Solutions for Introduction to Algorithms 2nd Edition

小高子 <gaochangjian@gmail.com> and Kight Wang <w.jq0722@gmail.com>

2011 年 7 月 12 日

目录

第一部分　基础知识 **5**

第一章　算法在计算中的作用 **7**

第二章　算法入门 **9**

第三章　函数的增长 **11**

第四章　递归式 **13**

4.1　代换法 13
4.2　递归树方法 13
4.3　主方法 14
思考题 15

第五章　概率分析和随机算法 **27**

5.1　雇用问题 27
5.2　指示器随机变量 28
5.3　随机算法 30
5.4　概率分析和指示器随机变量的进一步使用 32
思考题 33

第二部分　排序和顺序统计学 **35**

第六章　堆排序 **37**

6.1　堆 37
6.2　保持堆的性质 37
6.3　建堆 38
6.4　堆排序算法 38
6.5　优先级队列 38
思考题 38

第七章　快速排序 **39**

第八章　线性时间排序 41

第九章　中位数和顺序统计学 43

第三部分　数据结构 45

第十章　基本数据结构 47

第十一章　散列表 49

第十二章　二叉查找树 51

第十三章　红黑树 53
13.1 红黑树的性质 53
13.2 旋转 54
13.3 插入 54
思考题 54

第十四章　数据结构的扩张 57
14.1 动态顺序统计 57

第四部分　不是 CLRS 附录的附录 59

附录 A　排序算法比较 61

第一部分

基础知识

第一章　算法在计算中的作用

第二章　算法入门

第三章　函数的增长

第四章　递归式

4.1　代换法

4.2　递归树方法

4.2-1

4.2-2

4.2-3 **画出 $T(n) = 4T(\lfloor n/2 \rfloor) + cn$ 的递归树，并给出其解的渐近确界，其中 c 是一个常数。然后，用代换法证明你给出的界。**

第 i 层结点的值为 $c(\lfloor n/2^i \rfloor)$， $i \geqslant 0$，最后一层为 $T(1)$，即 $\lfloor n/2^i \rfloor = 1$，因此递归树一共有 $i = \lg n$ 层。

每一层的结点个数为 4^i，因此每一层的总代价为 $4^i \cdot c(\lfloor n/2^i \rfloor)$，其中最后一层的总代价为 $4^{\lg n} \cdot T(1) = \Theta(4^{\lg n}) = \Theta(n^{\lg 4}) = \Theta(n^2)$。于是可以得到整棵递归树的代价：

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n-1} 4^i \cdot c(\lfloor n/2^i \rfloor) + \Theta(n^2) \\
&\leqslant \sum_{i=0}^{\lg n-1} 4^i \cdot \frac{cn}{2^i} + \Theta(n^2) \\
&= cn \sum_{i=0}^{\lg n-1} 2^i + \Theta(n^2) \\
&= cn \cdot \frac{2^{\lg n} - 1}{2-1} + \Theta(n^2) \\
&= cn(n-1) + \Theta(n^2) \\
&= cn^2 - cn + \Theta(n^2) = \Theta(n^2) \qquad (4.1)
\end{aligned}
$$

因此这棵递归树的渐近确界为 $\Theta(n^2)$。现在用代换法进行证明，假设 $T(\lfloor n/2 \rfloor) \leqslant c(\lfloor n/2 \rfloor)^2$，则：

$$
\begin{aligned}
T(n) &= 4T(\lfloor n/2 \rfloor) + cn \\
&\leqslant 4c(\lfloor n/2 \rfloor)^2 + cn \\
&\leqslant 4c\left(\frac{n}{2}\right)^2 + cn = cn^2 + cn \leqslant cn^2 \qquad (4.2)
\end{aligned}
$$

4.3 主方法

4.3-1 用主方法来给出下列递归式紧确的渐近界:

a) $T(n) = 4T(n/2) + n$

在这个递归式中， $a = 4$， $b = 2$， $f(n) = n$，则 $n^{\log_b a} = n^{\log_2 4} = n^2$。因为 $f(n) < n^{\log_b a}$，尝试主定理的第一种情况是否成立。由于 $f(n) = O(n^{\log_b a-\varepsilon}) = O(n^{\log_2 4-1})$，其中 $\varepsilon = 1$，因此满足第一种情况，则 $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$。

b) $T(n) = 4T(n/2) + n^2$

在这个递归式中， $a = 4$， $b = 2$， $f(n) = n^2$，则 $n^{\log_b a} = n^{\log_2 4} = n^2$。因为 $f(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$，所以满足主定理的第二种情况，因此 $T(n) = \Theta(n^{\log_b a}\lg n) = \Theta(n^2\lg n)$。

c) $T(n) = 4T(n/2) + n^3$

在这个递归式中， $a = 4$， $b = 2$， $f(n) = n^3$，则 $n^{\log_b a} = n^{\log_2 4} = n^2$。因为 $f(n) > n^{\log_b a}$，尝试主定理的第三种情况是否成立。首先 $f(n) = \Omega(n^{\log_b a+\varepsilon}) = \Omega(n^{\log_2 4+1})$，其中 $\varepsilon = 1$。其次对足够大的 n，要使 $af(n/b) \leqslant cf(n)$ 成立，则:

$$\begin{aligned} 4f\left(\frac{n}{2}\right) &\leqslant cf(n) \\ 4\left(\frac{n}{2}\right)^3 &\leqslant cn^3 \\ c &\geqslant \frac{1}{2} \end{aligned} \tag{4.3}$$

存在常数 $\frac{1}{2} \leqslant c < 1$，因此满足第三种情况，则 $T(n) = \Theta(f(n)) = \Theta(n^3)$。

4.3-2 某个算法 A 的运行时间由递归式 $T(n) = 7T(n/2) + n^2$ 表示; 另一个算法 A' 的运行时间为 $T'(n) = aT'(n/4) + n^2$。若要 A' 比 A 更快，那么 a 的最大整数值是多少？

对于递归式 $T(n)$， $f(n) = n^2$， $n^{\log_b a} = n^{\log_2 7}$，且存在常数 ε 使得 $f(n) = O(n^{\log_b a-\varepsilon})$，满足主定理的第一种情况，因此 $T(n) = \Theta(n^{\log_2 7})$。

对于递归式 $T'(n)$， $f(n) = n^2$， $n^{\log_b a} = n^{\log_4 a}$。若要求出使算法 A' 比 A 快的最大整数 a，必定得使递归式 $T'(n)$ 满足主定理的第一种情况，因为只有第一种情况下 a 的值才是最大的。因此 $T'(n) = \Theta(n^{\log_4 a})$。

因为 A' 比 A 快，所以:

$$\begin{aligned} \Theta(n^{\log_4 a}) &< \Theta(n^{\log_2 7}) \\ n^{\log_4 a} &< n^{\log_2 7} \\ \log_4 a &< \log_2 7 \\ \frac{\log_2 a}{\log_2 4} &< \log_2 7 \\ \log_2 a &< \log_2 49 \\ a &< 49 \end{aligned} \tag{4.4}$$

因此若要 A' 比 A 更快，那么 a 的最大整数值是 48。

4.3-3 用主方法证明二分查找递归 $T(n) = T(n/2) + \Theta(1)$ 的解是 $T(n) = \Theta(\lg n)$。（二分查找的描述见练习 2.3-5）

$f(n) = \Theta(1)$， $n^{\log_b a} = 1$，所以 $f(n) = \Theta(n^{\log_b a})$，满足主定理的第二种情况，因此 $T(n) = \Theta(n^{\log_b a}\lg n) = \Theta(\lg n)$。

4.3-4 主方法能否应用于递归式 $T(n) = 4T(n/2) + n^2\lg n$？为什么？给出此递归式的渐近上界。

$f(n) = n^2\lg n$， $n^{\log_b a} = n^2$，尝试是否满足主定理的第三种情况。对任意正常数 ε，比值 $f(n)/n^{\log_b a} = \lg n$ 渐近小于 n^ε，因此不满足第三种情况，不能用主方法求解递归式。

现在用递归树方法求解此递归式的渐近上界。第 i 层结点的值为 $(n/2^i)^2\lg(n/2^i)$， $i \geqslant 0$，最后一层为 $T(1)$，即 $n/2^i = 1$，因此递归树一共有 $i = \lg n$ 层。

每一层的结点个数为 4^i，因此每一层的总代价为 $4^i \cdot (n/2^i)^2\lg(n/2^i) = n^2\lg(n/2^i)$，其中最后一层的总代价为 $4^{\lg n} \cdot T(1) = n^2T(1) = \Theta(n^2)$。于是可以得到整棵递归树的代价：

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n-1} n^2\lg\frac{n}{2^i} + \Theta(n^2) \\
&= \sum_{i=0}^{\lg n-1} n^2(\lg n - i) + \Theta(n^2) \\
&= \sum_{i=0}^{\lg n-1} n^2\lg n - n^2\sum_{i=0}^{\lg n-1} i + \Theta(n^2) \\
&= n^2(\lg n)^2 - n^2 \cdot \frac{(\lg n-1)\lg n}{2} + \Theta(n^2) \\
&= \frac{n^2\lg n(\lg n-1)}{2} + \Theta(n^2) \\
&= O(n^2\lg n) \qquad (4.5)
\end{aligned}$$

4.3-5 考虑在某个常数 $c < 1$ 时的规则性条件 $af(n/b) \leqslant cf(n)$，此条件是主定理第三种情况的一部分。举一个常数 $a \geqslant 1$， $b > 1$ 以及一个函数 $f(n)$，满足主定理第三种情况中的除了规则性条件之外的所有条件的例子。

思考题

4-1 递归式的例子

给出下列递归式的渐近上下界。假设 $T(n)$ 是个常数， $n \leqslant 2$。使所给出的界尽量紧确。并给出证明。

a) $T(n) = 2T(n/2) + n^3$

$f(n) = n^3$， $n^{\log_b a} = n$，有 $f(n) = \Omega(n^{\log_b a+2})$，且对足够大的 n，存在常数 $c < 1$ 满足 $af(n/b) \leqslant cf(n)$，其中 $1/4 \leqslant c < 1$，递归式符合主定理的第三种情况，因此 $T(n) = \Theta(f(n)) = \Theta(n^3)$。

b) $T(n)=T(9n/10)+n$

$f(n)=n$， $n^{\log_b a}=1$，有 $f(n)=\Omega(n^{\log_b a+1})$，且对足够大的 n，存在常数 $c<1$ 满足 $af(n/b)\leqslant cf(n)$，其中 $9/10\leqslant c<1$，递归式符合主定理的第三种情况，因此 $T(n)=\Theta(f(n))=\Theta(n)$。

c) $T(n)=16T(n/4)+n^2$

$f(n)=n^2$， $n^{\log_b a}=n^2$，有 $f(n)=n^{\log_b a}$，递归式符合主定理的第二种情况，因此 $T(n)=\Theta(n^{\log_b a}\lg n)=\Theta(n^2\lg n)$。

d) $T(n)=7T(n/3)+n^2$

$f(n)=n^2$， $n^{\log_b a}=n^{\log_3 7}$，因为 $\log_3 7<2$，所以存在常数 $\varepsilon>0$ 满足 $f(n)=\Omega(n^{\log_b a+\varepsilon})$。且对足够大的 n，存在常数 $c<1$ 满足 $af(n/b)\leqslant cf(n)$，其中 $7/9\leqslant c<1$，递归式符合主定理的第三种情况，因此 $T(n)=\Theta(f(n))=\Theta(n^2)$。

e) $T(n)=7T(n/2)+n^2$

$f(n)=n^2$， $n^{\log_b a}=n^{\lg 7}$，因为 $\lg 7>2$，所以存在常数 $\varepsilon>0$ 满足 $f(n)=O(n^{\log_b a-\varepsilon})$。递归式符合主定理的第一种情况，因此 $T(n)=\Theta(n^{\log_b a})=\Theta(n^{\lg 7})$。

f) $T(n)=2T(n/4)+\sqrt{n}$

$f(n)=\sqrt{n}$， $n^{\log_b a}=\sqrt{n}$，有 $f(n)=n^{\log_b a}$，递归式符合主定理的第二种情况，因此 $T(n)=\Theta(n^{\log_b a}\lg n)=\Theta(\sqrt{n}\lg n)$。

g) $T(n)=T(n-1)+n$

使用递归树方法进行求解， $T(n)=\sum_{i=1}^{n} i=\Theta(n^2)$。

再使用代换法进行证明，首先证明 $T(n)=O(n^2)$。假设 $T(n-1)\leqslant c(n-1)^2$，$c>0$，则:

$$\begin{aligned}T(n)&=T(n-1)+n\\&\leqslant c(n-1)^2+n\\&=cn^2-2cn+c+n\\&\leqslant cn^2\end{aligned}\tag{4.6}$$

当且仅当 $-2cn+c+n\leqslant 0$ 时成立，即 $n(1-2c)+c\leqslant 0$。此条件在 $c=1$ 及 $n\geqslant 1$ 时满足。

再证明 $T(n)=\Omega(n^2)$。同理可得 $T(n)\geqslant cn^2$，当且仅当 $-2cn+c+n\geqslant 0$ 时成立，此条件在 $0<c\leqslant 1/2$ 及 $n\geqslant 0$ 时满足。

综上， $T(n)=\Theta(n^2)$。

h) $T(n)=T(\sqrt{n})+1$

设 $n=2^m$，则 $T(2^m)=T(2^{m/2})+1$，再设 $T(2^m)=S(m)$，则 $S(m)=S(m/2)+1$。

运用主定理求解递归式 $S(m)$。 $f(m)=1$， $m^{\log_b a}=1$，满足主定理的第二种情况，则 $S(m)=\Theta(m^{\log_b a}\lg m)=\Theta(\lg m)$。因此 $T(n)=T(2^m)=S(m)=\Theta(\lg m)=\Theta(\lg\lg n)$。

4-2 找出所缺的整数

某数组 $A[1\,..\,n]$ 含有所有从 0 到 n 的整数，但其中有一个整数不在数组中。通过利用一个辅助数组 $B[0\,..\,n]$ 来记录 A 中出现的整数，很容易在 $O(n)$ 时间内找出所缺的整数。但在这个问题中，我们却不能由一个单一操作来访问 A 中的一个完整整数，因为 A 中的元素是以二进制表示的。我们所能用的唯一操作就是“取 $A[i]$ 的第 j 位”，这个操作所花时间为常数。

证明：如果访问数组 A 中信息的唯一方式是这种单一位操作，仍能在 $O(n)$ 时间内找出所缺的整数。 A 之外的任一完整整数仍可以由一个单一操作来访问。

先来看一个例子，假设这是一个 0~5 的序列，每一个数的二进制表示如下：

表 4.1　0~5 的二进制表示

数字（十进制）	对应的二进制	数字（十进制）	对应的二进制
0	000	3	011
1	001	4	100
2	010	5	101

为了更好地展现其中的规律，所有数的二进制表示都加上了前导 0，使得它们二进制表示的位数相同。这里规定二进制数左边第一位为“最高位”，右边第一位为“最低位”。可以得出，对于整数 n，它的二进制表示的位数最小为 $\lceil \lg(n+1) \rceil$。

先来观察所有数的最高位，最高位为“0”的数有 4 个，最高位为“1”的数有 2 个。其中最高位为“0”的数的个数恰好等于 $2^{(\lceil \lg(n+1) \rceil - 1)}$， n 代表序列的最大数，这里即是 5。

我们把最高位为“1”的两个数去掉，即 4 和 5，再来观察剩余数字的二进制表示：

表 4.2　0~3 的二进制表示

数字（十进制）	对应的二进制	数字（十进制）	对应的二进制
0	00	2	10
1	01	3	11

这次最高位为“0”的个数为 2，最高位为“1”的个数也为 2。而最高位为“0”的数的个数依旧等于 $2^{(\lceil \lg(n+1) \rceil - 1)}$，这里 $n = 3$。

现在回到最开始，仍然是一个 0~5 的序列，不过这次少了一个数：

表 4.3　0~5 的二进制表示

数字（十进制）	对应的二进制	数字（十进制）	对应的二进制
0	000	3	011
?	???	4	100
2	010	5	101

当然，我们都知道这个数是“1”，不过假设你是不知道的。用前面的方法再观察一次，这次最高位为“0”的数有 3 个，最高位为“1”的数有 2 个，你能猜出缺少的那个数的最高位是“0”还是“1”吗？肯定是“0”。刚才的几个例子已经能够得出对于一个 $0 \sim n$ 的序列，必定包含 $2^{(\lceil \lg(n+1) \rceil - 1)}$ 个最高位为“0”的数。

在确定了这个缺少的数的最高位为“0”之后，继续在其它所有最高位为“0”的数中观察：

表 4.4　0~3 的二进制表示

数字（十进制）	对应的二进制	数字（十进制）	对应的二进制
0	00	2	10
?	??	3	11

你恐怕已经知道这次该怎么做了，我们需要再次确定这个缺少的数的最高位。通过比较当前最高位为“0”的数的个数与 $2^{(\lceil \lg(n+1) \rceil - 1)}$ 的大小，便可以很方便地确定缺少数的最高位是“0”还是“1”。很顺利得，我们知道了这个缺少的数的最高位为“0”。以此类推，最终确定我们想要找的数就是“1”。

从上面的实例可以看出，通过不断地对序列所有数的最高位进行观察，便可以找到那个所缺的数字。这个过程用伪代码表示如下，注意这里对于数组下标的定义遵照 CLRS 书中的约定，即 $A[i]$ 代表数组 A 中的第 i 个元素，而实际编程中一般表示第 $i-1$ 个元素。

FIND-MISSING-INTEGER(A，n，M，i)

▷ A 代表缺少一个数的序列，n 代表序列 A 中的最大数（同时也是序列 A 的元素个数），M 代表存放所缺数各个二进制位的数组，i 代表上一次在 M 中存放的元素下标（初始为1）

▷ 递归式的结束条件，即序列 A 中只有一个数时，这时需要得到所缺数的最低位

if $n = 1$

then if $i \neq 2$

▷ 上一次在 M 中存放的二进制位并不是倒数第二位（如果把最低位看作倒数第一位），需要在存入最低位之前先把上一次存放的至倒数第二位之间的二进制位存入 M

```
                then for j ← 2 to i − 1
                          ▷ A[1]_j 表示序列 A 中第 1 个元素的二进制表示的第 j 位，下同
                          do M[j] ← A[1]_j
            M[1] ← 1 − A[1]_1
            return

  ▷ 初始化两个分别用来存放最高位为“0”和“1”的数的数组
  create arrays L[1..n] and R[1..n]

  ▷ 得到当前序列最大数的二进制表示的位数
  bits_num ← ⌈lg(n + 1)⌉

  ▷ 将最高位为“0”的数存放到数组 L 中，最高位为“1”的数存放到数组 R 中
  a ← 1
  b ← 1
  for j ← 1 to n
      do if A[j]_bits_num = 0
            then L[a] ← A[j]
                 a ← a + 1
            else R[b] ← A[j]
                 b ← b + 1

  ▷ 通过比较数组 L 的长度来确定所缺数的最高位是“0”还是“1”
  if length[L] < 2^(bits_num−1)
      then M[bits_num] ← 0
           i ← bits_num
           FIND-MISSING-INTEGER(L, 2^(bits_num−1) − 1, M, i)
      else if i = 1
                ▷ i = 1 即代表这是第一次执行
                then M[bits_num] ← 1
                     i ← bits_num
                     FIND-MISSING-INTEGER(R, n − 2^(bits_num−1), M, i)
                else ▷ last_bit 代表理论上上一次在 M 中存放的元素下标
                     last_bit ← ⌈lg(n − 2^(bits_num−1) + 1)⌉ + 1
                    ▷ 但有时还是会出现理论值（last_bit）与实际值（i）不相等的情况[1]，这时就需要把中间跳过的二进制位都补上
                     if i ≠ last_bit
```

[1] 比如$A[1..10]$，所缺数为 9，在第二次递归时就会发生这种情况。

```
        then for j ← last_bit to i − 1
                  do M[j] ← A[n]_j
    M[bits_num] ← 1
    i ← bits_num
    FIND-MISSING-INTEGER(R, n − 2^(bits_num−1), M, i)
```

通过上面的伪代码得出该算法的递归式为:

$$T(n) = T(2^{\lceil \lg(n+1) \rceil - 1} - 1) + cn \tag{4.7}$$

现在用代换法求解递归式，假设 $T(n) = O(n)$，当 $n = 2^{\lceil \lg(n+1) \rceil - 1} - 1$ 时成立，则:

$$\begin{aligned} T(n) &= T(2^{\lceil \lg(n+1) \rceil - 1} - 1) + cn \\ &\leqslant c(2^{\lceil \lg(n+1) \rceil - 1} - 1) + cn \\ &\leqslant c(2^{\lg(n+1)} - 1) + cn \\ &= c(n + 1 - 1) + cn \\ &= 2cn \end{aligned} \tag{4.8}$$

证得 $T(n) = O(n)$，Q.E.D.

算法参考资料: http://cs.nyu.edu/courses/summer08/G22.1170-001/hw02-soln.pdf

4-3 参数传递的代价

整个这本书中，我们都假定过程调用中的参数传递所花时间为常数，即使所传递的是个 N 个元素的数组也是一样。这个假设对大多数系统都是有效的，因为当参数为数组时，所传递的只是指向该数组的指针，而不是该数组本身。本题讨论三种参数传递策略:

1) 数组由一个指针来传递。时间 $= \Theta(1)$。
2) 参数数组通过复制而传递。时间 $= \Theta(N)$， N 是该数组的大小。
3) 一个数组在被传递时，仅拷贝被调用过程可能引用的数组的子域。若传递的是子树组 $A[p\,..\,q]$。时间 $= \Theta(p - q + 1)$。

a) 考虑在一个已排序的数组中找一个数的递归二叉查找算法（见练习 2.3-5)。针对上面的三种参数传递策略，给出最坏情况运行时间的递归式，并给出其解的上界。可以设 N 为原问题的规模， n 为子问题的规模。

b) 重做 2.3.1 节中 MERGE-SORT 的 a) 部分。

a) 二分查找用伪代码表示如下:

```
BINARY-SEARCH(A, n, p, q)
    ▷ A 为已排序数组，n 为想要查找的数字，p 为数组的下界，q 为数组的上界

    a ← ⌊(p + q) / 2⌋
```

```
if n = A[a]
    then return a
    else if p = q
             then return NIL

if n < A[a]
    then return BINARY-SEARCH(A, n, p, a)
    else return BINARY-SEARCH(A, n, a+1, q)
```

下面分别就上面三种参数传递策略进行讨论。

1) 这种策略就是通常考虑的情况，因此和4.3-3所说的一样，递归式为 $T(n) = T(n/2) + \Theta(1)$，递归式的解为 $T(n) = \Theta(\lg n)$。
2) 这种策略参数传递的时间从 $\Theta(1)$ 变成了 $\Theta(N)$，因此递归式也变为了 $T(n) = T(n/2) + \Theta(N)$，用主方法求出递归式的解为 $T(n) = O(N)$。
3) 因为子问题的规模为 n，因此递归式为 $T(n) = T(n/2) + \Theta(n)$，递归式的解为 $T(n) = O(n)$。

b) MERGE-SORT 也分为三种传递策略讨论：

1) 根据 2.3.2 节的分析，MERGE-SORT 的递归式为 $T(n) = 2T(n/2) + \Theta(n)$，递归式的解为 $T(n) = \Theta(n\lg n)$。
2) 虽然增加了参数传递的时间，但根据 2.3.1 节中 MERGE-SORT 的伪代码，递归式仍为 $T(n) = 2T(n/2) + \Theta(n)$，因此解依然是 $T(n) = \Theta(n\lg n)$。
3) 第三种情况其实和第二种很类似，虽然问题规模减小了，但分解 –合并所花的时间还是 $\Theta(n)$，解依旧是 $T(n) = \Theta(n\lg n)$。

4-4 更多递归式的例子

给出下列递归式 $T(n)$ 的渐近上下界。假设对足够小的 n， $T(n)$ 是常量。使所给出的界尽量紧确，并加以证明。

a) $T(n) = 3T(n/2) + n\lg n$

$f(n) = n\lg n$， $n^{\log_b a} = n^{\lg 3}$， $\because f(n) = O(n^{\log_b a - \varepsilon})$， $\therefore T(n) = \Theta(n^{\lg 3})$。

b) $T(n) = 5T(n/5) + n/\lg n$

方法同e)

c) $T(n) = 4T(n/2) + n^2\sqrt{n}$

$f(n) = n^2\sqrt{n}$， $n^{\log_b a} = n^2$， $\because f(n) = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{2+1/2})$，且有 $af(n/b) \leqslant cf(n)$， $\sqrt{2}/2 \leqslant c < 1$， $\therefore T(n) = \Theta(n^{5/2})$。

d) $T(n) = 3T(n/3 + 5) + n/2$

使用递归树方法求解。

e) $T(n)=2T(n/2)+n/\lg n$

使用递归树方法求解，第 i 层结点的值为 $\frac{n/2^i}{\lg(n/2^i)}$，最后一层为 $n/2^i=1$，因此递归树一共有 $\lg n$ 层。每一层的结点个数为 2^i，因此每一层的总代价为 $2^i \cdot \frac{n/2^i}{\lg(n/2^i)} = n/(\lg n-i)$。整棵递归树的代价为:

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n-1} \frac{n}{\lg n - i} \\
&= n\sum_{i=1}^{\lg n} \frac{n}{i} \text{（参考自"Instructor's Manual"，但不清楚是怎样得出的）} \\
&= n\sum_{i=1}^{\lg n} \frac{1}{i} \text{（参考自"Instructor's Manual"，但不清楚是怎样得出的）} \\
&= n\cdot\Theta(\lg\lg n) \text{（公式 A.7）} \\
&= \Theta(n\lg\lg n) \qquad (4.9)
\end{aligned}$$

证明略。(用数学归纳法分别证明 $T(n)\leqslant n(1+H_{\lfloor \lg n\rfloor})$ 与 $T(n)\geqslant n\cdot H_{\lceil \lg n\rceil}$， H_k 代表第 k 个调和级数。)

f) $T(n)=T(n/2)+T(n/4)+T(n/8)+n$

递归树一共有 $\lg n$ 层，每一层的总代价为 $(7/8)^i$，因此递归树的代价为:

$$\begin{aligned}
T(n) &= n\sum_{i=0}^{\lg n}\left(\frac{7}{8}\right)^i \\
&= \Theta(n) \qquad (4.10)
\end{aligned}$$

证明略。

g) $T(n)=T(n-1)+1/n$

这是调和级数的递归式，根据附录 A 中的公式 A.7，此递归式的渐近上下界为 $\Theta(\lg n)$。

h) $T(n)=T(n-1)+\lg n$

猜测此递归式的解为 $\Theta(n\lg n)$，需要分别证明 $T(n)\leqslant cn\lg n$ 和 $T(n)\geqslant cn\lg n$。

证明第一个不等式:

$$\begin{aligned}
T(n) &= T(n-1)+\lg n \\
&\leqslant c(n-1)\lg(n-1)+\lg n \\
&= cn\lg(n-1)-c\lg(n-1)+\lg n \\
&\leqslant cn\lg(n-1)-c\lg(n/2)+\lg n \\
&= cn\lg(n-1)-c\lg n+c+\lg n \\
&< cn\lg n-c\lg n+c+\lg n \\
&\leqslant cn\lg n \qquad (4.11)
\end{aligned}$$

当且仅当 $-c\lg n + c + \lg n \leqslant 0$ 时上式成立。

证明第二个不等式：

$$\begin{aligned} T(n) &= T(n-1) + \lg n \\ &\geqslant c(n-1)\lg(n-1) + \lg n \\ &= cn\lg(n-1) - c\lg(n-1) + \lg n \\ &\geqslant cn\lg(n/2) - c\lg(n-1) + \lg n \\ &= cn\lg n - cn - c\lg(n-1) + \lg n \\ &\geqslant cn\lg n \end{aligned} \tag{4.12}$$

当且仅当 $-cn - c\lg(n-1) + \lg n \geqslant 0$ 时上式成立。

因此 $T(n) = \Theta(n\lg n)$。

i) $T(n) = T(n-2) + 2\lg n$

方法同上。

j) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

参考4-1 的 h)及递归树方法。

4-5 斐波那契数

我们已在递归式 (3.21) 中定义了斐波那契数，现在进一步介绍它们的性质。我们将用生成函数技术来解斐波那契递归式。定义生成函数（或形式幂级数） $\mathcal{F}$ 如下

$$\mathcal{F}(z) = \sum_{i=0}^{\infty} F_i z^i = 0 + z + z^2 + 2z^3 + 3z^4 + 5z^5 + 8z^6 + 13z^7 + 21z^8 + \cdots$$

其中， F_i 是第 i 个斐波那契数。

a) 证明：$\mathcal{F}(z) = z + z\mathcal{F}(z) + z^2\mathcal{F}(z)$。

b) 证明

$$\mathcal{F}(z) = \frac{z}{1-z-z^2} = \frac{z}{(1-\phi z)(1-\hat{\phi} z)} = \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} - \frac{1}{1-\hat{\phi} z}\right)$$

其中， $\phi = \frac{1+\sqrt{5}}{2} = 1.61803\cdots$， $\hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803\cdots$。

c) 证明

$$\mathcal{F}(z) = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i)z^i$$

d) 证明：对 $i > 0$， $F_i = \phi^i/\sqrt{5}$ 截至最近的整数。（提示：$|\hat{\phi}| < 1$）。

e) 证明：对 $i \geqslant 0$， $F_{i+2} \geqslant \phi^i$。

a)

$$\begin{aligned} z\mathcal{F}(z) &= z^2 + z^3 + 2z^4 + 3z^5 + \cdots \\ z^2\mathcal{F}(z) &= \qquad\; z^3 + z^4 + 2z^5 + \cdots \\ \therefore z + z\mathcal{F}(z) + z^2\mathcal{F}(z) &= z + z^2 + 2z^3 + 3z^4 + 5z^5 + \cdots \\ &= \mathcal{F}(z) \end{aligned} \tag{4.13}$$

b) 根据 a) 的结论，求出 $\mathcal{F}(z) = \frac{z}{1-z-z^2}$，同时:

$$\begin{aligned} \frac{z}{(1-\phi z)(1-\hat{\phi} z)} &= \frac{z}{1-\hat{\phi}z-\phi z+\phi\hat{\phi}z^2} \\ &= \frac{z}{1-(\phi+\hat{\phi})z+\phi\hat{\phi}z^2} \\ &= \frac{z}{1-(\frac{1+\sqrt{5}}{2}+\frac{1-\sqrt{5}}{2})z+\frac{1+\sqrt{5}}{2}\cdot\frac{1-\sqrt{5}}{2}\cdot z^2} \\ &= \frac{z}{1-z-z^2} \end{aligned} \tag{4.14}$$

$$\begin{aligned} \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z}-\frac{1}{1-\hat{\phi}z}\right) &= \frac{1}{\sqrt{5}}\cdot\frac{(\phi-\hat{\phi})z}{(1-\phi z)(1-\hat{\phi}z)} \\ &= \frac{1}{\sqrt{5}}\cdot\frac{\sqrt{5}z}{1-z-z^2} \\ &= \frac{z}{1-z-z^2} \end{aligned} \tag{4.15}$$

即得证。

c) 根据公式 (3.23)，得证。

d) 略。

e) 根据 d) 的结论， $F_{i+2} = \phi^{i+2}/\sqrt{5}$，若结论成立，则:

$$\begin{aligned} \frac{\phi^{i+2}}{\sqrt{5}} &\geqslant \phi^i \\ \frac{\phi^2}{\sqrt{5}} &\geqslant 1 \\ \left(\frac{1+\sqrt{5}}{2}\right)^2 &\geqslant \sqrt{5} \\ 6 &\geqslant 2\sqrt{5} \end{aligned} \tag{4.16}$$

Q.E.D.

4-6 VLSI 芯片测试

Diogenes 教授有 n 个被认为是完全相同的 VLSI[2]芯片，原则上它们是可以互相测试的。教授的测试装置一次可测二片，当该装置中放有两片芯片时，每一片就对另一片作测试

[2]VLSI 代表“超大规模集成”，当今用于制作大多数微处理器的集成电路芯片技术。

并报告其好坏。一个好的芯片总能够报告另一片的好坏，但一个坏的芯片的结果是不可靠的。这样，每次测试的四种可能结果如下:

A 芯片报告	B 芯片报告	结论
B 是好的	A 是好的	都是好的，或都是坏的
B 是好的	A 是坏的	至少一片是坏的
B 是坏的	A 是好的	至少一片是坏的
B 是坏的	A 是坏的	至少一片是坏的

a) 证明若多于 $n/2$ 的芯片是坏的，在这种成对测试方式下，使用任何策略都不能确定哪个芯片是好的。假设坏的芯片可以联合起来欺骗教授。

b) 假设有多于 $n/2$ 的芯片是好的，考虑从 n 片中找出一个好芯片的问题。证明 $\lfloor n/2 \rfloor$ 对测试就足以使问题的规模降至近原来的一半。

c) 假设多于 $n/2$ 片芯片是好的，证明好的芯片可用 $\Theta(n)$ 对测试找出。给出并解答表达测试次数的递归式。

4-7 Monge 矩阵

一个 $m \times n$ 的实数矩阵 A，如果对所有 i， j， k 和 l， $1 \leqslant i < k \leqslant m$ 和 $1 \leqslant j < l \leqslant n$，有

$$A[i,j] + A[k,l] \leqslant A[i,l] + A[k,j]$$

那么，此矩阵 A 为 Monge 矩阵。换句话说，每当我们从 Monge 矩阵中挑出两行与两列，并且考虑行列交叉处的 4 个元素，左上角与右下角元素的和小于或等于左下角与右上角元素的和。例如下面的矩阵是一个 Monge 阵。

$$\begin{array}{ccccc}
10 & 17 & 13 & 28 & 23 \\
17 & 22 & 16 & 29 & 23 \\
24 & 28 & 22 & 34 & 24 \\
11 & 13 & 6 & 17 & 7 \\
45 & 44 & 32 & 37 & 23 \\
36 & 33 & 19 & 21 & 6 \\
75 & 66 & 51 & 53 & 34
\end{array}$$

a) 证明一个矩阵为 Monge 阵，当且仅当对所有 $i = 1$， 2， $\cdots$， $m-1$ 和 $j = 1$， 2， $\cdots$， $n-1$，有

$$A[i,j] + A[i+1,j+1] \leqslant A[i,j+1] + A[i+1,j]$$

（提示: 在“当”部分，对行、列分别使用归纳法。）

b) 下面的矩阵不是 Monge 阵。改变一个元素，把它变成 Monge 阵（提示：利用 a) 的结论）

$$\begin{matrix} 37 & 23 & 22 & 32 \\ 21 & 6 & 7 & 10 \\ 53 & 34 & 30 & 31 \\ 32 & 13 & 9 & 6 \\ 43 & 21 & 15 & 8 \end{matrix}$$

c) 假设 $f(i)$ 是第 i 行包含最左端最小值的列的索引值。证明对任何的 $m \times n$ Monge 矩阵，有 $f(1) \leqslant f(2) \leqslant \cdots \leqslant f(m)$。

d) 以下是一段关于分治算法的描述，用来计算 $m \times n$ Monge 矩阵 A 的每一行的最左端最小值:

构造一个包含所有 A 的偶数行的子矩阵 A'。递归地计算 A' 中每一行的最左端最小值。然后计算 A 中奇数行的最左端最小值。

解释如何能在 $O(m+n)$ 时间内计算出 A 的奇数行的最左端最小值？(假设偶数行的最左端最小值已知)

e) 写出 d) 部分所描述算法的运行时间的递归式，并证明其解为 $O(m+n\log m)$。

解答参考：http://www.cise.ufl.edu/class/cot5405sp09/hw1_soln.pdf

第五章　概率分析和随机算法

5.1　雇用问题

5.1-1 证明: 假设在程序 HIRE-ASSISTANT 的第 4 行中，我们总是能够决定哪一个应聘者最佳，这就蕴含我们知道应聘者排名的总次序。

Show that the assumption that we are *always able to* (一定可以) determine which candidate is best in line 4 of procedure HIRE-ASSISTANT implies that we know a *total order* (排序) on the ranks of the candidates.

必要性: 如果我们有所有应聘者能力的排名，那么显然可以知道当前的应聘者们中谁是最好的。

充分性: 如果我们始终能从当前的应聘者们找到最好的，有了这个操作，就可以对应聘者进行排序 (选择排序)，排序后得到应聘者能力排名。

5.1-2 描述 RANDOM(a, b) 过程的一种实现，它只调用 RANDOM(0, 1)。作为 a 和 b 的函数，你的程序的期望运行时间是多少？

算法描述如下:

```
RANDOM(a, b)
    num ← 0
    for i ← 0 to ⌈log₂(b − a)⌉
        do val ← num | (RANDOM(0, 1) ≪ i)
            if val + a ⩽ b
               then num ← val
               else return RANDOM(a, b)   ▷ Try again
    return a + num
```

上述算法调用 $\lceil log_2(b-a)\rceil + 1$ 次 RANDOM(0, 1)。第 i 次 RANDOM(0, 1) 的结果构成 num 的二进制第 i 位。若结果大于 b，则重新算。

时间复杂度为 $O(\lceil log_2(b-a)\rceil + 1)$。

错误算法:

```
RANDOM(a, b)
```

```
num ← a
for i ← 1 to b − a
    do num ← num + RANDOM(0, 1)
return num
```

以上算法调用 $b-a$ 次 RANDOM(0, 1) 并求和，这样得出的结果服从二项分布，而不是均匀分布。

5.1-3 假设你希望以各 $1/2$ 的概率输出 0 和 1。你可以自由使用一个输出 0 或 1 的过程 BIASED-RANDOM。它以概率 p 输出 1，以概率 $1-p$ 输出 0，其中 $0<p<1$，但是你并不知道 p 的值。给出一个利用 BIASED-RANDOM 作为子程序的算法，返回一个无偏向的结果，即以概率 $1/2$ 返回 0，以概率 $1/2$ 返回 1。作为 p 的函数，你的算法的期望运行时间是多少？

算法描述如下:

```
UNBIASED-RANDOM
    while TRUE
        do x ← BIASED-RANDOM
           y ← BIASED-RANDOM
           if x ≠ y
              then return x
```

上述算法只可能返回 0 和 1 两种值，其中返回 0 的概率为 $\Pr\{x=0 \text{ and } y=1\}=(1-p)p$，返回 1 的概率为 $\Pr\{x=1 \text{ and } y=0\}=p(1-p)$。因此返回 0 或 1 的概率是相等的。

算法的每一次迭代都相当于是一次 Bernoulli trial，成功的概率为 $2p(1-p)$。算法的运行时间取决于迭代次数，即 Bernoulli trial 取得一次成功所要进行的试验次数。这一系列试验的概率分布满足 geometric distribution，分布的期望即是迭代次数。根据 (C.31) 公式得出期望为 $1/(2p(1-p))$，因此算法的期望运行时间为 $O(1/(2p(1-p)))$。

算法参考资料: Instructor’s Manual

5.2 指示器随机变量

5.2-1 在 HIRE-ASSISTANT 中，假设应聘者以随机的顺序出现，正好雇用一次的概率是多少？正好雇用 n 次的概率又是多少？

Instructor’s Manual:

Since HIRE-ASSISTANT always hires candidate 1, it hires exactly once if and only if no candidates other than candidate 1 are hired. This event occurs when candidate 1 is the best candidate of the n, which occurs with probability $1/n$.

HIRE-ASSISTANT hires n times if each candidate is better than all those who were interviewed (and hired) before. This event occurs precisely when the list of ranks given to the algorithm is $\langle 1, 2, \ldots, n \rangle$, which occurs with probability $1/n!$.

正好雇佣一次事件 A：第一个面试者为能力最强的人，后面 $n-1$ 个人随意排列 (反正也不会被录用)。

$$\begin{aligned} \Pr\{A\} &= \frac{(n-1)!}{n!} \\ &= \frac{1}{n} \end{aligned} \tag{5.1}$$

正好雇用 n 次事件 B：面试者按照能力由弱到强顺序登场，这个排列出现概率为：

$$\Pr\{B\} = \frac{1}{n!} \tag{5.2}$$

5.2-2 在 HIRE-ASSISTANT 中，假设应聘者以随机的顺序出现，正好雇用两次的概率是多少？

第一个应聘者是肯定会被雇用的，但不可以是所有应聘者中最强的那个，因为这样就导致只能雇用一次，因此第一个应聘者的 rank 值只能是 $[1, n-1]$。当第一个应聘者取得区间中的任意一个值时， n 个应聘者都有可能等于那个值，概率为 $1/n$。

当第一个应聘者被雇用后，下一个被雇用的肯定是 rank 值为 n 的人。但在雇用这个最强的人（以下简称 Superman）之前不能把第一个应聘者淘汰了，这就得要求所有比第一个应聘者 rank 值高的人都必须在 Superman 之后面试。假设第一个应聘者的 rank 值为 i，于是就有 $n-i$ 个应聘者比他的 rank 值高，而 Superman 必须是这 $n-i$ 个应聘者中第一个被面试的人，这个事件发生的概率是 $1/(n-i)$，设这个事件用 F 表示。

设 E_i 代表第一个应聘者的 rank 值为 i 的事件， A 代表正好雇用两次这个事件，因此：

$$\begin{aligned} \Pr\{A\} &= \Pr\{F \cap (E_1 \cup E_2 \cup \cdots \cup E_{n-1})\} \\ &= \Pr\{(F \cap E_1) \cup (F \cap E_2) \cup \cdots \cup (F \cap E_{n-1})\} \\ &= \sum_{i=1}^{n-1} \Pr\{F \cap E_i\} \\ &= \sum_{i=1}^{n-1} \left(\frac{1}{n-1} \cdot \frac{1}{n} \right) \\ &= \frac{1}{n} \cdot H_{n-1} \end{aligned} \tag{5.3}$$

其中 H_{n-1} 代表第 $n-1$ 个调和级数。

5.2-3 利用指示器随机变量来计算掷 n 次骰子总和的期望值。

5.2-4 利用指示器随机变量来解帽子保管问题（hat-check problem）：有 n 位顾客，他们每个人给餐厅负责保管帽子的服务生一顶帽子。服务生以随机的顺序将帽子归还给顾客。请问拿到自己帽子的客户的期望数目是多少？

令指示器随机变量 X_i 对应于第 i 个顾客拿到自己帽子的事件，随机变量 X 表示 n 位顾客中拿到自己帽子的人数，则

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] \\
&= \sum_{i=1}^{n} \frac{1}{n} \\
&= 1 \qquad (5.4)
\end{aligned}$$

因此拿到自己帽子的客户的期望数目是 1。

5.2-5 假设 $A[1\,..\,n]$ 是由 n 个不同的数构成的数组。如果 $i < j$ 且 $A[i] > A[j]$，则称 (i, j) 对为 A 的逆序对（inversion）。（思考题 2-4 中有更多关于逆序对的例子。）假设 A 的元素形成 $<1, 2, \cdots, n>$ 上的一个均匀随机排列。利用指示器随机变量来计算 A 中逆序对的期望数目。

令指示器随机变量 X_k 对应于第 k 个 (i, j) 对是逆序对的事件，随机变量 X 表示 A 中逆序对的数目，其中 $\mathrm{E}[X_k] = 1/2$，则

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{k=1}^{\binom{n}{2}} X_k\right] \\
&= \sum_{k=1}^{\binom{n}{2}} \mathrm{E}[X_k] \\
&= \sum_{k=1}^{\binom{n}{2}} \frac{1}{2} \\
&= \binom{n}{2} \frac{1}{2} \\
&= \frac{n(n-1)}{4} \qquad (5.5)
\end{aligned}$$

因此 A 中逆序对的期望数目为 $n(n-1)/4$。

5.3 随机算法

5.3-1 Marceau 教授对引理 5.5 证明过程中使用的循环不变式表示异议。他对在第 1 次迭代之前循环不变式是否为真理提出质疑。他的理由是人们可以容易地宣称空数组不包含 0 排列。因此空数组包含 0 排列的概率应该是 0，所以在第 1 次迭代之前循环不变式无效。请改写过程 RANDOMIZE-IN-PLACE，使其相关的循环不变式在第 1 次迭代之前对非空数组仍适用，并未你的过程修改引理 5.5 的证明。

修改后的算法为:

```
RANDOMIZE-IN-PLACE(A)
    n ← length[A]
    swap A[1] ↔ A[RANDOM(1, n)]
    for i ← 2 to n
        do swap A[i] ↔ A[RANDOM(i, n)]
```

这样在第一次循环迭代之前，子数组 $A[1..1]$ 包含一个 1 排列的概率为 $(n-i+1)!/n! = 1/n$。

5.3-2 Kelp 教授决定写一个过程来随机产生非同一排列（identity permutation）的任意排列。他提出了如下的过程:

```
PERMUTE-WITHOUT-IDENTITY(A)
    n ← length[A]
    for i ← 1 to n − 1
        do swap A[i] ↔ A[RANDOM(i + 1, n)]
```

这段代码实现了 Kelp 教授的意图了吗？

答案是没有实现。虽然上面这个算法不会产生与原排列一模一样的随机排列，但同时也造成了其它一些排列无法产生。比如序列为 $<1,2,3>$，上面这个算法是绝对不会得到 $<1,3,2>$ 这种排列的。

5.3-3 假设不是将元素 $A[i]$ 与子数组 $A[i..n]$ 中的随机一个元素相交换，而是将它与数组任何位置上的随机元素相交换:

```
PERMUTE-WITH-ALL(A)
    n ← length[A]
    for i ← 1 to n
        do swap A[i] ↔ A[RANDOM(1, n)]
```

这段代码会产生均匀随机排列吗？为什么会？或为什么不会？

不会产生均匀随机排列。还是以序列 $<1,2,3>$ 为例，这个序列一共有 $3! = 6$ 种排列，而 PERMUTE-WITH-ALL 算法可以产生 $3^3 = 27$ 种情况，也就是说这 27 种情况有很多是重复的，而每一种排列的重复次数必须是 27/6 才行，显然这个除法得到的结果不可能是整数，因此 PERMUTE-WITH-ALL 是错误的。

5.3-4 Armstrong 教授建议使用下列过程来产生均匀随机排列:

```
PERMUTE-BY-CYCLIC(A)
    n ← length[A]
    offset ← RANDOM(1, n)
    for i ← 1 to n
```

```
        do dest ← i + offset
           if dest > n
              then dest ← dest − n
           B[dest] ← A[i]
     return B
```

证明任意元素 $A[i]$ 出现在 B 中任何特定位置的概率都是 $1/n$。然后通过证明其结果不是均匀随机排列来表明 Armstrong 教授错了。

一旦 $offset$ 确定，随机排列也就确定，而 $offset$ 取得任何一个值的概率为 $1/n$，因此任意元素 $A[i]$ 出现在 B 中任何特定位置的概率都是 $1/n$。

由于 $offset$ 只有 n 种取值，导致最终产生的随机排列也最多只有 n 种，这显然是不正确的，因为一共有 $n!$ 种。

5.3-5 证明程序 PERMUTE-BY-SORTING 的数组 P 中，所有元素都唯一的概率至少为 $1-1/n$。

5.3-6 解释如何实现算法 PERMUTE-BY-SORTING，来处理两个或更多优先级相同的情况。亦即，即使有两个或更多的优先级相同，你的算法也必须产生一个均匀随机排列。

5.4 概率分析和指示器随机变量的进一步使用

5.4-1 一个房间里必须要有多少人，才能让某人和你生日相同的概率至少为 1/2？必须要有多少人，才能让至少两个人生日为 7 月 4 日的概率大于 1/2？

5.4-2 假设将球投入到 b 个盒子里。每一次投掷都是独立的，并且每个球落入任何盒子的机会都相等。在至少有一个盒子包含两个球之前，期望的投球次数是多少？

5.4-3 在生日悖论的分析中，要求各生日彼此独立这一点是否是很重要的？或者，是不是只要每两个人的生日互相独立就足够了？证明你的答案。

5.4-4 一个聚会需要邀请多少人，才能让其中很可能有 3 个人的生日相同？

5.4-5 在大小为 n 的集合中，一个 k 串实际上是一个 k 排列的概率是多少？这个问题和生日悖论有什么关系？

5.4-6 假设将 n 个球投入 n 个盒子里，每次投球都是独立的，并且每个球落入任何盒子的机会都相等。空盒子的期望数量是多少？正好有一个球的盒子的期望数量又是多少？

5.4-7 为使序列长度的下界变得更加准确，请证明在 n 次均匀硬币的抛掷中，不出现比 $\lg n - 2\lg\lg n$ 更长的连续正面序列的概率小于 $1/n$。

思考题

5-1 概率计数

利用一个 b 位的计数器，一般只能计数到 2^b-1，而用 R. Morris 的概率计数法，则可以计到一个大得多的值，但代价是精度有所损失。

对 $i=0,1,\ldots,2^b-1$，令计数器的值 i 表示计数 n_i，且各 n_i 构成了一个非负的递增数列。假设计数器的初值为 0，代表计数 $n_0=0$。作用于计数器上的 INCREMENT 操作以概率的方式包含值 i。如果 $i=2^b-1$，则报告溢出错误。否则，计数器以概率 $1/(n_{i+1}-n_i)$ 增加 1，以概率 $1-1/(n_{i+1}-n_i)$ 保持不变。

对于所有的 $i\geqslant 0$，如果选择 $n_i=i$，则此计数器就是一个平常的计数器。如果选择 $n_i=2^{i-1}$（$i>0$），或 $n_i=F_i$（第 i 个斐波那契数，见 3.2 节），则会出现一些有趣的情况。

对这个问题，假设 n_{2^b-1} 已足够大，从而使得发生溢出错误的概率可以忽略。

a) 证明在执行了 n 次 INCREMENT 操作之后，计数器所表示的数的期望值正好是 n。

b) 对有计数器所表示的计数的方差的分析要依赖于 n_i 的序列。让我们来看一下简单情况: 对所有 $i\geqslant 0$，$n_i=100i$。在执行了 n 次 INCREMENT 操作之后，估计计数器所表示的数的方差。

a) 设 X_j 表示在第 j 次执行 INCREMENT 操作之后计数器增加的值，S_n 表示执行 n 次 INCREMENT 操作之后计数器表示的数。

因此 $S_n=X_1+X_2+\cdots+X_j$，$j=1,2,\ldots,n$，$\mathrm{E}[S_n]=\sum_{j=1}^{n}\mathrm{E}[X_j]$。而根据期望的定义

$$\begin{aligned}\mathrm{E}[X_j]&=\left((n_{j+1}-n_j)\cdot\frac{1}{n_{j+1}-n_j}\right)+\left(0\cdot\left(1-\frac{1}{n_{j+1}-n_j}\right)\right)\\&=1\end{aligned}\tag{5.6}$$

因此 $\mathrm{E}[S_n]=n$。

b) 定义随机变量 X_j 与 S_n，含义同上。因为 X_j 都是两两独立的变量，根据公式 (C.28)，$\mathrm{Var}[S_n]=\sum_{j=1}^{n}\mathrm{Var}[X_j]$。

$n_{i+1}-n_i=100(i+1)-100i=100$，根据公式 (C.26)

$$\begin{aligned}\mathrm{Var}[X_j]&=\mathrm{E}[X_j^2]-\mathrm{E}^2[X_j]\\&=\left(\left(100^2\cdot\frac{1}{100}\right)+\left(0\cdot\frac{99}{100}\right)\right)-1^2\\&=99\end{aligned}\tag{5.7}$$

因此 $\mathrm{Var}[S_n]=99n$。

第二部分

排序和顺序统计学

第六章　堆排序

6.1　堆

6.1-1 在高度为 h 的堆中，最多和最少的元素个数是多少？

最多为 $2^{h+1}-1$，最少为 2^h。

6.1-2 证明：含 n 个元素的堆的高度为 $\lfloor \lg n \rfloor$。

由上题的结论可得： $2^h \leqslant n \leqslant 2^{h+1}-1$，因此 $h \leqslant \lg n$，即 $h = \lfloor \lg n \rfloor$。

6.1-3 证明：在一个最大堆的某棵子树中，最大元素在该子树的根上。

6.1-4 在一个最大堆中，假设其所有元素都不相同，那么其最小元素可能存在于堆的哪些地方？

6.1-5 一个已排好序的数组是一个最小堆吗？

是

6.1-6 序列 <23, 17, 14, 6, 13, 10, 1, 5, 7, 12> 是一个最大堆吗？

不是

6.1-7 证明：当用数组表示存储了 n 个元素的堆时，叶子结点的下标是 $\lfloor n/2 \rfloor + 1$， $\lfloor n/2 \rfloor + 2$， $\cdots$， n。

6.2　保持堆的性质

6.2-1 利用图 6-2 作为范例，图示出 MAX-HEAPIFY(A, 3) 作用于数组 A = <27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0> 的过程。

6.2-2 由过程 MAX-HEAPIFY 开始，写出进行对应的最小堆操作的 MIN-HEAPIFY(A, i) 过程的伪代码，并比较 MIN-HEAPIFY 与 MAX-HEAPIFY 的运行时间。

```
MIN-HEAPIFY(A, i)
    l ← LEFT(i)
    r ← RIGHT(i)
    if l ⩽ heap-size[A] and A[l] < A[i]
```

6.3 建堆

6.4 堆排序算法

6.5 优先级队列

思考题

第七章　快速排序

第八章　线性时间排序

第九章 中位数和顺序统计学

第三部分

数据结构

第十章　基本数据结构

第十一章　散列表

第十二章　二叉查找树

第十三章　红黑树

13.1　红黑树的性质

13.1-1

13.1-2 对图 13-1 中的红黑树，画出调用 TREE-INSERT 插入关键字 36 后的结果。如果插入的结点被标为红色，所得的树是否还是一颗红黑树？如果该结点被标为黑色呢？

标为红色则还是红黑树，黑色就不是了。

13.1-3 定义松弛红黑树为满足红黑性质 1，3，4 和 5 的二叉查找树。换言之，根部可以是红色或是黑色。考虑一棵根是红色的松弛红黑树 T。如果将 T 的根部标为黑色而其他都不变，则所得到的是否还是一棵红黑树？

是

13.1-4 假设将一棵红黑树的每一个红结点"吸收"到它的黑色父结点中，来让红结点的子女变成黑色父结点的子女（忽略关键字的变化）。当一个黑结点的所有红色子女都被吸收后，其可能的度是多少？此结果树的叶子深度怎样？

可能的度有:

- 2，该结点的子女都是黑结点。
- 3，子女中一个黑结点，一个红结点。
- 4，子女都是红结点。

叶子的深度都一样。(根据红黑性质 5)

13.1-5 证明：在一棵红黑树中，从某结点 x 到其后代叶结点的所有简单路径中，最长的一条是最短一条的至多两倍。

直观上来说，最短路径肯定全是黑结点，而最长路径肯定是红黑结点交替出现。

由红黑性质 4 可以得出一条路径中至少有一半是黑结点，即 $\mathrm{bh}(x) \geqslant h/2$。于是 $h \leqslant 2\mathrm{bh}(x)$，其中 h 也是最长路径的长度，$\mathrm{bh}(x)$ 也是最短路径的长度，即得证。

13.1-6 在一棵黑高度为 k 的红黑树中，内结点最多可能有多少个？最少可能有多少个？

由上题可知 h 最大为 $2k$，因此内结点最多为 $\sum_{i=0}^{2k-1} 2^i = 4^k - 1$。

而 h 的最小值应为 k，因此内结点最少为 $\sum_{i=0}^{k-1} 2^i = 2^k - 1$。

13.1-7

13.2 旋转

13.2-1

13.2-2

13.2-3 **设在图 13-2 的左边一棵树中，a，b 和 c 分别为子树 α，β 和 γ 中的任意结点。如果将结点 x 左旋，则 a，b 和 c 的深度会如何变化？**

a 深度加 1，b 不变，c 减 1。

13.3 插入

13.3-1 **在 RB-INSERT 的第 16 行中，假设新插入的结点 z 是红的。注意如果将 z 着为黑色，则红黑树的性质 4) 就不会被破坏。那么我们为什么没有选择将 z 着为黑色呢？**

显然着为黑色，性质 5) 肯定被破坏了。

思考题

13-1 持久动态集合

a) 对一棵一般的持久二叉查找树，为插入一个关键字 k 或删除一个结点 y，确定需要改变哪些结点。

插入一个结点需要改变途径的所有结点，删除一个结点需要改变沿该结点向上至根的所有结点。

b) 请写出一个程序 PERSISTENT-TREE-INSERT，使得在给定一棵持久树 T 和一个要插入的关键字 k 时，它返回将 k 插入 T 后新的持久树 T'。

```
PERSISTENT-TREE-INSERT(T, k)
    用 root[T] 结点构造 root[T'] 结点
    用 k 构造结点 z
    x ← root[T]
    y ← root[T']
    a ← NIL
    while x ≠ NIL
        do a ← y
           if key[z] < key[x]
              then 用 left[x] 结点构造一个 tmp 结点
                   left[y] ← tmp
```

```
                  x ← left[x]
                  y ← left[y]
             else 用 right[x] 结点构造一个 tmp 结点
                  right[y] ← tmp
                  x ← right[x]
                  y ← right[y]
if a = NIL
    then root[T'] ← z
    else if key[z] < key[a]
            then left[a] ← z
            else right[a] ← z
return T'
```

c) 如果持久二叉查找树 T 的高度为 h，所实现的 PERSISTENT-TREE-INSERT 的时间和空间要求分别是多少？(空间要求与新分配的结点数成正比。)

时间要求为 $O(h)$，空间要求也为 $O(h)$。

d) 假设我们在每个结点中增加一个父亲结点域。这样一来，PERSISTENT-TREE-INSERT 需要做一些额外的复制工作。证明在这种情况下，PERSISTENT-TREE-INSERT 的时空要求为 $\Omega(n)$，其中 n 为树中的结点个数。

增加父亲结点域，插入新结点时就需要复制整棵树，因此时空要求为 $\Omega(n)$。

e) 说明如何利用红黑树来保证每次插入或删除的最坏情况运行时间为 $O(\lg n)$。

第十四章　数据结构的扩张

14.1　动态顺序统计

14.1-1

14.1-2

14.1-3 写出 **OS-SELECT** 的非递归形式。

```
OS-SELECT(x, i)
    r ← size[left[x]] + 1
    while i ≠ r
        do if i < r
                then x ← left[x]
                else x ← right[x]
            r ← size[left[x]] + 1
    return x
```

14.1-4 写出一个递归过程 **OS-KEY-RANK**(T, k)，使之以一棵顺序统计树 T 和某个关键字 k 为输入，返回在由 T 表示的动态集合中 k 的秩。假设 T 的所有关键字都是不同的。

```
OS-KEY-RANK(T, k)
    r ← size[left[x]] + 1
    if k = key[T]
        then return r
    else if k < key[T]
        then return OS-KEY-RANK(left[T], k)
    else return r + OS-KEY-RANK(right[T], k)
```

14.1-5 给定含 n 个元素的顺序统计树中的一个元素 x 和一个自然数 i，如何在 $O(\mathbf{lg}n)$ 时间内，确定 x 在该树的线性序中第 i 个后继？

```
OS-SUCCESSOR(T, x, i)
    r ← OS-RANK(T, x) + i
    return OS-SELECT(root[T], r)
```

14.1-6 在 OS-SELECT 或 OS-RANK 中，每次引用结点的 $size$ 域都仅是为了计算在以结点为根的子树中该结点的秩。假设我们将每个结点的秩（对于以该结点为根的子树而言）存于该结点自身之中。说明在插入和删除时如何来维护这个信息（注意这两种操作可能引起旋转）。

对于插入操作，第一阶段新结点若为经过结点左子树中的结点，则该经过结点的 $rank$ 值加 1。第二阶段，针对左旋情况， $rank[x]$ 不变， $rank[y] \leftarrow rank[x]+1$，右旋情况是对称的。

对于删除操作，第一阶段同插入操作，只是 $rank$ 值需要变为减 1。第二阶段与插入操作完全相同。

第四部分

不是 CLRS 附录的附录

附录 A　排序算法比较

算法	平均时间复杂度	空间复杂度	稳定性	原地排序
插入排序	$O(n^2)$	$O(1)$	稳定	$\surd$
合并排序	$O(n\lg n)$	$O(n)$	稳定	$\times$
堆排序	$O(n\lg n)$	$O(1)$	不稳定	$\surd$
快速排序	$O(n\lg n)$	$O(\lg n)$	不稳定	$\surd$
计数排序	$O(n)$		稳定	$\times$
基数排序	$O(n)$		稳定	$\times$