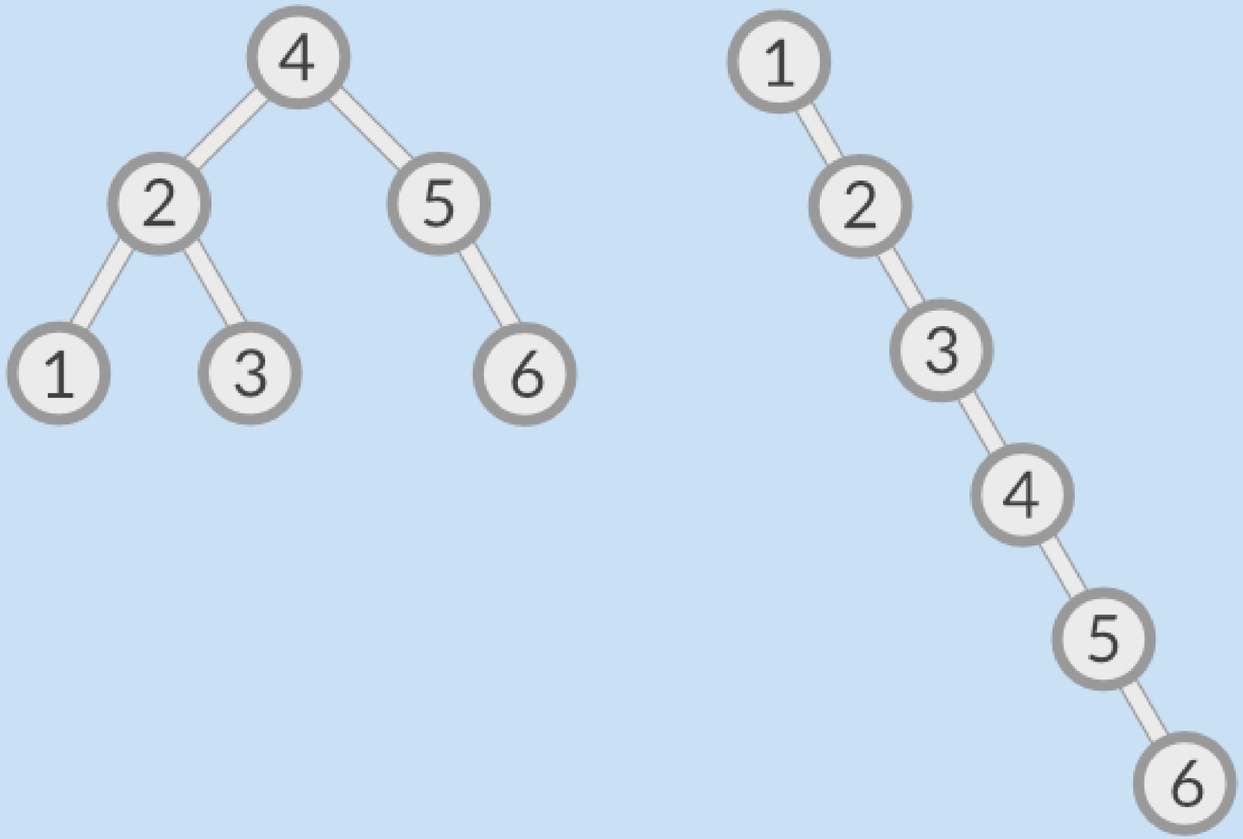
UNDERSTANDING RED-BLACK TREES

THE NEED FOR BALANCE IN TREES



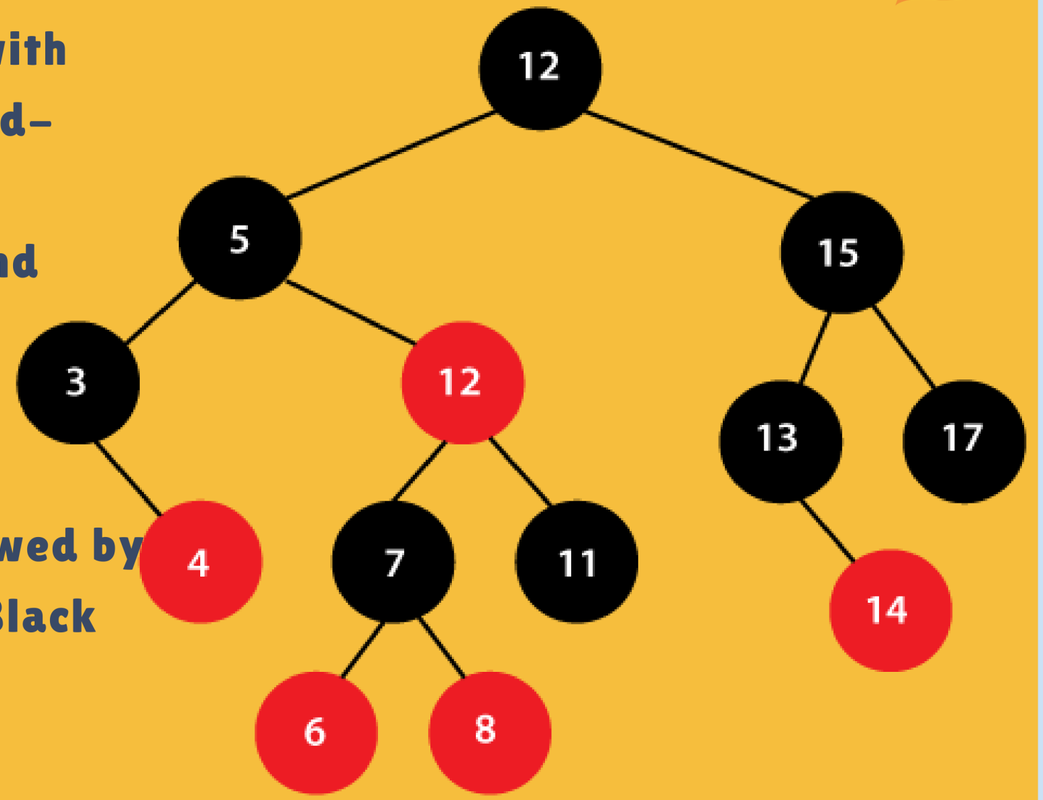
- A balanced BST offers the best search efficiency because the path from the root to any leaf node is of minimal length, which ideally is O(log n).

- An unbalanced tree can degenerate into a linked list structure in the worst case, making the search time complexity linear, or O(n).

RED-BLACK TREE - AN OVERVIEW

- A Red–Black Tree is a type of binary search tree with extra properties that lead to a balanced tree. A Red–Black Tree is self–balancing, which means it automatically rearranges itself after insertions and deletions to maintain balance.

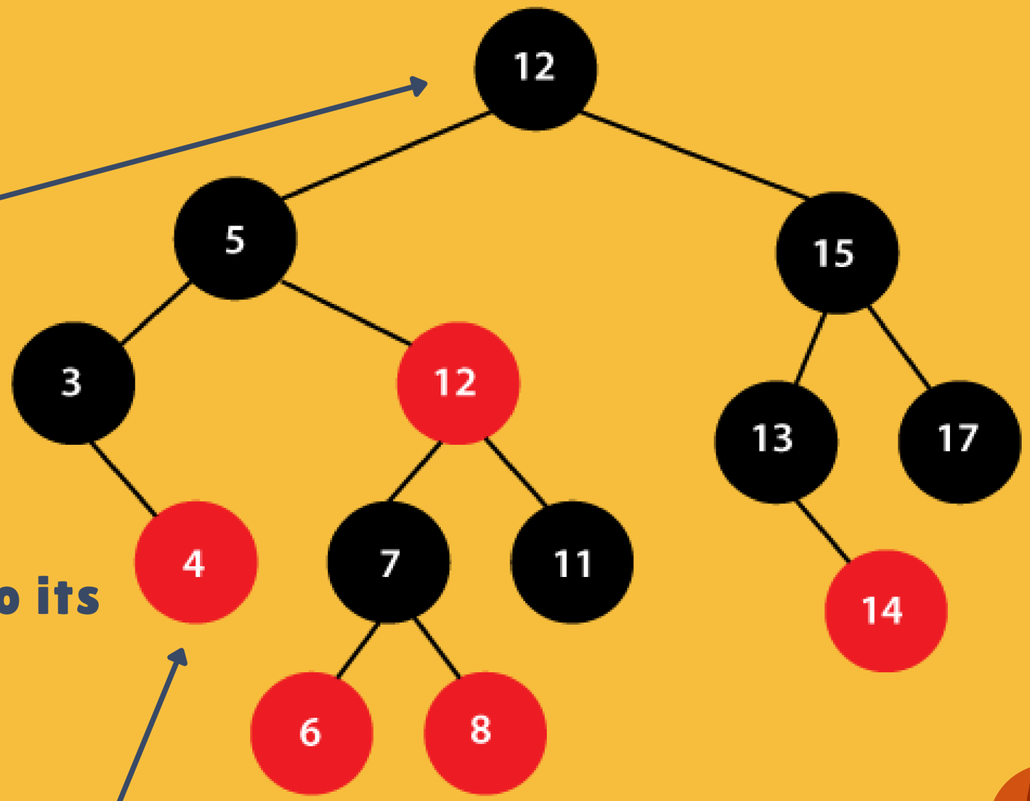
- Rotation Rules: Insertions and deletions are followed by rotations and color changes to maintain the Red–Black properties.



Red Black Tree

RED-BLACK TREE PROPERTIES

- Every node is either red or black.
- All leaves (NIL nodes) are black.
- The root of the tree is always black.
- Red node property: Red nodes cannot have red children.
- Black Height Property: Every path from a node to its descendant NULL nodes has the same number of black nodes.
- New Node Property: Every new node inserted is red.

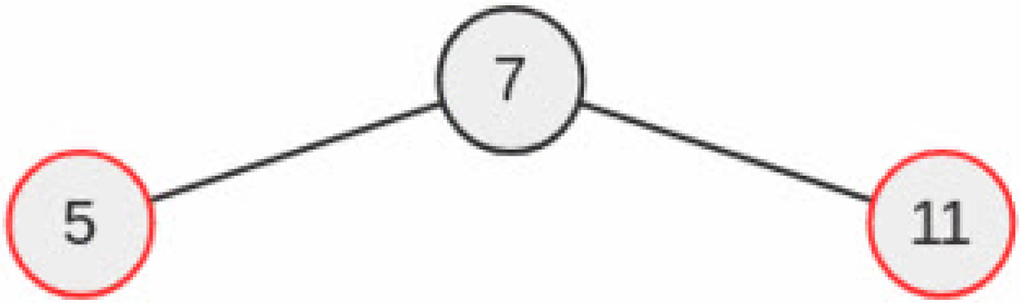


Red Black Tree

INSERTION IN A RED-BLACK TREE

- **Similarity to BST: Insertion begins just like in a regular Binary Search Tree, by placing the new node in the correct position according to its value.**

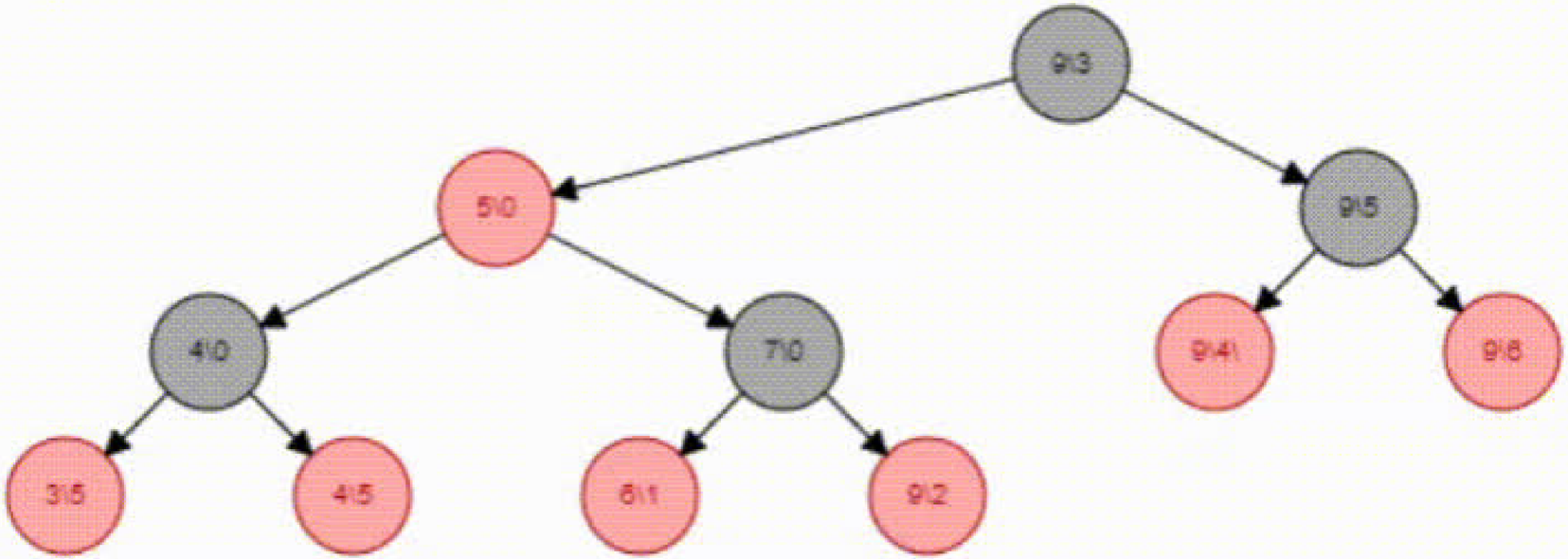
- **Initial Coloring: The new node is always colored red at first, to maintain the black-height property.**



DELETION IN A RED-BLACK TREE

- **Preliminary Note: Deletion in a Red-Black Tree can be complex because it may disrupt the tree's balancing properties.**

- **Initial Step: Just like in a Binary Search Tree, we locate the node to be deleted first.**



ADVANTAGES OF BALANCED TREES

- Efficient search, insertion, and deletion operations
- Improved performance in dynamic, real-world data usage scenarios
- Ensured log(n) height for balanced data distribution

BIG-O ANALYSIS: BALANCED VS. UNBALANCED TREES

- **Balanced Trees: Search operations in a balanced tree, like a Red-Black Tree, have a time complexity of O(log n).**

- **Unbalanced Trees: In contrast, an unbalanced tree can degrade to a linked list in the worst case, with a time complexity of O(n).**

PRACTICAL APPLICATION

- Red–Black Trees are particularly effective in scenarios where we need to maintain a balanced tree structure after frequent insertions and deletions.

- Databases:
- Red–Black Trees provide quick search, insert, and delete operations, making them suitable for database indexes where balance is crucial for performance.

- Associative Arrays:
- Languages like C++ use Red–Black Trees in implementing ordered maps, which require elements to be sorted and allow for efficient retrieval by key.

- Priority Queues:
- While binary heaps are more common, Red–Black Trees can be used for priority queues that require an ordered sequence of elements and support for other associative operations.

CONCLUSION

Red-Black Trees are a testament to the power of balanced tree structures, offering adaptability and efficiency, which makes them suitable for a wide array of applications where balanced search times are crucial.