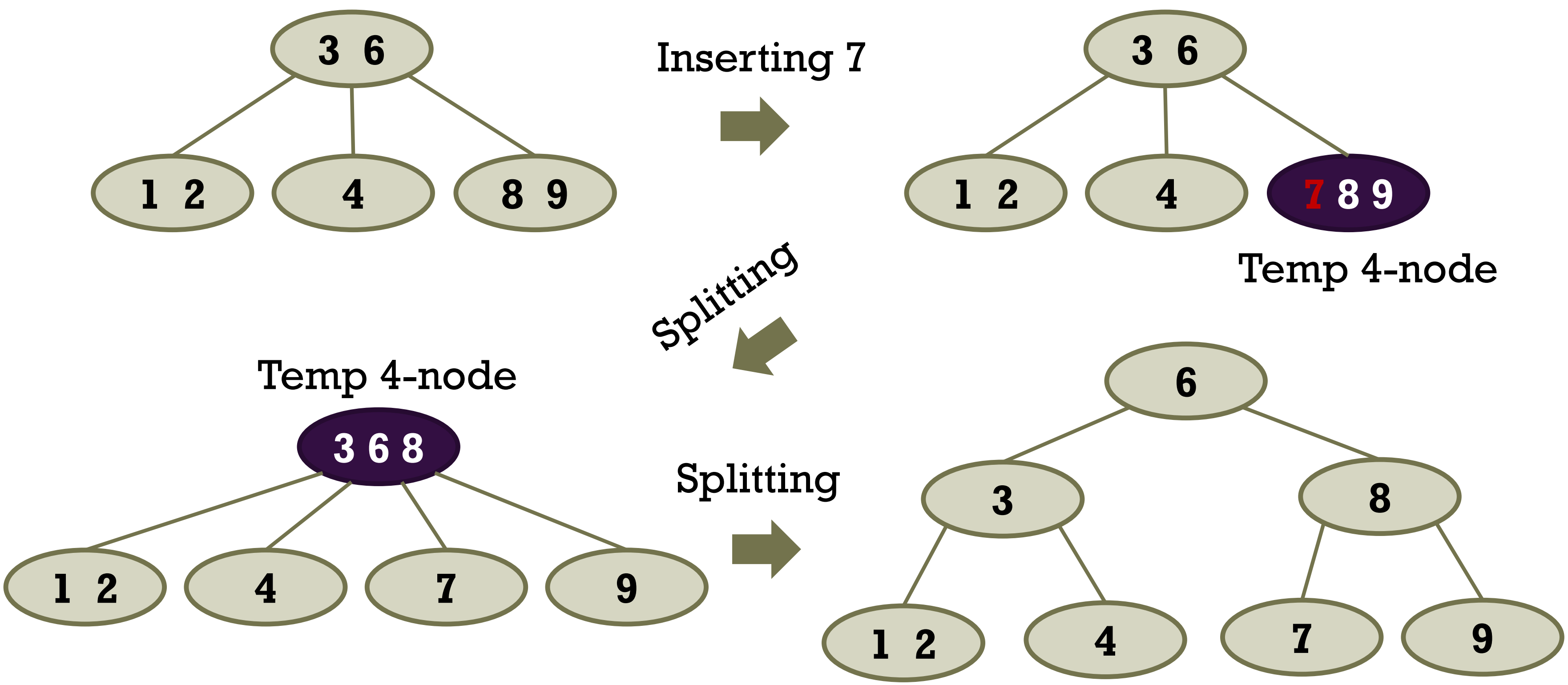
Insertion in 2-3 Tree

- Case 3: Inserting an element to a 3-node
  - If the leaf node has **two values**, **split the node** and **promote the median** of the three values to parent.



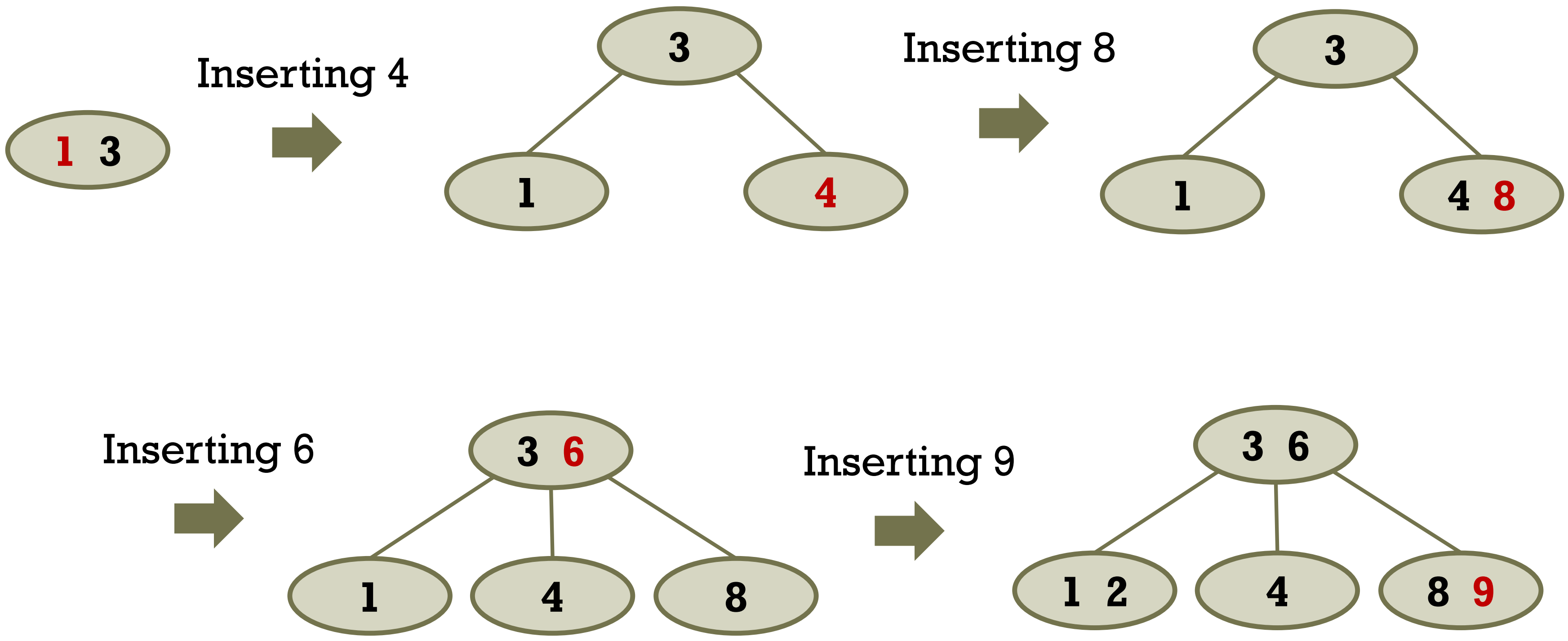
Insertion in 2-3 Tree

- Case 3: Inserting an element to a 3-node
  - If the leaf node has **two values**, **continue to split** and **promote**, forming a new root node if necessary.



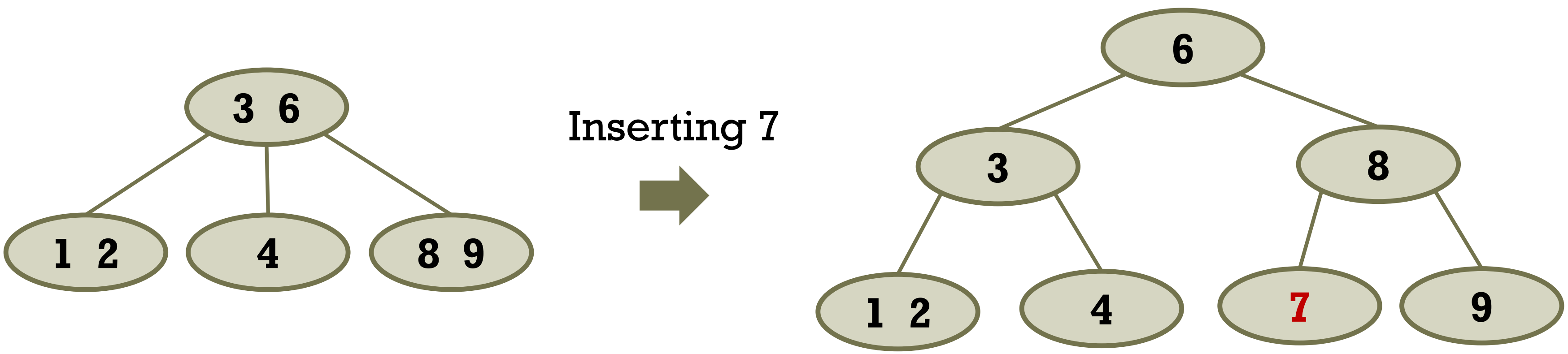
Insertion Example in 2-3 Tree

- Inserting 3, 1, 4, 8, 6, and 9



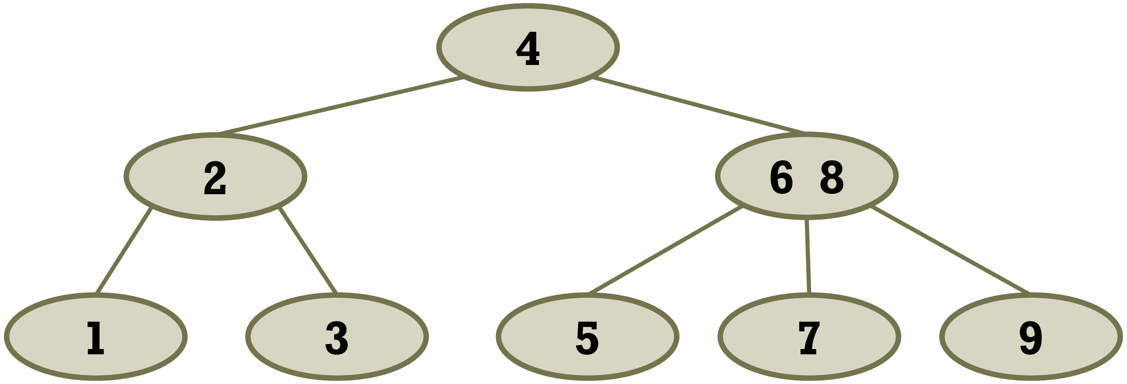
Insertion Example in 2-3 Tree

- Inserting 7
  - Inserting an element to a 3-node
    - If the leaf node has **two values**, **continue to split** and **promote**, forming a new root node if necessary.



Exercise: Insertion in 2-3 Tree

- Inserting 1, 2, 3, 4, 5, 6, 7, 8, and 9
  - https://www.cs.usfca.edu/~galles/visualization/BTree.html



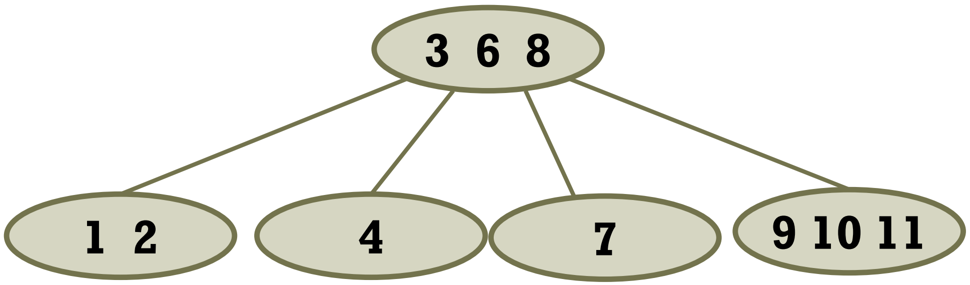
How to Perform Deletions?

- There are three possible cases for deletions.
  - **Case 1**: If the node to be removed is a **leaf node**.
    - If the leaf node is 3-node, simply delete the element.
    - Otherwise, …
  - **Case 2**: If the node to be removed is an **internal node**.
    - Borrow an element from one of its children.

- How to implement the 2-3 tree?
  - **Try to implement the 2-3 tree for yourself!**

B-Tree as Extension of 2-3 Tree

- Extending 2-3 tree into 2-3-4 tree
  - **4-node**: it has **three elements** with **four children**.



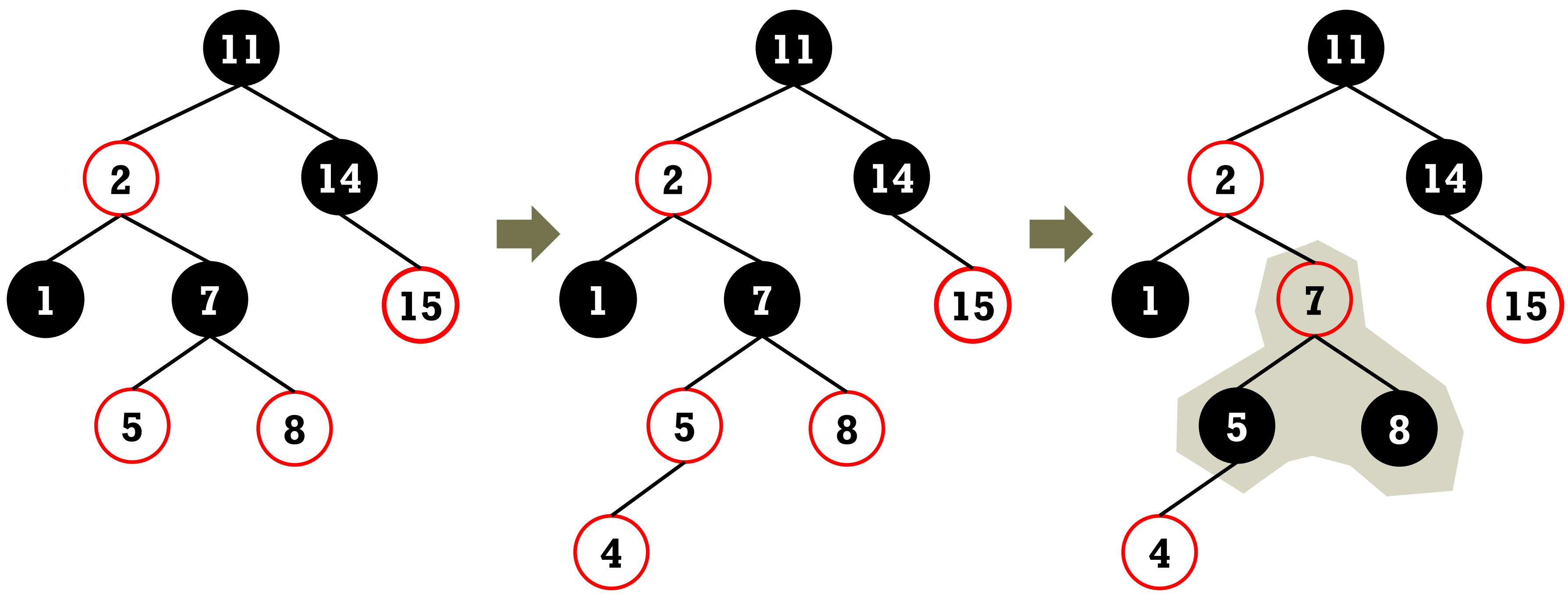
- B-tree
  - An generalized extension of the 2-3 tree and the 2-3-4 tree
  - This has been widely used for indexing records in database.
  - https://en.wikipedia.org/wiki/B-tree

Summary of Balanced Trees

- AVL tree vs. red-black tree vs. B-tree
  - Because they are all self-balanced trees, the operations for searching, insertion, and deletion are bound by $\boldsymbol{O(\log n)}$.
    - n is the number of nodes.
  - Practically, there are some trade-off factors.
    - Speed of retrieval vs. speed of updates
      - The AVL tree is more effective for **frequent retrieval**.
      - The red-black tree is more effective for **frequent updates**.
    - The B-tree is effective for managing a **large-scale dataset** and need to access the data from the disk.

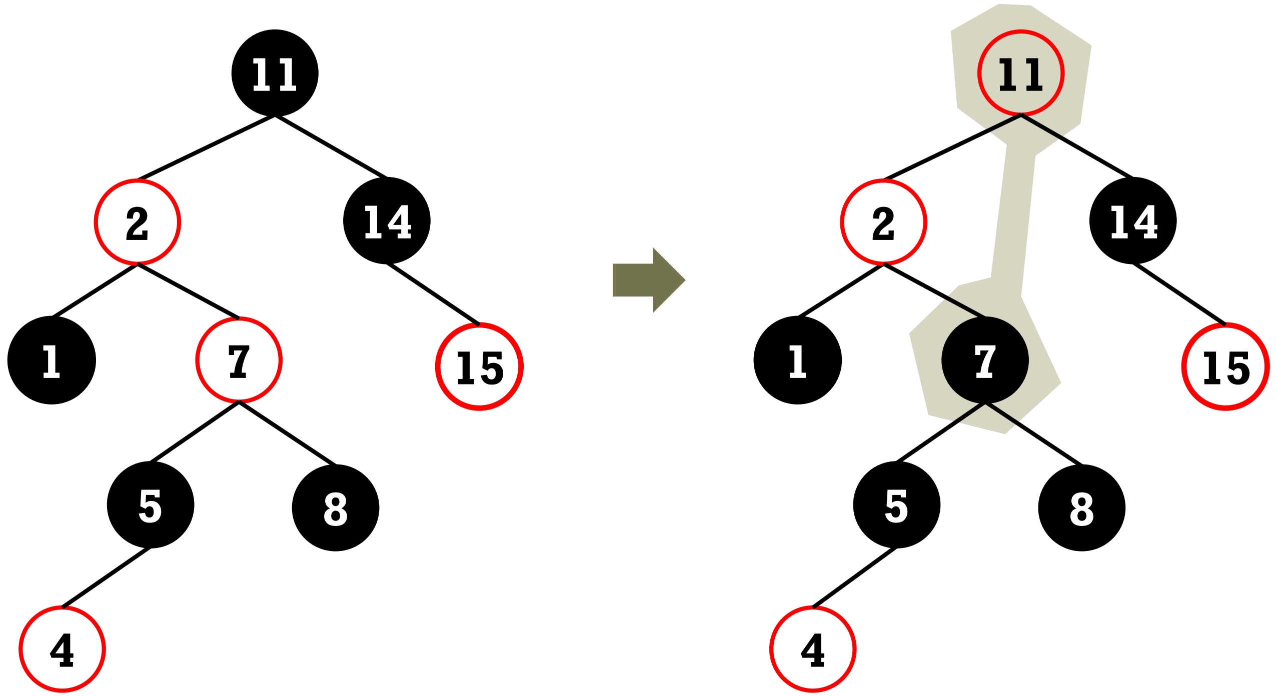
Exercise: Insertion in Red-Black Tree

- Inserting 4 to this red-black tree
  - Insert 4 as the left child of 5.
  - As the case 1, perform color promotion for 5, 7, and 8.



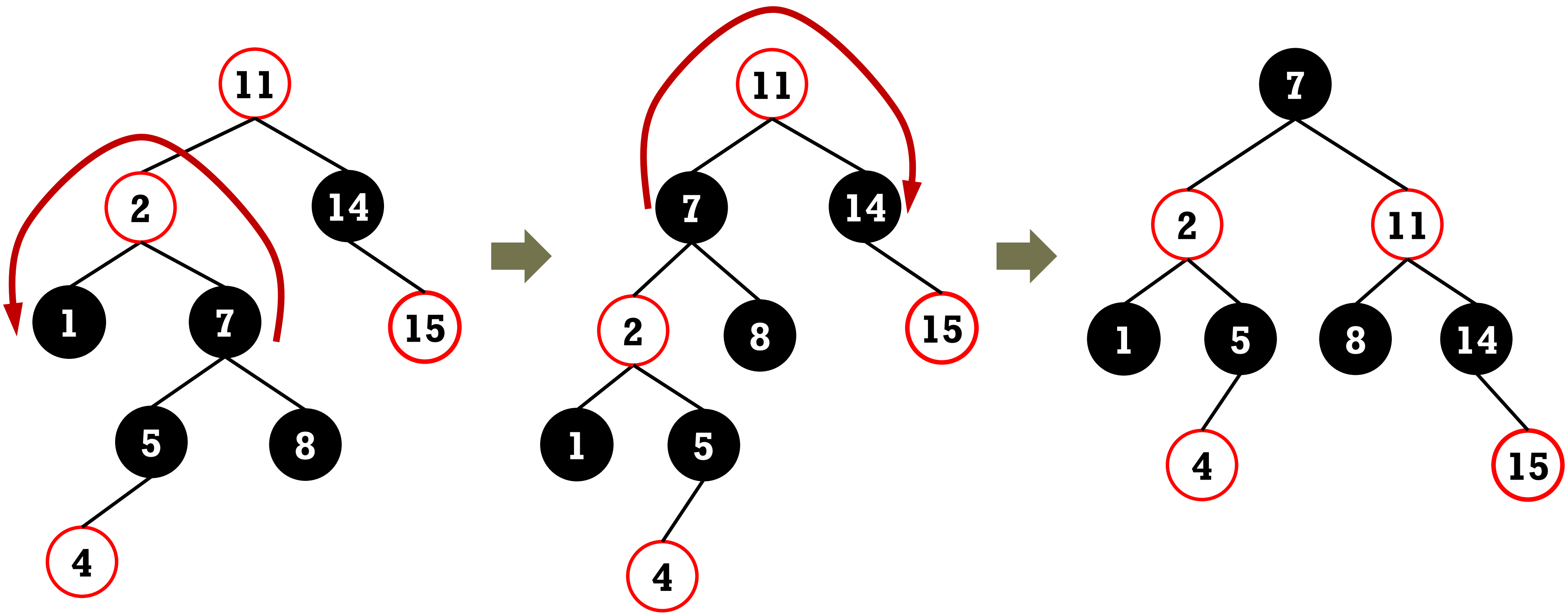
Exercise: Insertion in Red-Black Tree

- Inserting 4 to this red-black tree
  - Because of the case 5 for 7, change the colors of 7 and 11.
  - Then, perform LR rotation for 7.



Exercise: Insertion in Red-Black Tree

- Inserting 4 to this red-black tree
  - Because of the case 5 for 7, change the colors of 7 and 11.
  - Then, perform LR rotation for 7.



Exercise: Insertion in 2-3 Tree

- Inserting 1, 2, 3, 4, 5, 6, 7, 8, and 9
  - https://www.cs.usfca.edu/~galles/visualization/BTree.html