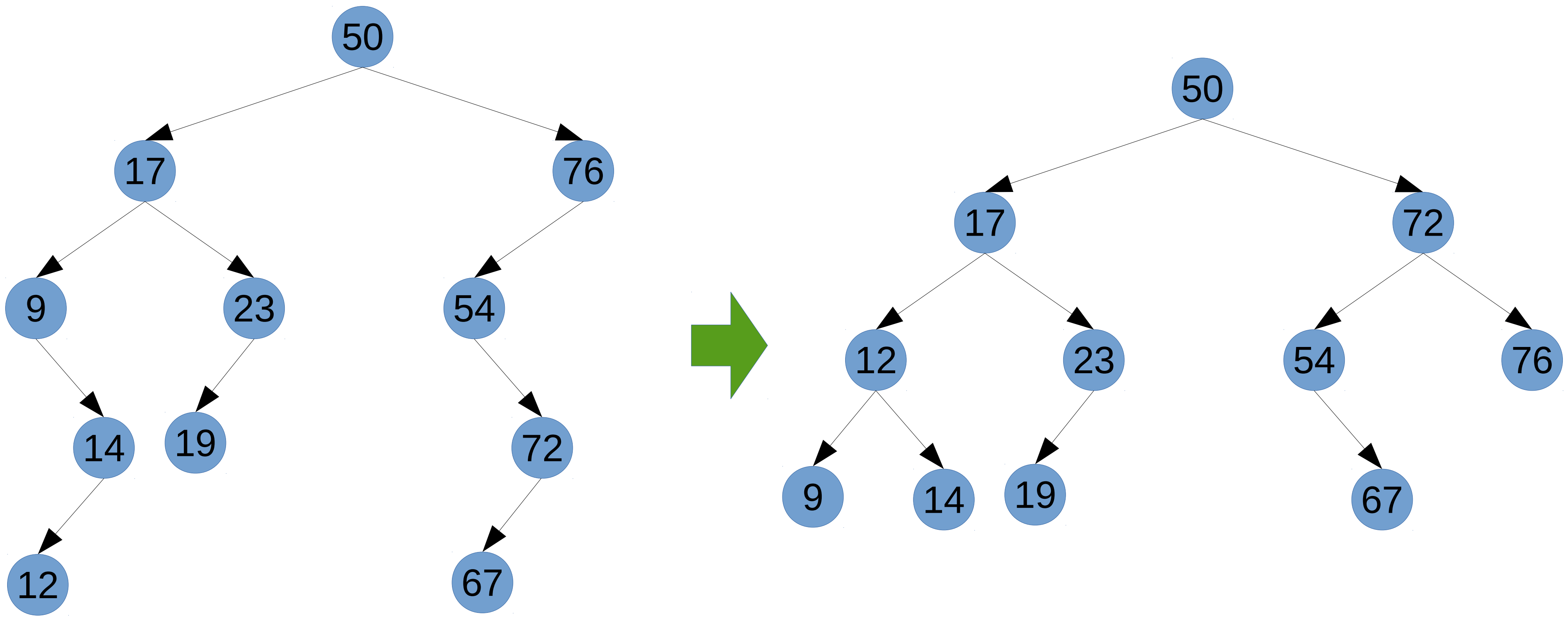
Self Balancing Trees

Angel R. Perez

Balanced Tree



Considerations

- Most operations on a BST take time directly proportional to the height of the tree: Operations: O(**h**) | **h** hight of the tree
  - For a tree degenerated in a linked list (not balanced):
    - Operations: O(**n**) | **n** number of nodes
  - For a balanced tree:
    - Operations: O(**Log$_2$(n)**) | **n** number of nodes

Advantages & Disadvantages

- A self-balanced tree does not degenerate in a list.
  - A self-balanced tree guarantee efficient operations
    - Lookup, Deletion, Insertion: ~O($\mathbf{Log_2(n)}$)

- A self-balanced tree has overhead in some operations
  - Deletion, Insertion (due to rotations and recoloring)
- Harder to implement

Self-Balanced Trees

- **Red-Black**
- **AVL**
- **B**
- **Scapegoat**

Red-Black Trees: Properties

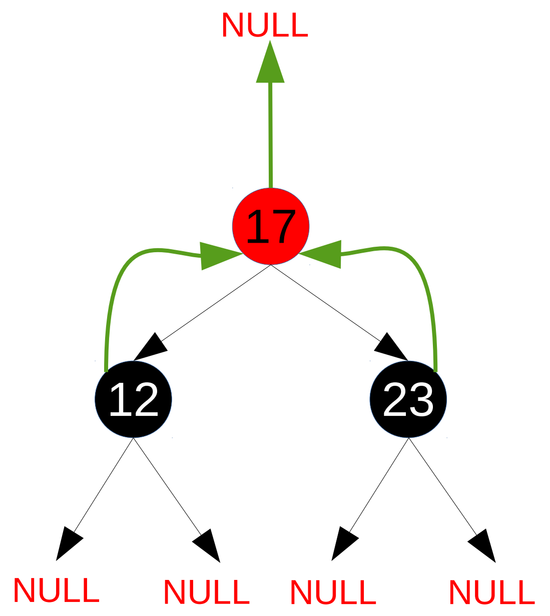
1) A node is either red or black.
2) The root is always black.
3) All leaves(NIL) are black.
4) There are no two adjacent red nodes.

   A red node has two black children.
5) Every path from root to NIL node has same number or black nodes.

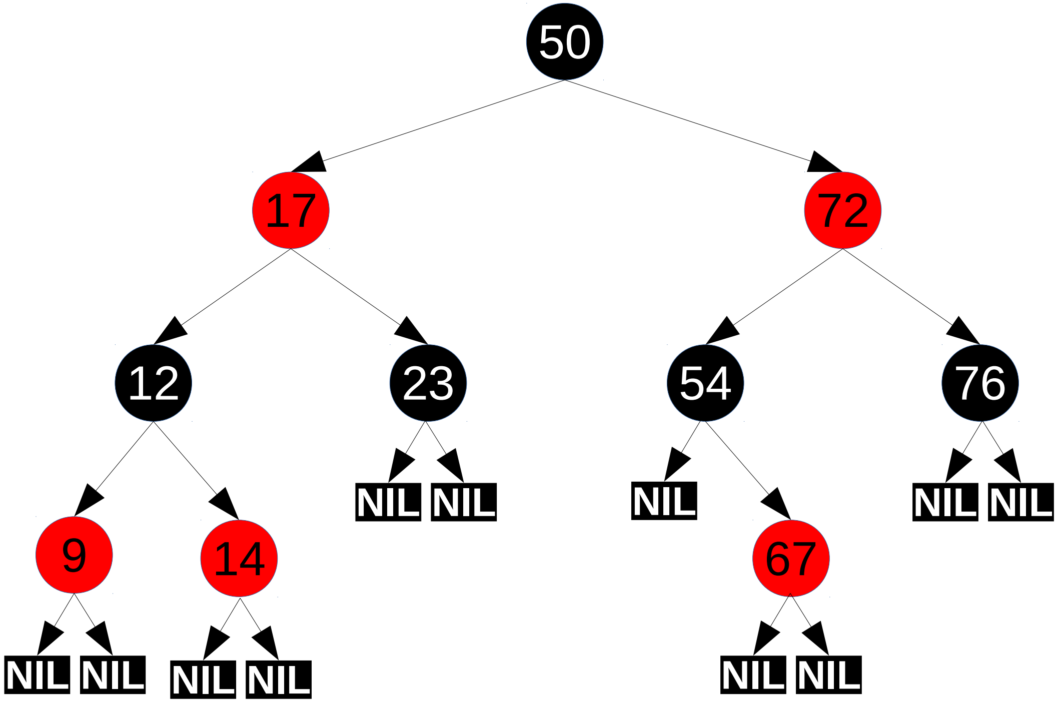
A red black tree is a BST. Lookup in an RBT is just lookup in a BST. The colors don't matter.

Representation

```
typedef struct node {
    int data;
    char color;
    struct node *parent;
    struct node *left;
    struct node *right;
} NODE;
```



Representation

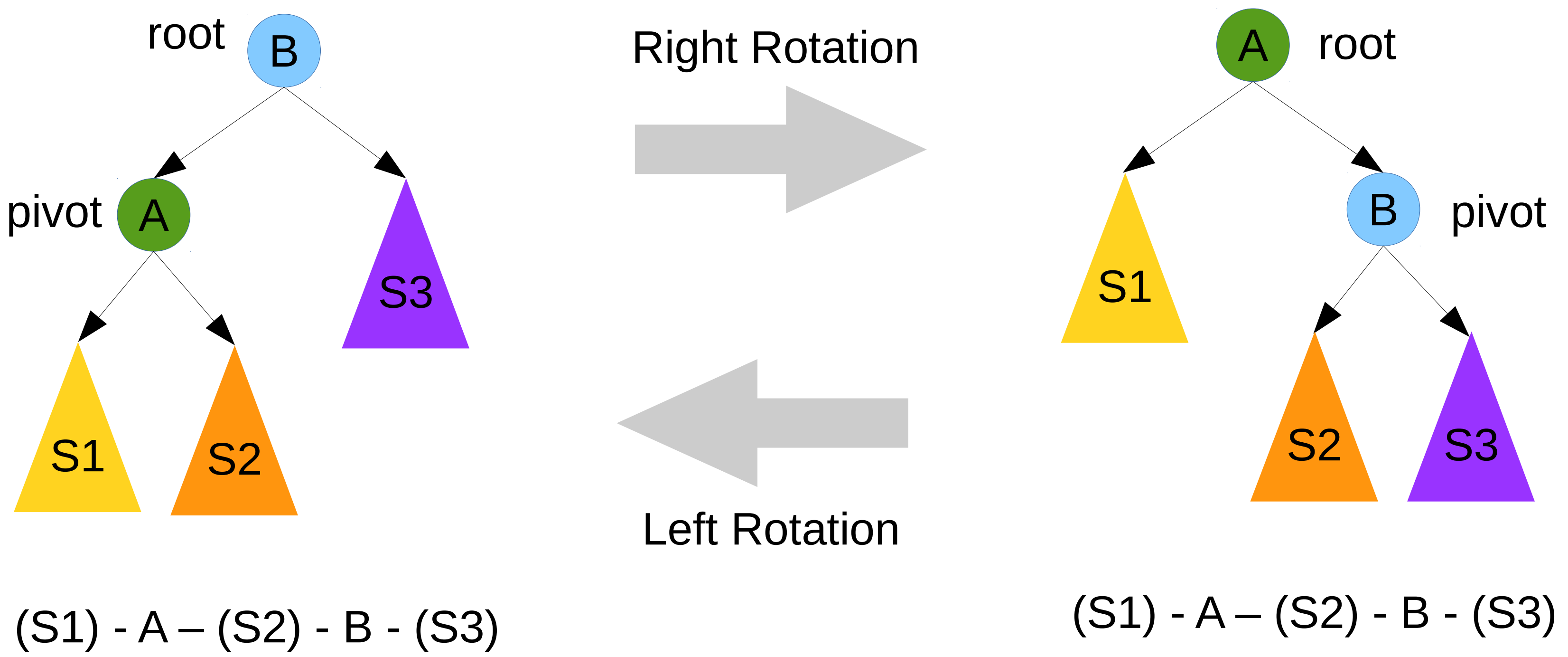


Operations

- **Insertion**
- **Deletion**
- Traversals
  - Pre-Order
  - In-Order
  - Post-Order
  - BFS (breadth first search)
  - DFS (depth first search)
- Lookup
- **Rotations***

Rotations

- Change the tree structure without interfering with the order of the elements.
- Used to change the shape of the tree, particularly by decreasing the hight of the tree.
- Move smaller subtrees down and larger subtrees up.
- The order of the elements is not affected (In-Order invariance)



Insertion

1. Insert as the new node as any BST insertion
2. Set the node color as RED
3. If the parent of the new node is RED, there is a double-RED problem that must be corrected
   - A double RED problem is corrected with zero or more recoloring followed by zero or one restructuring.
4. Color the root node BLACK

Applications

- Priority queues
- Associative arrays(key, value)
- Sets
- Hash tables
- Encoding (enumeration)
- Computational geometry
- Completely Fair Scheduler (Linux)

References

- https://en.wikipedia.org/wiki/Binary_search_tree
- https://en.wikipedia.org/wiki/Self-balancing_binary_search_tree
- https://en.wikipedia.org/wiki/Tree_rotation
- https://en.wikipedia.org/wiki/Red%E2%80%93black_tree
- https://en.wikipedia.org/wiki/AVL_tree
- https://en.wikipedia.org/wiki/B-tree
- http://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/
- http://quiz.geeksforgeeks.org/c-program-red-black-tree-insertion/
- http://cs.lmu.edu/~ray/notes/redblacktrees/
- https://www.cs.usfca.edu/~galles/visualization/Algorithms.html