Chapter 5

Probabilistic Analysis and Randomized Algorithms

dqwang@mail.hust.edu.cn

群名称：2019-算法

群　号：835135560

群名称：2019-算法
群　号：835135560

5.1 雇佣问题

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，于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。

- 假设雇用代理推荐过来的应聘候选人有n人，编号为1到n。
- 并假设你在面试完一个应聘者后，就能决定该应聘者是否是你目前见过的最佳人选，如果是，则立即雇用。

雇佣算法可描述如下：

雇佣算法

```
HIRE-ASSISTANT(n)
best = 0            // candidate 0 is a least-qualified dummy candidate
for i = 1 to n
    interview candidate i
    if candidate i is better than candidate best
        best = i
        hire candidate i
```

分析：

该过程中检查序列中的每个成员，所以面试的总人数就是n。其中若干人被雇用，记被雇用总人数为m。

设介绍一个人面试的费用为c_i，发生一次雇用的费用为c_h，则该算法的总费用是$O(c_i n+c_h m)$.

- 进一步会发现介绍面试的总费用$c_i n$是恒定的，因为不管雇用多少人，总会面试n个应聘者。
- 而发生雇用的总费用$c_h m$是变数，因为m是未知的。
- 所以我们只需关注于雇用总费用$c_h m$即可。**因此，求解雇佣问题就是对算法中best的更新频率m（次数）建立模型。**

- 最坏情形分析：
  - 当应聘者质量按出现的次序严格递增时，就会出现最坏情况： 最坏情形下实际雇用了所有参加面试的应聘者。
  - 此时面试了n次，雇用了n次，所以雇用总费用是$O(c_h n)$。

- 一般情况分析：
  - 事实上，一般情形下应聘者不会总以质量递增的次序出现。
  - 那么你估计整个过程会有多少人被雇用呢？

    一半吗？

- 要注意这样一个事实：一般情况下应聘者会以任意可能的次序来应聘，有好有坏，我们事先既不知道他们出现的次序，也不能控制这个次序（假设雇用代理也不关心好坏的话）
- 那么，在一般情形下，会发生什么呢?

引入“概率分析”对上述现象进行分析

概率分析：就是在问题分析中应用概率的理念。

我们用概率分析的方法分析上述问题的雇用费用。

为了进行概率分析，首先对输入分布做出假设。这里，假设雇佣问题中应聘者以随机顺序出现 —— 并且这种随机性由输入自身决定（不是你决定的）。

同时这也意味着所有应聘者是一种全序关系，即任意两个应聘者可以比较好坏并决定哪一个更有资格。

- 用rank(i)表示应聘者质量。
  - 不失一般性，设rank(i)为整数，取值1~n，1代表最差，n代表最好，且所有应聘者的rank各不相同；
  - 则任意序列 R=<rank(1), rank(2),…,rank(n)> 将是〈1, 2, …, n〉的一个排列。注：〈1, 2, …, n〉的排列共有n!种。
- 称**应聘者以随机顺序出现**，就是说R是数字1到n的n!种排列中的任一个，随机出现，且为**均匀随机排列**，即在n!种可能的排列中，每种排列以1/n! 的可能性“等概率”出现。

随机算法

目的：为了利用概率分析，就要了解关于**输入分布的一些信息**。但在许多情况下，我们对输入分布了解很少。而且即使知道输入分布的某些信息，也无法从计算上对这种认知建立模型——输入不可控。

输入如何可控?

使一个算法中的某部分的行为随机化，然后利用概率和随机性作为算法设计与分析的工具进行相关处理。

分析：在雇佣问题中，看起来应聘者好像以随机顺序出现，但我们无法知道是否如此。

怎么办?

为了分析雇佣问题一般情况下的解，我们必须对应聘者的出现次序进行**更大的控制**，使其达到一种**"随机"出现的样子**。

方法：**假设雇用代理推荐了n个应聘者，可以事先给我们一份名单，而我们每天随机选择某个应聘者来面试。**

——这有什么不同呢？

尽管除了应聘者名字外，对其他信息一无所知，但不再像以前依赖于"猜测"应聘者以随机次序出现。取而代之，我们获得了对流程的控制并加强了随机次序。

如果一个算法的行为不仅由输入决定，而且也由一个随机数生成器产生的数值决定，则称这个算法是随机化的（Randomized），把这样的算法称为随机算法。

随机数生成器RANDOM：

- 调用RANDOM(a, b)将返回一个介于a和b之间的整数，且每个整数以等概率出现。
- 而且每次RANDOM返回的整数都独立于前面调用的返回值。

例：RANDOM(0, 1)产生0和1的概率都为1/2；

RANDOM(3, 7)可以返回3，4，5，6，7，每个出现的概率为1/5；

期望运行时间:

我们将一个随机算法的平均运行时间称为期望运行时间。

- 此时，算法的最终输入由随机数发生器产生

- 一般而言，当概率分布是发生在算法的原始输入上时，我们讨论算法的“平均情况运行时间”，而当算法本身做出随机选择时，我们讨论算法的“期望运行时间”。

指示器随机变量

为了建立概率(probabilities)和期望(expectations)之间的联系，这里引进**指示器随机变量**(indicator random variables)，用于实现概率与期望之间的转换。

指示器随机变量:

给定一个样本空间S(sample space)和一个事件A(event)，那么事件A对应的指示器随机变量 I{A} 定义为:

$$\mathrm{I}\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs}, \\ 0 & \text{if } A \text{ does not occur}. \end{cases}$$

例：硬币有正反两面，抛掷一枚硬币，求正面朝上的期望次数。

分析：

求正面朝上的期望次数就是说如果硬币抛了n次，“平均”有几次正面朝上。

将硬币正面朝上的事件记为H，反面朝上的事件记为T。从而有样本空间S={H，T}，且Pr(H)=Pr(T)=1/2.

对于正面朝上的事件H，定义一个指示器随机变量X_H，记录在一次抛硬币时正面朝上的次数：

$$X_H = \mathrm{I}\{H\} = \begin{cases} 1 & \text{如果 } H \text{ 发生} \\ 0 & \text{如果 } T \text{ 发生} \end{cases}$$

则，一次抛掷硬币正面朝上的期望次数是：

一次抛掷硬币正面朝上的期望次数：

$$
\begin{aligned}
\mathrm{E}[X_H] &= \mathrm{E}[\mathrm{I}\{H\}] \\
&= 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{T\} \\
&= 1 \cdot (1/2) + 0 \cdot (1/2) \\
&= 1/2 .
\end{aligned}
$$

可以看到，一个事件对应的**指示器随机变量**的**期望值**等于该事件发生的概率。

引理5.1 给定一个样本空间S和S中的一个事件A，设$X_A=I\{A\}$，那么$E[X_A]=Pr\{A\}$。

证明：由指示器随机变量的定义以及期望值的定义，有：

$$
\begin{aligned}
E[X_A] &= E[I\{A\}] \\
&= 1*Pr\{A\} + 0*Pr\{\sim A\} \\
&= Pr\{A\}
\end{aligned}
$$

其中，~A表示S-A，即A的补。

n次抛掷硬币正面朝上的期望次数是多少?

- 设随机变量X表示n次抛硬币中出现正面朝上的总次数。
- 设指示器随机变量X_i对应第i次抛硬币时正面朝上的事件，

即：X_i=I{第i次抛掷时出现事件H}。

则显然有： $$X=\sum_{i=1}^{n} X_i .$$

则，两边取期望，计算正面朝上次数的期望，有：

$$\mathrm{E}[X]=\mathrm{E}\left[\sum_{i=1}^{n} X_i\right] .$$

即：总和的期望值等于n个指示器随机变量值和的期望，也等于n个指示器随机变量值期望的和。

由引理5.1，若每个指示器随机变量的期望值为1/2，则总和X的期望值为：

$$
\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] \\
&= \sum_{i=1}^{n} 1/2 \\
&= n/2 .
\end{aligned}
$$

用指示器随机变量分析雇佣问题

下面计算雇用新办公助理的**期望次数**：

- 假设应聘者以随机顺序出现。
- 设**X是一个随机变量**，其值等于雇用新办公助理的总次数。
- 定义n个**指示器随机变量X_i**，与应聘者i的一次面试相对应，根据i是否被雇用有：

$$X_i = \mathrm{I}\{\text{应聘者 } i \text{ 被雇用}\} = \begin{cases} 1 & \text{如果应聘者 } i \text{ 被雇用} \\ 0 & \text{如果应聘者 } i \text{ 不被雇用} \end{cases}$$

以及 $X = X_1 + X_2 + \cdots + X_n$

根据定理5.1，有 $\mathrm{E}[X_i] = \Pr\{\text{应聘者 } i \text{ 被雇用}\}$

应聘者i被雇用的概率是多少呢?

- 计算过程HIRE-ASSISTANT中第5~6行被执行的概率：

在第6行中，若应聘者i被雇用，则他就要比前面i-1个应聘者更优秀。

因为已经假设应聘者以随机顺序出现，所以这i个应聘者也以随机次序出现。而且在这当前的i个应聘者中，任意一个都可能是最有资格的。那么，应聘者i比前i-1个应聘者更有资格的概率是1/i，即它将有1/i的概率被雇用。

  - 故由引理5.1可得： $\mathrm{E}[X_i] = 1/i$

```
HIRE-ASSISTANT(n)
1  best = 0              // candidate 0 is a least-qualified dummy candidate
2  for i = 1 to n
3      interview candidate i
4      if candidate i is better than candidate best
5          best = i
6          hire candidate i
```

计算E[X]：

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] && \text{（根据等式(5.2)）} \\
&= \sum_{i=1}^{n} E[X_i] && \text{（根据期望的线性性质）} \\
&= \sum_{i=1}^{n} 1/i && \text{（根据等式(5.3)）} \\
&= \ln n + O(1) && \text{（根据等式(A.7)）}
\end{aligned}
\qquad \text{(式5.5)}
$$

亦即，尽管面试了n个人，但平均起来，实际上大约只雇用了他们之中的 $\ln n$ 个人。

引理5.2 假设应聘者**以随机次序出现**，算法HIRE-ASSISTANT总的雇用费用平均情形下为$O(c_h \ln n)$.

证明：

根据雇用费用的定义和等式(5.5)，可以直接推出这个界，说明雇用人数的期望值大约为$\ln n$。

可见，**平均情形下的雇用费用$c_h \ln n$比最坏情况下的雇用费用$O(c_h n)$有了很大的改进。**

5.3 随机算法

如前节所示，**输入的分布有助于分析一个算法的平均情况行为。**但很多时候是无法得知输入分布的信息的。

- 采用随机算法，分析算法的期望值。

雇用问题的随机算法:

- 随机算法在算法运行前**先随机地排列应聘者**，体现所有排列都是等可能出现的性质。
- **核心思路：**随机算法不是假设输入的分布，而是设定一个分布。
  - 根据前面的讨论，如果应聘者以随机顺序出现，则聘用一个新办公助理的平均情况下雇佣次数大约是$\ln n$。
  - 现在，虽然修改了算法，使得随机发生在算法上，但雇用一个新办公助理的期望次数仍大约是$\ln n$。

雇佣问题的随机算法

对于雇用问题，代码中唯一需要改变的是随机地变换应聘者序列。

```
RANDOMIZED-HIRE-ASSISTANT(n)
randomly permute the list of candidates
best = 0                // candidate 0 is a least-qualified dummy candidate
for i = 1 to n
    interview candidate i
    if candidate i is better than candidate best
        best = i
        hire candidate i
```

引理5.3 过程RANDOMIZED-HIRE-ASSISTANT的雇用费用期望是$O(c_h \ln n)$.

证明：对输入数组进行变换后，我们已经达到了和HIRE-ASSISTANT概率分析时相同的情况。 ■

关于算法的讨论：

- **如果不考虑随机处理，则算法就是“确定”的。**

此时，雇用新办公助理的次数依赖于**初始时各个应聘者的排名**，对于任何特定输入，雇用一个新办公助理的次数始终相同。

如：排名列表A_1=<1, 2, 3, 4, 5, 6, 7, 8, 9, 10>，新办公助理会雇用10次。

排名列表A_2=<10, 9, 8, 7, 6, 5, 4, 3, 2, 1>，新办公助理会只雇用1次。

排名列表A_3=<5, 2, 1, 8, 4, 7, 10, 9, 3, 6>，新办公助理会雇用三次。

（但不同的输入，雇用新办公助理的次数还是不同的）。费用依赖于雇用新办公助理的次数，所以可以看到，输入A_1是最昂贵的，输入A_2是最不贵的，而输入A_3适中。

- 算法RANDOMIZED-HIRE-ASSISTANT在算法运行前先随机地排列应聘者，是随机算法。
  - ✓ 此时，“随机”发生在算法上，而不是在输入分布上。
  - ✓ 随机算法中，任何一个给定的输入，如A_1或A_3，都无法说出best会被更新多少次的。因为随机发生在算法中，而不是直接在输入分布上。没有特别的输入会引出它的最坏行为（当然也没有特别的输入不会引出它的最坏行为）。
- 随机化处理使得输入次序不再相关。只有在随机数生成器产生一个”不走运“的排列时，随机算法才会运行得很差。

引理5.2和引理5.3的区别:

- 在引理5.2中，我们**在输入上做了随机分布的假设**，求的是**平均情形下的雇用费用**。
- 在引理5.3中，**随机化**作用在算法上，求的是**雇用费用的期望值**。

如何产生随机排列的数组?

- 随机算法需要通过**对给定的输入变换排列**以使输入随机化。
- 这里介绍两种随机化方法。

不失一般性，假设给定一个数组A，包含元素1到n。**随机化的目标是构造这个数组的一个均匀随机排列**。

方法一：为数组的每个元素A[i]赋一个随机的优先级P [i]，然后根据优先级对数组中的元素进行排序。

例：如果初始数组A=<1, 2, 3, 4>，随机选择的优先级是

P=<36, 3, 62, 19>，

则将产生一个数组：B=<2, 4, 1, 3>

上述策略的过程描述

```
PERMUTE-BY-SORTING(A)
1  n = A.length
2  let P[1..n] be a new array
3  for i = 1 to n
4      P[i] = RANDOM(1, n^3)
5  sort A, using P as sort keys
```

- 第4行**选取一个在1~n^3之间的随机数**。使用范围1~n^3是为了让P中所有优先级尽可能唯一。
- 第5步排序时间为O(nlogn)
- 排序后，如果P[i]是第j个最小的优先级，那么A[i]将出现在输出位置j上。最后得到一个”随机“排列。

引理5.4 假设所有优先级都不同，则过程PERMUTE-BY-SORTING产生输入的均匀随机排列 。

证明：

从考虑每个元素A[i]分配到第i小优先级的特殊排列开始讨论， 说明这个排列正好发生的概率是1/n!。

设E_i代表元素A[i]分配到第i小优先级的事件（i=1, 2, …, n）， 则对所有的E_i，整个事件发生的概率是：

$$
\begin{aligned}
&\Pr\{E_1 \cap E_2 \cap E_3 \cap \cdots \cap E_{n-1} \cap E_n\} \\
= \quad &\Pr\{E_1\} \cdot \Pr\{E_2 \mid E_1\} \cdot \Pr\{E_3 \mid E_2 \cap E_1\} \cdot \Pr\{E_4 \mid E_3 \cap E_2 \cap E_1\} \\
&\cdots \Pr\{E_i \mid E_{i-1} \cap E_{i-2} \cap \cdots \cap E_1\} \cdots \Pr\{E_n \mid E_{n-1} \cap \cdots \cap E_1\}
\end{aligned}
$$

这里，$\Pr\{E_1\}$是为第一个元素选择的优先级恰好为第1小优先级的概率， 故有$\Pr\{E_1\}=1/n$。

对于i=2, 3, . . . , n，一般有：

$$\Pr\{E_i \mid E_{i-1} \cap E_{i-2} \cap \cdots \cap E_1\} = 1/(n-i+1).$$

即：给定元素A[1]到A[i-1]的前i-1个小的优先级，在剩下的n-(i-1)个元素中，每个都有可能是第i小优先级，而恰好选中第i小优先级的概率是1/(n-i+1)。

最终有：

$$\Pr\{E_1 \cap E_2 \cap E_3 \cap \cdots \cap E_{n-1} \cap E_n\} = \left(\frac{1}{n}\right)\left(\frac{1}{n-1}\right)\cdots\left(\frac{1}{2}\right)\left(\frac{1}{1}\right) = \frac{1}{n!}$$

说明**获得等同排列的概率是1/n!**。

因此PERMUTE-BY-SORTING过程能产生一个**均匀随机排列**。

- 扩展上述证明过程，使其对任何优先级的排列都有效：
  - 考虑集合{1, 2, . . . , n}的任意一个确定排列

$$\sigma=\langle\sigma(1),\sigma(2),\cdots,\sigma(n)\rangle。$$

  - 由于优先级各不相同，所以可以重新定义映射：$r(\sigma(i))\rightarrow i$。
  - 定义E_i为“元素A[i]分配到优先级是第σ(i)小的事件”，则其等价于“元素A[i]分配到优先级是第r(σ(i))小的事件”，从而还原到“元素A[i]分配到第i小优先级”的情形，所以同样的证明仍适用。
  - 因此，如果要计算得到任何特定排列的概率，该计算与前面的计算恰好得到第i小优先级的概率完全相同，于是得到此排列的概率也是1/n!

方法二：原址排列给定数组。第i次迭代时，元素A[i]从元素A[i]到A[n]中随机选取。

过程如下：

RANDOMIZE-IN-PLACE(A)

```
1  n = A.length
2  for i = 1 to n
3      swap A[i] with A[RANDOM(i, n)]
```

第i次迭代后，A[i]不再改变

引理5.5 过程RANDOMIZE-IN-PLACE可计算出一个均匀随机排列。

证明：使用循环不变式来证明该过程能产生一个均匀随机排列。

（自学）

生日悖论

当人数达到多少，可使得这些人中有相同生日的可能性达到50%?