Red-Black Tree Cheatsheet

Overview

- Red-Black Tree is a self-balancing binary search tree.
- It provides worst-case O(log n) time complexity for search, insert, and delete operations.
- Each node is either red or black.
- The root node is always black.
- If a node is red, its children must be black.
- Every path from a given node to any of its descendant leaf nodes contains the same number of black nodes.

Operations

Insertion

```
def insert_node(root, key):
    # Create a new node
    new_node = Node(key)
    # Insert the new node as a normal BST
    root = bst_insert(root, new_node)

    # Fix the Red-Black Tree properties
    fix_violation(root, new_node)

    # Return the root of the modified tree
    return root
```

Deletion

```
def delete_node(root, key):
    # Find the node to delete
    node = bst_delete(root, key)

    # Fix the Red-Black Tree properties
    fix_violation(root, node.parent)

    # Return the root of the modified tree
    return root
```

Traversal

In-order Traversal

```
def in_order_traversal(node):
    if node:
        in_order_traversal(node.left)
```

```
        print(node.key)
        in_order_traversal(node.right)
```

Time Complexity

- Insertion: O(log n)
- Deletion: O(log n)
- Traversal: O(n)

Resources

- Red-Black Tree Wikipedia
- GeeksforGeeks: Red-Black Tree
- Red-Black Tree Visualization