#ADS13 Randomized Algorithm

The Hiring Problem

假如你要雇佣一名新的办公室助理，但是你先前的雇佣尝试都失败了，你打算找一个雇佣代理。雇佣代理每天给你推荐一个应聘者。你面试这个人，然后决定是否雇佣他，同时你需要付给雇佣代理一定的费用，以便面试应聘者。除此之外，雇佣一个人也需要花费一大笔钱，因为你必须辞掉目前的办公室助理，同时付给雇佣代理一大笔中介费。

你承诺：任何时候都找最合适的人来担任这项职务，因此你决定在面试完应聘者之后，如果该应聘者比目前的办公室助理更合适，就会辞掉当前的办公室助理，然后聘用新的。

同时你愿意为该策略付费，但希望能够估算出该费用。

Hiring　　C　　复制代码

```
int Hiring(EventType C[],int N)
{
    int Best = 0;
    int BestQ = the quality of candidate 0;
    for(int i=1;i<=N;i++)
    {
        Qi = interview(i);
        if(Qi > BestQ){
            BestQ = Qi;
            Best = i;
            hire(i);
        }
    }
    return Best;
}
```

Worst case: The candidates come in increasing qualty order.

Total cost: $O(c_i n + c_h m)$

面试和雇佣都会产生一定的费用，面试费用较低，记为ci，雇佣费用很高。假设有n个应聘者，m个被雇佣的人，算法总费用为 $O(c_i n + c_h m)$， 由于面试费用总是固定的，因此我们可以只关注chm就可以了。

但是在大多数情况下，我们并不知道应聘者是否是随机出现的，此时可以在输入和算法之间加入一个随机数生成器，这是我们可以称这个算法是随机的。

当分析一个随机算法的运行时间时，我们以运行时间的期望值来衡量。一个随机算法的运行时间成为期望运行时间；当概率分布发生在算法的输入的时候，讨论的时平均情况运行时间。

我们经常采用指示器随机变量来进行分析，为概率与期望之间的转换提供了一个便利的方法。

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

$$I\{A\} = \begin{cases} 1 & \text{如果A发生} \\ 0 & \text{如果A不发生} \end{cases}$$

假设我们来确定抛一枚标准硬币时 正面朝上的期望次数。样本空间S= {T,F}, 接下来定义一个指示器随机变量XT，对应硬币朝上的事件T，

$$X_T = I\{T\} = \begin{cases} 1 & \text{如果T发生} \\ 0 & \text{如果F发生} \end{cases}$$

$$E[X_T] = E[I\{T\}] = 1 * Pr\{T\} + 0 * Pr\{F\} = 1/2$$

> 一个事件A对应的指示器随机变量的期望值等于事件A的发生的概率。
>
> 给定一个样本空间S和事件A，设XA = I{A}, 那么E[XA] = Pr{A}。

X对引发雇佣新的办公室助理的次数，Xi对应第i个应聘者被雇佣的这个随机器指示变量

$$X_i = I\{i\text{被聘用}\} = \begin{cases} 1 & \text{应聘者}i\text{被聘用} \\ 0 & \text{应聘者}i\text{未被聘用} \end{cases}$$

$$X = X_1 + X_2 + ... + X_n$$

假设应聘者是随机出现的，那么前i个应聘者中任意一个都可能是目前最有资格的，那么应聘者i比其他应聘者更有资格的概率是1/i。

$$E[X_i] = Pr\{应聘者i被雇佣\} = 1/i$$

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n} X_i] \\ &= \sum_{i=1}^{n} E[X_i] \quad (期望的线性性质) \\ &= \sum_{i=1}^{n} 1/i \\ &= ln(n) + O(1) \quad (调和级数) \end{aligned}$$

根据上述分析的结果，我们可以知道，尽管我们需要面试n个人，但平均起来，实际上大约只雇佣了他们之中的ln(n)个人。

因此，假设应聘者是随机次序的，算法HIRE—ASSISTANT总的雇佣费用平均情况下为$O(c_h lnn)$

我们不能限制输入的随机性，但可以加一个中间层，不管输入的序列是什么，都产生一个均匀随机的序列，这样执行就不依赖于输入了，而是依赖于随机选择。

Randon-Hiring

```
int Hiring(EventType C[],int N)
{
    int Best = 0;
    int BestQ = the quality of candidate 0;

    randomly permute the list of candidates/*takes time*/

    for(int i=1;i<=N;i++)
    {
        Qi = interview(i);
        if(Qi > BestQ){
            BestQ = Qi;
            Best = i;
            hire(i);
        }
    }
    return Best;
}
```

许多随机算法通过对驶入变换排列使得输入随机化。讨论两种随机方法并给与证明。

第一种，随机排列数组。为数组的每一个元素A[i]赋予一个随机的优先级P[i]，然后根据优先级随数组A中的元素进行排序。即为置换策略。

Assign each element A[i] a random priority P[i], and sort.

例如，A = {1,2,3,4}， 随机的优先级 P = {45,23,96,12}， 那么就会产生一个数组B = {4,2,1,3}。 这个过程叫做PERMUTE-BY-SORTING

Randomized Permutation Algorithm

```
void PermuteBySorting(ElemType A[],int N)
{
    for(i=1;i<=N;i++)
        A[i].P = 1 + rand%(N^3);
    Sort A, using P as the sort keys;
}
```

第二种，原地排列给定数组。在进行第i次迭代的时候，元素A[i] 是从元素A[i]到A[n]中随机选取的，第i次迭代之后，A[i]不再改变。RANDOMIZE–IN–PLACE过程的时间复杂度为O (n) 。

RANDOMIZE–IN–PLACE　C　复制代码

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