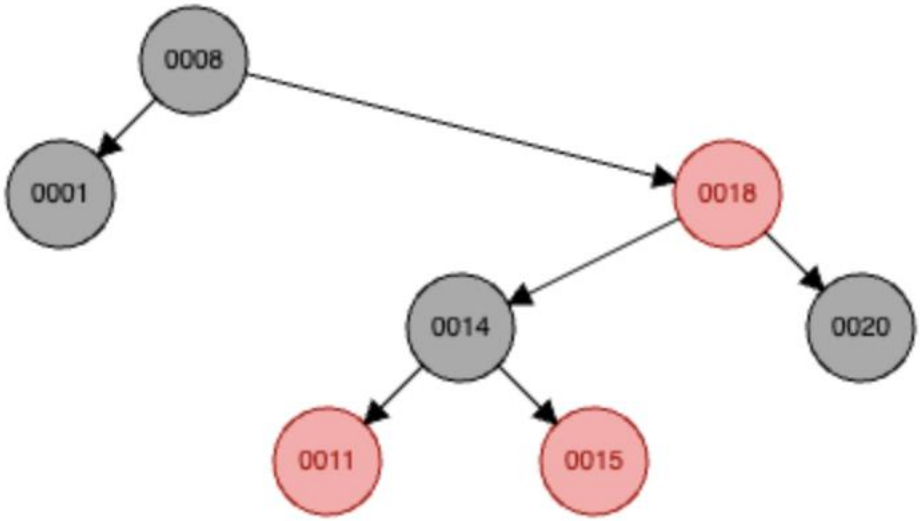
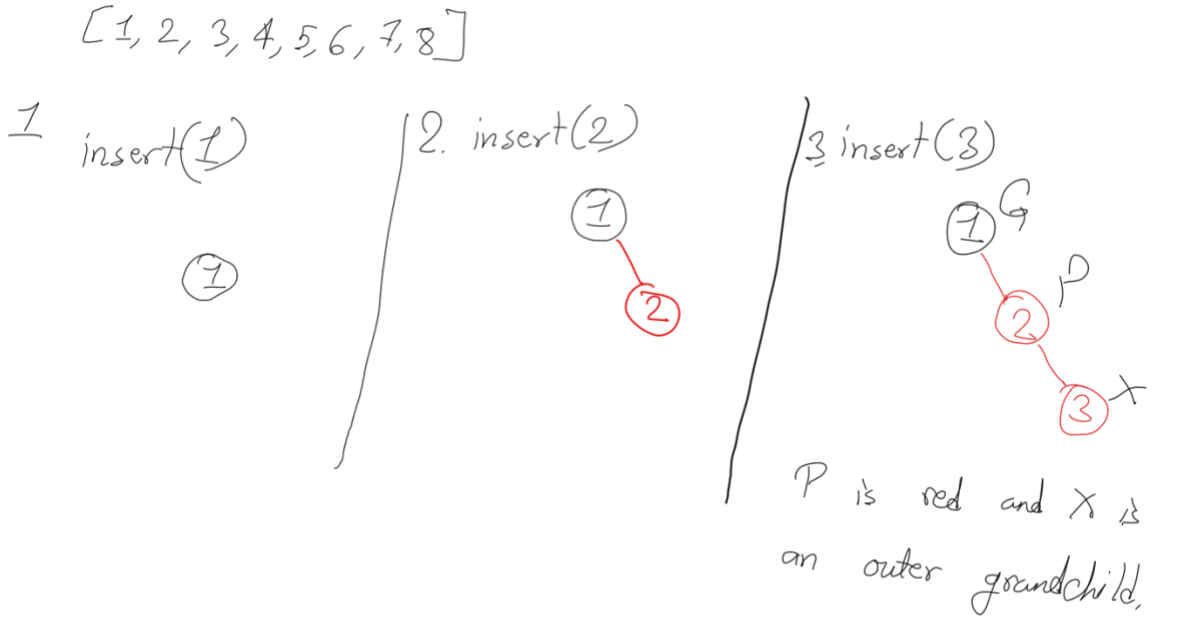
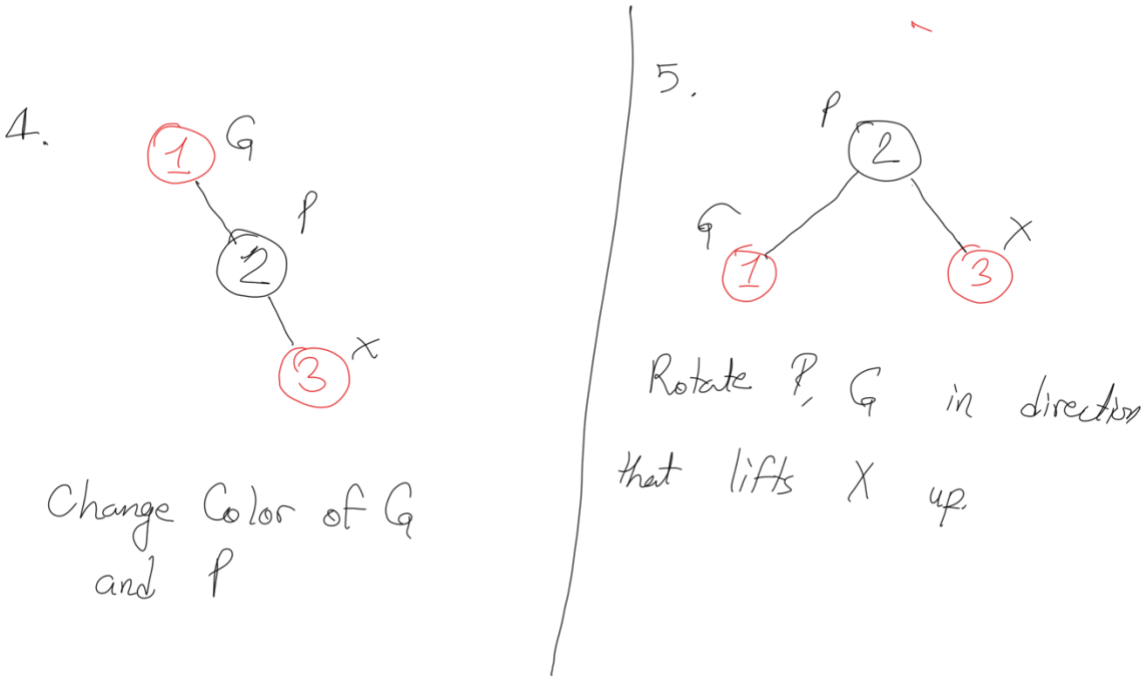
Answer of 1



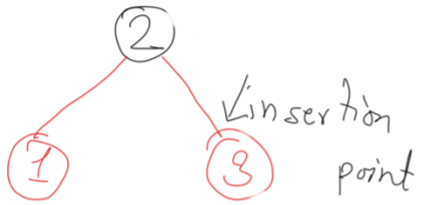
Answer of 2

a. Insert [1,2,3,4,5,6,7,8]



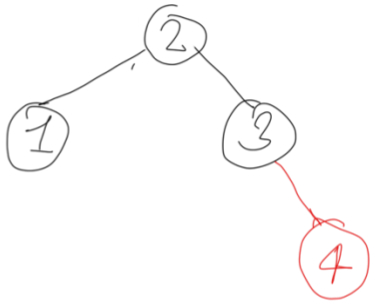


6. insert(4)



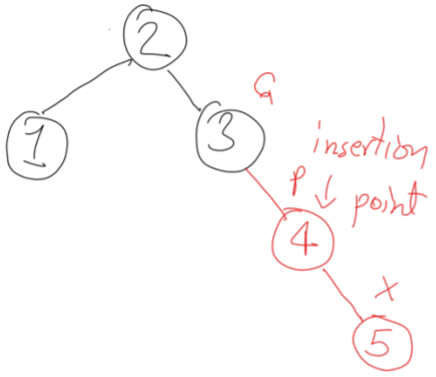
black node with 2 red nodes, Flip color & insert.

7.



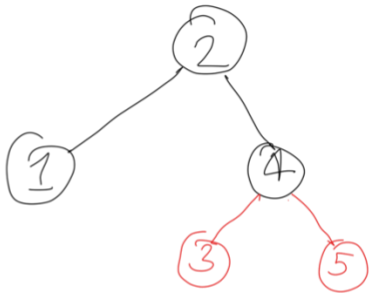
2 is root node, so color is not flipped

8. insert(5)

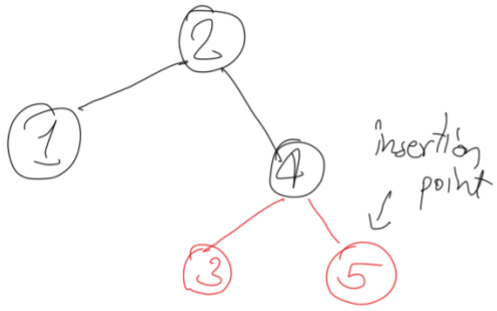


Outer grand child violation. Change color of G, P and rotate P, G to lift X.

9.

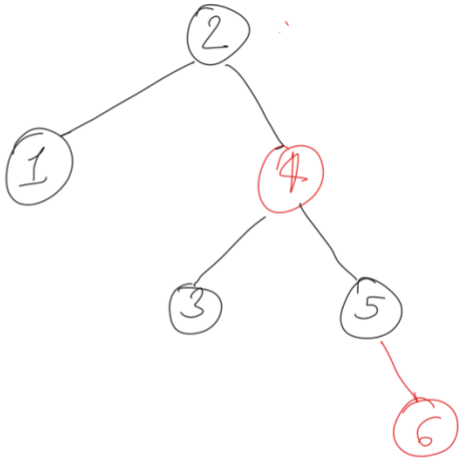


10. insert(6)

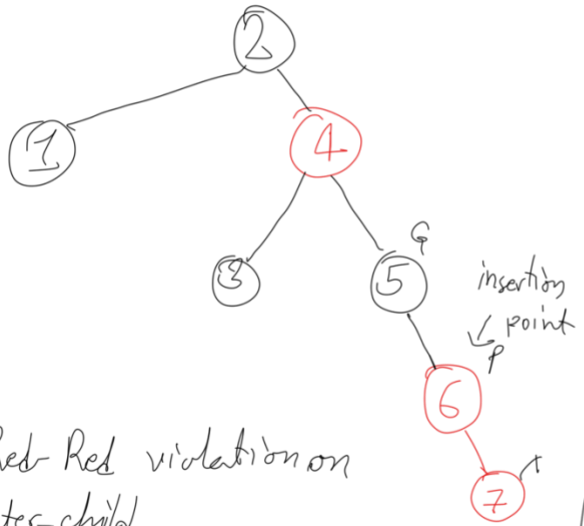


A black node with two red nodes detected, flip colors. And insert 6.

11.

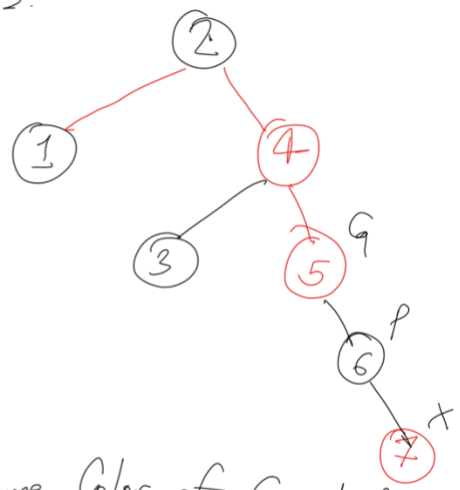


12. insert(7)



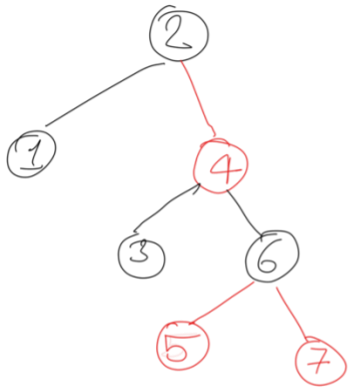
Red-Red violation on outer-child.

13.



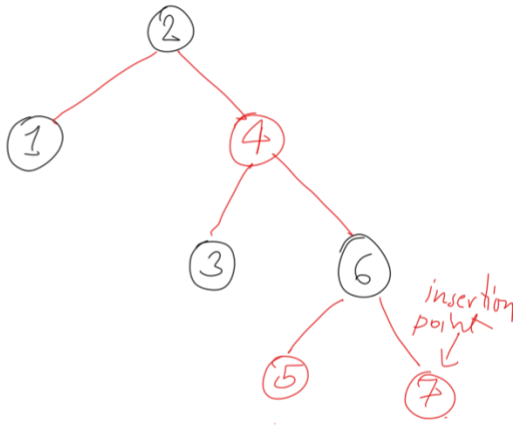
Change Color of G and P.

14.



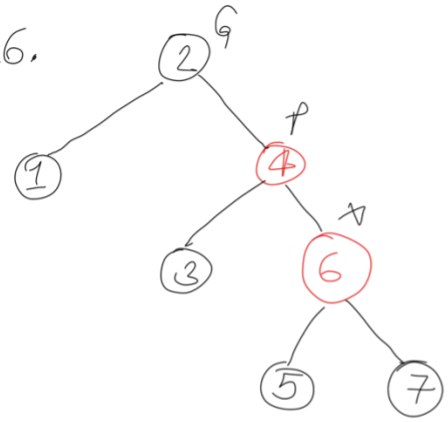
Rotate P and G that lifts X up.

15. insert (8)



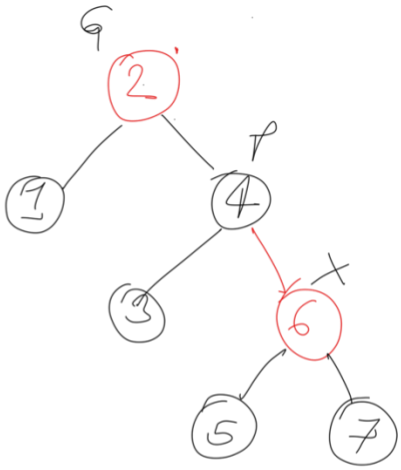
A node with two red child is detected. Flip colors and insert 8.

16.



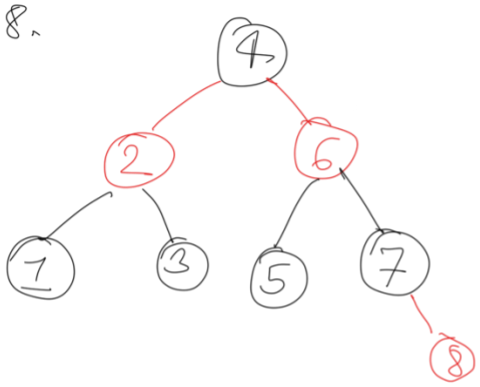
Outer grand-child red-red violation, change color of G and P

17.



Rotate P & G such that X is lifted.

18.



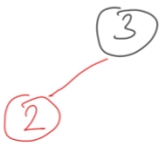
Insert 8. Final Red black tree is as above.

b. [3,2,1 5,4,6]

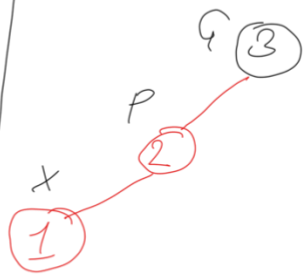
[3, 2, 1, 5, 4, 6]

1. insert (3)

2. insert (2)

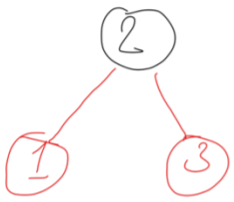


3. insert (1)

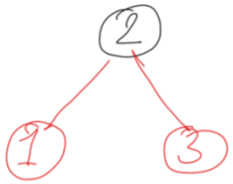


We encounter a red-red outer grand child violation. Change color of P, G and rotate to lift X.

4.

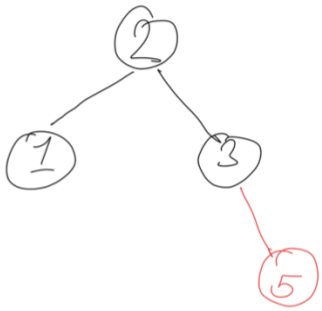


5. insert(5)

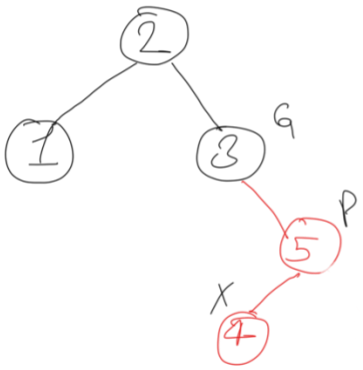


Black node with 2 red nodes detected. Flip Colors and insert 5.

6.

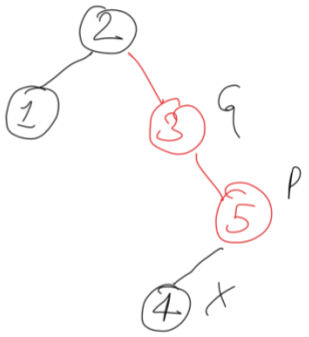


7. insert (4)



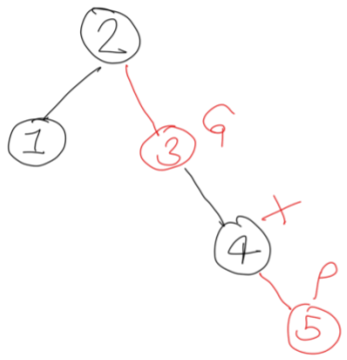
insertion causes red-red inner grandchild violation, Change color of G and X.

8.



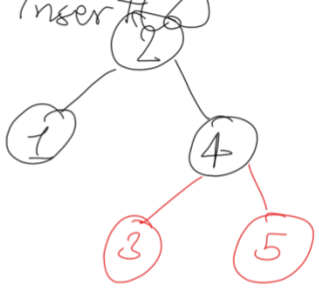
Rotate P and X in the direction that lifts up X.

9.



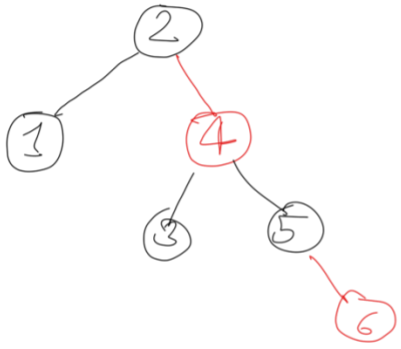
Perform G and X rotation in the direction that lifts X up.

10. insert(6)



A black node with 2 red nodes detected. Flip Colors,

11.



6 is inserted. Final tree looks like above,

Answer of 3

Hashmap Implementation:

```
class Trie {

    static class TrieNode {
        public Map<Character, TrieNode> children;
        public boolean isEndOfWord;

        TrieNode() {
            children = new HashMap<>();
            isEndOfWord = false;
        }
    }

    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode current = root;

        for (var c : word.toCharArray()) {
            current = current.children.computeIfAbsent(c,k ->
new TrieNode());
        }

        current.isEndOfWord = true;
    }

    public boolean search(String word) {
        TrieNode current = root;

        for (var c : word.toCharArray()) {
            current = current.children.getOrDefault(c, null);

            if (current == null) {
                return false;
            }
        }

        return current.isEndOfWord;
    }
```

```
    public boolean startsWith(String prefix) {
        TrieNode current = root;

        for (var c : prefix.toCharArray()) {
            current = current.children.getOrDefault(c, null);

            if (current == null) {
                return false;
            }
        }

        return true;
    }
}

/**
 * Your Trie object will be instantiated and called as such:
 * Trie obj = new Trie();
 * obj.insert(word);
 * boolean param_2 = obj.search(word);
 * boolean param_3 = obj.startsWith(prefix);
 */
```

Array Implementation:

```
class Trie {

    static class TrieNode {
        public TrieNode[] children;
        public boolean isEndOfWord;

        TrieNode() {
            children = new TrieNode[26];
            isEndOfWord = false;
        }
    }

    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode current = root;
```

```
        for (var c : word.toCharArray()) {
            int index = c - 'a';

            if (current.children[index] == null){
                current.children[index] = new TrieNode();
            }

            current = current.children[index];
        }

        current.isEndOfWord = true;
    }

    public boolean search(String word) {
        TrieNode current = root;

        for (var c : word.toCharArray()) {
            int index = c - 'a';

            current = current.children[index];

            if (current == null) {
                return false;
            }
        }

        return current.isEndOfWord;
    }

    public boolean startsWith(String prefix) {
        TrieNode current = root;

        for (var c : prefix.toCharArray()) {
            int index = c - 'a';

            current = current.children[index];

            if (current == null) {
                return false;
            }
        }

        return true;
    }
}

/**
```

```
* Your Trie object will be instantiated and called as such:
* Trie obj = new Trie();
* obj.insert(word);
* boolean param_2 = obj.search(word);
* boolean param_3 = obj.startsWith(prefix);
*/
```