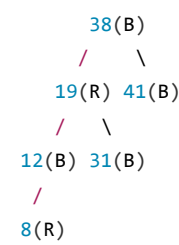


# HW4

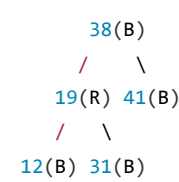
## Q1解答

### (a)插入后的红黑树

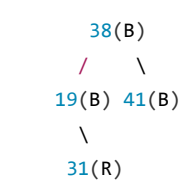


### (b)按步骤删除的红黑树

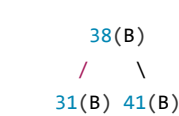
1. 删除8



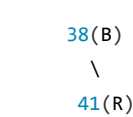
2. 删除12



3. 删除19



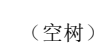
4. 删除31



5. 删除38



6. 删除41



## Q2 一棵黑高为 k 的红黑树

### 1.最小内部结点数

为了使红黑树的结点尽可能少，即尽量少使用红色结点，同时保证树的黑高是k。

- 所有内部结点都是**黑色**的。
- 这是一个**完全黑色二叉树**，即每个黑色非叶结点都有两个黑色子节点。
- 总结点数为  $2^k - 1$ 。

### 2.最大内部结点数

为了使红黑树的结点尽可能多，即尽量使用红色结点，同时保证树的黑高是k。

- 内部结点交替使用**黑色**和**红色**结点。
- 根结点是**黑色**结点。
- 这是一个**完全二叉树**，实际树高达到  $2k$ 。
- 总结点数为  $4^k - 1$ 。

## Q3 区间集合的最大重叠点

### (a) 证明：在最大重叠点中，一定存在一个点是其中一个区间的端点。

1. **假设：** 存在一个  $x$ ，它是最大重叠点，且被  $a$  个区间覆盖。
2. **考虑覆盖  $x$  的区间：**  
对于每个区间  $[l_i, r_i] \in S$ ，有

$$l_i \leq x \leq r_i$$

3. **定义左端点的最大值  $y$ ：**  
设所有这些区间的左端点的最大值为：

$$y = \max_{[l_i, r_i] \in S} \{l_i\}$$

因为  $x$  被所有  $a$  个区间覆盖，所以必有：

$$y \leq x$$

4. **证明  $y$  也是被  $a$  个区间覆盖的点：**  
对于任意  $[l_i, r_i] \in S$ ，由于  $l_i \leq y \leq x \leq r_i$ ，因此：

$$l_i \leq y \leq r_i$$

这说明  $y$  也被所有  $a$  个区间覆盖。

5. **结论：**  
因为  $y$  是这些区间的左端点之一，所以在最大重叠点中，存在一个点是区间的端点。

### (b)设计数据结构

#### 设计思路

1. 使用红黑树作为整体存储的结构：
  - 将所有区间的端点作为红黑树的节点。
  - 利用红黑树基本平衡的特性，可以快速插入、删除和找到最大重叠点。
2. 差分思想：
  - 区间左端点处+1，表示从这里开始被覆盖；区间右端点的下一个位置处-1，表示从这里开始不再被覆盖。
  - 插入区间时：左端点  $l$  处，记录+1；右端点下一个位置  $r + 1$  处，记录-1。

- 删除区间时：左端点 $l$ 处，记录-1；右端点下一个位置 $r + 1$ 处，记录+1。因为删除区间本质是把区间标记为未被覆盖，我们可以用抵消的方式实现。
3. 维护前缀和：
- 前缀和 $sum$ ，记录从最小的位置（即 $\infty$ ）到当前节点的差分值累加和，即当前的覆盖区间数。
  - 基于 $sum$ ,维护 $maxOverlapTimes$ ，记录最大重叠次数，以及维护 $maxOverlapPoint$ ,记录最大重叠次数对应的点。
4. 节点信息：
- key: 点位置
  - val: 差分值，即当前点位置被覆盖的次数
  - sum: 前缀和，即从 $\infty$ 到当前点的覆盖次数累加和
  - maxOverlapTimes: 当前点位置被覆盖的最大次数(以当前节点为根)
  - maxOverlapPoint: 当前点位置被覆盖的最大次数对应的点

## 算法细节

### 1. 插入区间：

```
FUNCTION INTERVAL_INSERT(root, l, r):  
    root = INSERT(root, l, +1)  
    root = INSERT(root, r, -1)  
    RETURN root  
END FUNCTION
```

### 2. 删除区间

```
FUNCTION INTERVAL_DELETE(root, l, r):  
    root = INSERT(root, l, -1)  
    root = INSERT(root, r, +1)  
    RETURN root  
END FUNCTION
```

### 3. 查找最大重叠点：

```
FUNCTION FIND_POM(root):  
    IF root IS NULL:  
        RETURN NULL    // 没有区间时返回空  
    ELSE:  
        RETURN root.maxPrefixPoint  
    END FUNCTION
```

### 4. 插入节点：

```
FUNCTION INSERT(root, key, val):
  IF root IS NULL:
    CREATE NEW NODE(key, val)
    SET sum = val
    SET maxPrefixSum = val
    SET maxPrefixPoint = key
    RETURN NEW NODE
  END IF

  ELSE IF key < root.key:
    root.left = INSERT(root.left, key, val)
    root.left.parent = root
  ELSE IF key > root.key:
    root.right = INSERT(root.right, key, val)
    root.right.parent = root
  ELSE:
    root.val += val
  END IF

  UPDATE(root) //更新节点信息

  BALANCE(root) // 平衡树

  RETURN root
END FUNCTION
```

5. 更新节点信息:

```

FUNCTION UPDATE(root):
    // 更新前缀和
    IF root.left IS NOT NULL:
        root.sum += root.left.sum
    IF root.right IS NOT NULL:
        root.sum += root.right.sum
    // 计算左子树的最大前缀和
    leftMaxSum = NEGATIVE_INFINITY
    leftMaxPoint = NULL
    IF root.left IS NOT NULL:
        leftMaxSum = root.left.maxPrefixSum
        leftMaxPoint = root.left.maxPrefixPoint
    // 计算经过当前节点的前缀和
    currentSum = root.val
    IF root.left IS NOT NULL:
        currentSum += node.left.sum
    // 计算右子树的最大前缀和
    rightMaxSum = NEGATIVE_INFINITY
    rightMaxPoint = NULL
    IF root.right IS NOT NULL:
        rightMaxSum = currentSum + root.right.maxPrefixSum
        rightMaxPoint = root.right.maxPrefixPoint
    // 更新最大重叠次数和最大重叠点
    IF leftMaxSum > currentSum AND leftMaxSum > rightMaxSum:
        root.maxPrefixSum = leftMaxSum
        root.maxPrefixPoint = leftMaxPoint
    ELSE IF currentSum >= leftMaxSum AND currentSum >= rightMaxSum:
        root.maxPrefixSum = currentSum
        root.maxPrefixPoint = root.key
    ELSE:
        root.maxPrefixSum = rightMaxSum
        root.maxPrefixPoint = rightMaxPoint
    END IF
END FUNCTION

```

注：可以调用库中平衡树的函数，这里不赘述。

## Q4

### (a) 将 $x$ 的子链表添加到堆的根列表中在 $O(1)$ 时间内可能无法完成

主要原因如下：

1. 将  $x$  的子链表加入根链表时，需要将他们的父指针更新为 NIL，因为他们现在成为根节点。
2. 如果  $x$  有  $x.degree$  个子节点，那么时间复杂度为  $O(x.degree)$ 。

### (b) 计算 PISANO-DELETE 实际时间复杂度的紧凑上界

当  $x \neq H.min$  时，PISANO-DELETE 的主要操作：

1. CUT 和 CASCADING-CUT 操作：
  - CUT 操作的时间复杂度为  $O(1)$ ，因为只需要将  $x$  从父节点  $y$  的子链表中移除并加入到根链表中，这个过程不涉及比较或其他操作。
  - CASCADING-CUT 递归地对需要的节点进行剪切操作，每次的时间复杂度是  $O(1)$ ，因此总时间复杂度为  $O(c)$ 。
2. 更新子节点的父指针并将其加入根链表：
  - 在(a)问中我们已经得到其时间复杂度为  $O(x.degree)$ 。
3. 从根链表中移除  $x$ ：

- 时间复杂度为  $O(1)$ 。

综上, 总时间复杂度为  $O(c + x.degree)$