Date: Sep 24, 2025

Red black tree – deletion

- We start deletion from a red black tree in the same way as we do in case of a BST, i.e. when we delete a node with two non-leaf children, we find either the left-most element of the right subtree or the right-most element of the left subtree and move its value to the node being deleted, then delete the node from which we had copied the value.
- When the node to be deleted is red, then no problem arises.
- When the node to be deleted is black, then N is made the child of P and coloured black. Here P is coloured white as the colour of P is unknown.
- Problem arises when the node to be deleted, X, is black and its child N is also black.

There are 6 special cases to be considered:

1. **The current node is the root**

   Here nothing needs to be done.

   ```
   void deleteCase1(Node n) {
           if (n.parent == NULL) return;
           else deleteCase2(n);
   }
   ```

2. **The sibling S of the current node is red**

   Here interchange the colours of S and P, and rotate left at P and proceed to case 3.

   ```
   void deleteCase2(Node n) {
           Node S = n.sibling();
           if (!S.isBlack) {
                   n.parent.isBlack = false;
                   S.isBlack = true;
                   if (n == n.parent.left) n.parent.rotateLeft();
                   else n.parent.rotateRight();
           }
           deleteCase3(n);
   }
   ```

3. **Both parents and siblings are black**

   If this is the case, then colour the sibling red so all paths through S will have one less black node. Therefore, all the paths through P now have one fewer black node than the paths that do not pass through P. To fix this problem, rebalancing of P from case 1 should be started.

```
void deleteCase3(Node n) {
      Node S = n.sibling();
      if (n.parent.isBlack && S.isBlack && S.left.isBlack &&
S.right.isBlack) {
            S.isBlack = false;
            deleteCase1(n.parent);
      } else {
            deleteCase4(n);
      }
}
```

4. **Both N and S are black, S's children are black but P is red**

   Here we interchange the colours of S and P.

```
void deleteCase4(Node n) {
      Node S = n.sibling();
      if (!n.parent.isBlack && S.isBlack && S.left.isBlack &&
S.right.isBlack) {
            S.isBlack = false;
            n.parent.isBlack = true;
      } else {
            deleteCase5(n);
      }
}
```

5. **N is left-child of P and S is black, S's left-child is red and right-child is black**

   Here, a right rotation is performed at S and interchange the colours of S and its new parent. Now all the paths still have equal number of black nodes but N has a black sibling whose right-child is red so we go to case 6.

```
void deleteCase5(Node n) {
      Node S = n.sibling();
      if (n == n.parent.left && S.isBlack && !S.left.isBlack &&
S.right.isBlack) {
            S.isBlack = false;
            n.parent.isBlack = true;
            S.rotateRight();
```

```
        } else if (n == n.parent.right && S.isBlack && !S.right.isBlack &&
S.left.isBlack {
                S.isBlack = false;
                S.right.isBlack = true;
                S.rotateLeft();
        }
        deleteCase6(n);
}
```

6. **When current node's sibling is black, S's right-child is red and N is the left-child of its parent P**

   Here left rotation is performed at P, then the colours of P and S are interchanged and S's right child is coloured black.

```
void deleteCase6(Node n) {
        Node S = n.sibling();
        S.isBlack = n.parent.isBlack;
        n.parent.isBlack = true;
        if (n == n.parent.left) {
                S.right.isBlack = true;
                n.parent.rotateLeft();
        } else {
                S.left.isBlack = true;
                n.parent.rotateRight();
        }
}
```