

Algorithm - P35

Monge 阵列 (Monge array, Monge matrix), 指对于 $m \times n$ 的实数矩阵 A

对于正整数 $1 \leq i < k \leq m, 1 \leq j < l \leq n$, 都满足

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

即对于矩阵中任意两行与两列确定的 4 个元素

左上与右下之和小于等于左下与右上之和, 或者说主对角线之和小于等于反对角线之和

对于 $m \times n$ 的实数矩阵 A , A 是 Monge 阵列 当且仅当

对任意 $1 \leq i < m, 1 \leq j < n$, 有 $A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j]$

证明过程有: 如果 A 是 Monge 阵列, 则 A 满足

对任意 $1 \leq i < k \leq m, 1 \leq j < l \leq n$, 有 $A[i, j] + A[k, l] \leq A[i, l] + A[k, j]$

则且又 $k = i+1, l = j+1$, 即有 对任意 $1 \leq i < m, 1 \leq j < n$

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

如果 A 满足对任意 $1 \leq i < m, 1 \leq j < n$, 有 $A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j]$

对于给定的 $1 \leq j < n$, 考虑连续两行的情形

基础步骤: 对任意 $1 \leq i < m$, 有 $P(i, j, i+1, j+1)$ 为真

递归步骤: 假设对任意 $1 \leq i < k < m-1$, $P(i, j, k+1, j+1)$ 为真

则有 $A[i, j] + A[k+1, j+1] \leq A[i, j+1] + A[k+1, j]$

考虑 $k+1$ 的情形, 有 $k+1 < m, k+2 \leq m$

于是有 $A[k+1, j] + A[k+2, j+1] \leq A[k+1, j+1] + A[k+2, j]$

则 $A[i, j] + A[k+1, j+1] + A[k+1, j] + A[k+2, j+1]$

$\leq A[i, j+1] + A[k+1, j] + A[k+1, j+1] + A[k+2, j]$

即 $A[i, j] + A[k+2, j+1] \leq A[i, j+1] + A[k+2, j]$, 即 $P(i, j, k+2, j+1)$ 为真

根据数学归纳法, 对于任意 $1 \leq i < k \leq m, 1 \leq j < n$

以此为基础步骤: $A[i, j] + A[k, j+1] \leq A[i, j+1] + A[k, j]$

递归步骤: 假设对任意 $1 \leq j \leq l < n-1$, $P(i, j, k, l+1)$ 为真

则有 $A[i, j] + A[k, l+1] \leq A[i, l+1] + A[k, j]$

考虑 $l+1$ 的情形, 有 $l+1 < l+2 \leq n$,

于是有 $A[i, l+1] + A[k, l+2] \leq A[i, l+2] + A[k, l+1]$

$A[i, j] + A[k, l+1] + A[i, l+1] + A[k, l+2]$

$\leq A[i, l+1] + A[k, j] + A[i, l+2] + A[k, l+1]$

即 $A[i, j] + A[k, l+2] \leq A[i, l+2] + A[k, j]$, 即 $P(i, j, k, l+2)$ 为真

根据数学归纳法, 对于任意 $1 \leq i < k \leq m, 1 \leq j < l \leq n$

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

即 矩阵 A 为 Monge 阵列

Algorithm - P36

Monge 阵列
最左最小元素

对于任意 $m \times n$ 的 Monge 阵列，且令 $f(i)$ 表示第 i 行的最左最小的元素列号

则有 $f(1) \leq f(2) \leq \dots \leq f(m)$

证明过程有：对于任意行号 i ，有 $1 \leq f(i) \leq n$

如果存在 $1 \leq i < j \leq m$ ，使得 $f(i) > f(j)$

则可知 $A[i, f(j)] > A[i, f(i)]$

且 $A[j, f(j)] > A[j, f(i)]$

于是 $A[i, f(j)] + A[j, f(i)] > A[i, f(i)] + A[j, f(j)]$

与 Monge 阵列的性质相矛盾

于是有 $1 \leq f(1) \leq f(2) \leq \dots \leq f(m) \leq n$

可以由此设计求 Monge 阵列每一行最左最小元素的分治算法

分解

: 提取 $m \times n$ 的矩阵阵的偶数行组成子矩阵 A'

解决

: 求子矩阵 A' 中每一行最左最小元素

合并

: 通过已知的偶数行的最左最小元素的列号以求出奇数行的最左最小元素

def mongeArray(carr):

if len(carr) == 1:

 ind, min = 0, float('inf')

 for i, ele in enumerate(carr[0]):

 if ele < min:

 ind, min = i, ele

 return [ind]

else:

 ret = [0 for i in range(len(carr))]

 even_ret = mongeArray(carr[1::2])

 for i, ele in enumerate(even_ret):

 ret[2*i+1] = ele

 left = 0

 根据已获得的偶数行最左最小元素求奇数行的最左最小元素

 for i, line in enumerate(carr):

 if i % 2 == 0:

 min = float('inf')

 for ind in range(left, right):

 if arr[i][ind] < min:

 min = arr[i][ind]

 ret[i] = ind

 right = len(carr[i])

 left = right - 1

 else:

 right = ret[i] + 1

基础步骤，在合并步骤中

当只有一行时 由于奇数行的最左最小元素列号

则顺序搜索 已经限制在偶数行划分的区间

每个元素以获得 所以从列号来看只需顺序遍历一次

最左最小元素 而搜索次数不低于行数

复杂度为 $\Theta(n)$ 复杂度为 $\Theta(n+m)$

递归式为 $T(m+n)$

$T(m+n) = \begin{cases} \Theta(n), & n=1 \\ T(\frac{m}{2}, n) + \Theta(m+n), & n>1 \end{cases}$

于是有 $T(m+n) = O(m+n \lg m)$

Algorithm - P37

主定理补充：注意在主定理中，在 $f(n) = O(n^{\log_b a - \varepsilon})$, $f(n) = \Theta(n^{\log_b a})$, $f(n) = \Omega(n^{\log_b a + \varepsilon})$ 之间存在空隙，即 $f(n)$ 并未多项式大于 $n^{\log_b a}$ 或多项式小于 $n^{\log_b a}$ 在此类情况下无法应用主定理进行多项式求解

扩展情形 2 可以弥补 $f(n) = \Theta(n^{\log_b a})$ 和 $f(n) = \Omega(n^{\log_b a + \varepsilon})$ 之间空隙

即如果存在非负整数 k , 使得 $f(n) = \Theta(n^{\log_b a} (\lg^k n))$

则有递归式的解为 $T(n) = \Theta(n^{\log_b a} (\lg^{k+1} n))$

证明过程可由引理 2 的情形 2 开始

即有 $f(n) = \Theta(n^{\log_b a} (\lg^k n))$, 其中 $k \in \mathbb{N}$

$$\text{则 } g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

$$= \Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \cdot \left(\frac{n}{b^j}\right)^{\log_b a} \cdot (\lg^{k+1} n)^k\right)$$

$$= n^{\log_b a} \cdot \Theta\left(\sum_{j=0}^{\log_b n - 1} (\lg^n - j \lg b)^k\right)$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \Theta((\lg^n - j \lg b)^k)$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \Theta(\lg^k n)$$

$$= \Theta(n^{\log_b a} (\lg^{k+1} n))$$

$$\text{于是有 } T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

$$= \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} (\lg^{k+1} n))$$

$$= \Theta(n^{\log_b a} (\lg^{k+1} n))$$

Akra-Bazzi 方法，相比于主定理给出了更一般的求解递归式方法，用于解决子问题划分不均衡的算法

$$\text{对于递归式形如 } T(x) = \begin{cases} \Theta(1) & , 1 \leq x \leq x_0 \\ \sum_{i=1}^k a_i T(b_i x) + f(x) & , x > x_0 \end{cases}$$

其中实数 $x \geq 1$, 且常数 x_0 满足 对任意 $i=1, \dots, k$, 有 $x_0 \geq \frac{1}{b_i}$ 且 $x_0 \geq \frac{1}{1-b_i}$

对于 $i=1, 2, \dots, k$, a_i 是正常数, 常数 $0 < b_i < 1$

k 为正整数常数

$f(x)$ 为满足多项式增长条件的非负函数

即存在正常数 C_1, C_2 , 使得对任意 $x \geq 1$, $i=1, 2, \dots, k$

以及任意 $b_i x \leq u \leq x$, 有 $C_1 f(u) \leq f(u) \leq C_2 f(u)$

或者说 $|f'(x)|$ 的上界为 x 的一个多项式

即对于任意实数常数 α, β , $f(x) = x^\alpha (\lg^\beta x)$ 满足条件

求解时, 寻找满足 $\sum_{i=1}^k a_i b_i^\beta = 1$ 的实数 P 注意实数 P 总是存在

则递归式的解为 $T(n) = \Theta(x^P (1 + \int_1^x \frac{f(u)}{u^{P+1}} du))$

Algorithm - P38

证明过程：

对于正整数 $n \geq 1$, 有 $e\left(\frac{n}{e}\right)^n \leq n! \leq e^n\left(\frac{n}{e}\right)^n$

证明过程有：基础步骤：当 $n=1$ 时， $e\left(\frac{1}{e}\right)^1 = 1! = 1$, 即 $e\left(\frac{n}{e}\right)^n \leq n! \leq e^n\left(\frac{n}{e}\right)^n$ 成立

递归步骤：假设对于任意 $n > 1$, $P_1 \wedge \dots \wedge P_{n-1}$ 成立，则考虑 P_n

首先考虑 $n!$ 与 $e^n\left(\frac{n}{e}\right)^n$, 则有

$$n! = n(n-1)!$$

$$\stackrel{(IH)}{\leq} n \cdot e \cdot (n-1)\left(\frac{n-1}{e}\right)^{n-1}$$

$$= e^n \cdot \left(\frac{n}{e}\right)^n \cdot \left(\frac{e}{n}\right)^n \cdot \frac{(n-1)^n}{e^{n-1}}$$

$$= e^n\left(\frac{n}{e}\right)^n \cdot e \cdot \left(\frac{n-1}{n}\right)^n$$

$$= e^n\left(\frac{n}{e}\right)^n \cdot e(1-\frac{1}{n})^n$$

又根据 $1+x \leq e^x$, 对所有 $x \in \mathbb{R}$ 成立, 则取 $x = -\frac{1}{n}$

$$n! \leq e^n\left(\frac{n}{e}\right)^n \cdot e(1-\frac{1}{n})^n$$

$\leq e^n\left(\frac{n}{e}\right)^n \cdot e(e^{-1/n})^n = e^n\left(\frac{n}{e}\right)^n$, 即 P_n 右半部分成立

再考虑 $n!$ 与 $e\left(\frac{n}{e}\right)^n$, 则有

$$n! = n(n-1)! \stackrel{(IH)}{\geq} n \cdot e \left(\frac{n-1}{e}\right)^{n-1}$$

$$= e \cdot \left(\frac{n}{e}\right)^n \cdot \left(\frac{e}{n}\right)^n \cdot n \cdot \left(\frac{n-1}{e}\right)^{n-1}$$

$$= e \cdot \left(\frac{n}{e}\right)^n \cdot e \cdot \left(\frac{n-1}{n}\right)^{n-1}$$

$$= e \cdot \left(\frac{n}{e}\right)^n \cdot e \cdot (1-\frac{1}{n})^{n-1}, \text{ 取 } x = \frac{1}{n-1}$$

$$= e \cdot \left(\frac{n}{e}\right)^n \cdot (e^{1/(n-1)})^{n-1} (1-\frac{1}{n})^{n-1}$$

$$\geq e\left(\frac{n}{e}\right)^n \cdot (1+\frac{1}{n-1})^{n-1} (1-\frac{1}{n})^{n-1}$$

$$\text{且 } n = n-1+1, \text{ 则 } = e\left(\frac{n}{e}\right)^n \cdot \left(\frac{n}{n-1} \cdot \frac{n-1}{n}\right)^{n-1} = e\left(\frac{n}{e}\right)^n, \text{ 即 } P_n \text{ 左半部分成立}$$

于是根据数学归纳法, 对任意正整数 n , 有 $e\left(\frac{n}{e}\right)^n \leq n! \leq e^n\left(\frac{n}{e}\right)^n$

另外有对于当正整数 $n \geq 1$ 时, 对 $n! \geq e\left(\frac{n}{e}\right)^n$ 部分的积分证明

对于正整数 $n \geq 1$, $\sum_{k=1}^n \ln^k$ 可看作图中矩形面积之和

而对于函数 $y = \ln x$, $\int_1^n \ln x dx$ 为曲线与 x 轴, $x=n$ 围成面积

于是有 $\sum_{k=1}^n \ln^k \geq \int_1^n \ln x dx$. 其中等号仅有 $n=1$ 时取得

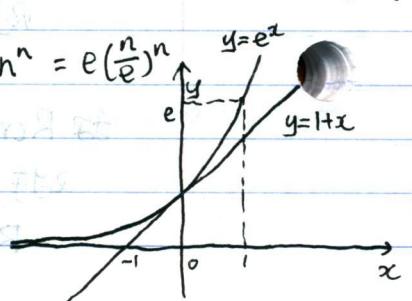
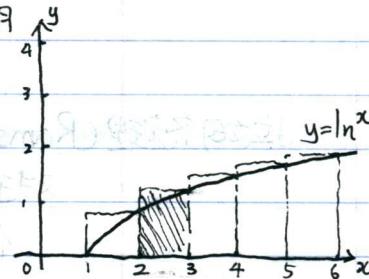
$$\text{又 } \int_1^n \ln x dx = [x \ln x - x]_1^n = n \ln^n - n + 1$$

$$\text{于是 } \sum_{k=1}^n \ln^k \geq n \ln^n - n + 1$$

$$\text{即 } n! = e^{\ln n!} = e^{\sum_{k=1}^n \ln^k} \geq e^{n \ln^n - n + 1} = e \cdot e^{-n} \cdot n^n = e\left(\frac{n}{e}\right)^n$$

而对于任意实数 x , 有 $1+x \leq e^x$.

可通过观察图形得到, 且等号仅在 $x=0$ 时取得



Algorithm - P39

本既率分析 (probability analysis) 指在问题分析中应用本既率的理念。为了使用本既率分析，必须使用或者假设关于输入的分布。如果可以对所有可能的输入集合做某种假定，则采用本既率分析来设计一个高效算法，并加深对问题的认知。如果无法描述一个合理的输入分布，则不能采用本既率分析。

平均情况运行时间，指对所有可能输入产生的运行时间取平均 (average-case running time)

均匀排列 (uniform random permutation)，在数字1到n的 $n!$ 种可能排列中，每一种均等概率出现。如果某个输入的排列可等价于数字1到n的 $n!$ 种排列，则称为以随机顺序出现 (in a random order)。

随机的 (randomized)，指算法不仅由输入决定，还由随机数生成器决定。

随机数生成器 (random-number generator, RNG)，通常记为 $\text{RANDOM}(a, b)$ ，指等概率地返回一个介于 a, b 之间的整数 $t \in [a, b]$ 。可以视随机数生成器为一个 d ($b-a+1$) 的均匀骰子。

伪随机数生成器 (pseudorandom-number generator)，由编程环境提供的确定性算法返回的值从统计上看是随机的 (statistically random)。

期望运行时间 (expected running time)，指随机算法的运行时间的期望值衡量。当分析随机算法的运行时间时，以运行时间的期望值衡量以此区分输入随机的确定性算法。

利用不均匀的硬币模拟均匀的硬币

即对于过程 BIASED-RANDOM()，以 $0 < p < 1$ 的概率返回 1，以 $1-p$ 的概率返回 0。

以 BIASED-RANDOM() 为基础子程序，设计算法返回一个无偏的结果，即 $\frac{1}{2}$ 为 1， $\frac{1}{2}$ 为 0。

def half-by-bias(p): | 以 BIASED-RANDOM() 为准的复杂度为

$$\text{while } 1: \quad E(T) = 2 \cdot \frac{1}{2p(1-p)} = \frac{1}{p(1-p)}$$

$d_1, d_2 = \text{BIASED-RANDOM}(), \text{BIASED-RANDOM}()$ | 连续掷两次硬币

if $d_1 == 1$ and $d_2 == 0$: return 1 | 如果为 10 则返回 1

if $d_1 == 0$ and $d_2 == 1$: return 0 | 如果为 01 则返回 0

注意：对于任意 $p \in (0, 1)$ ， $P\{10\} = P\{01\} = p(1-p)$ 否则重掷两次硬币

Algorithm - P40

指示器随机变量(indicator random variable),又称示性变量,

对于给定的样本空间 (sample space) S 和事件 (event) A

定义事件A对应的指示器随机变量 $I_{\{A\}} = \begin{cases} 1, & \text{当 } A \text{ 发生} \\ 0, & \text{当 } A \text{ 不发生} \end{cases}$

对于样本空间 S 和事件 A , 设随机变量 $X_A = I\{A\}$

则有 X_A 的期望 $E[X_A] = \Pr\{A\}$, 其中 $\Pr\{A\}$ 表示事件 A 发生的概率

证明过程有，令 \bar{A} 为事件 A 的补，即集合 $\{A\}$ 相对于事件集合 P_S 的补集

则有 $E[X_A] = E[I\{A\}] = 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\}$

$$c_0 + c_1 d^{1/2} + \dots + c_d d^{\frac{1}{2}} = \Pr[A]$$

$$(C_0(\mathcal{V}_0) \oplus \mathbb{C})^{1-\dim_{\mathbb{R}} \mathcal{V}_0} \otimes \mathbb{C}^{\dim_{\mathbb{R}} \mathcal{V}_0} + (C_0(\mathcal{V}_0) \oplus \mathbb{C})^{\dim_{\mathbb{R}} \mathcal{V}_0} \otimes \mathbb{C}$$

》雇用问题 (hiring problem), 假定通过面试雇用代理所推荐的应用者来寻找最合适的人选

假设总是选择最优的应聘者。

即当应聘者优于当前人员时，辞去当前人员，并**雇佣**应聘者。

假设对于每一位应聘者，需要向雇用代理支付较少的费用 c_i 。

对于雇用一位新的应聘者，需要付出一笔较大的费用 C_h

则考虑该策略的成本

HIRE-ASSISTANT(n)] 假设应聘者编号为从 1 到 n

创建一个比所有应聘者都差的虚拟应聘者。

~~for i=1 to n~~

(not) interview candidate i] 支付雇用代理费用 C_i

if candidate i better than candidate best

$$best = i^{(n)} + ({}^{\text{pred}}\!m_{n+1}) = const$$

hire candidate i] 支付雇用应聘者成

则在策略执行过程中, interview 执行了 n 次。

假设 hire 执行了 m 次，其中 $1 \leq m \leq n$

则策略 i 的总成本为 $O(c_i n + c_h m)$, 且最坏情形为 $O(c_h n)$

假设雇用问题中的应聘者以随机顺序出现，则总成本的平均情形为 $O(C_1 \ln n)$

证明过程有,令 X_i 为指示器随机变量(雇用应聘者*i*)

则 $E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n \Pr\{\text{使用耳塞者}\}$

对于第*j*个应聘者, $Pr\{\text{录用应聘者 } j\} = Pr_{S_j}$

現有為求大約而得情形如下 $\ln n + O(1)$

Algorithm - P41

069 - randomized

随机算法

指让随机发生在算法，而不是在输入分布上
randomize in the algorithm, not in the input distribution

现在每次运行算法时

执行#依赖于随机选择，而且很可能与前一次算法执行不同
the execution depends on the random choices made
is likely to differ from the previous execution of the algorithm

注意：对于随机算法，没有特别的输入会引出最坏情况行为
no particular input elicits its worst-case behavior

雇用问题 RANDOMIZED-HIRE-ASSISTANT(n):

应聘者序列

randomly permute the list of candidate] 唯一的改变是随机地变换
best := 0] 与原始程序完全相同的进程
for $i := 1$ to n
interview candidate i
if candidate i is better than candidate best
 best := i
hire candidate i] 支付雇用代理费用 c_i
] 支付雇用成本 C_h , 其中 $C_h \gg c_i$

注意 RANDOMIZED-HIRE-ASSISTANT 过程中雇用费用期望也是 $O(c_h \ln n)$

证明过程为，当对输入序列进行随机排列后，

实际上产生了与 HIRE-ASSISTANT 采用概率分析时相同情况

随机排列 目标是构造一个输入数组的均匀随机排列 (uniform random permutation)

如为每个元素 $A[i]$ 指派一个随机的优先级 $P[i]$ ，然后依据优先级对数组进行排序

PERMUTE-BY-SORTING(A):

$n := A.length$

let $P[1..n]$ be a new array

$[1, n^3]$

for $i := 1$ to n

$P[i] := \text{RANDOM}(1, n^3)$

为 $A[i]$ 指派 的随机整数为 $P[i]$

使用范围 $[1, n^3]$ 使尽量不出现相同

sort A , using P as sort keys

优先级

注意：耗时最大的步骤为排序，排序将花费 $\Omega(n \log n)$ 的时间

排序完成后，如果 $A[i]$ 对应优先级 $P[i]$ 为第 j 小的优先级

则 $A[i]$ 出现在位置 j 上

Algorithm - P42

对于 PERMUTE-BY-SORTING 算法，假设所有优先级均不相同，则产生均匀随机证明过程有。首先考虑元素 $A[i]$ 获得第 i 个最小优先级的特殊排列， $i=1, 2, \dots, n$ 。

即令事件 E_i 表示元素 $A[i]$ 获得第 i 小的优先级， $i=1, 2, \dots, n$

则考虑概率率 $\Pr\{E_1 \cap E_2 \cap \dots \cap E_n\}$

根据条件概率率 $\Pr\{H | E\} = P\{H \cap E\} / P\{E\}$

则 $\Pr\{E_1 \cap E_2 \cap \dots \cap E_n\} = \Pr\{E_n | E_1 \cap E_2 \cap \dots \cap E_{n-1}\} \cdot \Pr\{E_1 \cap E_2 \cap \dots \cap E_{n-1}\}$

$= \Pr\{E_1\} \cdot \Pr\{E_2 | E_1\} \cdot \dots \cdot \Pr\{E_{n-1} | E_1 \cap E_2 \cap \dots \cap E_{n-2}\}$

对于下两个事件，又可知 $\Pr\{E_i\} = 1/n$ ，即从 n 个优先级中选择最小的

$\Pr\{E_2 | E_1\} = 1/(n-1)$ ，即从剩余 $n-1$ 个优先级中选择最小的

则可得 $\Pr\{E_1 \cap E_2 \cap \dots \cap E_{n-1}\} = (1/n) + 1/(n-1) \cdot (1/n-1)$

$\Pr\{E_n | E_1 \cap E_2 \cap \dots \cap E_{n-1}\} = 1/1$

于是有 $\Pr\{E_1 \cap E_2 \cap \dots \cap E_n\} = \frac{1}{n} \times \frac{1}{n-1} \times \dots \times \frac{1}{1} = \frac{1}{n!}$

再考虑对集合 $\{1, 2, \dots, n\}$ 的任意排列 $6(1), 6(2), \dots, 6(n)$

即令事件 E'_i 表示元素 $A[i]$ 获得了第 $6(i)$ 小的优先级

则可以以同样方式证明 $\Pr\{E'_1 \cap E'_2 \cap \dots \cap E'_n\} = \frac{1}{n!}$

于是可知对于集合 $\{1, 2, \dots, n\}$ 的 $n!$ 个任意排列中的任意一个

出现的概率均为 $\frac{1}{n!}$ ，即算法产生了均匀随机排列

$$1 - \frac{1}{n}$$

对于 PERMUTE-BY-SORTING 算法，优先级数组 P 中每个元素，都唯一的概率是证明过程有，对于分配给集合 $\{1, 2, \dots, n\}$ 的优先级 $6(1), 6(2), \dots, 6(n)$

令事件 E_i 为， $6(i)$ 与其余 $n-1$ 个优先级中的值重复

由于其余 $n-1$ 个优先级中有不超过 $n-1$ 个在 $[1, n^3]$ 之间的值

于是有 $\Pr(E_i) \leq (n-1)/n^3$

又根据概率论，有 $\Pr(U_{i=1}^n E_i) \leq \sum_{i=1}^n \Pr(E_i)$

即令事件 E 为存在 $1 \leq i < j \leq n$ ， $6(i) = 6(j)$ ，于是 $E = E_1 \cup E_2 \cup \dots \cup E_n$

于是 $\Pr(E) = \Pr(U_{i=1}^n E_i) \leq \sum_{i=1}^n \Pr(E_i) \leq n(n-1)/n^3 < n^2/n^3 = 1/n$

于是有优先级数组 P 中每个元素都唯一的概率 $\Pr(\bar{E}) \geq 1 - \frac{1}{n}$

对于 PERMUTE-BY-SORTING 算法，修改算法以应对出现两个或者更多优先级相同的修改排序算法，对拥有相同优先级的元素集合。

递归地调用 PERMUTE-BY-SORTING 算法。

另外在集合中有两个元素时，改为仅随机一个数字，并决定两个元素的先后次序

Algorithm - P43

post-condition

随机排列

对于证明一个排列是均匀随机排列，存在一个不充分的弱条件
即证明对于每个元素 $A[i]$ 在排列中排在位置 j 的概率为 $\frac{1}{n}$

证明过程为：考虑以下算法产生的结果

PERMUTE-BY-CYCPLIC(A):

1. 定义 $h := A.length$

2. 定义 $B[1..n]$ 为一个新数组

3. $offset := RANDOM(1, n)$

4. $for i := 1 \text{ to } n$

5. $dest := i + offset$

6. $if dest > n$

7. $dest := dest - n$

8. $B[dest] := A[i]$

9. $return B$

注意：此处生成的随机数为偏移量

如果考虑 $A[i], i = 1, 2, \dots, n$

其最终位置 $dest = (i + offset) \% n$

由于 $offset$ 在 $[1, n]$ 上

随机取的整数

所以 $dest$ 在 $[1, n]$ 上均为 $\frac{1}{n}$

但是从整体来看， B 只有 $n!$ 种可能

所以并不是均匀随机排列

产生随机排列的一个更好方法是原地排列给定数组

RANDOMIZE-IN-PLACE(A):

1. $h := A.length$

2. $for i := 1 \text{ to } n$

3. $swap A[i] \text{ with } A[RANDOM}(i, n)]$

在 $O(n)$ 时间复杂度内完成

第 i 次迭代中，从 $A[i..n]$ 随机选取 $A[i]$

在第 i 次迭代后， $A[i]$ 不再改变

过程 RANDOMIZE-IN-PLACE 可计算出一个均匀随机排列

考虑使用循环不变式证明

循环不变式：在 for 循环的第 i 次迭代之前

对于每个可能的 $(i-1)$ 排列，子数组 $A[1..i-1]$ 包含这个排列的概率为 1

初始：在第 1 次迭代之前，考虑的排列是 0 排列

则对于每个可能的 0 排列，考虑子数组 $A[1..0]$ 包含 0 排列的概率为 1

可知 $(n-i+1)!/n! = n!/n! = 1$

而由于 0 排列没有元素，而空数组 $A[1..0]$ 也不包含任何元素

则可以认为 $A[1..0]$ 包含任何 0 排列的概率为 1

于是循环不变式对于 $i=1$ 成立

$$\frac{(n-i+1)!}{n!}$$

但是注意：空数组包含 0 排列实际上可以任意声明概率

即如果宣称空数组不包含 0 排列也是可以的

量通过由 $(n-i+1)!/n!$ 引起的如此即使得循环不变式的初值失效

Algorithm - P44

随机排列 对于原址排列给定数组算法做出修改以避免循环不变量初始化失效

过程 RANDOMIZE-IN-PLACE-SC(A):

$n := A.length$

if $n = 0 \vee n = 1$:] 当 A 为空数组或仅有一个元素时
return] 不对数组进行修改而直接返回

else: [在循环开始前先从数组中随机选择一个元素置入 $A[0]$]

for $i := 2$ to n : [随机选择一个元素置入 $A[i]$]

swap $A[i]$ with $A[RANDOM(1, n)]$] 与原算法相同的循环.

过程 RANDOMIZE-IN-PLACE-SC 能产生一个均匀随机排列

证明 运行过程有: 当 $n=0$ 或 $n=1$ 时,

数组 A 的随机排列仅包含其自身, 即每个排列的概率为 $\frac{1}{1} = \frac{1}{1} = 1$

当 $n > 1$ 时, 定义循环不变量为

在 for 循环的第 j 次迭代以前, 有循环计数器 $i = j + 1$

对每个可能的 j 排列, 即 $(i-1)$ 排列

子数组 $A[1..j]$ 包含这个 j 排列的概率是 $(n-j)!/n!$.

初始化: 在循环的第 1 次迭代之前, 程序执行了 1 次 swap 操作

则此时对于 $A[1..j]$, 数组中每个元素出现在 $A[1..j]$ 的概率

均为 $1/n = (n-1)!/n!$

保持: 假设在第 j 次迭代以前, 每种可能的 j 排列

在 $A[1..j]$ 中出现的概率均为 $(n-j)!/n!$

则考虑在第 j+1 次迭代以后, 第 j+1 次迭代之前的 $A[1..j+1]$ 的情形

$A[1..j+1]$ 包含两部分, 前 j+1 次迭代生成的子数组 $A[1..j]$

及由第 j+1 次迭代选出的元素 $A[j+1]$

令事件 E_1 为特定 j 排列包含于 $A[1..j]$

事件 E_2 为迭代选择了特定的元素 $A[j+1]$

于是 $Pr\{E_1 \cap E_2\} = Pr\{E_2 | E_1\} Pr\{E_1\}$

由于给定 E_1 , 发生 E_2 的概率为 $\frac{1}{n-j} = \frac{1}{n-j} \cdot \frac{(n-j)!}{n!} = \frac{(n-j+1)!}{n!}$

于是可知循环不变量在第 j+1 次迭代前也为真

终止: 在第 n 次迭代前, 循环计数器 $i = n+1$, 循环终止

此时数组 $A[1..n]$ 包含任何给定 n 排列的概率均为 $\frac{(n-n)!}{n!} = \frac{1}{n!}$

于是可知过程 RANDOMIZE-IN-PLACE-SC 产生一个均匀随机排列

Algorithm - P45

随机排列

考虑一个随机产生除恒等排列(identity permutation)外任意排列

PERMUTE - WITHOUT - IDENTITY(A):

int i, j, t; int n := A.length

for i := 1 to n-1 do t = A[i];

swap A[i] with A[RANDOM(i+1, n)]

注意这个过程实际上生成的是错排(derangement)

(cont.) 即所有元素均不在其原本的位置上, 考虑被移开的时间

如果在第*i*次迭代前 $A[i]$ 已被移动, 在 $A[i]$ 移动后不再会被选中

如果在第*i*次迭代 $A[i]$ 移动到其他位置, 之后也不会移动回到 $A[i]$

考虑在交换过程中, 不是将 $A[i]$ 与 $A[i..n]$ 中随机元素交换, 而是与 $A[1..n]$ 中随机元素

PERMUTE - WITH - ALL(A):

n := A.length

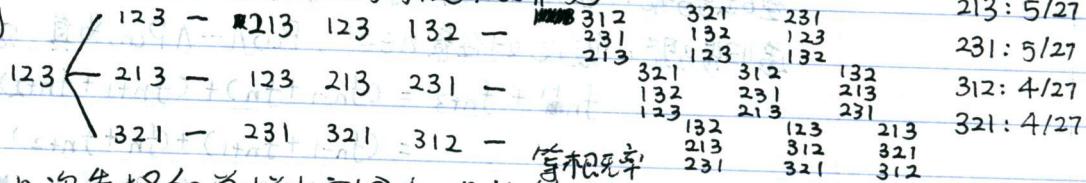
for i := 1 to n do

swap A[i] with A[RANDOM(1, n)]

注意这个过程在 $n=0, 1, 2$ 时是可以产生均匀随机排列的

但是在 $n \geq 3$ 时, 过程无法产生均匀随机排列

如 $n=3$ 时



证明过程有, n 次选择的总样本空间有 n^n 种可能, 而一共有 $n!$ 种不同排列

当 $n \geq 3$ 时, $n! \neq n^n$, 于是不可能产生均匀随机排列

随机样本 (random sample), 对于集合 $\{1, 2, \dots, n\}$, 具有 m 个元素的集合 $S \subseteq \{1, 2, \dots, n\}$ 等可能地出现
如果先排列 $\{1, \dots, n\}$ 再选择前 m 个元素, 需要调用 n 次 RANDOM(), 当 $n \gg m$ 时成本过高

RANDOM - SAMPLE(m, n):

if $m = 0$:] 基础步骤聚: 当 $m=0$ 时

return \emptyset] \emptyset 本身即均匀随机样本

else $S := \text{RANDOM-SAMPLE}(m-1, n-1)$] 假设生成了 $(m-1, n-1)$ 的随机样本

$i := \text{RANDOM}(1, n)$] 递归步骤聚,

if $i \in S$:]

$S = S \cup \{n\}$

else $S = S \cup \{i\}$

考虑返回的 S 包括 n , 则概率为 $\frac{m}{n}$

而对于不包含 n 的 S , 概率为 $\frac{n-m}{n}$

及 $\binom{n-1}{m} / \binom{n}{m} = \frac{(n-1)!}{m!(n-1-m)!} / \frac{n!}{m!(n-m)!} = \frac{n-m}{n}$

于是可以产生一个均匀随机样本