Programming Assignment 4: Binary Search Trees and Red-Black Trees

CS212 : Design and Analysis of Algorithms

Objective

This assignment deepens your understanding of Binary Search Trees (BST) and Red-Black Trees (RBT).

Part 1: Binary Search Tree (BST)

Tasks

1. **Node Structure (Augmented with Additional Data):**
   - Define a class/function/method `BSTNode` to represent a node in a binary search tree. Each node should contain:
     - A key (integer).
     - Left child, right child and parent references/pointers.
2. **BST Class with Augmentations:**
   - Create a `BinarySearchTree` class with the following methods:
     - `insert(key: int)`: Inserts a key into the BST.
     - `search(key: int) -> bool`: Returns `True` if the key is found in the tree, otherwise `False`.
     - `delete(key: int)`: Deletes a key from the BST.
     - `inorder_traversal() -> List[int]`: Returns a list of keys using in-order traversal.
     - `find_minimum() -> int`: Returns the minimum key in the BST.
     - `find_maximum() -> int`: Returns the maximum key in the BST.
     - **Advanced Operations**:
       - `select(k: int) -> int`: Returns the k-th smallest element in the tree.
       - `rank(key: int) -> int`: Returns the rank of a given key in the tree (i.e., the number of elements smaller than the given key).

* `find_predecessor(key: int) -> int`: Returns the in-order predecessor of a key in the tree.
* `find_successor(key: int) -> int`: Returns the in-order successor of a key in the tree.

Example Usage

```
bst = BinarySearchTree()
bst.insert(10)
bst.insert(20)
bst.insert(5)

print(bst.select(2))  # 10 (2nd smallest element)
print(bst.rank(20))  # 3 (rank of 20)
print(bst.find_predecessor(10))  # 5
print(bst.find_successor(10))  # 20
```

Part 2: Red-Black Tree (RBT)

Tasks

1. **Node Structure (Augmented with Additional Data):**
   - Define a class `RBTNode` to represent a node in a Red-Black Tree. Each node should contain:
     - A key (integer).
     - Left and right child references.
     - A parent reference.
     - A color (`RED` or `BLACK`).
2. **RBT Class with Balancing and Augmentations:**
   - Create a `RedBlackTree` class with the following methods:
     - `insert(key: int)`: Inserts a key into the Red-Black Tree and maintains its balancing properties.
     - `search(key: int) -> bool`: Returns `True` if the key is found in the tree, otherwise `False`.
     - `delete(key: int)`: Deletes a key from the Red-Black Tree and rebalances the tree.
     - `inorder_traversal() -> List[int]`: Returns a list of keys using in-order traversal.
     - `find_minimum() -> int`: Returns the minimum key in the RBT.
     - `find_maximum() -> int`: Returns the maximum key in the RBT.
     - **Advanced Operations**:

* `select(k: int) -> int`: Returns the k-th smallest element in the tree (1-based indexing).
* `rank(key: int) -> int`: Returns the rank of a given key in the tree (i.e., the number of elements smaller than the given key).
* `find_predecessor(key: int) -> int`: Returns the in-order predecessor of a key in the tree.
* `find_successor(key: int) -> int`: Returns the in-order successor of a key in the tree.

Balancing Logic and Augmentations

Ensure that rotations and recoloring are implemented correctly to maintain Red-Black Tree properties during insertion and deletion.

Example Usage

```
rbt = RedBlackTree()
rbt.insert(10)
rbt.insert(20)
rbt.insert(5)

print(rbt.select(2))  # 10 (2nd smallest element)
print(rbt.rank(20))  # 3 (rank of 20)
print(rbt.find_predecessor(10))  # 5
print(rbt.find_successor(10))  # 20
```

Note

- You are expected to work individually on this assignment.
- Plagiarism will not be tolerated. Any suspected plagiarism will result in penalties as per the course policy.