Intro to Algorithm
Report

109550198

Environment (Os, compiler version, IDE)

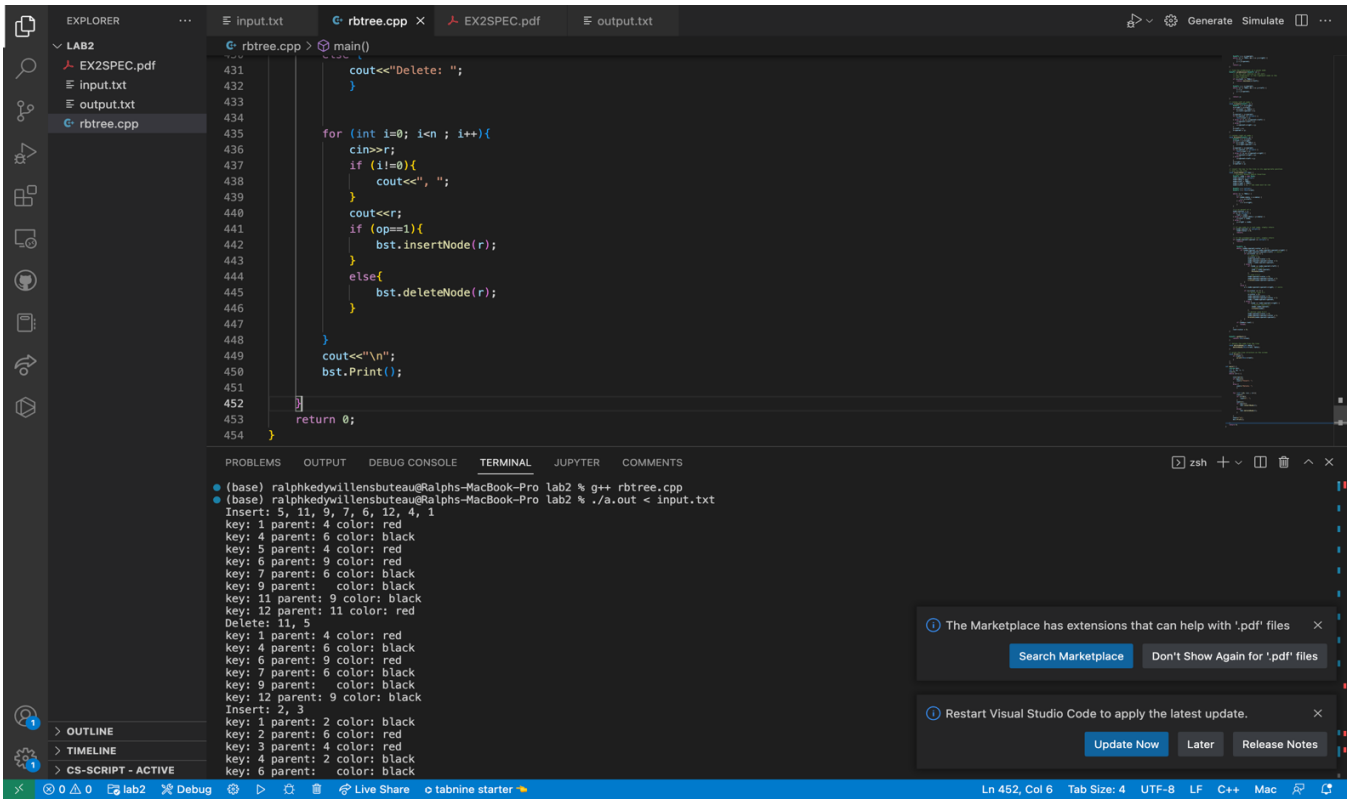
To implement this program, which is a red-black tree, including the following two operations:

- Insert a node.
- Delete a node.
  The OS used for this program is MacOS, the compiler version is apple clang version 14.0.0, and the IDE is visual studio code for mac.

How to run your program

To run the program, I used the terminal of vs code because I was already in the file directory and typed *g++ rbtree.cpp* After I clicked enter and typed *./a.out < input.txt* because the program uses standard input and output, I pressed enter. The terminal shows me the result

Here is the screenshot:



Result:

Method or solutions **:**

- *Operations:*

  Except for insertion and deletion, all operations are identical to those in a standard binary search tree. A red-black tree's operations all take O (log N) time to complete.

- *Insertion:*

  The pseudocode of insertion:

```
RB-INSERT (T, k)
    BST-INSERT (T, k) //normal BST insertion
    while k.parent.color == RED
       if k.parent == k.parent.parent.right
          u = k.parent.parent.left //uncle
          if u.color == RED
             u.color = BLACK
             k.parent.color = BLACK
             k.parent.parent.color = RED
             k = k.parent.parent
          else if k == k.parent.left
                k = k.parent
                LEFT-ROTATE(T, k)
             k.parent.color = BLACK
             k.parent.parent.color = RED
             RIGHT-ROTATE(T, k.parent.parent)
       else (same as then clause with "left" and "right" exchanged)
    T.root.color = BLACK
```

- *Deletion:*

  The pseudocode of deletion:

```
RB-DELETE (T, x)
    BST-DELETE (T, x)
    while x ≠ T.root and x.color == BLACK
       if x == x.parent.left
          s = x.parent.right
          if s.color == RED
             s.color = BLACK // case 3.1
             x.parent.color = RED // case 3.1
             LEFT-ROTATE (T, x.parent) // case 3.1
```

```
        s = x.parent.right // case 3.1
      if s.left.color == BLACK and s.right.color == BLACK
        s.color = RED // case 3.2
        x = x.parent //case 3.2
      else if s.right.color == BLACK
            s.left.color = BLACK // case 3.3
            s.color = RED //case 3.3
            RIGHT-ROTATE (T, s) // case 3.3
            s = x.parent.right // case 3.3
        s.color = x.parent.right // case 3.4
        x.parent.color = BLACK // case 3.4
        s.right.color = BLACK // case 3.4
        LEFT-ROTATE (T, x.parent) // case 3.4
        x = T.root
    else (same as then close with "right" and "left" exchanged)
  x.color = BLACK
```

Anything you want to share:

References:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (n.d.). Introduction to algorithms (3rd ed.). The MIT Press.

Red Black Tree Deletion. https://www.youtube.com/watch?v=CTvfzU_uNKE