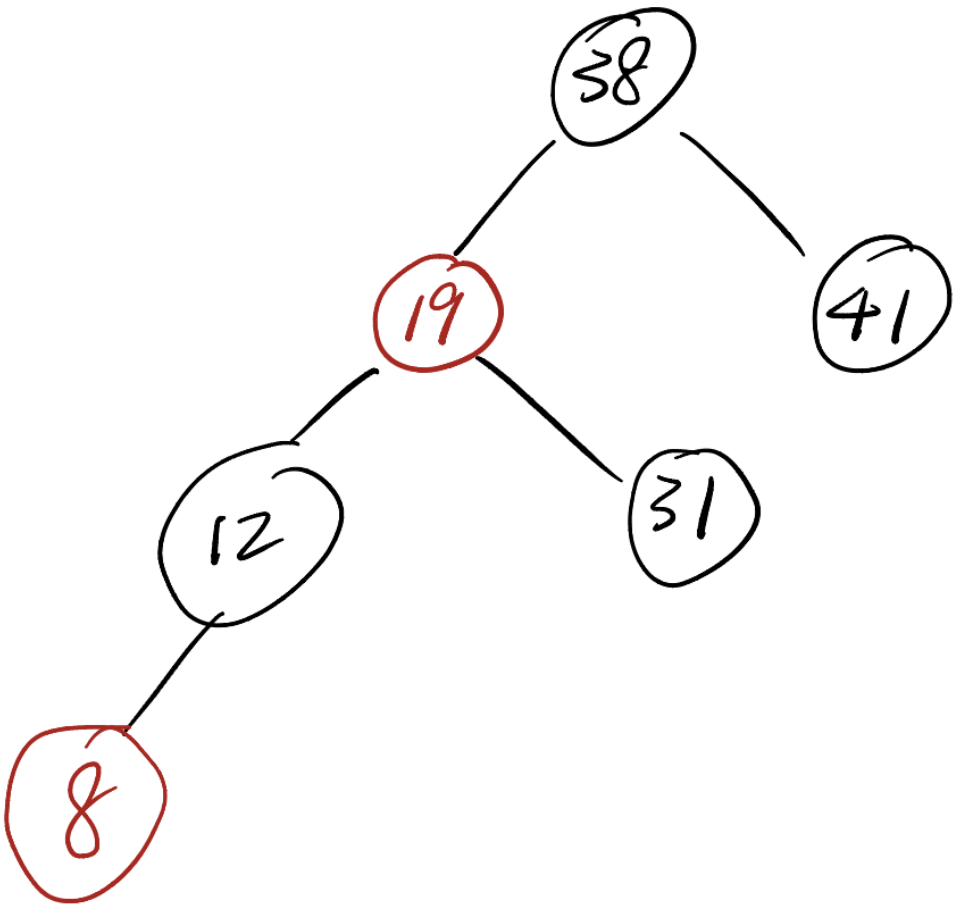
HW4

林宸昊 PB20000034

1. 考虑两种情况：
   - `x = y.left` 由于x为叶节点，此时调用`TreeSuccessor`将直接跳过while循环直接执行

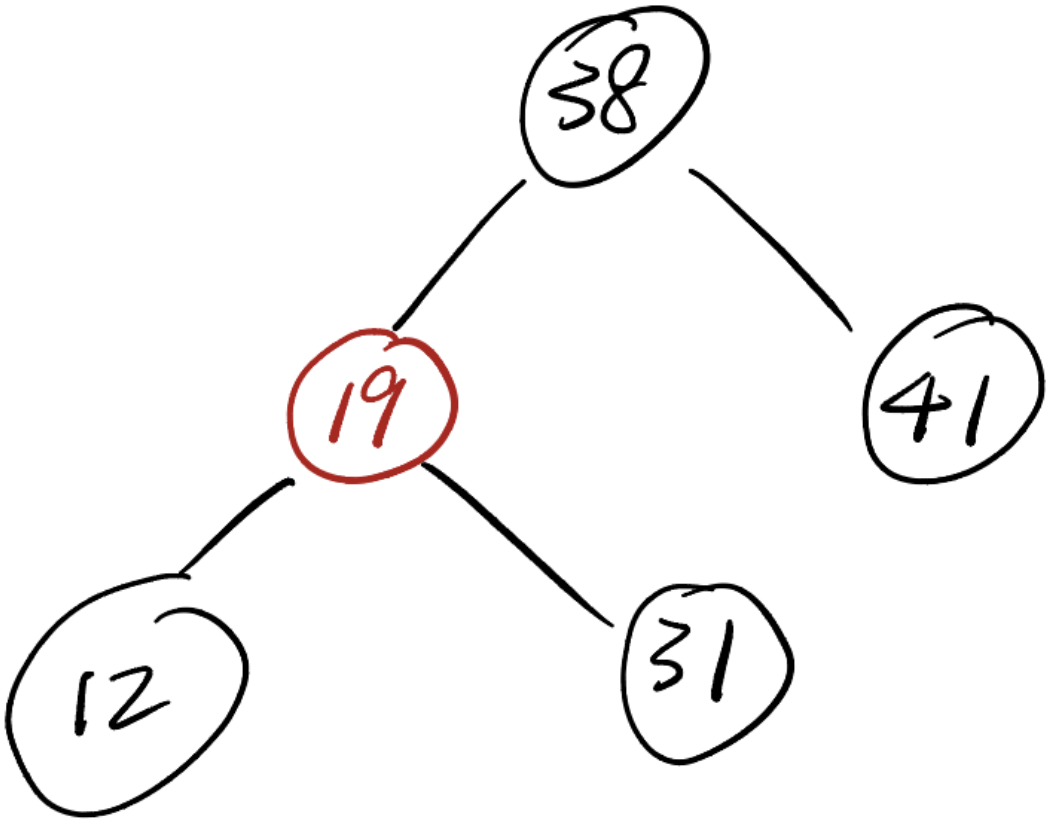
     ```
     y <- p[x];
     ...
     return y;
     ```

     即y为x的后继，也即y.key为大于x.key的最小关键字；
   - `x = y.right` 同样，由于x为叶节点，此时y为x的前驱，即y.key是小于x.key的最大关键字。
2. (a)

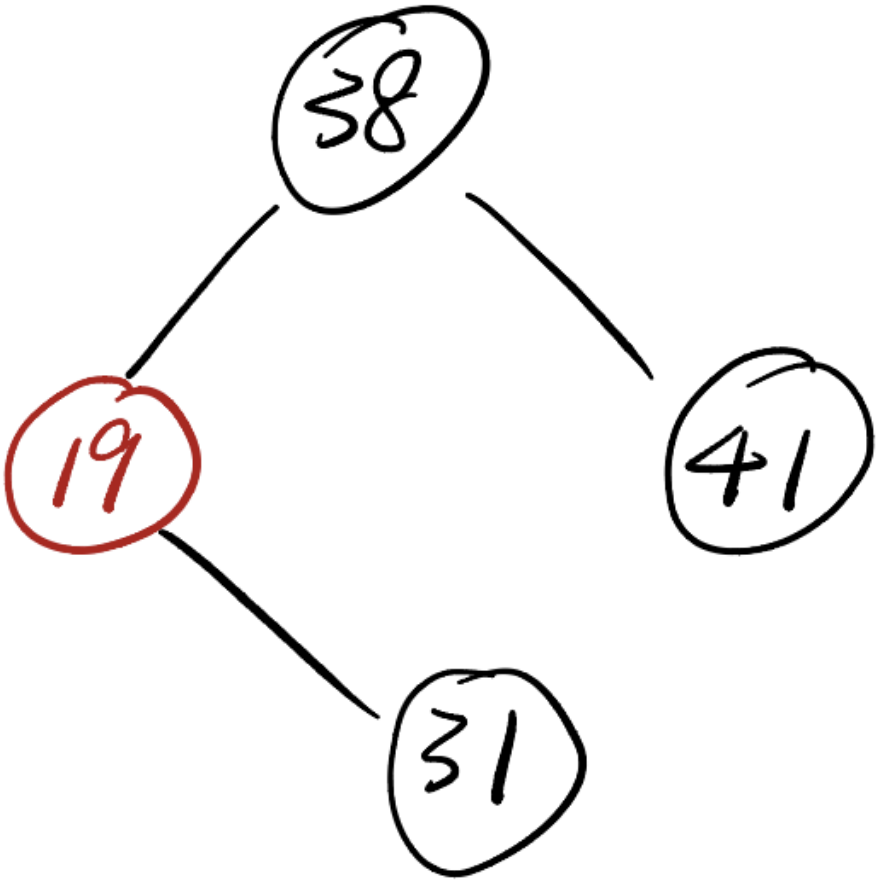


(b)

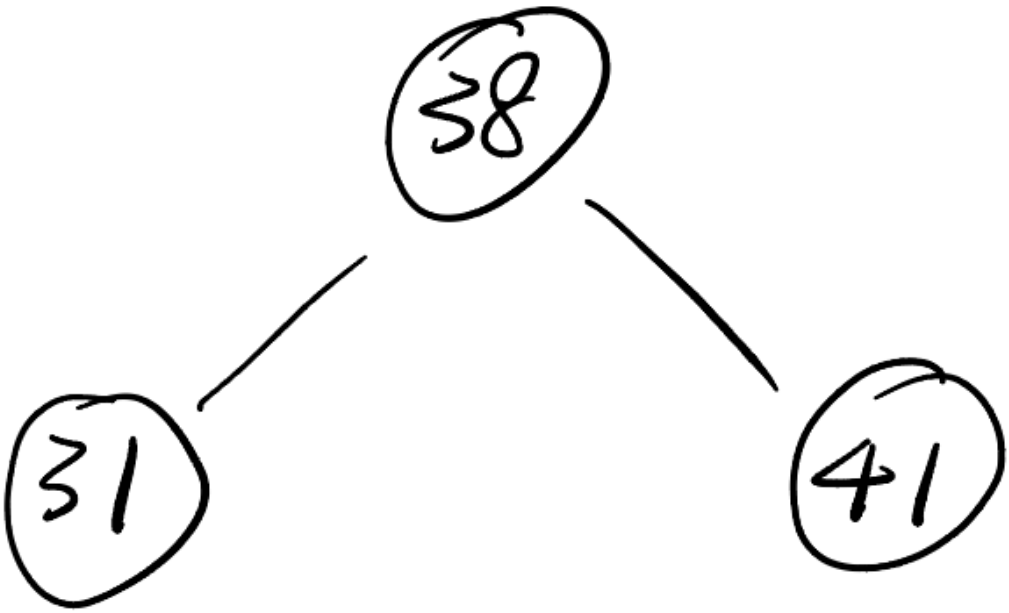
- 删除8：



- 删除12：



- 删除19：



3. (a)

- 假设取到的最大重叠点不是区间端点，那么将该点逐渐增大，直到第一次碰到某个区间端点停止。在这个过程中，由于碰到端点的这个区间就是包含它的最小区间，因此这个过程中这个点持续增大且包含它的区间不会减少，与假设矛盾。故最大重叠点一定是区间端点。

(b)

- 采用红黑树作为基础数据结构。
  - 节点值为所有输入区间的端点值;
  - 每个节点附带并维护数个新信息:
    - pos(x): 若x为左端点则值为1，右端点值为-1;
    - value(x): 以x为根的所有节点的pos值之和;

      例如x的左子树中理论上所有节点均为小于x值的端点，但是可能存在子树亦有左右子树，即某个区间的左右端点均在x的左子树中。此时pos值的-1就正好可以处理这种情况，此时得到的value亦正好为x此时的区间重叠数。
    - max(x): 以x为根的树中所有节点中最大的区间重叠数，并且储存对应最大重叠数的那个节点的信息。
  - 采用DC方法，对根节点的左右子树递归寻找最后与自身比较得到最大重叠点。
- INTERVAL-INSERT:

  与红黑树的基本插入方式一致，插入过程中经过的每个节点的value值都加上插入节点的pos值;
- INTERVAL-DELETE:

  与红黑树的基本删除方式一致，删除过程中经过的每个节点的value值都减去删除节点的pos值;
- FIND-POM算法:

  由于采用DC方法，可以自底向上计算每个节点的max(x)并记录对应位置。对于max的计算可以利用新增的两个节点信息:
  - 对于某节点x自身，其重叠数即为左子树的value值加上它自身的pos值;
  - 对于某节点x左子树，其最大（因为是对于整个子树而言）重叠数即为max(left(x));
  - 对于某节点x右子树，可以把x看作它的左子树，其最大重叠数即为x自身的重叠数加上max(right(x));

  由于使用DC方法自底向上寻找最大重叠点，每次比较之后都能得到对应x点的位置最后返回到根节点进行比较得到最终的最大重叠点。