郑源泽19307130077hw3

5.2.4

Use indicator random variables to solve the following problem, which is known as the hat-check problem. Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

随机变量X表示得到正确帽子的顾客数量

定义一个指示器随机变量 $X_i = I\{customer\ i\ gets\ his\ own\ hat\}$

$X = X_1 + X_2 + X_3 + \ldots + X_n$

由于帽子顺序是随机的， Pr{Xi=1}=1/n，由引理5.1，E[Xi]=1/n

E[X] = E[X1]+E[X2]+E[X3]+...+E[Xn] = 1

5.3.4

5.3-4

Professor Armstrong suggests the following procedure for generating a uniform random permutation:

```
PERMUTE-BY-CYCLIC(A)
n = A.length
let B[1..n] be a new array
offset = RANDOM(1, n)
for i = 1 to n
    dest = i + offset
    if dest > n
        dest = dest - n
    B[dest] = A[i]
return B
```

Show that each element $A[i]$ has a $1/n$ probability of winding up in any particular position in B. Then show that Professor Armstrong is mistaken by showing that the resulting permutation is not uniformly random.

这个算法首先确定一个随机的位移量offset，在1-n之间，而后相当于将整个A数组向右平移了offset个单位，超出范围的补在新数组的前面。

由于offset是随机的，1-n的offset每个的概率是1/n, 那么不失一般性，以原数组中第 i 个元素为例，

他将在新数组中出现的位置是A[((i+offset-1) mod n)+1]，对于每一个不同的offset，新位置等可能地涵盖了B中的任何一个位置

因此，A[i]出现在B中任何位置的概率都是1/n

然而，这个算法不是uniformly random，因为它只包含了n种可能的情况（即A中的元素平移），每种概率1/n。

而全随机的算法会出现n!种不同的结果，每种概率1/n！。

5-1

5-1 Probabilistic counting

With a b-bit counter, we can ordinarily only count up to $2^b - 1$. With R. Morris's ***probabilistic counting***, we can count up to a much larger value at the expense of some loss of precision.

We let a counter value of i represent a count of n_i for $i = 0, 1, \ldots, 2^b - 1$, where the n_i form an increasing sequence of nonnegative values. We assume that the initial value of the counter is 0, representing a count of $n_0 = 0$. The INCREMENT operation works on a counter containing the value i in a probabilistic manner. If $i = 2^b - 1$, then the operation reports an overflow error. Otherwise, the INCREMENT operation increases the counter by 1 with probability $1/(n_{i+1} - n_i)$, and it leaves the counter unchanged with probability $1 - 1/(n_{i+1} - n_i)$.

If we select $n_i = i$ for all $i \geq 0$, then the counter is an ordinary one. More interesting situations arise if we select, say, $n_i = 2^{i-1}$ for $i > 0$ or $n_i = F_i$ (the ith Fibonacci number—see Section 3.2).

For this problem, assume that n_{2^b-1} is large enough that the probability of an overflow error is negligible.

a. Show that the expected value represented by the counter after n INCREMENT operations have been performed is exactly n.

b. The analysis of the variance of the count represented by the counter depends on the sequence of the n_i. Let us consider a simple case: $n_i = 100i$ for all $i \geq 0$. Estimate the variance in the value represented by the register after n INCREMENT operations have been performed.

a. 设 X_j 为第j次 INCREMENT 操作后计数器代表的 value 的增量

假设 INCREMENT 之前，counter 的真实值是 i，它代表 value 为 n_i

那么，它有 $\frac{1}{n_{i+1}-n_i}$ 的概率增加，这样它代表的 value 为 n_{i+1}

有 $1-\frac{1}{n_{i+1}-n_i}$ 的概率不变

$E[X_j] = (0(1-\frac{1}{n_{i+1}-n_i}) + ((n_{i+1}-n_i)\cdot(\frac{1}{n_{i+1}-n_i})) = 1$

∴ n次 INCREMENT 后的增量期望为 $E[X_0]+\cdots+E[X_n] = n$

设最终值为 V_n，则 $E[V_n] = 0+n = n$

b. $Var[V_n] = Var[X_1+X_2+\cdots+X_n]$，由于 $X_1, X_2 \cdots X_n$ 是成对独立的

$Var[V_n] = Var[X_1]+Var[X_2]+\cdots+Var[X_n]$

$n_i = 100i \rightarrow$ 有 $\frac{1}{100}$ 的概率增加 100，有 $\frac{99}{100}$ 概率保持不变

$Var[X_j] = E(X_j^2) - E^2(X_j) = ((0^2\cdot\frac{99}{100}) + (\frac{1}{100}\cdot 100^2)) - 1^2 = 99$

∴ $Var[V_n] = 99n$

Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a *local minimum* if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T, but the labeling is only specified in the following *implicit* way: for each node v, you can determine the value x_v by *probing* the node v. Show how to find a local minimum of T using only $O(\log n)$ *probes* to the nodes of T.

SOLUTION(A, i)

```
if i.value < i.left.value AND i.value < i.right.value //如果根节点满足条件直接返回
    return i
else if i > (Integer)i.heapsize/2 //如果当前节点是叶子，直接返回
    return i
```

```
// 在小于根节点的儿子节点中选择一个，递归调用即可
else if i.value > i.left.value
    return solution(A, i.left)
else if i.value > i.right.value
    return solution(A, i.right)
```

由于每次调用SOLUTION都是向下进一层或者返回，因此O(lgn)

6.4-3

6.4-3
What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

increasing

heapsort算法首先建立最大堆，花费O(n)，而后n次循环，每次调用MAX-HEAPIFY花费O(lgn)

总花费： $O(n) + O(nlgn) = O(nlgn)$

decreasing

即使已经是最大堆，BUILD-MAX-HEAP还是会从第一个非叶节点向上扫描并调用MAX-HEAPIFY，MAX_HEAPIFY花费常数级别时间，但是BUILD-MAX-HEAP仍然和输入是线性关系

对于堆排序循环中，仍然是每次取得最大值和最后元素交换，调用MAX-HEAPIFY花费O(lgn)

综上，总花费： $O(n) + O(nlgn) = O(nlgn)$

6.4-4

6.4-4
Show that the worst-case running time of HEAPSORT is $\Omega(n \lg n)$.

最坏情况：每次调用MAX-HEAPIFY，都会覆盖整个树的高度。

$$\sum_{i=1}^{n-1} \lfloor \lg i \rfloor \leq \sum_{i=1}^{n-1} \lg i = \lg((n-1)!) = \Theta((n-1)\lg(n-1)) = \Omega(n \lg n)$$