- 书面作业讲解
  - TC第6.1节练习2、4、7
  - TC第6.2节练习2、5、6
  - TC第6.3节练习3
  - TC第6.4节练习2、4
  - TC第6.5节练习5、7、9

TC第6.1节练习2

- 用n来表示h的范围比较繁琐，不如改用h来表示n的范围

$$2^h \le n \le 2^{h+1} - 1 < 2^{h+1}$$

$$h \le \lg n < h + 1$$

$$\lfloor \lg n \rfloor = h$$

TC第6.3节练习3

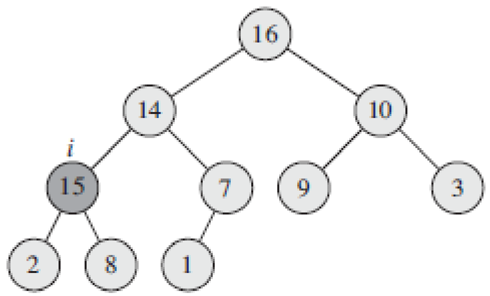
- 数学归纳法
  - h=0时，高度为0的数量=叶子数量=$\left\lceil \frac{n}{2} \right\rceil \le \left\lceil \frac{n}{2^{h+1}} \right\rceil$
  - 假设h=k时，高度为h的数量$\le \left\lceil \frac{n}{2^{h+1}} \right\rceil$
  - 则h=k+1时，高度为h+1的数量$\le \left\lceil \frac{\left\lceil \frac{n}{2^{h+1}} \right\rceil}{2} \right\rceil = \left\lceil \frac{n}{2^{(h+1)+1}} \right\rceil$

TC第6.4节练习4

- 要证的是Ω，不是O
- $n\Omega(\lg n) = \Omega(n \lg n)$，这样对吗？
- $\sum_{i=1}^{n-1} \Omega(\lg i) = \Omega\left(\sum_{i=1}^{n-1} \lg i\right) = \Omega(\lg(n-1)!) = \Omega(\lg n!) = \Omega(n \lg n)$

TC第6.5节练习5

- errata： loop invariat需要增加以下条件：
  - A[PARENT(i)]≥A[LEFT(i)] and A[PARENT(i)]≥A[RIGHT(i)], if these nodes exist



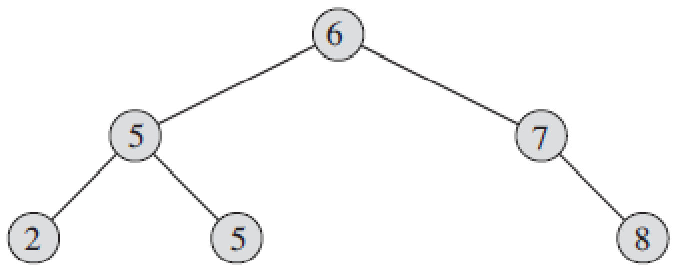
TC第6.5节练习9

k个sorted list的首元素移入同一个堆	O(k)
反复地： -将堆顶元素移出并输出 -将该元素所属sorted list的首元素移入堆	O(nlgk)

- 教材答疑和讨论
  - TC第12、13章

问题1： binary search trees

- binary search tree是怎样的一个结构？
- 它支持dynamic set的哪些操作？运行时间分别是多少？



Search
Insert
Delete
Minimum
Maximum
Successor
Predecessor

问题1：binary search trees (续)

```
INORDER-TREE-WALK(x)
if x ≠ NIL
    INORDER-TREE-WALK(x.left)
    print x.key
    INORDER-TREE-WALK(x.right)
```

- 这个算法的作用是什么？
- 你能简要概括它的基本原理吗？
- 如何证明它的正确性？
- 它的运行时间是多少？

问题1：binary search trees (续)

```
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)
```

- 这个算法的作用是什么？
- 你能简要概括它的基本原理吗？
- 如何证明它的正确性？
- 它的运行时间是多少？

问题1：binary search trees (续)

```
ITERATIVE-TREE-SEARCH(x, k)
while x ≠ NIL and k ≠ x.key
    if k < x.key
        x = x.left
    else x = x.right
return x
```

- 这个算法的作用是什么？
- 你能简要概括它的基本原理吗？
- 如何证明它的正确性？
- 它的运行时间是多少？

问题1：binary search trees (续)

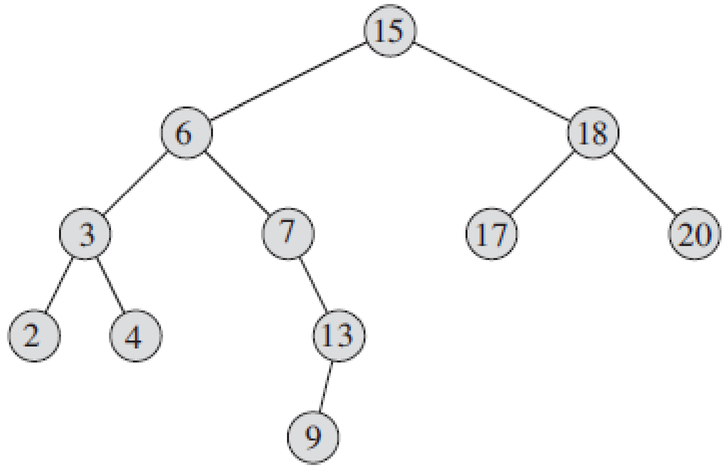
```
TREE-MINIMUM(x)
while x.left ≠ NIL
    x = x.left
return x
```

```
TREE-MAXIMUM(x)
while x.right ≠ NIL
    x = x.right
return x
```

- 这个算法的作用是什么？
- 你能简要概括它的基本原理吗？
- 如何证明它的正确性？
- 它的运行时间是多少？

问题1：binary search trees (续)

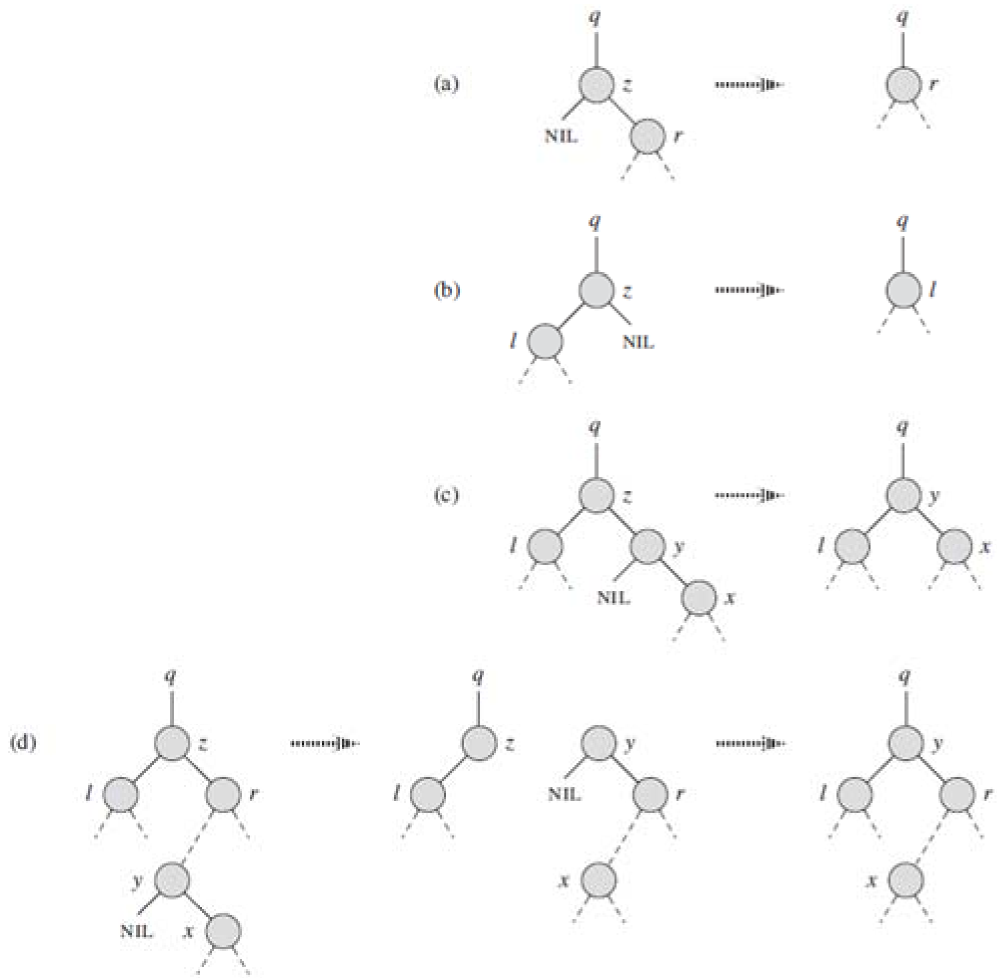
```
TREE-SUCCESSOR(x)
if x.right ≠ NIL
    return TREE-MINIMUM(x.right)
y = x.p
while y ≠ NIL and x == y.right
    x = y
    y = y.p
return y
```



- 这个算法的作用是什么？
- 你能简要概括它的基本原理吗？
- 为什么结果是正确的？
- 它的运行时间是多少？

问题1：binary search trees (续)

- 如何删除一个顶点？
- 为什么结果是正确的？

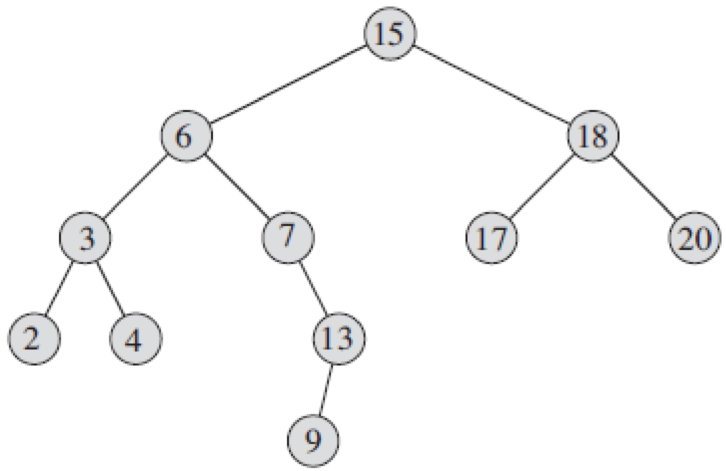


问题1：binary search trees (续)

- 什么样的输入会导致一个糟糕的binary search tree？
- 如果有人恶意这么做，该怎么应对？

问题1：binary search trees (续)

- binary search tree和hash table各有什么优缺点？
  - 运行时间：search, insert, delete
  - successor/predecessor
  - range search
  - sorting
  - 对输入的要求
    - hash function vs. total order
    - size/resize



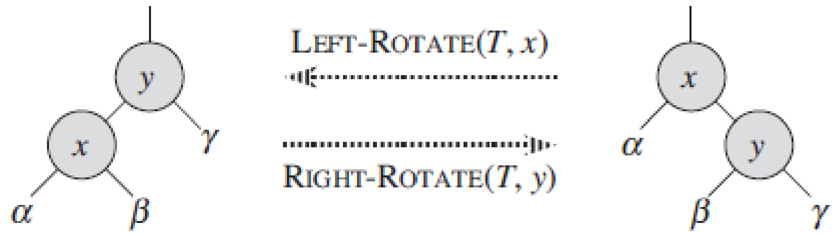
问题2： red-black trees

- 你怎么理解red-black tree的平衡性？
  - No simple path from the root to a leaf is more than twice as long as any other.
- 为什么会具有这种平衡性？

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

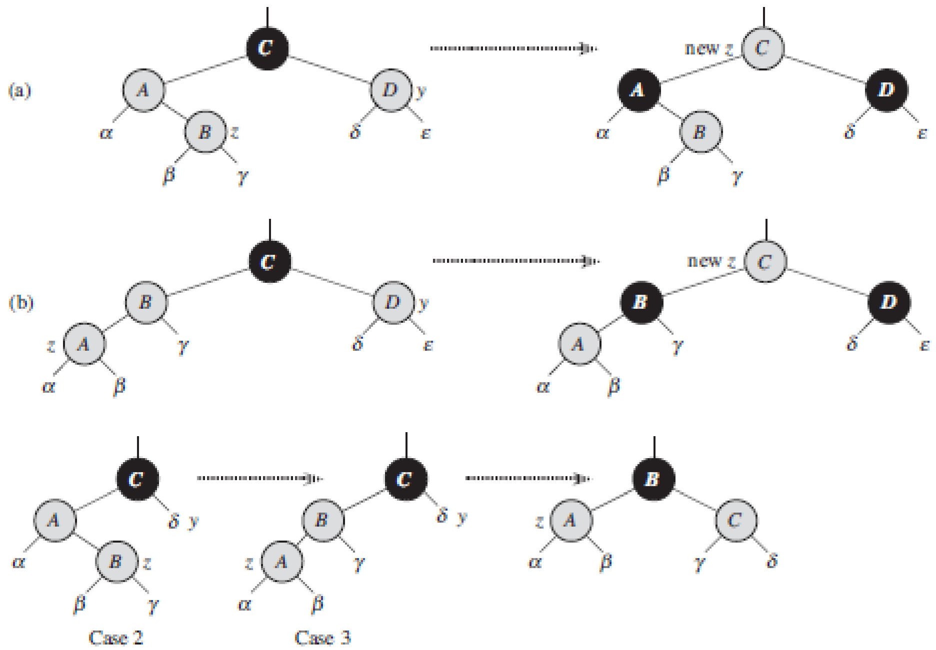
问题2： red-black trees (续)

- 经过insert和delete中的各种复杂操作之后，为什么binary search tree的性质不会丢失？
- 为什么rotation之后仍能保持binary search tree的性质？



问题2： red-black trees (续)

- 将z (red)插入之后，fixup的主要目标是什么？
  – 保持每条path上的black数量
  – 消除相连的red
- 对每种case分别是如何实现的？

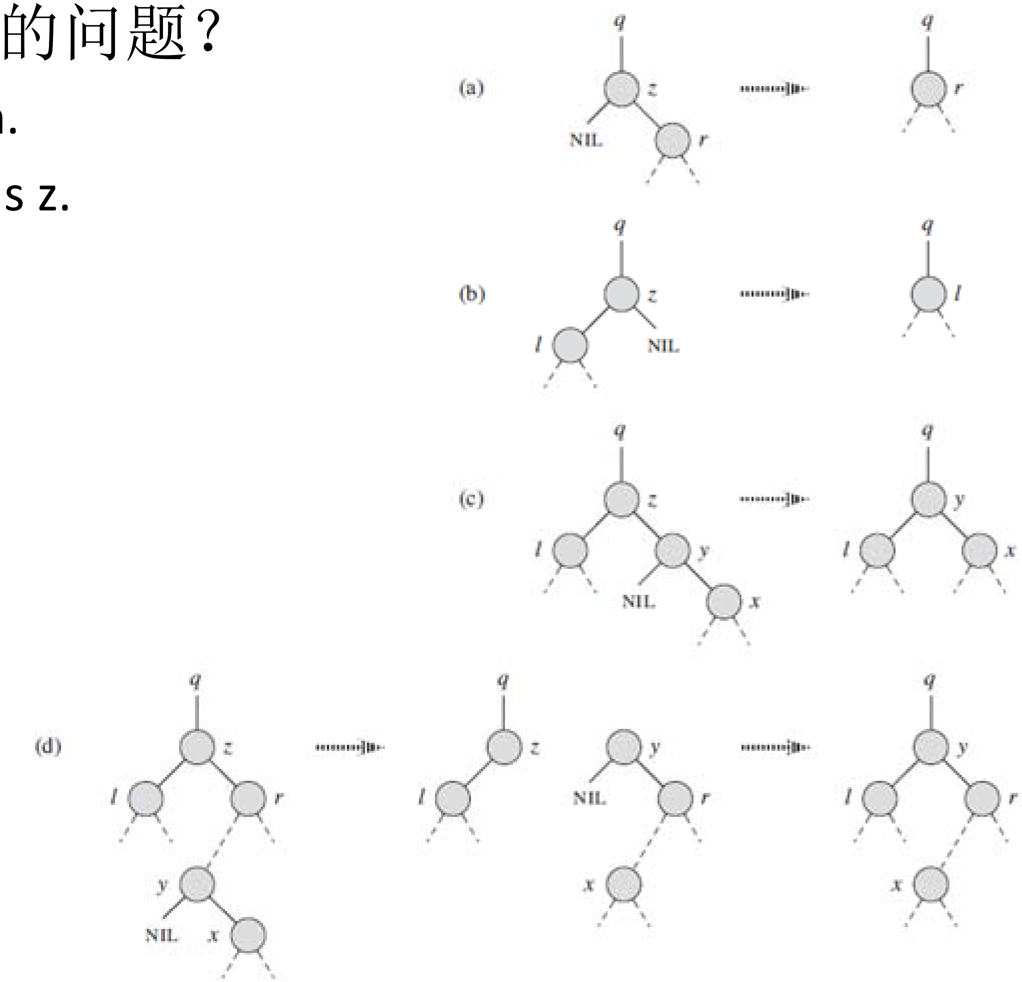


问题2： red-black trees (续)

- 将z删除之后，fixup的主要目标是什么？
  - 保持每条path上的black数量
  - 消除相连的red
- 这里涉及到的z, y, x分别表示什么？
  - y moves into z’s position.
  - x moves into y’s position.

问题2： red-black trees (续)

- 如何先修复删除z带来的问题？
  - y moves into z’s position.
  - Gives y the same color as z.
- 副作用是什么？
  - y原来的位置会出问题
- 什么时候会出问题？
  - y=red？
  - y=black



问题2： red-black trees (续)

- 如何再修复移走y (black)带来的问题？
  - x moves into y’s position.
  - Push y’s blackness onto x.
- 副作用是什么？
  - x可能有超额blackness需要摊出去
- 对每种case分别是如何解决的？