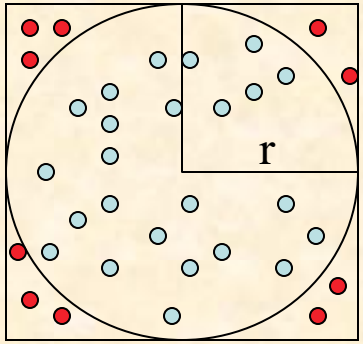
8 随机算法
Random Algorithm

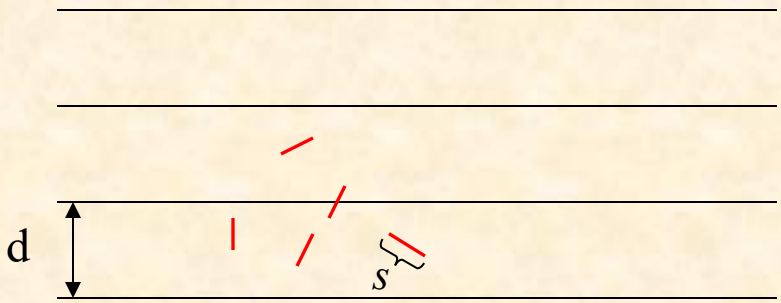
给一枚硬币，试计算圆周率。

圆周率计算



$$\frac{S_{circle}}{S_{square}} = \frac{\pi \cdot r^2}{4 \cdot r^2} \approx \frac{n_{in\_circle}}{n_{in\_square}} \qquad \pi \approx 4 \cdot \frac{n_{in\_circle}}{n_{in\_square}}$$

```
public static double darts(int n)
{   // 用随机投点法计算pi值
    int k=0;
    for (int i=1;i <=n;i++) {
        double x=dart.fRandom();
        double y=dart.fRandom();
        if ((x*x+y*y)<=1) k++;
    }
    return 4*k/(double)n;
}
```



18世纪，法国数学家布丰和勒可莱尔提出的“投针问题”，记载于布丰1777年出版的著作中：“在平面上画有一组间距为d的平行线，将一根长度为s（s<d）的针任意掷在这个平面上，求此针与平行线中任一条相交的概率。” 具体步骤如下：

1) 取一张白纸，在上面画上许多条间距为d的平行线。

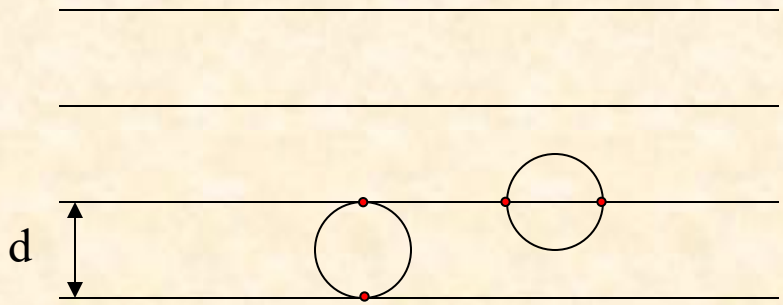
2) 取一根长度为s（s<d） 的针，随机地向画有平行直线的纸上掷n次，观察针与直线相交的次数，记为m

3) 计算针与直线相交的概率p

布丰本人证明了，这个概率是

$$p = \frac{2s}{\pi d} \approx \frac{m}{n}$$

原因是什么呢？



- 一个直径为d的圆圈，不管如何投掷，必定是有两个交点。那么投掷n，共有2n个交点。
- 把圆圈拉直，变成一条长为πd的线段。显然，这样的线段扔下时与平行线相交的情形要比圆圈复杂些，可能有4个交点，3个交点，2个交点，1个交点，甚至于都不相交。由于圆圈和线段的长度同为πd，根据机会均等的原理，当它们均投掷n次(n是一较大的数)，**两者与平行线组交点的总数期望值**也是一样的，即均为2n。
- 长度为s的线段，投掷n次后，交点的个数为记为m，那么： $m/2n = s/\pi d$

$$\pi d \text{ ---------- } 2n$$
$$s \text{ ---------- } m$$

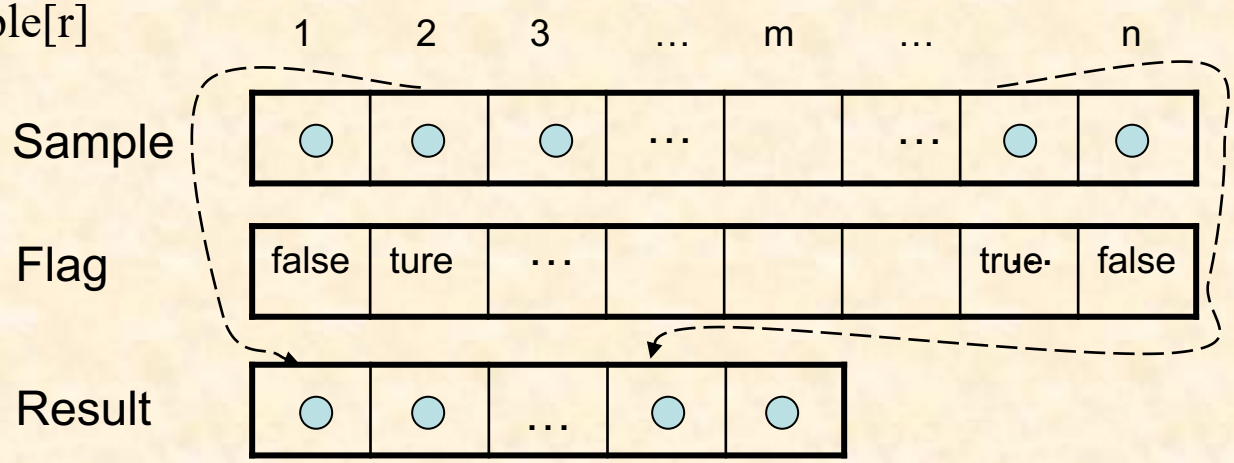
- 布丰投针实验是第一个用几何形式表达概率问题的例子，他首次使用随机实验处理确定性数学问题，为概率论的发展起到一定的推动作用。
- 像投针实验一样，用通过概率实验所求的概率来估计我们感兴趣的一个量，这样的方法称为蒙特卡罗方法（Monte Carlo method）。蒙特卡罗方法是在第二次世界大战期间随着计算机的诞生而兴起和发展起来的。这种方法在应用物理、原子能、固体物理、化学、生态学、社会学以及经济行为等领域中得到广泛利用。
- 蒙特·卡罗方法（Monte Carlo method），也称为统计模拟方法，是二十世纪四十年代中期由于科学技术的发展和电子计算机的发明，而被提出的一种以概率统计理论为指导的一类非常重要的数值计算方法。是指使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。蒙特·卡罗方法的名字来源于摩纳哥的一个城市蒙地卡羅，该城市以赌博业闻名，而蒙特·卡罗方法正是以概率为基础的方法。

随机非重复采样问题

- 设有n 个样本，要求从中随机选出m 个样本(m<n/2)。请设计一个随机算法来完成该任务，并分析该算法的时间复杂度。
- 思路：
  - 将n个样本存放于数组Sample[1…n]中。数组Result[1…m]存放选中的m个样本。
  - 定义一个长度为n的标志数组Flag[1…n]。Flag[i]=true表示对应位置的样本已经被选中；否则为未选中。
  - 随机生成一个落在区间[1,n]的随机数r。若Flag[r]为false，Sample[r]追加到数组Result中,并将Flag[r]置为true；若Flag[r]为true,则重新生成随机数r，直到Flag[r]为false。重复此步骤，直到m个样本选择完成。

输入：样本数组Sample[1…n], 选择样本的个数m, 且m<n/2
输出：选中的样本Result[1..m]

```
1.for i←1 to n
2.  Flag[i] ←false //boolean数组，表示对应的样本是否已被选中
3. end for
4. k←0
5. while k<m
6.     r←random(1,n)
7.     if not Flag[r] then
8.         k←k+1
9.         Result[k] ← Sample[r]
10.        Flag[r] ← true
11.    end if
12.end while
13.return Result[1…m]
```



时间复杂度分析

- 很显然，这是一个典型的迭代算法，因而可以使用计算迭代次数的技术来分析。
- 观察：然而每次运行此算法，迭代次数都可能是不同的。

- 引理1：在某个空间中抛掷硬币，设正面朝上的概率是p，反面朝上的概率为q(q=1-p)。用随机变量X表示“出现正面朝上”需要连续抛掷的次数，则随机变量X的分布满足几何分布

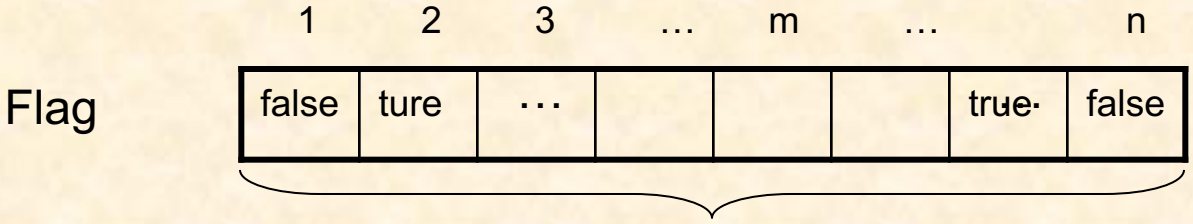
$$\mathbf{Pr}(X=k)=\begin{cases} pq^{k-1} & if\ k\geq 1 \\ 0 & if\ k<1 \end{cases}$$

X的数学期望是

$$\mathbf{E}(X)=\sum_{k=1}^{\infty}k\cdot\mathbf{Pr}(X=k)=\sum_{k=1}^{\infty}kpq^{k-1}=\frac{1}{p}$$

E(X)的含义是：平均连续抛掷1/p次，会出现一次正面。

- 分析：假设已经选中了k-1个样本，当前需要选择第k个样本。



“空位置个数”： n-(k-1)

“全部位置个数”： n

“随机数r落在空位置的概率”：n-(k-1) /n

用随机变量X_k 表示“落在空位置”需要连续生成随机数的次数(也就是，为选中第k个样本算法需要的迭代次数)。

由引理可知：$E(X_k)=n/(n-k+1)$

- 用随机变量Y表示为了从n个样本中选出m个样本而生成的总的随机数个数(即算法总的迭代次数)，根据数学期望的线性性质，我们有

$$E(Y) = E(X_1) + E(X_2) + \ldots + E(X_m)$$

$$= \sum_{k=1}^{m} \mathbf{E}(X_k)$$

$$= \sum_{k=1}^{m} \frac{n}{n-k+1}$$

$$= n\sum_{k=1}^{\mathbf{n}} \frac{1}{n-k+1} - n\sum_{k=\mathbf{m+1}}^{\mathbf{n}} \frac{1}{n-k+1}$$

$$= n\sum_{k=1}^{\mathbf{n}} \frac{1}{k} - n\sum_{k=1}^{\mathbf{n-m}} \frac{1}{k}$$

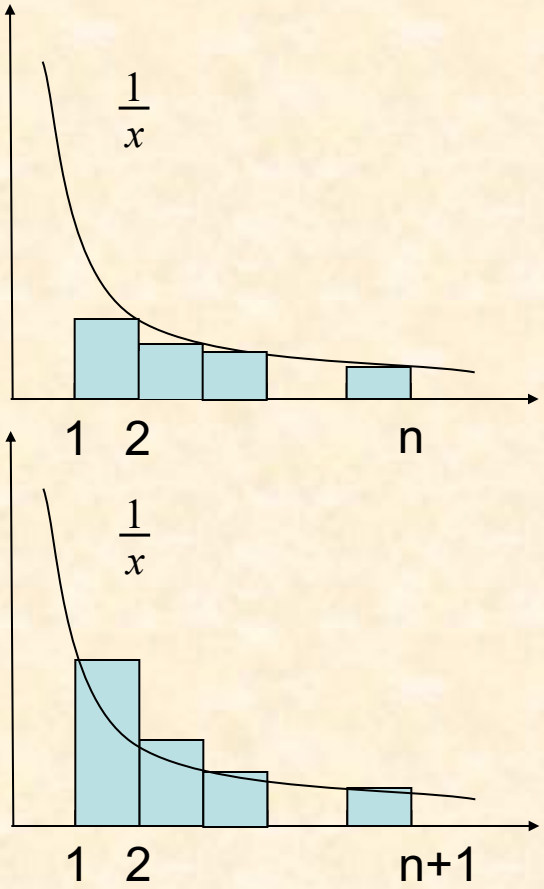
补充

$$n\sum_{i=1}^{n}\frac{1}{i}=\Theta(n\log n)$$

解:

a. $$\sum_{i=1}^{n}\frac{1}{i}=1+\sum_{i=2}^{n}\frac{1}{i}\le 1+\int_{1}^{n}\frac{1}{x}dx=1+\ln n$$

b. $$\sum_{i=1}^{n}\frac{1}{i}\ge \int_{1}^{n+1}\frac{1}{x}dx=\ln(n+1)$$



所以 $$\ln(n+1)\le\sum_{i=1}^{n}\frac{1}{i}\le 1+\ln n$$

换底公式 $$\frac{\log(n+1)}{\log e}\le\sum_{i=1}^{n}\frac{1}{i}\le 1+\frac{\log n}{\log e}$$

所以 $$\sum_{i=1}^{n}\frac{1}{i}=O(1+\frac{\log n}{\log e})=O(\log n)$$

$$\sum_{i=1}^{n}\frac{1}{i}=\Omega(\frac{\log(n+1)}{\log e})=\Omega(\log n)$$

亦即 $$n\sum_{i=1}^{n}\frac{1}{i}=\Theta(n\log n)$$

由于：$\sum_{k=1}^{n}\frac{1}{k}\le \ln n+1,\quad \sum_{k=1}^{n-m}\frac{1}{k}\ge \ln(n-m+1)$

则有：$E(Y)=n\sum_{k=1}^{\mathbf{n}}\frac{1}{k}-n\sum_{k=1}^{\mathbf{n-m}}\frac{1}{k}$

$$\le n(\ln n+1)-n\ln(n-m+\mathbf{1})$$

$$\le n(\ln n+1)-n\ln(n-m)$$

$$\le n(\ln n+1)-n\ln(n-\mathbf{n/2}) \qquad (\because \mathbf{m}\le \mathbf{n/2})$$

$$= n(\ln n+1-\ln(n/2))$$

$$= n(\ln 2+1)$$

$$\approx 1.69n$$

算法的期望运行时间T(n)主要有两部分组成：1)将boolean数组S[1…n]置为false耗时Θ(n)，2)产生随机数r←random(1,n)的期望运行时间E(Y)≈1.69n；因此，期望运行时间

$$T(n) \approx \Theta(n)+ 1.69n = \Theta(n)$$

快速排序（重新来看一下）

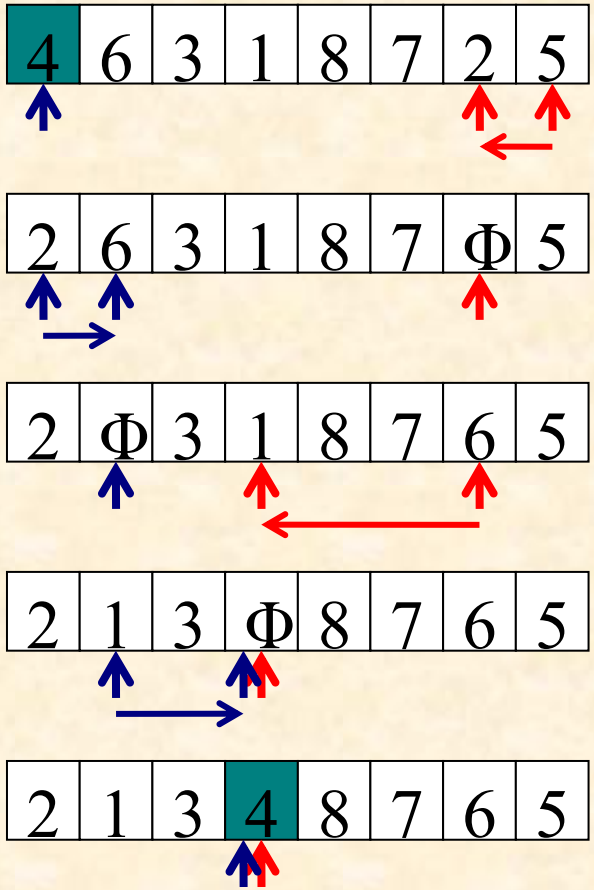
快速排序是一个非常流行而且高效的算法，其平均时间复杂度为Θ(nlogn). 其优于合并排序之处在于它在原位上排序，不需要额外的辅助存贮空间(合并排序需Θ(n)的辅助空间)。Charles A. R. Hoare 1960 年发布了使他闻名于世的快速排序算法(Quicksort)，这个算法也是当前世界上使用最广泛的算法之一，当时他供职于伦敦一家不大的计算机生产厂家。1980 年，Hoare 被授予 Turing 奖，以表彰其在程序语言定义与设计领域的根本性的贡献。在 2000 年，Hoare 因其在计算机科学和教育方面的杰出贡献被英国皇家封为爵士。

将 A[1...8]={4, 6, 3, 1, 8, 7, 2, 5} 按照非降序方式进行排序

- SPLIT:以待排序数组的**首元素**作为**基准元素**，将待排序数组**分成**左右**两个子数组**。使得左边子数组中的元素都小于等于基准元素；右边子数组中的元素都大于等于基准元素。换句话说，通过SPLIT，已经将首元素放置到正确的排序位置。
- 对左、右子数组进行相同的操作。

Algorithm: SPLIT(A[low,...high])
输入：数组A[low,...high]
输出：用A[low]作基准元素划分后的数组A
及基准元素新的位置w

1. x←A[low]
2. while (low<high)
3. while (low<high && A[high]>x) --high;
4. A[low] ←A[high]
5. while (low<high && A[low]≤x) ++low;
6. A[high] ←A[low]
7. end while
8. A[low] ← x
9. w←low
10. return A and w //新数组A与x的新位置w



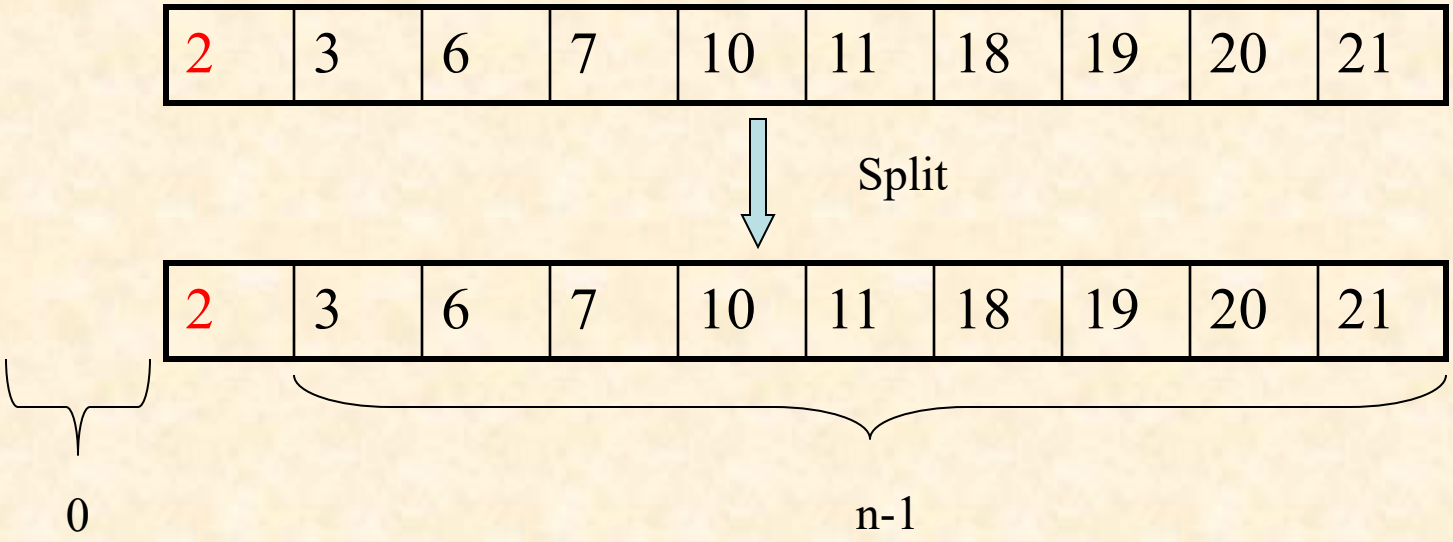
```
Algorithm: QUICKSORT(A[low…high])
输入: n个元素的数组A[low…high]
输出：按非降序排列的数组A[low…high]
1.  if low<high then
2.     w ← SPLIT(A[low…high])  {w为基准元素A[low]的新位置}
3.     QUICKSORT(A, low, w-1)
4.     QUICKSORT(A, w+1, high)
5.  end if
```

理想情形：每次SPLIT后得到的左右子数组规模相当，因此有：

$$T(n) = 2T(n/2) + n \ = 2(2T(n/2^2) + n/2) + n$$
$$= 2^2 T(n/2^2) + 2n = 2^3 T(n/2^3) + 3n = \cdots$$

$$\cdots = 2^{\log n} T(n/2^{\log n}) + (\log n) \cdot n = \Theta(n \log n)$$

最差情形(数组已经按照升序或是降序排好)：每次SPLIT后，只得到左或是右子数组。
不妨以已经按照升序拍好的数组为例：



$$T(n)=\begin{cases}1 & if\ n=1\\ T(n-1)+n & if\ n>1\end{cases} \quad \Rightarrow \quad T(n)=\Theta(n^2)$$

怎么解决？

- **Randomized QuickSort**
- 对于每个待排序的数组(子数组)，在SPLIT前，先执行下面的步骤1和2。

A

2	3	6	7	10	11	18	19	20	21

r

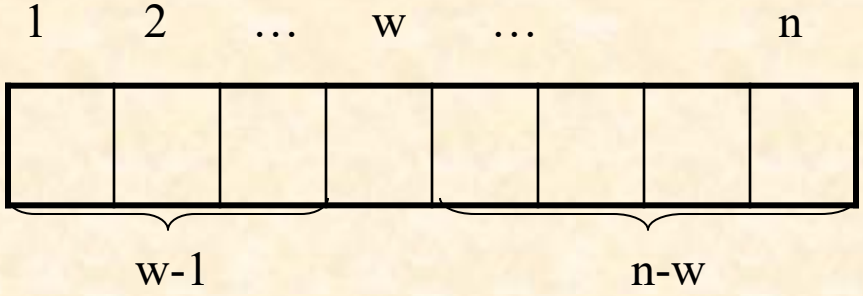
1. Generate a random integer $r \in [1, n]$
2. Exchange A[1] with A[r]

Algorithm: RandomizedQUICKSORT(A[low…high])
输入: n个元素的数组A[low…high]
输出：按非降序排列的数组A[low…high]

1. if low<high then
2. Generate a random integer $r \in$ [low,high]
3. Exchange A[low] with A[r]
4. w ← SPLIT(A[low…high]) {w为基准元素A[low]的新位置}
5. QUICKSORT(A, low, w-1)
6. QUICKSORT(A, w+1, high)
6. end if

Randomized QuickSort的时间复杂度：

我们用$C(n)$ 表示对一个n个元素的数组进行快速排序所需要的总的比较次数。



因此，我们有：

$$C(n) = (n-1) + \frac{1}{n}\sum_{w=1}^{n}\left(C(w-1) + C(n-w)\right)$$

$$\because \sum_{w=1}^{n} C(n-w) = C(n-1) + C(n-2) + \cdots + C(0) = \sum_{w=1}^{n} C(w-1)$$

$$\therefore C(n) = (n-1) + \frac{2}{n}\sum_{w=1}^{n} C(w-1)$$

$$n \cdot C(n) = n(n-1) + 2\sum_{w=1}^{n} C(w-1) \ldots\ldots (a)$$

⇓ n-1替换n

$$(n-1)C(n-1) = (n-1)(n-2) + 2\sum_{w=1}^{n-1} C(w-1) \ldots\ldots (b)$$

(a)-(b),并适当变换 ⟹

$$\frac{C(n)}{n+1} = \frac{C(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

令 $D(n) = \frac{C(n)}{n+1}$ ⇓

$$D(n) = D(n-1) + \frac{2(n-1)}{n(n+1)}, D(1) = 0$$

⇓

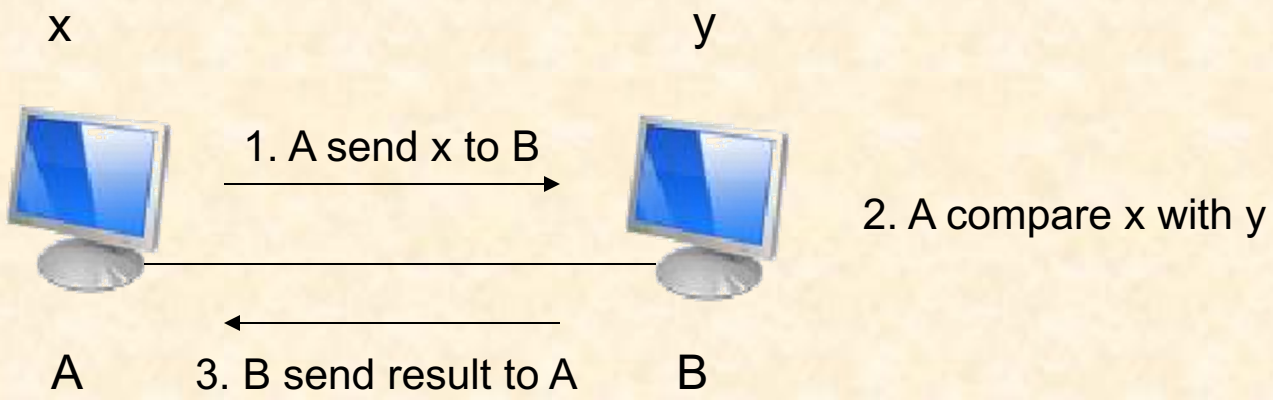
$$D(n) = 2\sum_{j=1}^{n} \frac{j-1}{j(j+1)} = 2\sum_{j=1}^{n} \frac{2}{(j+1)} - 2\sum_{j=1}^{n} \frac{1}{j}$$

$$= 4\sum_{j=2}^{n+1} \frac{1}{j} - 2\sum_{j=1}^{n} \frac{1}{j} = 2\sum_{j=1}^{n} \frac{1}{j} - \frac{4n}{n+1} = \Theta(\log n)$$

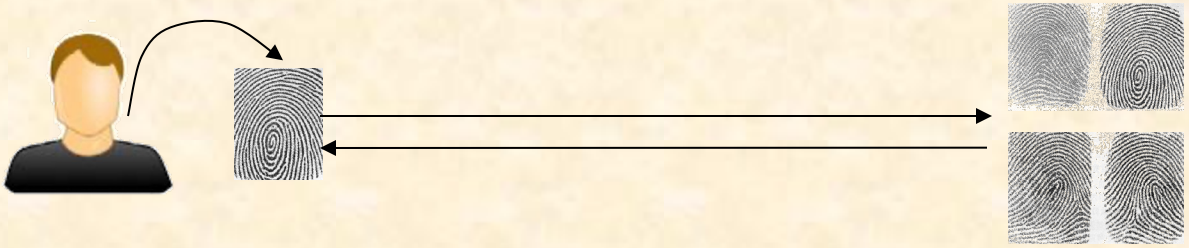
$$\therefore C(n) = (n+1)D(n) = \Theta(n \log n)$$

测试串的相等性

- 通信问题：发射端A发送信号x（x一般为很长的二进制串），接收端B收到信号y，要确定是否x = y。



目标：降低通信量，以减少对通信资源的占用。

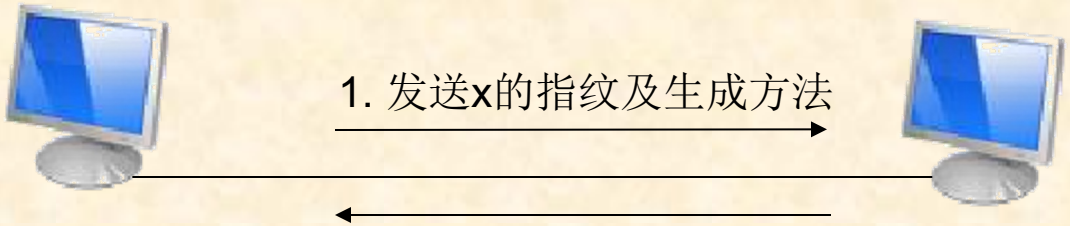


嫌疑人
(A地)

指纹库
(B地)

x ⟶ 生成x的指纹

y



1. 发送x的指纹及生成方法

3. 回传比对结果

A

B

2. 利用同样的方法生成y的指纹，并和x的指纹比对：**如果指纹不相同，**认为$x \neq y$；**否则，认为** $x = y$。

生成指纹

对于一个n位(bit)的二进制串x，I(x)为该二进制串所表示的一个整数。一种生成指纹的方法是：选择一个素数p，令

$$I_p(x) = I(x) \pmod{p}$$

$I_p(x)$即作为x的指纹。

因为$I_p(x)<p$，则$I_p(x)$可用一个不超过$\lfloor \log p \rfloor+1$位的二进制串来表示。因此，如果p 不是很大，那么，指纹$I_p(x)$可以作为一个较短的串来发送，串长度为O(log p)。

例如：$x=(101011)_2=(43)_{10}$, 取$p=(101)_2=(5)_{10}$,

则$I(x) \pmod{p} = 43 \pmod 5 = 3 = (11)_2$, 仅需2比特.

分析

- 上述算法如果给出$x \neq y$，肯定正确; 如果给出$x = y$,则可能出错。
- 出现错误匹配情形是：

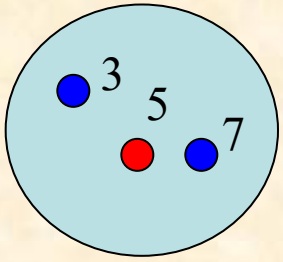
$$x \neq y，但\ I_p(x) = I_p(y)$$

$$\Updownarrow$$

$$x \neq y，\ p\ 整除 I(x) - I(y)$$

例：$x=(101011)_2=(43)_{10}$, $y=(110101)_2=(53)_{10}$。若取 $p=(101)_2=(5)_{10}$, 则会出现错误匹配，即：

$$x \neq y, 但\ I_p(x) = 3 = I_p(y).$$



下面我们先给出算法的框架，再分析出错的概率。

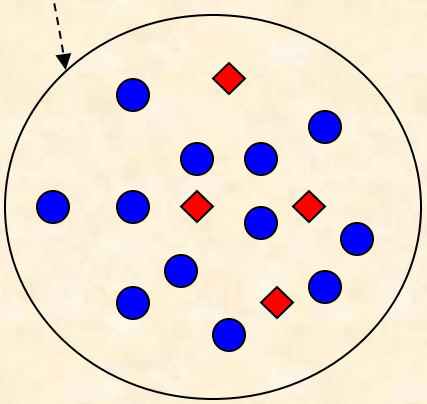
算法：

1．从小于M 的素数集中随机选择一个p

2．A 将p 和 $I_p(x)$发送给B

3．B 检查是否$I_p(x) = I_p(y)$，从而确定x 是否和y 相等。

当算法报告x=y时，可能会出错。是否会出错取决于所选择的素数p。因此，该算法的出错概率为：



$$P_{error} = \frac{\text{会导致出错的素数个数(红色)}}{\text{候选素数个数(红色}+\text{蓝色)}}$$

1. 用π(n)表示小于整数n 的不同素数的个数。那么π(n)渐趋近于n/ln(n)。
2. 如果整数$k < 2^n$，那么能够整除k 的素数的个数小于π(n)。

结论

$$P_{error} = \frac{\text{会导致出错的素数个数}}{\text{候选素数个数}} = \frac{\left|\{p \mid x \neq y, p\text{整除}I(x) - I(y)\}\right|}{\pi(M)}$$

由于，x, y均为n位二进制串，所以：

$$0<I(x),I(y)<2^n, \quad -2^n < \overbrace{I(x)-I(y)}^{k} < 2^n.$$

则由结论2可知，整除I(x)-I(y)的素数个数小于π(n) 。

$$P_{error} \leq \frac{\pi(n)}{\pi(M)}$$

若取M=2n^2，则：

$$P_{error} \leq \frac{\pi(n)}{\pi(M)} = (n \ln n) / (2n^2 / \ln 2n^2) \approx \frac{1}{n}$$

假如算法**连续执行k次**的运行结果为“相等”，则出错概率可以降低为：

$$P_{error} \le \left(\frac{1}{n}\right)^k$$

例子：传输一个百万位的串即n=1,000,000，取$M=2n^2=2\times10^{12}$，传输素数p至多需$\lfloor\log(M)\rfloor+1=41$位，传输指纹也至多41位，共82位。验证1次发生假匹配的概率为1/1,000,000，重复验证5次，假匹配概率可减小到$(10^{-6})^5=10^{-30}$.

例：$x=(101011)_2=(43)_{10}$, $y=(110101)_2=(53)_{10}$,

取$p=(101)_2=(5)_{10}$, $I_p(x)=3=I_p(y)$，出现假匹配；

再取p=3，$I_p(x)=1\neq I_p(y)=2$，验证不通过；

模式匹配问题

- 问题描述：给一个字模式串$X=x_1x_2\ldots x_n$，模式串$Y=y_1y_2\ldots y_m$，其中m<n，问：模式串Y是否在模式串X中出现？不失一般性，假设模式串定义在字符集{0,1}上。
- 最直观的方法：沿模式串X滑动Y，逐个比较子模式串$X(j)=x_j\ldots x_{j+m-1}$和Y。时间复杂度：最坏情况下为O(mn)，例如：

  X=“00000000000000000000000000000000000001”

  Y=“000001”

随机算法

- 思想：同样沿着模式串X滑动Y，但不是直接将模式串Y与每个子模式串$X(j)=x_j \ldots x_{j+m-1}$进行比较；而是借鉴指纹匹配的思想，将Y的指纹与子模式串X(j)的指纹比较，从而判断Y是否与X(j)相同。
- 直接使用上述方法时间复杂性不能有效降低，因为指纹计算同样要耗费时间。
- 然而，分析可以发现：新串X(j+1)的指纹可以很方便地从X(j)的指纹计算出来，即：

$$I_p(X(j+1))=(2I_p(X(j)) - 2^m x_j + x_{j+m}) \pmod p$$

- 为何有上述结论？

$$X(j) = x_j x_{j+1} \cdots x_{j+m-1}$$

$$I(X(j)) = x_j 2^{m-1} + x_{j+1} 2^{m-2} + \cdots + x_{j+m-1} 2^0 \quad (1)$$

$$I(X(j+1)) = x_{j+1} 2^{m-1} + x_{j+2} 2^{m-2} + \cdots + x_{j+m-1} 2^1 + x_{j+m} 2^0 \quad (2)$$

2×(1): $$2I(X(j)) = x_j 2^m + x_{j+1} 2^{m-1} + \cdots + x_{j+m-1} 2^1 \quad (3)$$

(3) - (2): $$2I(X(j)) - I(X(j+1)) = x_j 2^m - x_{j+m} 2^0$$

$$\Downarrow$$

$$I(X(j+1)) = I(X(j)) - x_j 2^m + x_{j+m} 2^0$$

$$\Downarrow$$

$$I_p(X(j+1)) = (2I_p(X(j)) - 2^m x_j + x_{j+m}) (\mathrm{mod}\ p)$$

输入：模式串X，长度为n，模式串Y，长度为m
输出：若Y在X中出现，则返回Y在X中的第1个位置，否则返回-1
1. 从小于M的素数集中随机选择一个素数p
2. j←1
3. 计算$W_p=2^m(\mathrm{mod}\ p)$, $I_p(Y)$ 和 $I_p(X_1)$ //O(m)+O(m)+O(m)
4. while $j\le n-m+1$ //$X_j = x_j\ldots x_{j+m-1}$, 须有$j+m-1\le n$，即$j\le n-m+1$
5. if $I_p(X_j) = I_p(Y)$ then return j //(可能)找到了匹配子串, O(log$p$)
6. $I_p(X_{j+1}) \leftarrow (2I_p(X_j) - x_jW_p + x_{j+m})$ //每步O(1)时间，共O(n)
7. j←j+1
8. end while
9. return -1 //Y肯定不在X中(若在，已由第5步返回j值)

时间复杂度分析：O(m) + O(n) + O(log$p$) = O(m+n)