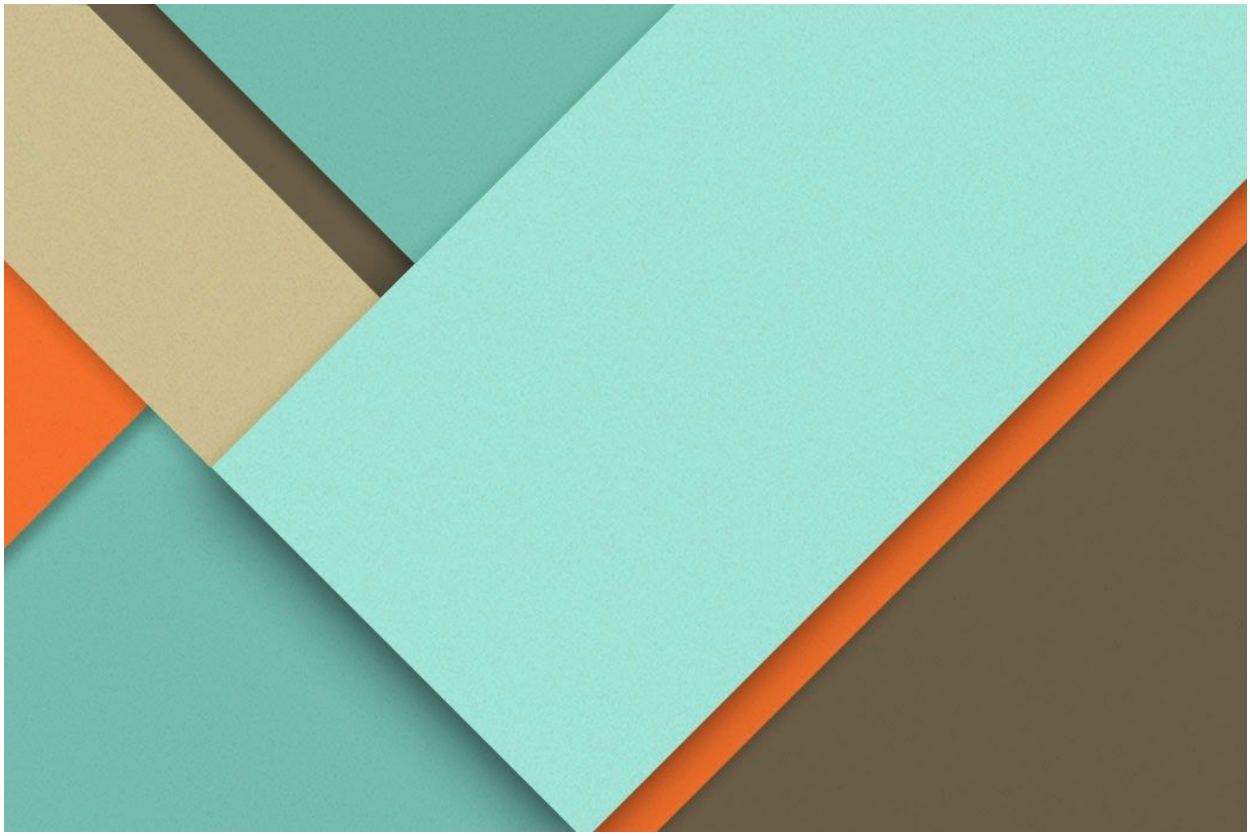


Data Structures 2 - Lab 2

Implementing Red Black Tree & Treemap interface

Afnan Mousa Mabrouk 15

Shimaa kamal 34

Overview

A redblack tree is a kind of self-balancing binary search tree in computer science. Each node of the binary tree has an extra bit interpreted as the color (red or black) of the node. These color bits are used to ensure the tree remains approximately balanced during insertions and deletions.

Tree Map is A Red-Black tree based NavigableMap implementation. The map is sorted according to the natural ordering of its keys.

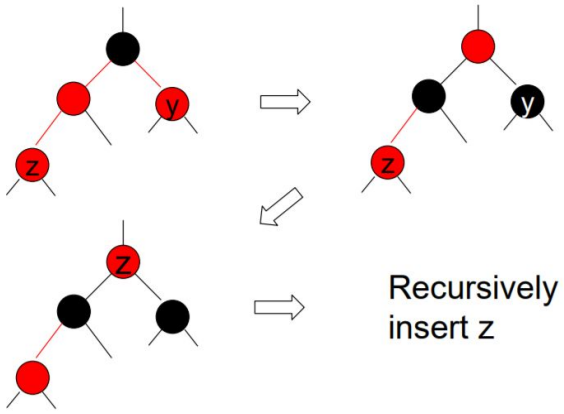
Design Decisions

Insertion and deletion in Red-Black-Tree aren't easy in implementation as they have many cases as the tree must be always balanced and follows some specifications which are:

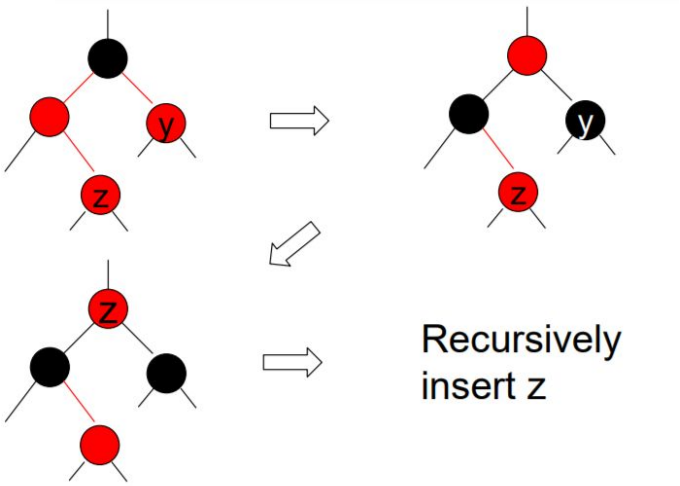
1. Every node is either red or black.
2. The root is black.
3. If a node is red, then both its children are black.
4. For each node, all paths from the node to descendant leaves contain the same number of black nodes.

Insertion cases

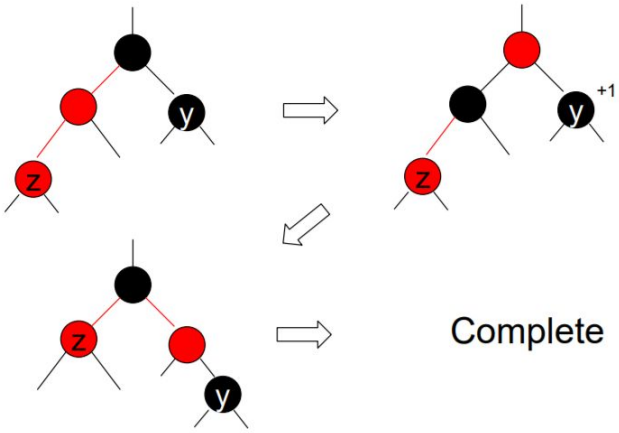
Case 1: y is red and z is a left child



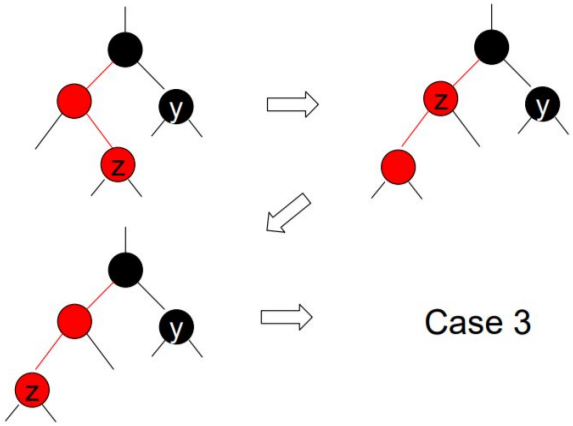
Case 2: y is red and z is a right child



Case 3: y is black and z is a left child



Case 4: y is black and z is a right child



Deletion cases

1. Node x is red
2. Node x is black and its sibling w is red.
3. Node x is black and its sibling w is black and both of w's children are black.
4. Node x is black and its sibling w is black and
   - If x is the left child, w's left child is red and w's right child is black.
   - If x is the right child, w's right child is red and w's left child is black.
5. Node x is black and its sibling w is black and
   - If x is the left child, w's right child is red.
   - If x is the right child, w's left child is red.

Time analysis

Balance is preserved by painting each node of the tree with one of two colors in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

The balancing of the tree is not perfect. Let's see the complexity of its important functions.

Red-Black-tree functions

I. insert(key,value)

Since the tree originally has O(log n) height, there are O(log n) iterations. The tree_insert routine also has **O(log n)** complexity, so overall the insertion routine also has **O(log n)** complexity.

II. search(key)

The balancing of the tree is not perfect, but it is good enough to allow it to guarantee searching in **O(log n)** time, where *n* is the total number of elements in the tree.

III. delete(key)

Delete a kew takes **O(log n)** time but will need special care since it can modify tree as it has many special cases.

IV. rotate(node)

Left and right rotation take only **O(1)** time. Since a constant number of pointers are modified.

Tree-Map

- TreeMap has complexity of **O(log n)** for insertion and lookup.
- TreeMap does not allow null keys but allows multiple null values.
- TreeMap maintains order. It stores keys in sorted and ascending order.