Date: Sep 18, 2025

Properties

Distinguishing characteristic of red black trees are the five properties that are preserved throughout the lifetime, called the red-black properties.

1. Each node must be coloured either red or black.
2. The colour of the root node is always black.
3. All leaf nodes are black.
4. Both the children of a red node must be black.
5. Each path down from the root to a leaf node contains the same number of black nodes.

These constraints enforce a critical property of red-black tree – the longest part from the root node to any leaf node is no more than twice as long as the shortest path from the root to any other leaf in that tree.

The five properties together guarantee that the total height of the red-black tree is no more than half the height of the tree which is enough to ensure that the height of the tree is $O(\log n)$.

Insertion

We shall denote the new node to be inserted as N. The parent of N as P and the grandparent of N as G and the uncle of N as U.

While restoring the red-black properties, there are five cases to be considered. In each special case, the tree is locally fixed relative to some red node.

Initially this is the red node inserted in the red-black tree. So each local change is performed either to make the red parent of the current node black or to make the root of the tree black.

I. The current node is the root of the tree.

   In this case, we just change the colour of the current node to black.

II. When parent P of the current node is black.

   In this case, no alteration is required to preserve the red-black property.

III. If both the parent P and uncle U of the new node N is red, then the grandparent must be black.

When a new node of colour red is inserted, then the red-black property is violated as a red parent cannot have red children. In order to locally restore the red-black property, we colour P and U with black and G with red.

Now N has a black parent and any path through the parent and the uncle must pass through the grandparent will have the same number of black nodes on these paths is unchanged.

However, G may now violate property 2, i.e. root is always black or property 4, i.e. both children of every red node is black. To fix this problem, the entire procedure is recursively performed on G from Case I.

```
void insertCase3(Node n) {
      Node u = n.uncle();
      if (u != NULL && !u.isBlack) {
            n.parent.isBlack = true;
            u.isBlack = true;
            Node g = n.grandfather();
            g.isBlack = flase;
            insertCase1(g);
      }
      else insertCase4(n);
}
```

IV. When parent P is red, uncle U is black and P is the left child of the grandparent G and N is the right child of P. In other words, G, P and N are not in a straight line. To fix this problem, a left rotation is done so that N now becomes the parent of P and then proceed to case 5.

```
void insertCase4(Node n) {
      Node g = n.grandparent();
      if (n == n.parent.right && n.parent == g.left) {
            n.parent.rotateLeft();
            n = n.left;
      } else if (n == n.parent.left && n.parent == g.right) {
            n.parent.rotateRight();
            n = n.right;
      }
      insertCase5(n);
}
```

V. When the parent P is red and the uncle U is black. Also when G, P and N are in a straight line. Here P and N are both red, violating the red-black

property, then push up P and push down G and colour P black and G red. This ensures no red node has a red child without changing the number of black nodes from the root to the leaf.

```
void insertCase5(Node n) {
      Node g = n.grandparent();
      n.parent.isBlack = true;
      g.isBlack = false;
      if (n == n.parent.left) g.rotateRight();
      else g.rotateLeft();
}
```