Visualization: https://www.cs.usfca.edu/~galles/visualization/RedBlack.html
Source code: https://junboom.tistory.com/18

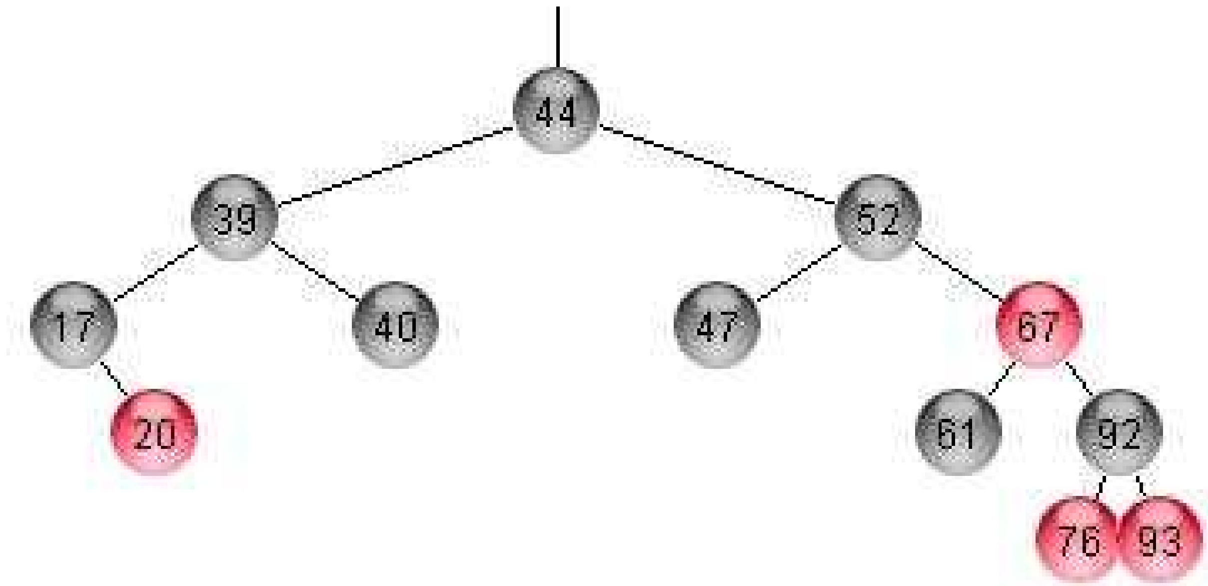
Red-Black tree

Balanced BST

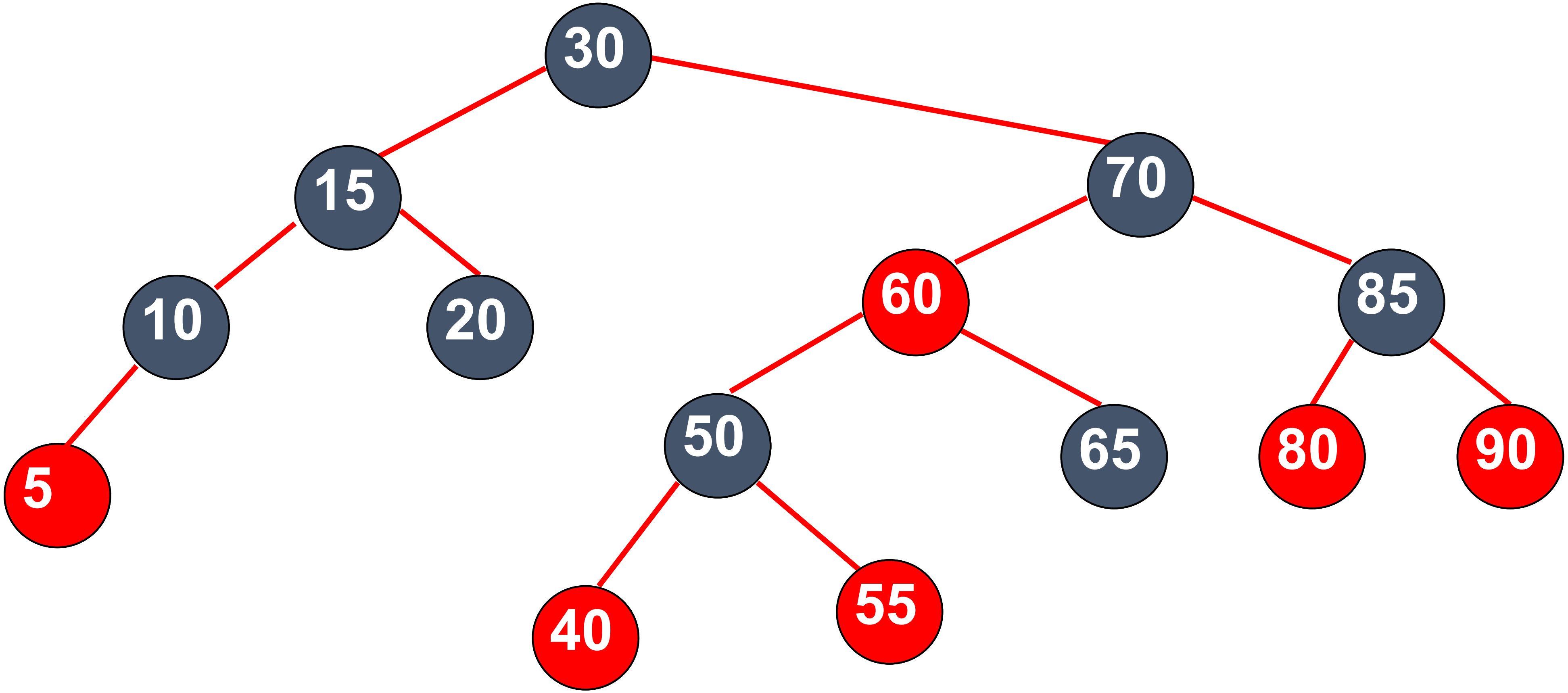
- |depth(leftChild) – depth(rightChild)| <= 1
- Example
  - AVL Trees – Maintain a three-way flag at each node (-1,0,1) determining whether the left sub-tree is longer, shorter or the same length. Restructure the tree when the flag would go to –2 or +2.
  - Red-black trees – Restructure the tree when rules among nodes of the tree are violated as we follow the path from root to the insertion point.

A Red-Black Tree

- Red-black trees are tress that conform to the following rules:
  1. Every node is colored (either red or black)
  2. The root is always black
  3. If a node is red, its parent and children must be black
  4. Every path from the root to leaf, or to a null child, must contain the same number of black nodes.

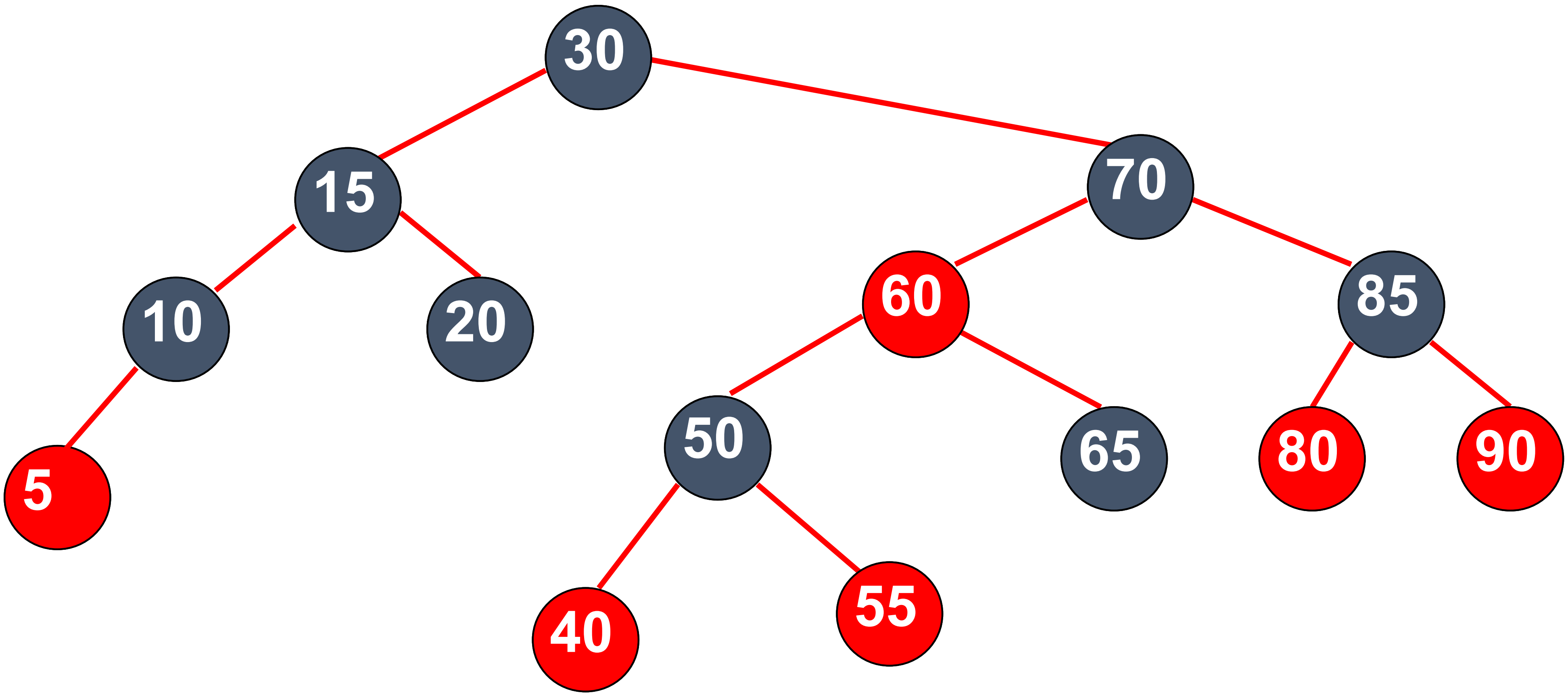


A Red-Black Tree



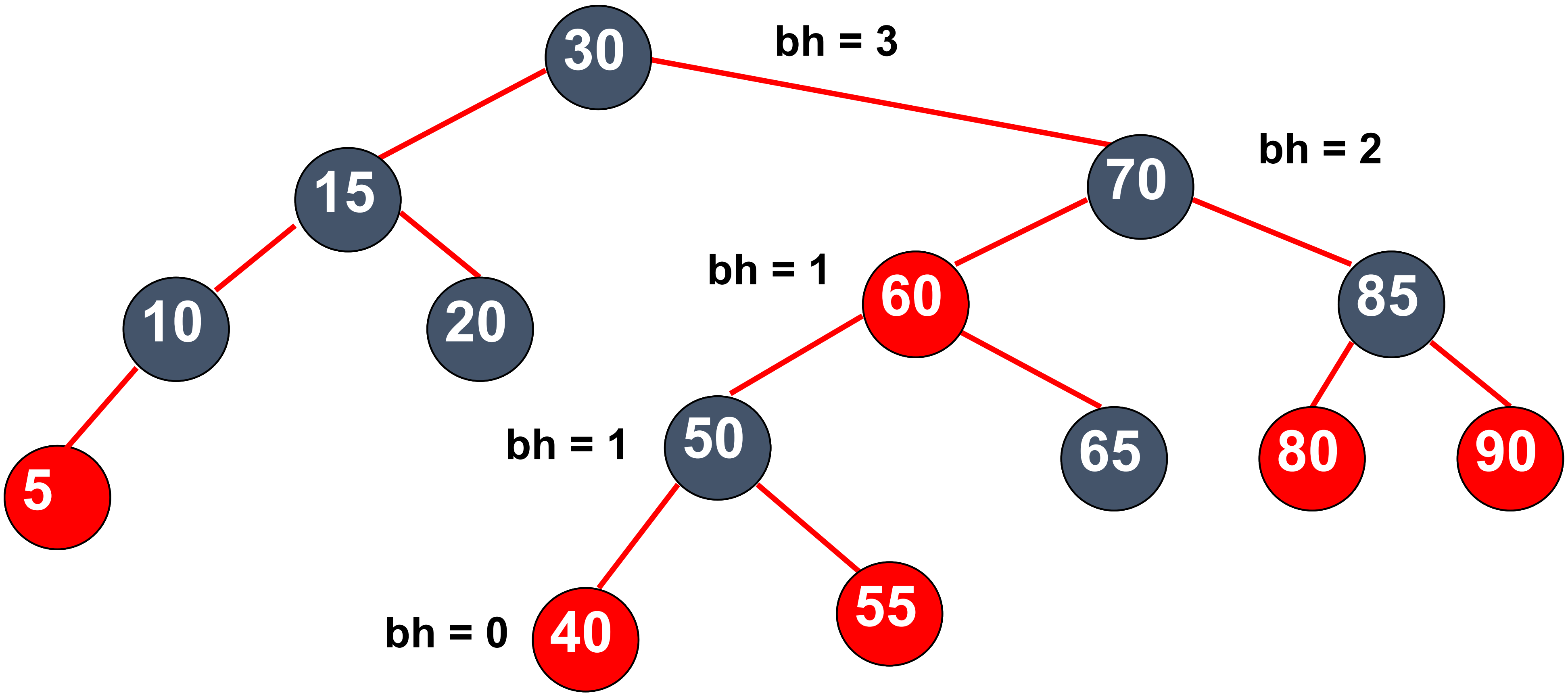
1. Every node is colored either red or black
2. The root is black

A Red-Black Tree



3. If a node is red, its children must be black

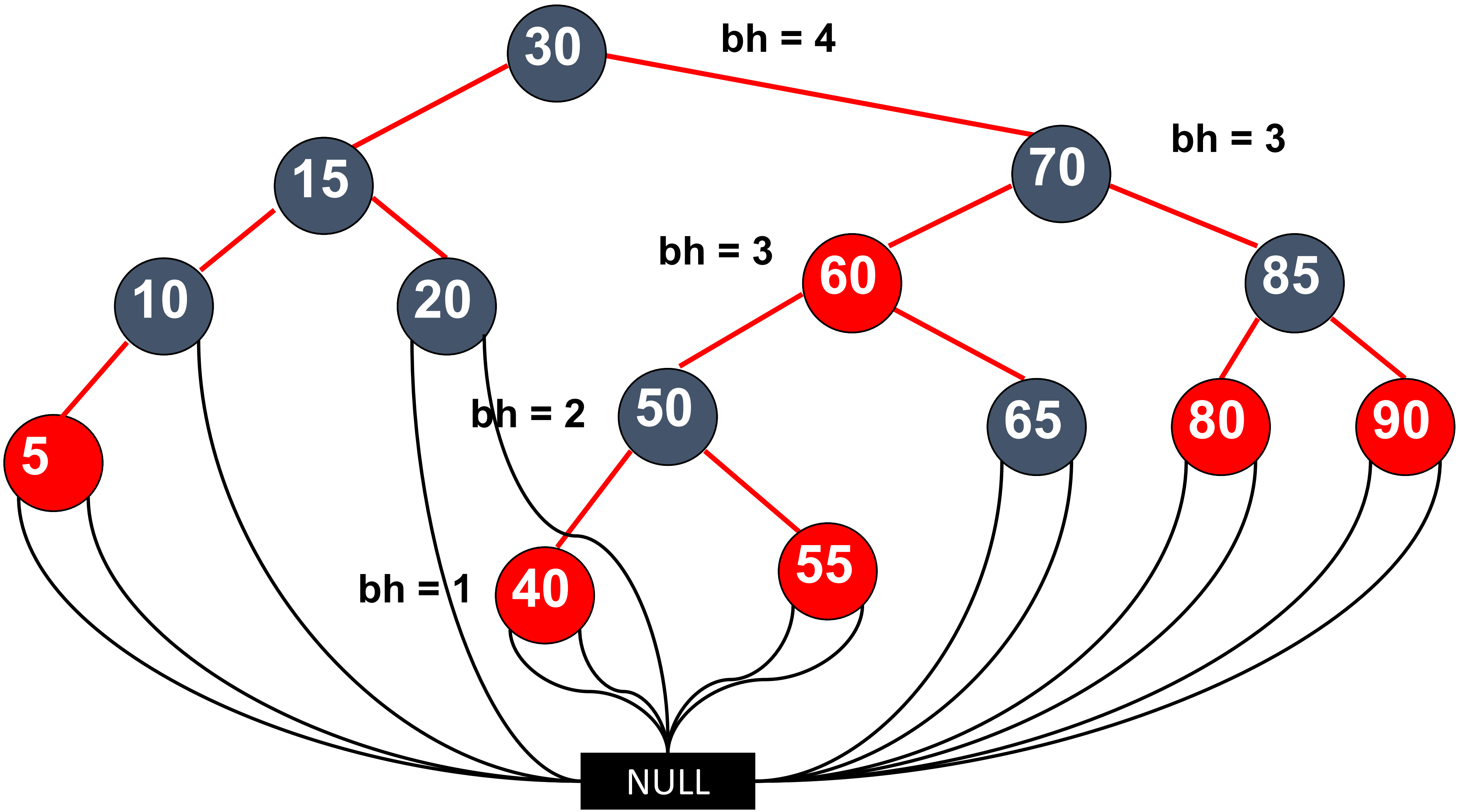
A Red-Black Tree



4. All simple paths from any node x to a descendent leaf must contain the same number of black nodes

= *black-height(x)*

A Red-Black Tree



5. We assume a null point is black.

Insertion Algorithm

- Where
  - New node: *x*
    - *x* is always red
  - The parent of *x*: *p*
  - The parent of *p*: p^2
  - The sibling of *p(uncle)*: *u*
- Case 1: empty tree – insert black node
- Case 2: *p* is black – insert red node
- Case 3: *p* is red
  - Case 3-1: *u* is red
  - Case 3-2: *u* is black
    - Case 3-2-1: *x* is the right child of *p*
    - Case 3-2-2: *x* is the left child of *p*

We consider only when p is the left child of p^2.

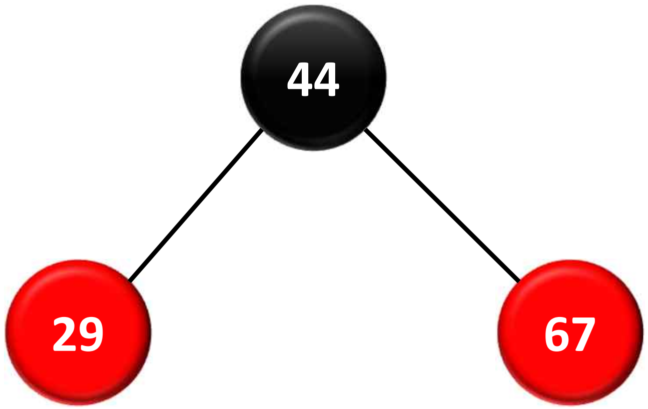
If p is the right child of p^2, left ←→ right

Case 1

- Empty tree: insert black node
- Insert 44

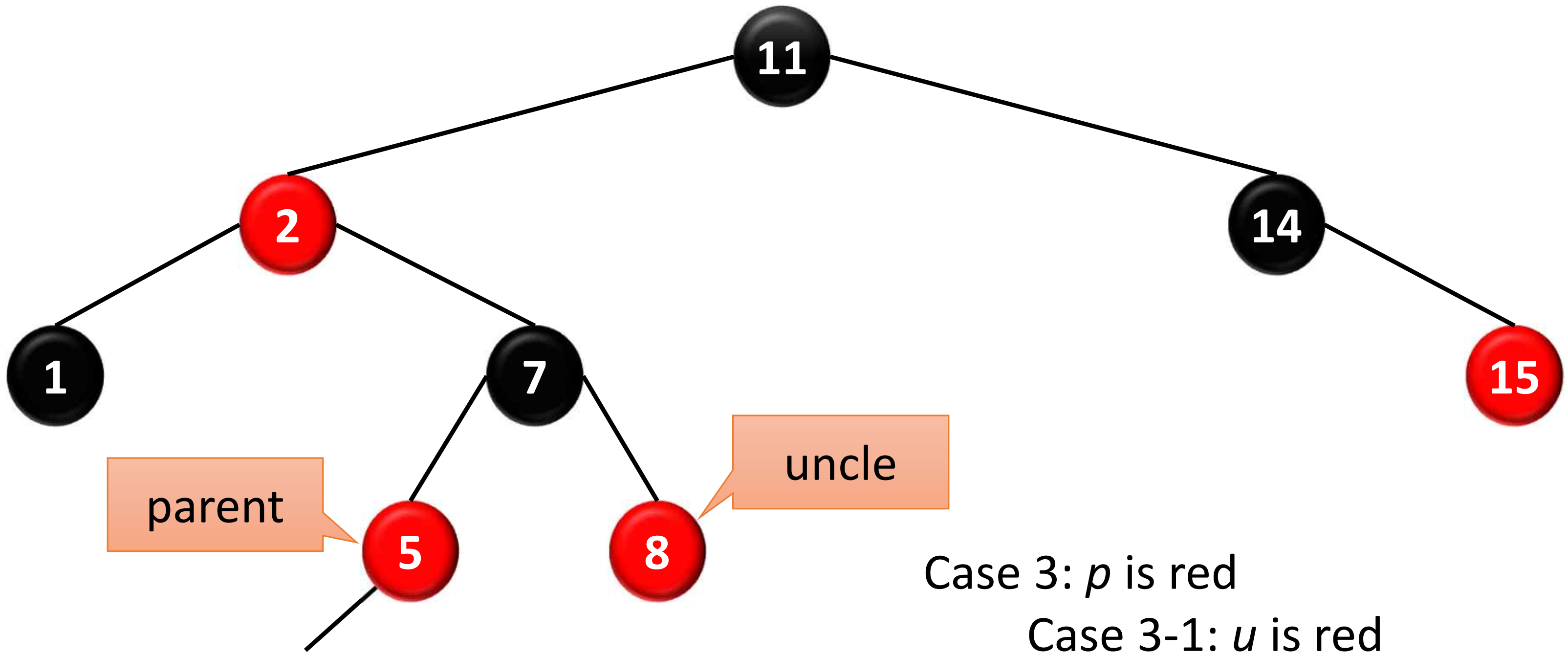
Case 2

- Parent node is black: Insert red node
- Insert 29 and 67



Case 3

- Insert 4



Case 3: *p* is red

Case 3-1: *u* is red

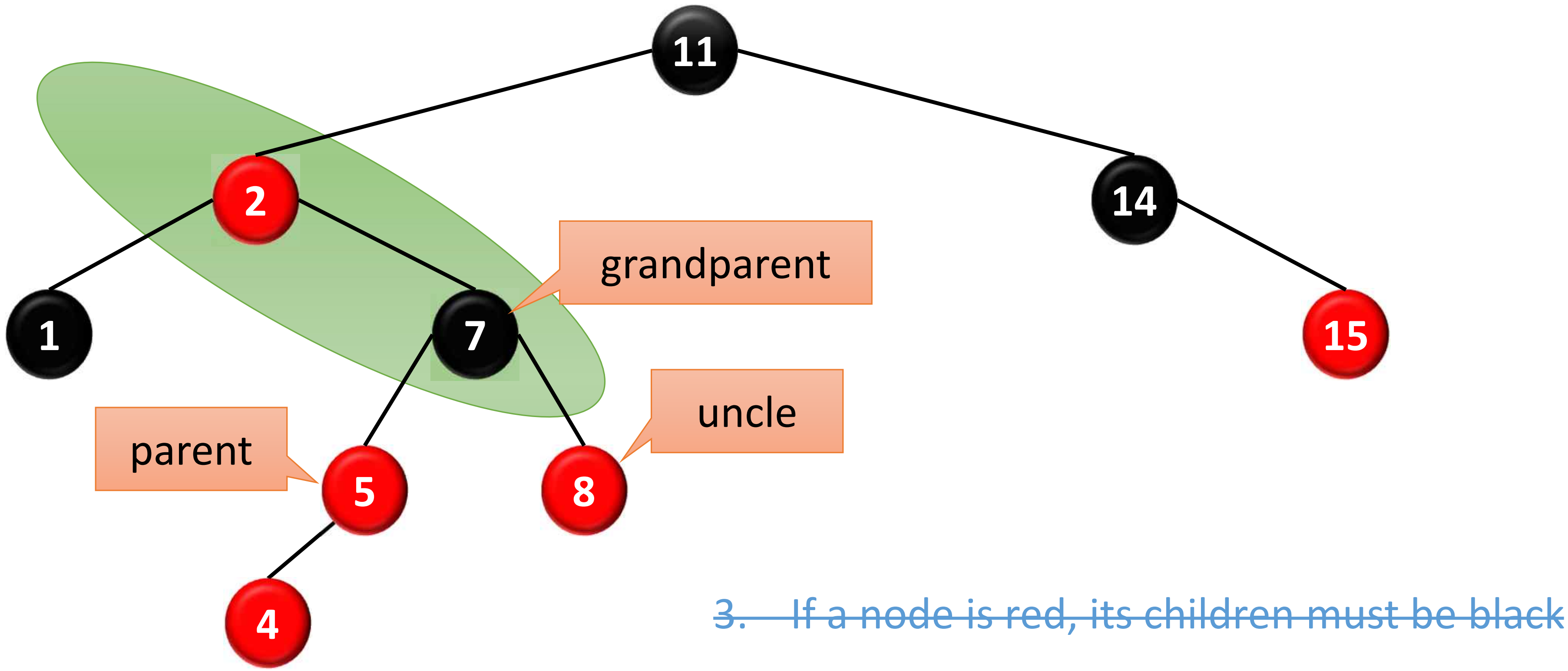
Case 3-2: u is black

Case 3-2-1: x is the right child of p

Case 3-2-2: x is the left child of p

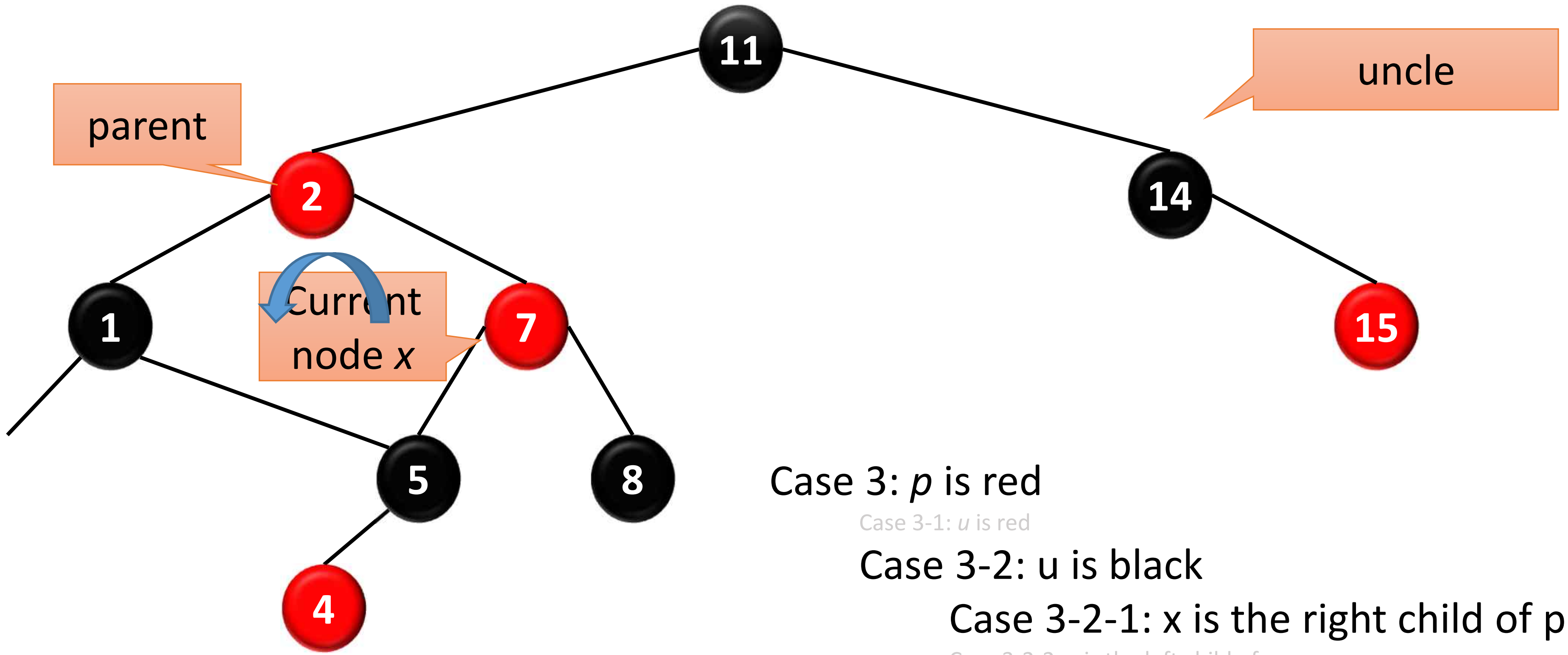
Case 3-1

- Change colors
  - Parent & uncle : black
  - Grandparent: red



Case 3-2-1

- Left-rotation on parent



Case 3: *p* is red

Case 3-1: *u* is red

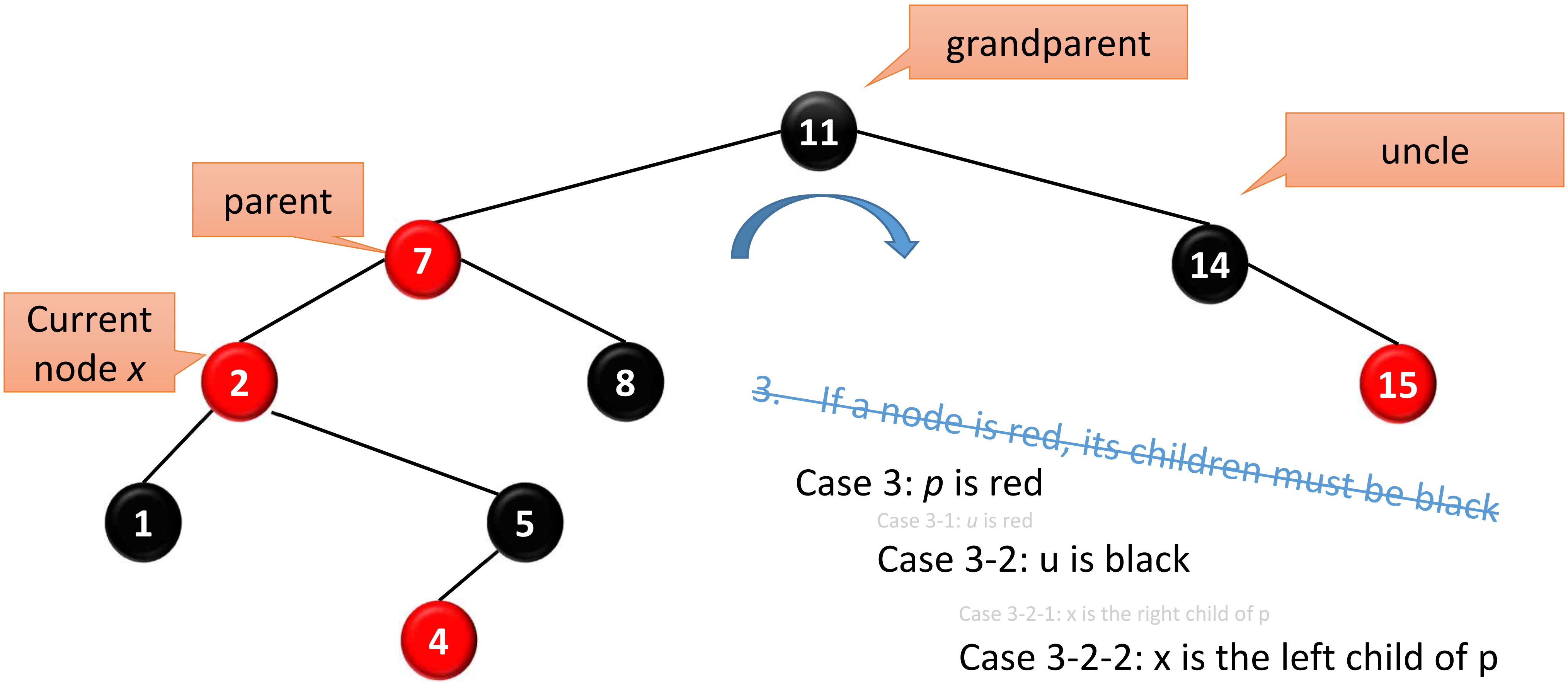
Case 3-2: u is black

Case 3-2-1: x is the right child of p

Case 3-2-2: x is the left child of p

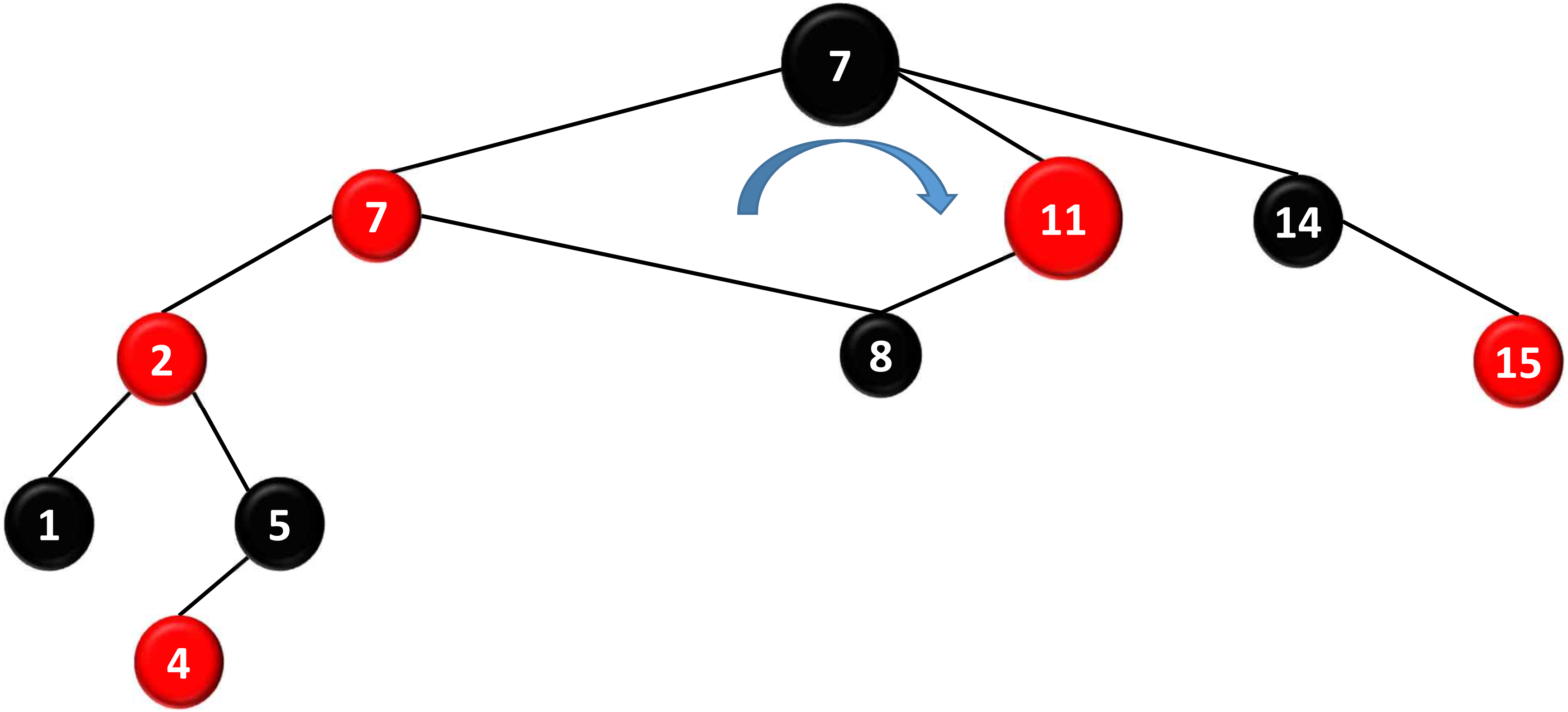
Case 3-2-2

- Right-rotation on grandparent
- Change colors
  - Parent(7) & grandparent(11)



Case 3-2-2

- Right-rotation on grandparent
- Change colors
  - Parent(7) & grandparent(11)



Time complexity

- Insert: $O(\log n)$: maximum height of tree
- Color red: $O(1)$
- Fix violations: $O(\log n)$
  - Const # of:
    - Recolor: O(1)
    - Rotation: O(1)

- Total: $O(\log n)$

Pro and Con of Red-black Trees

- Advantages
  - AVL: relatively easy to program. More balanced. Insert requires only one rotation.
  - Red-Black: Fastest in practice, no traversal back up the tree on insert
- Disadvantages
  - AVL: Repeated rotations are needed on deletion, must traverse back up the tree.
  - Red-Black: Multiple rotates on insertion, delete algorithm difficult to understand and program