**概率论与数理统计课程小论文**

随机算法的概率分析

|  |  |
| --- | --- |
| **班 号** | **2201110** |
| **学 号** | **220111012** |
| **姓 名** | **王靳** |
| **日 期** | **2023.11.17** |
| **成 绩** |  |

**摘 要**

我们先对一个平凡的算法进行了分析，发现了其缺点。我们再引入洗牌算法，让输入不再受制于人。其中洗牌算法有两种做法，但第二种洗牌算法更优一些。洗牌算法中，需要一个随机数发生器，我们设计了 “线性同余法”，最后我们对 “线性同余法” 产生的结果进行统计分析。

关键词：随机算法、概率分析、线性同余法、洗牌算法、随机数发生器、雇佣问题

## 课题背景及研究的目的和意义

我们会经常对数组进行排序操作，在“快速排序算法”中，会随机的选取一个值，作为基准，再递归的进行快速排序。“随机选取一个值”，这就涉及到对算法进行概率分析：这个算法的时间复杂度的期望是多少？我这个随机的值是什么样的分布？如果不知道分布，我们能不能用统计的方法来评估这个分布？

**正文**

为了明确下面我们所要阐述的共同话题，我们先定义以下内容：

#### 定义 1 - 大 O（渐进上界）

这种渐进记号有其实际意义，通常用来衡量一个算法运行时间的上界

#### 定义 2 - 全序关系

全序关系具有以下性质：[6]

1. 反对称性： 若 a <= b 且 b <= a 则 a = b
2. 传递性：若 a<= b 且 b <= c 则 a <= c
3. 完全性：a <= b 或 b <= a

#### 定义 3 - 循环不变式

循环不变式用来证明“迭代”的正确性

循环不变式的性质：

1. 初始化：It is true prior to the first iteration of the loop
2. 保持：If it is true before an iteration of the loop, it remainds true before next iteration
3. 终止：The loop terminates, and when it terminates, the invariant-usually along with the reason that the loop terminated-gives us a useful property that helps show that the algorithm is correct.

具体严格的定义，涉及到“霍尔逻辑”[7]，本文不做过多讨论。我们只需要知道，这个“循环不变式”与“数学归纳法”类似即可。

#### 定义 4 - 指示器随机变量

给定一个样本空间 S 和一个时间 A，那么时间 A 对应的“指示器随机变量” I{A} 定义为

性质：

## 雇佣问题

问题描述：[1]

假如你要雇用一名新的办公助理。你先前的雇用尝试都失败了，千是你决定找一个雇用代理。雇用代理每天给你推荐一个应聘者。你面试这个人，然后决定是否雇用他。你必须付给雇用 代理一小笔费用，以便面试应聘者。然而要真的雇用一个应聘者需要花更多的钱，因为你必须辞 掉目前的办公助理，还要付一大笔中介费给雇用代理。你承诺在任何时候，都要找最适合的人来 担任这项职务。因此，你决定在面试完每个应聘者后，如果该应聘者比目前的办公助理更合适， 就会辞掉当前的办公助理，然后聘用新的。你愿意为该策略付费，但希望能够估算该费用会是多少。

令：有 n 个人来应聘，前来应聘的人的编号为 [1..n]，面试的费用是 ，雇佣的费用是 ，其中

对于上面流程，我们可以抽象出一个 HIRE-ASSISTANT(n)算法：

HIRE-ASSISTANT(n):  
 best = 0  
 for i = 1 to n:  
 interview candidate i // 花费c\_i的代价  
 if candidate i is better than candidate best:  
 best = i  
 hire candidate i // 花费c\_h的代价(高昂)

上面这个算法中的if candidate i is better than candidate best:意味着，这 n 个应聘者的质量，是一种全序关系。令rank(i)表示为第 i 个应聘者的质量，那么这 n 个人接踵而至的面试，可以看成一个输入序列A=<rank(1), rank(2)..rank(n)>，其中 A 是<1, 2.. n>的一种排列方式

然后我们就要开始评估这个算法了。

如果应聘人员的质量呈现递增的趋势，那么就会导致出现最坏的结果，每一个人都需要支付“雇佣费”（中介费），那么需要支付的费用为

如果应聘人员的质量，第一个就是最佳人选，那么就只用花费最小的代价即可，也就是

这个时候，就会出现一个令人困惑的地方：如果我作为公司的管理层，我需要支付的中介费是与 A 强相关的，并且我们这个算法显示：A 并不受公司的控制，万一那个“中介所”中所有的应聘者串通好，想要赚取其中的份中介费呢？这就很麻烦了。

所以，我们不希望算法的代价由输入决定。我们可以对输入序列进行 shuffle 的操作。

RANDOMIZED-HIRE-ASSISTANT(n):  
 randomly permute the list of candidates  
 best = 0  
 for i = 1 to n:  
 interview candidate i  
 if candidate i is better than candidate best:  
 best = i  
 hire candidate i

上面这段“伪代码”中，我们暂且认为 shuffle 算法是正确的，能够产生均匀分布的随机序列。后面我们会在对 shuffle 进行讨论。

“均匀分布的随机序列”，意思是：我们都知道由 n 个元素组成的序列，有种排列方式，所以对于一个确定的序列 A，他能在全排列中被选择的概率是

那么，我们可以求出来这个 RANDOMIZED-HIRE-ASSISTANT 的“代价的期望”

令随机变量，那么需要支付的中介费的次数是

又，第 i 位面试者被雇佣，意味着他/她是前 i 位面试者中，最优秀的质量最高的那个（质量是全序关系），这个概率是

则，

因此，我们可以知道，公司需要支付的费用的期望是

但是我们仍然有一个问题，就是如何生成均匀分布的随机排列呢？

## 均匀分布的随机排列 1 - 通过排序生成随机序列

伪代码如下：[1]

PERMUTE-BY-SORTING(A):  
 n = len(A)  
 let P[1..n] be a new array  
 for i = 1 to n:  
 P[i] = RANDOM(1, n^3) // 闭区间  
 sort A, using P as sort keys

sort A, using P as sort keys，意思是：假设我们的排序是通过交换来实现的（假设顺序是从小到大），那么有：

if P[i] > P[j]:  
 swap(P, i, j)  
 swap(A, i, j) // A 是 P 的 “卫星数据”

(上面的 P 是 priority 的意思)

这个算法看起来没太多问题，实际上在 n 较大的时候，也确实是对的。下面我们进行概率分析与算法正确性证明。

#### 定理 - 算法的正确性 - 序列是均匀分布的

假设上面通过P[i] = RANDOM(1, n^3)生成的“优先级”都不同，那么PERMUTE-BY-SORTING(A)产生的序列是均匀随机的

即证：对于每个序列，它(序列)出现的概率是$1/n!$

解：

A[1..n]并不是一个确定的排列，因为它是外部的输入。我们找到A[1..n]的任意一个确定的排列B[1..n]。

于是，我们只需要证明，sort A, using P as sort keys，sorting 完成以后， 即可

（这里的 Pr 就是概率的意思，因为上面 P 已经表示为“优先级”了）

因为我们是 sort A, using P as sort keys，于是乎，B[i]一定是P中第 $i$ 小的元素。

设：B[i]分配到第 i 小的“优先级”为事件 $E\_i$

我们只需要证明：可

又根据 n 变量的乘法公式，我们有：

因此，只需要证明可

其中 表示：B[i]是B[i..n]中，优先级最小的元素，这个事件的概率是 （B[i..n]有 $n-i+1$ 个元素，在这些元素中，选取“优先级”最小的元素，放入B[i]这个位置）

然后我们将 代入乘法公式，得到 得证

但是我们上面这个算法是有问题的：「B[i..n]有 n – i + 1 个元素，在这些元素中，选取“优先级”最小的元素，放入B[i]这个位置。」如果我这里，“优先级”最小的元素不止 1 个呢？这也就是我们的「P[i] = random(1, n^3)」中n^3的来源。

#### 引理 - 优先级尽量不相同

通过P[i] = random(1, n^3)得到的 P 数组，所有元素都唯一的概率至少是$1 - 1/n$

解：

令：P[i] = P[j] 为事件 $X\_{ij}$，由于P[i]和P[j]是 1 到 n^3 之间的随机数，于是我们有：

（P[i] = r 与 P[j] = s，显然是独立事件，其中 r 与 s 可以是任意值，可以相等，也可以不相等）。

令 P 中存在某两个相等的元素为事件 $X$，且任意两个 $X\_{ij}$ 之间不相交，故而：

因此，所有元素都唯一的概率 。

当 n 足够大的时候，两个元素的"优先级"相等就是小概率事件。

对于算法分析而言，上面这个算法需要再分配一个数组，并且还要进行排序，不免有些麻烦。并且上面，只是大概率的会生成每个元素都不相同的优先级 P，从而大概率的让每个导出序列 B 的概率是1/n!。

下面我们将设计一个更好的算法并证明其正确性。

## 均匀分布的随机排列 2 - Fisher–Yates shuffle Algorithm

伪代码如下：[4]

RANDOMIZE-IN-PLACE(A):  
 n = len(A)  
 for i = 1 to n:  
 swap A[i] with A[RANDOM(i, n)]

显然，这个算法产生的序列是随机的，但是序列的分布是否是均匀的呢？需要我们来证明。

#### 定理 - 证明 Fisher-Yates shuffle 算法：

证明：在 for 循环的每次迭代开始前，对每个可能的 (i-1) 排列，子数组A[1..i-1]包含这个 (i-1) 排列的概率是：

一种**错误**的思路：

分母是 $n!$，显然，因为是 n 个元素的全排列，即

分子是 ，因为前面的 个元素已经确定，剩下有 个元素尚未确定，一共有 种不同的排列方式

因为，我们相当于是先用了要证明的东西：等概率

**正确**的分析思路：

这个是一个增量算法（每次循环都比上一次多做了“一点”事情），证明增量算法，需要我们使用：循环不变式。

1. 初始化：i=1，A[1..0]包含 0 排列的概率是 1，平凡的
2. 保持：

* 第 i 次迭代时，考虑任意一个特殊的 i 排列<x1, x2..x3>，令 A[1..i-1]包含<x1, x2..x3>这个 (i-1) 排列为事件 ，记在 A[i] 位置放置 为事件 ，下面证明：（这是下一次循环的）
* 假设第 i 次迭代前不变式成立，则
* 在 发生的条件下， 发生的概率是：，含义就是：从剩下的 n-i+1 个元素中，选择到了 的概率是 }，选择完了以后，将 交换到 A[i]
* 最后，根据乘法公式：}，状态保持，得证

1. 终止：最后一次循环结束后，，根据循环不变式，对每个可能的 n 排列，数组A[1..n]包含这个 n 排列的概率是，是均匀随机排列

我们现在是分析了这个算法的正确性。

## 线性同余法（Linear congruential generator） - 伪随机数生成器 - 统计分析

上面，我们论述了两种洗牌算法。但是，实际上还是有一点令人疑惑的：

1. 在证明基于 sorting 的随机序列的过程中，有：

* 这里我们是承认了P[i] = RANDOM(1, n^3)中，RANDOM 生成的就是均匀分布的随机数

1. 在证明 Fisher–Yates shuffle Algorithm 算法的过程中，有：

* 这里，我们还是承认了swap A[i] with A[RANDOM(i, n)]中，RANDOM 生成的就是均匀分布的随机数

但是如果 RANDOM 生成的不是“均匀分布的随机数”，上面的证明也是无稽之谈。

那么我们要如何生成均匀分布的随机数呢？我们有以下伪代码：[3]

RANDOM(seed, num, a, c, m):  
 // a是乘法常数，c是增量，m是模数  
 x = seed  
 while n < num:  
 x = (a \* x + c) % m  
 return

用一个公式可以总结 ，其中是初始化的种子

由于本人并没有学过《密码学》与《数论》，因此这里便不会使用 数论的分析方法 来证明 LCG 产生的是均匀的随机序列；但是我们会使用 “统计学”的方法 来验证这种方法。

我们从**方差**、**均值**这两个角度，将 LCG 与 "均匀分布" 进行比较。[2]

下面是一段 python 代码，我们让这个 LCG 算法，生成 1w 个从 0 到 99 的元素，并且，我们将 LCG 相关的参数，都从 1 到 99 进行了遍历。我们将不同参数得出来的均值与方差求平均，运行结果，得到：方差的均值为 824.912，平均值的均值为 49.592。而 0 到 99 的均匀分布，其方差为 833.25，均值为 49.5。[5]可见，我们的这个 LCG 算法，或许有一定的合理性。至于是否正确，需要我本人继续深造。

import numpy[8] as np  
class LCG:  
 def \_\_init\_\_(self, seed, a, c, m):  
 self.state = seed  
 self.a = a  
 self.c = c  
 self.m = m  
  
 def next(self):  
 self.state = (self.a \* self.state + self.c) % self.m  
 return self.state  
  
  
vars = []  
means = []  
a = [i for i in range(1, 100)]  
c = [i for i in range(1, 100)]  
seed = [i for i in range(1, 100)]  
n = 10000 # 生成1w个数  
m = 100  
  
for x in a:  
 for y in c:  
 for z in seed:  
 l = LCG(z, x, y, m)  
 arr = np.array([l.next() for \_ in range(n)])  
 temp = np.array(arr)  
 vars.append(temp.var())  
 means.append(temp.mean())  
  
print(np.array(vars).mean())  
print(np.array(means).mean())

## 总结

到目前为止，我们完整的解决了 “雇佣问题”。我们先对一个平凡的算法进行了分析，发现了其缺点。我们再引入洗牌算法，让输入不再受制于人。其中洗牌算法有两种做法，但第二种洗牌算法更优一些。洗牌算法中，需要一个随机数发生器，我们设计了 “线性同余法”，最后我们对 “线性同余法” 产生的结果进行统计分析。

**参考文献**

[1] https://mitpress.mit.edu/9780262046305/introduction-to-algorithms 算法导论

[2] 茆诗松. 概率论与数理统计

[3] https://en.wikipedia.org/wiki/Linear\_congruential\_generator 线性同余法

[4] https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates\_shuffle Fisher-Yates洗牌算法

[5] https://zhuanlan.zhihu.com/p/357524886 概率论之使用线性同余法生成任意随机分布随机数

[6] https://en.wikipedia.org/wiki/Total\_order 全序关系

[7] https://en.wikipedia.org/wiki/Hoare\_logic 霍尔逻辑

[8] https://numpy.org/doc/stable numpy文档