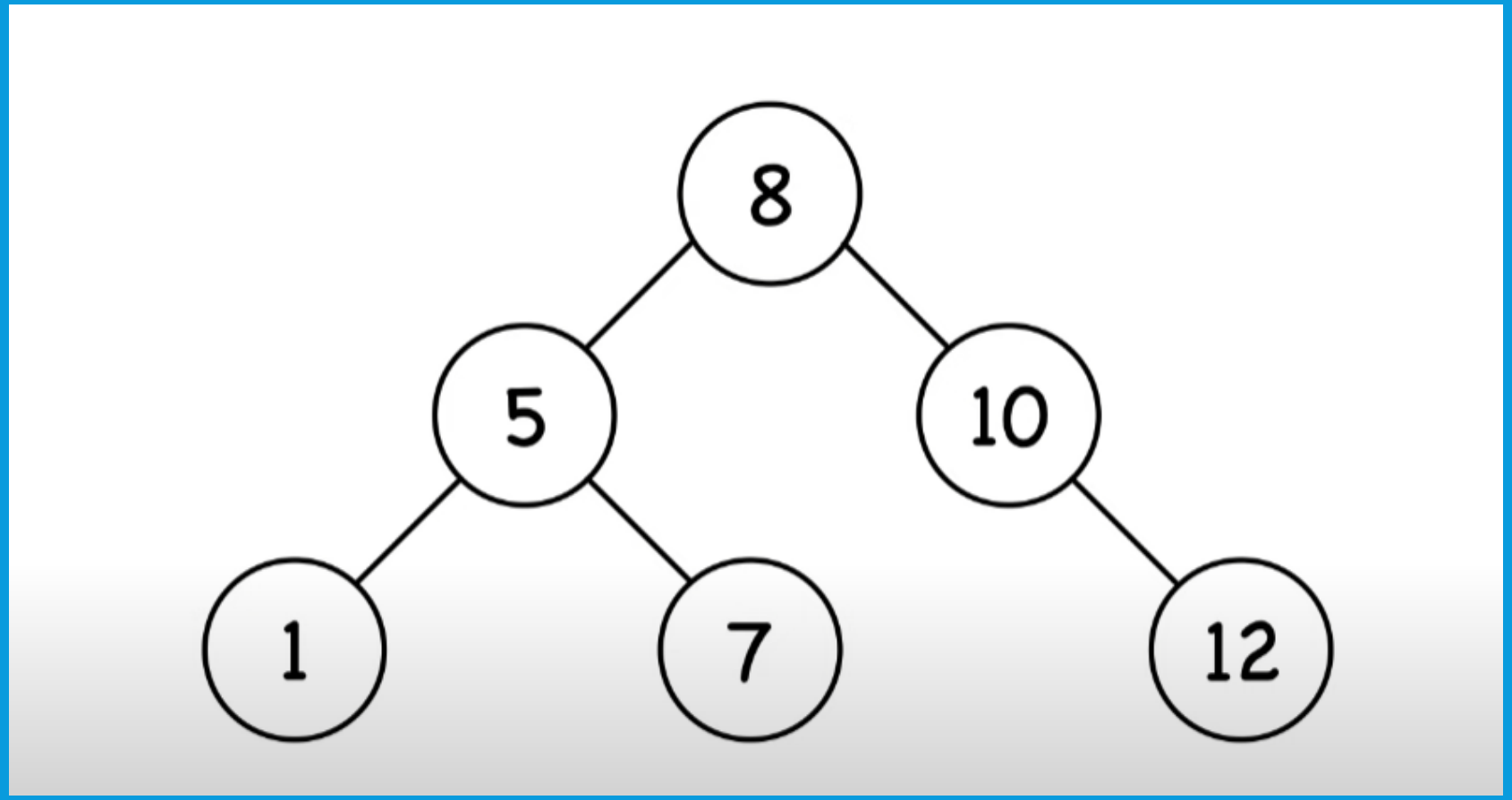
RED BLACK TREE

BINARY SEARCH TREES

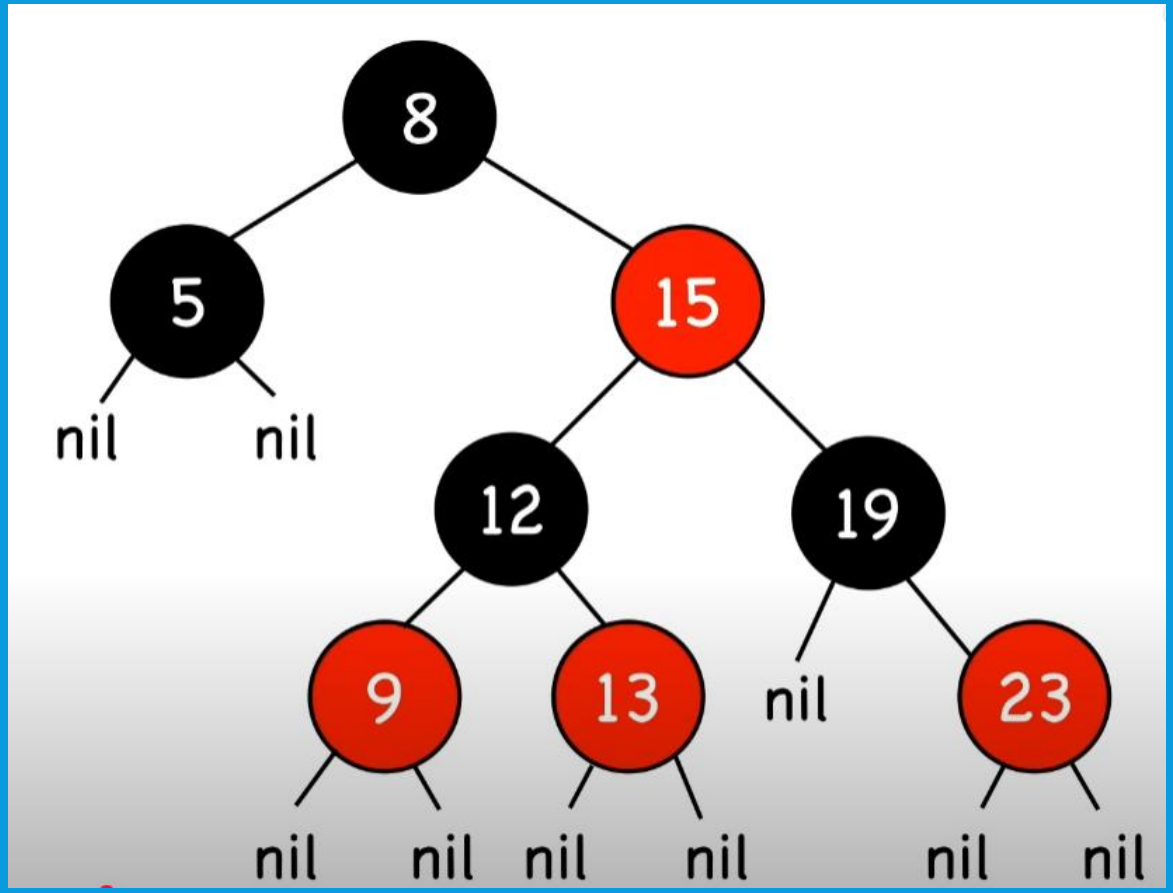
- Ordered, or sorted, binary trees.
- Nodes can have 2 subtrees.
- Items to the left of a given node are smaller.
- Items to the right of a given node are larger.


8
5
10
1
7
12

RED-BLACK TREE

- A node is either red or black.
- The root and leaves (NIL) are black.
- If a node is red, then its children are black.
- All paths from a node to its NIL descendants contain the same number of black nodes.

RED-BLACK TREE



RED-BLACK TREE

- Nodes require one storage bit to keep track of color.
- The longest path (root to farthest NIL) is no more than twice the length of the shortest path (root to nearest NIL).
- Shortest path: all black nodes
- Longest path: alternating red and black

OPERATIONS

- Search
- Insert and Delete require rotation and recoloring

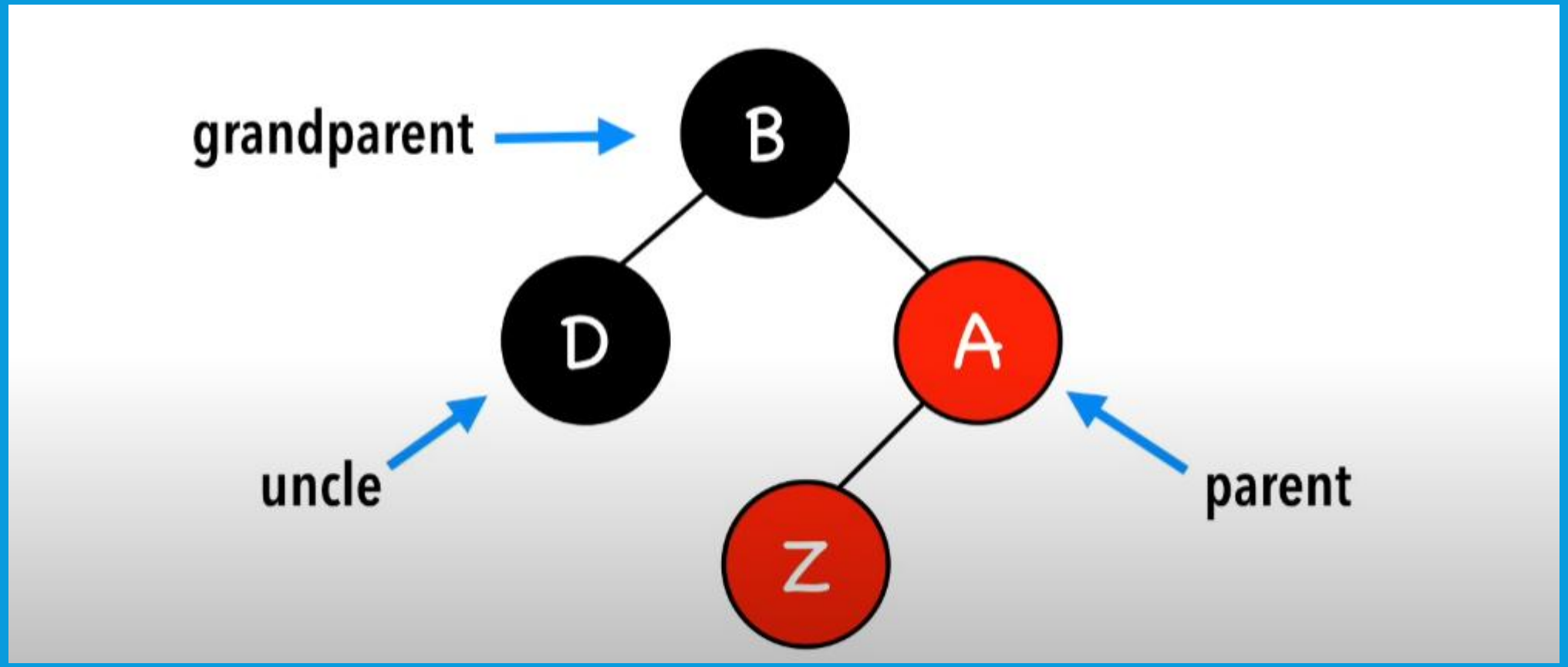
TIME COMPLEXITY, SPACE COMPLEXITY

- Search O(log n)
- Insert O(log n)
- Remove O(log n)
- Space Complexity O(n)

ROTATION

- Alters the structure of a tree by rearranging subtrees
- Goal is to decrease the height of the tree
- Red-black trees- maximum height of O(log n)
- Larger subtrees up, smaller subtrees down
- Does not affect the order of elements

Z NODE'S RELATIONSHIP



INSERTION

- Insert Z and color it red
- Recolor and rotate nodes to fix violation

4 scenarios

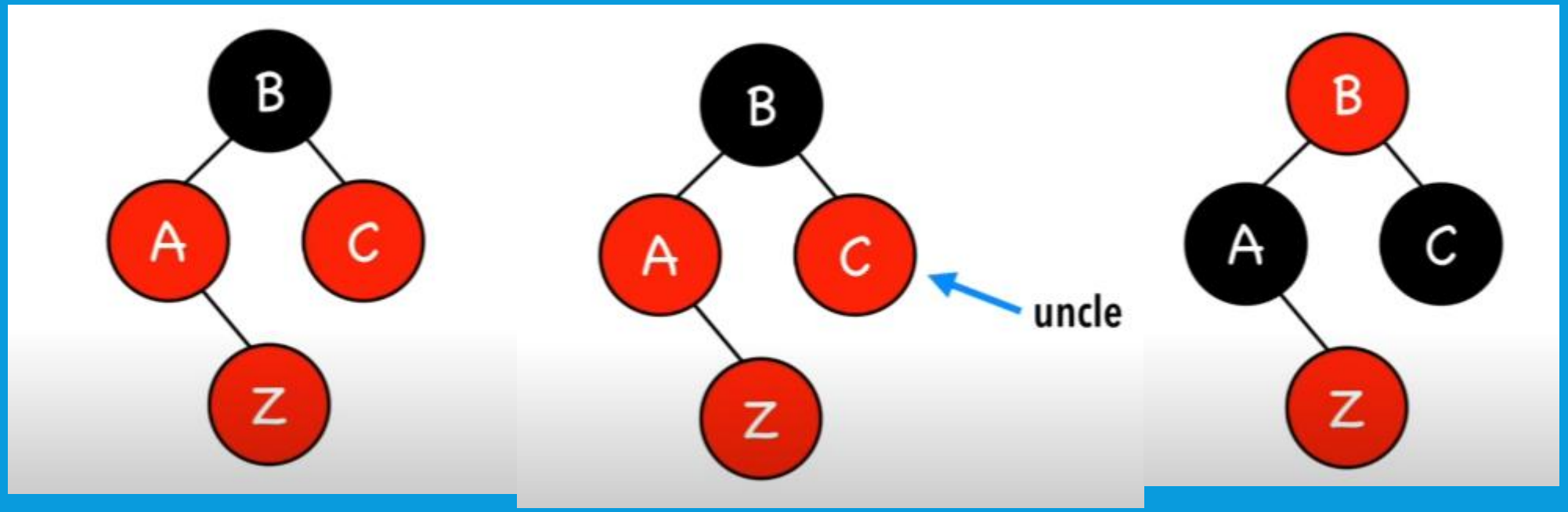
- Z = root
- Z.uncle = red
- Z.uncle = black (triangle)
- Z.uncle = black (line)

CASE 1

- Z = root - color black

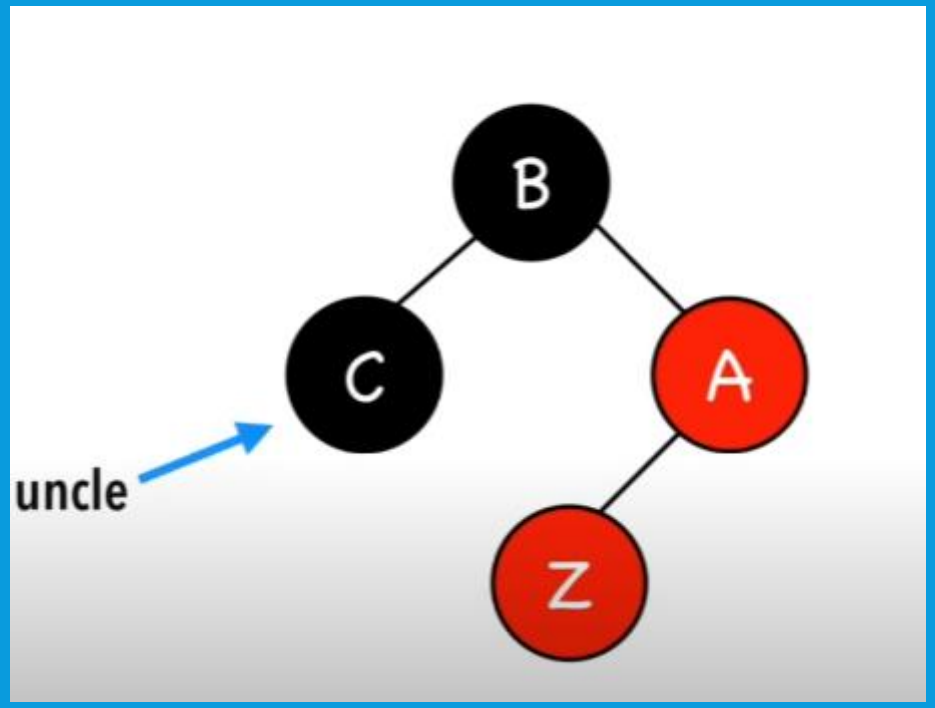
CASE 2

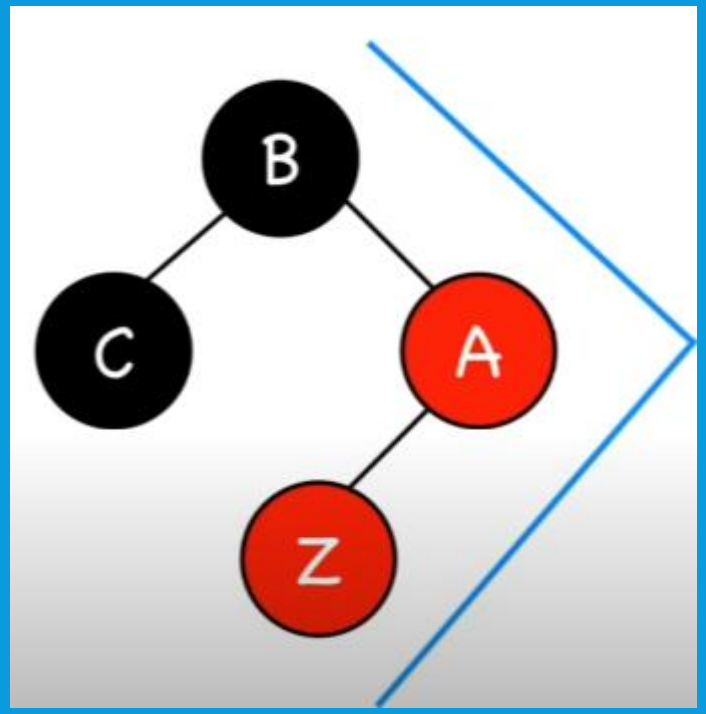
- Z.uncle = red – recolor

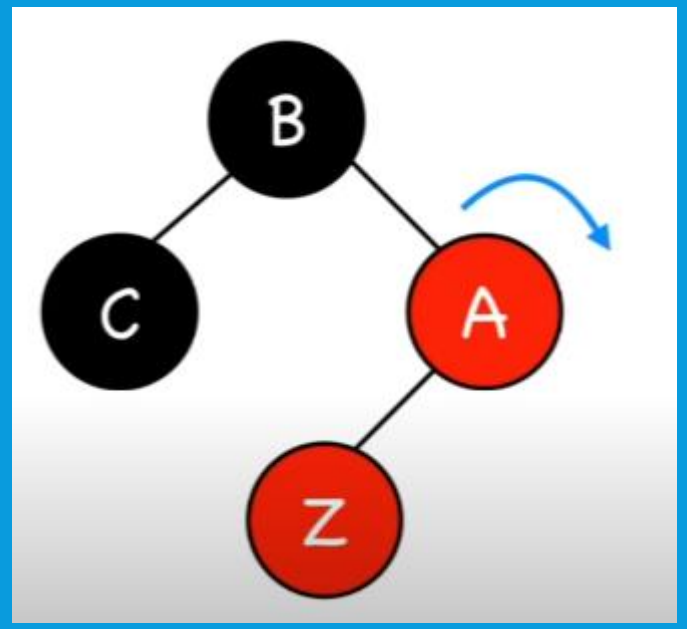


CASE 3

- Z.uncle = black (triangle) –Double rotate Z.parent

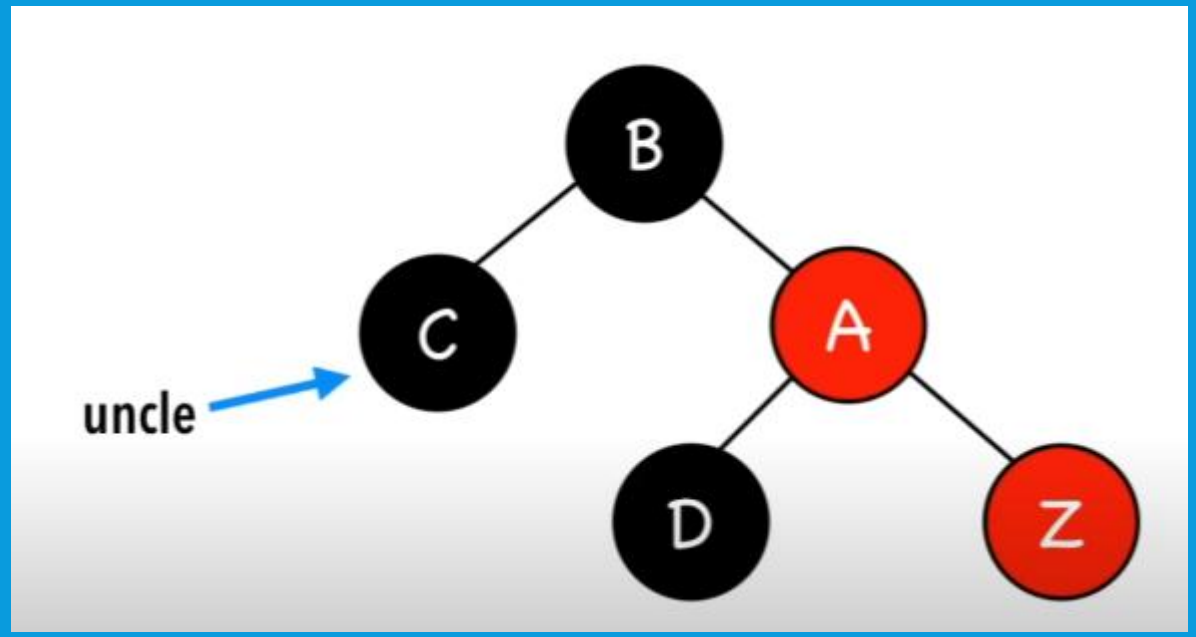


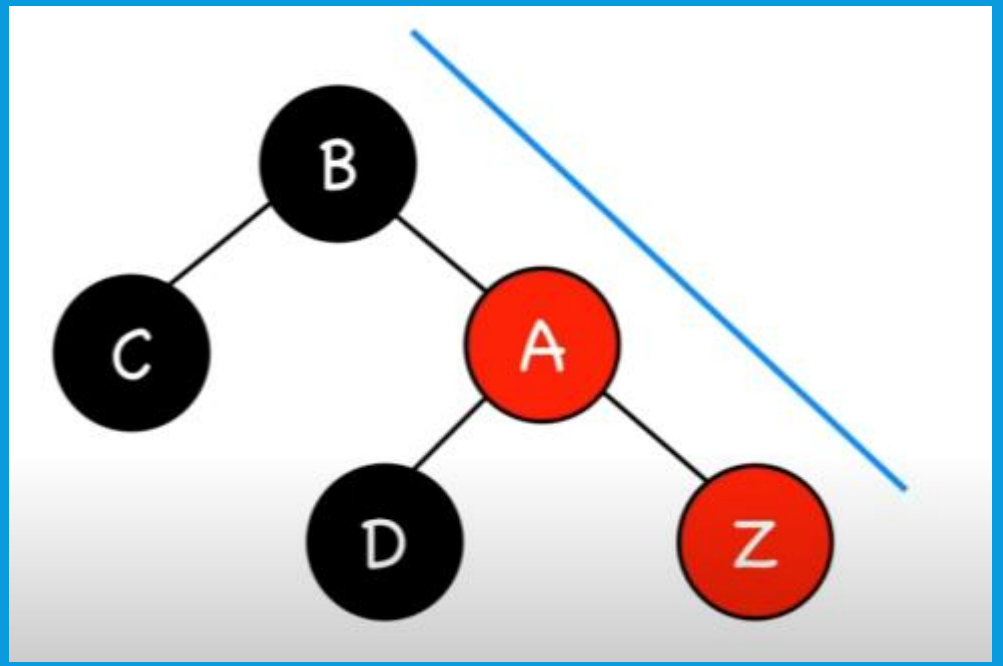




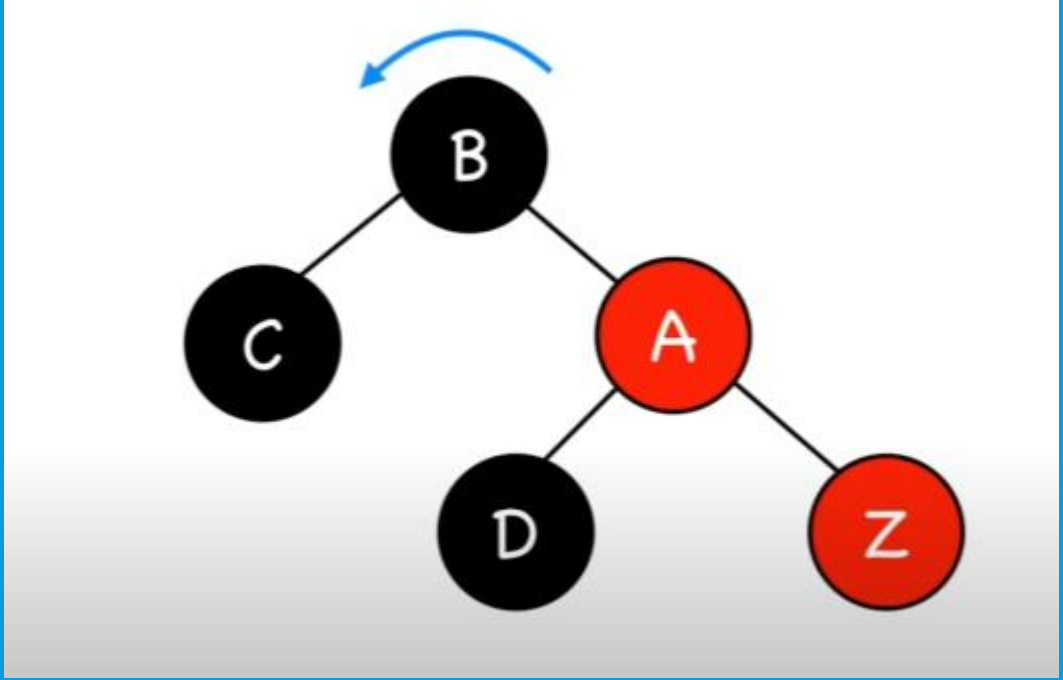
CASE 4

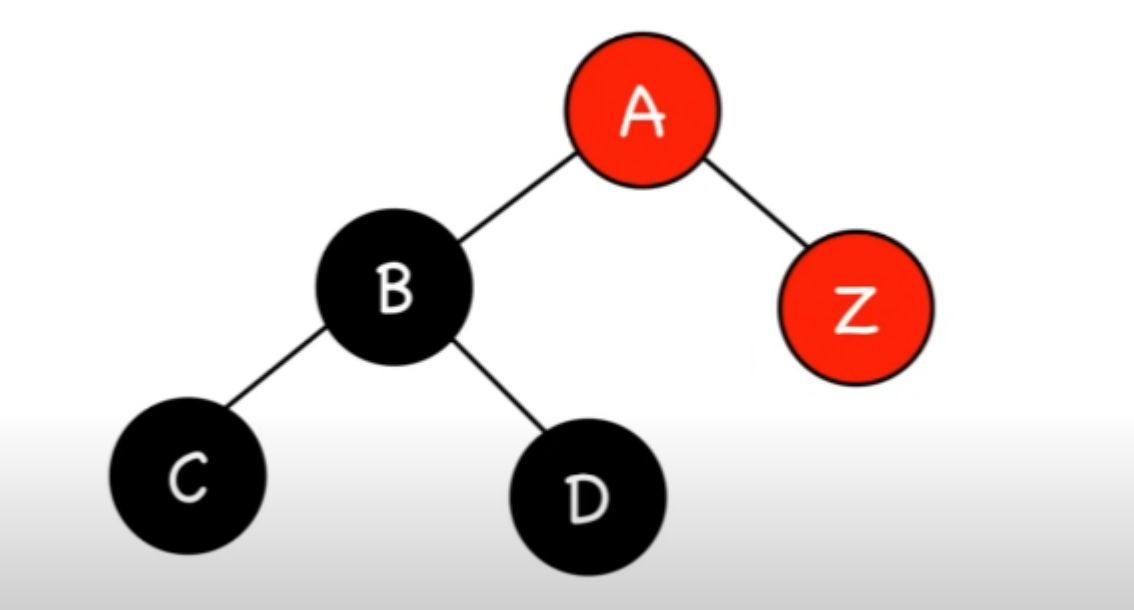
- Z.uncle = black (line) –rotate Z's grandparent and recolor



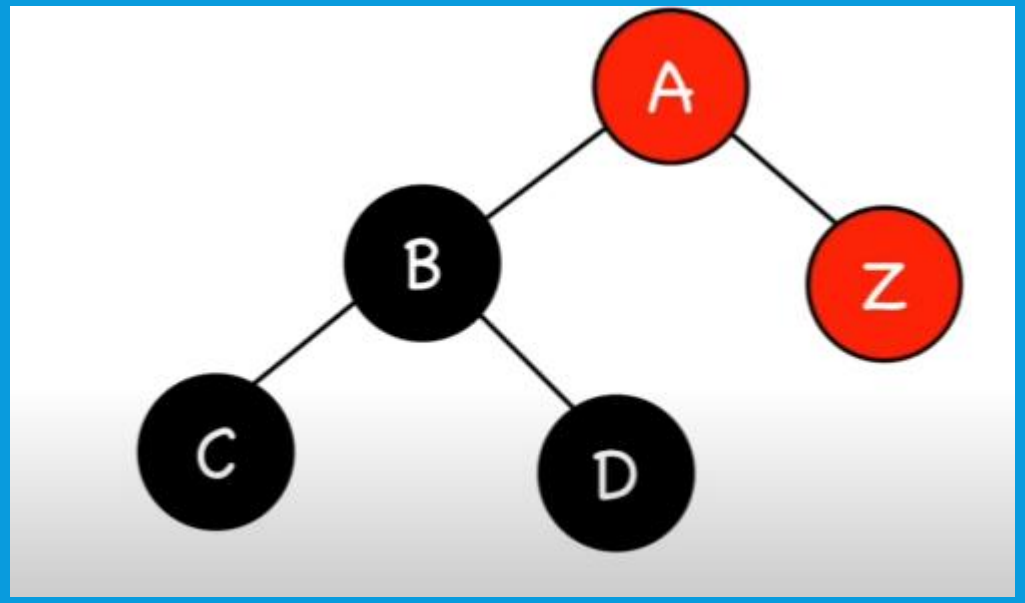


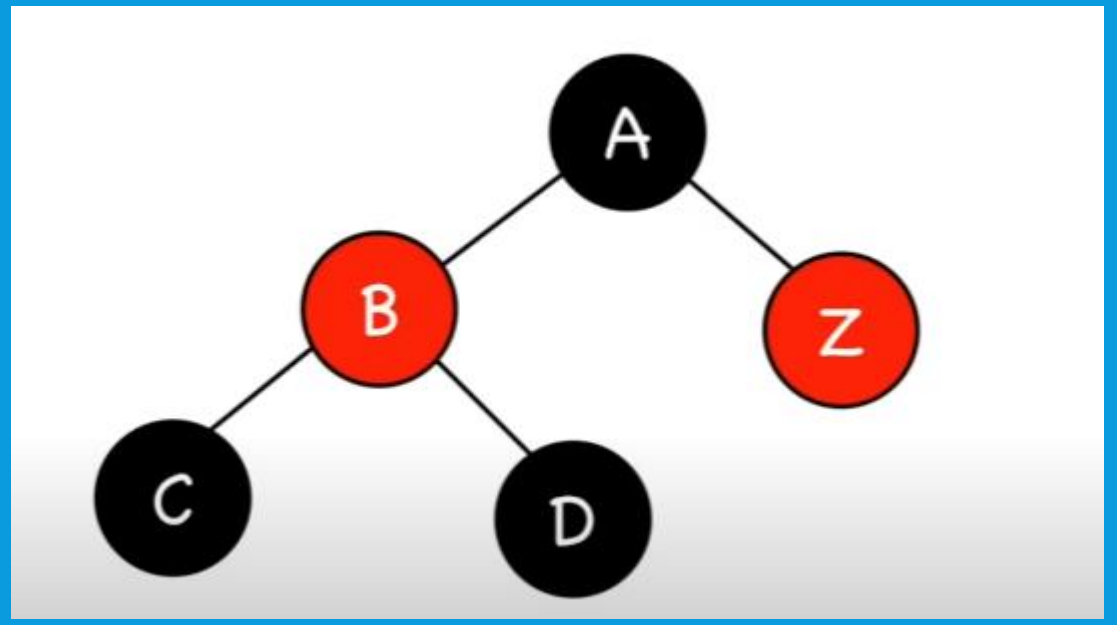
CASE 4 -ROTATION





CASE 4 -RECOLOR





4 SCENARIOS

- Z = root -> color black
- Z.uncle = red -> recolor
- Z.uncle = black (triangle) -> rotate Z.parent
- Z.uncle = black (line) -> rotate Z.grandparent & recolor

SEARCH

- Starts at the root of the tree and compares the search key with the current node's key.
- If the key matches, it returns the value of the current node.
- If the key is smaller, it moves to the left subtree.
- If the key is larger, it moves to the right subtree.
- If it reaches a None node, the key is not present in the tree.

DELETION

- Search for the Node

Find the node containing the key to be deleted.

- Handle Red-Black Properties

If the node or its replacement violates Red-Black Tree properties (e.g., two consecutive red nodes, imbalance), apply rotations and color flips to restore balance.

- Replace or Remove the Node

If the node has two children, replace it with its in-order successor (smallest node in the right subtree).

If the node has one or no children, adjust pointers to bypass the node.

THANK YOU