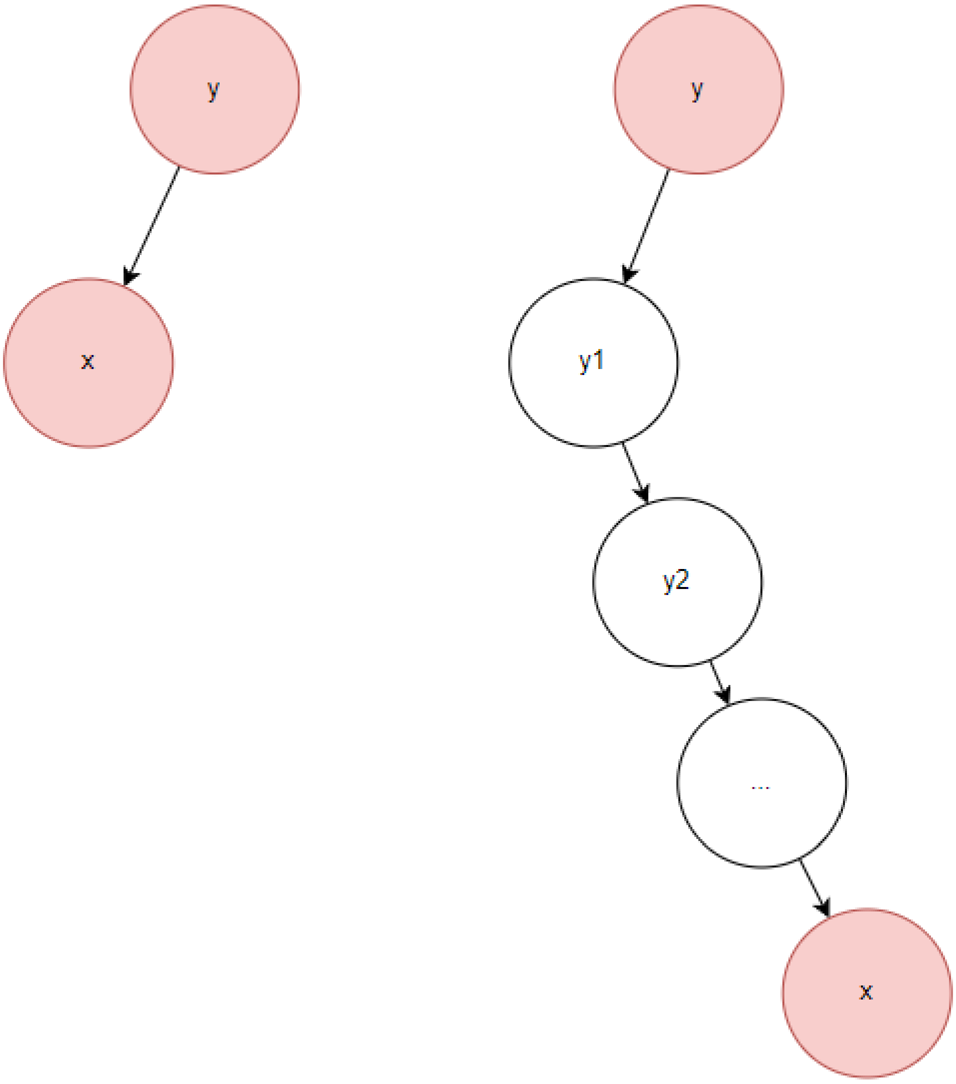
郑源泽19307130077hw6

12.2-6

证明：由于x没有右子树，故y和x的相对位置只有如图所示的几种情况：



其中，第二种情况白色的中间节点可以有任意个，但是不能向左拐，否则y就不满足是最低的条件。

我们寻找小于y的最大元素：小于y的元素可能在y的祖先（当y是右子节点时），也可能在y的左子树

但是祖先的情况肯定祖先节点会小于y的左子树（二叉搜索树性质）

这样看来，从y向x走，第一种情况由于没有右子树，明显x是小于y的最大的元素

第二种情况同理，向左走之后找最大的元素，就是y。

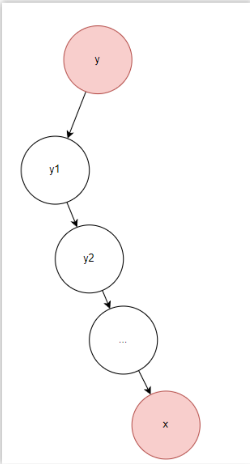
于是：x是小于y的最大的元素得证

反之：y是x的后继successor得证

12.3-5

PARENT

```
PARENTS(T, x)
//返回树T中x的亲属节点
//利用节点的后继信息
if x == T.root
    return x
y = TREE-MAXIMUM(x).succ
if y == NIL
    y = T.root
else if y.left == x
    return y
    y = y.left
while y.right != x
    y = y.right
return y
```



SEARCH

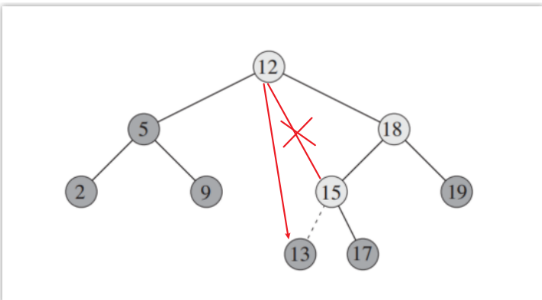
和书中的一样

TREE-SEARCH(x, k)

if x == NIL or k == $x.key$
  return x
if $k < x.key$
  return TREE-SEARCH$(x.left, k)$
else return TREE-SEARCH$(x.right, k)$

INSERT

```
INSERT(T, x)
//插入x会影响上层的某个
//  以x的后继为后继的节点的后继
//插入的肯定是叶子节点
y = NIL
x = T.root
pred = NIL
while x != NIL
    y = x
    if z.key < x.key
        x = x.left
    else
        pred = x
        x = x.right
if y == NIL
    T.root = z
    z.succ = NIL
elseif z.key < y.key
    y.left = z
    z.succ = y
    if pred != NIL
        pred.succ = z
else
    y.right = z
    z.succ = y.succ
    y.succ = z
```



DELETE

首先重写TRANSPLANT(T, u, v)

transplant不涉及更改节点的succ属性

```
TRANSPLANT(T, u, v)
    p = PARENT(T, u)
    if p == NIL
        T.root = v
    else if u == p.left
        p.left = v
    else
        p.right = v
```

delete的过程和原文基本一致，只是会影响之前以z为succ的节点的属性

我们需要预先将z的前驱更改为z的后继

```
DELETE(T, z)
//  pred = TREE-MINIMUM(T.root)
//  while pred.succ != z
//      pred = pred.succ
//  pred.succ = z.succ
//  更正，不满足要求O(h)
    pred = TREE-PREDECESSOR(T, z)
    pred.succ = z.succ
    if z.left == NIL
        TRANSPLANT(T, z, z.right)
    else if z.right == NIL
        TRANSPLANT(T, z, z.left)
    else
        y = TREE-MIMIMUM(z.right)
        if PARENT(T, y) != z
            TRANSPLANT(T, y, y.right)
            y.right = z.right
        TRANSPLANT(T, z, y)
        y.left = z.left
```

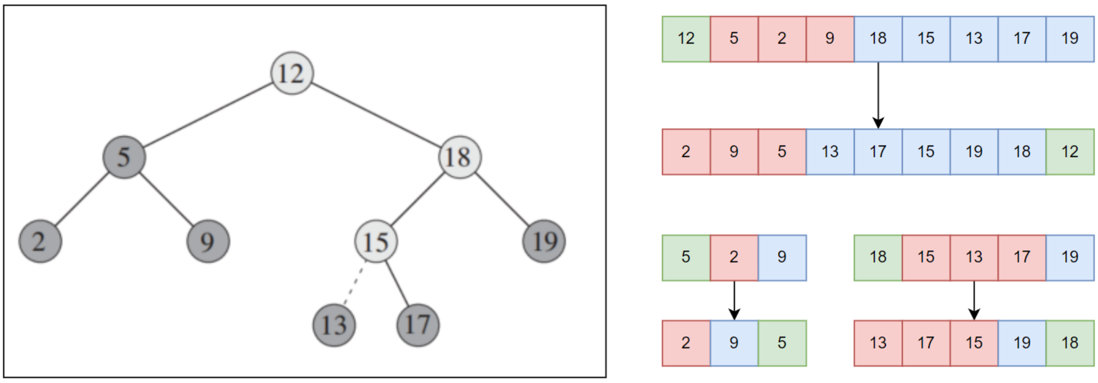
```
TREE-PREDECESSOR(T, z)
    if x.left != NIL
        return TREE-MAXIMUM(x.left)
    y = T.root
    pred = NIL
    while y != NIL
        if y.key == x.key
            break
        if y.key < x.key
            pred = y
            y = y.right
        else
            y = y.left
    return pred
```

3

Without a binary search tree, is it possible to give the result of the post-order tree-walk if both pre-order and in-order tree walk results are given? Justify your answer.

中序遍历结果没什么用，就是一个简单的递增顺序

从先序遍历得到后序遍历



如图所示，先序可以把后面的分为两隔子树：小于根（绿色）的就是左子树（红色），大于的就是右子树（蓝色）

根放在最后，递归解决左右子树问题

伪代码如下

```
let B[0...n] be a new array //全局变量
    fill B with 0
int index = n               //全局变量
Solution(A, star, end)
    if start == end
        B[index] = A[start]
        index--
        return
    B[index] = A[start]
    index--
    //遍历判断大小得到四个索引
    easily find the index left-start, left-end, right-start, right-end
    //先填右子树，再填左子树
    Solution(A, right-start, right-end)
    Solution(A, left-start, left-end)
```