随机算法

东南大学计算机学院 方效林

本章内容

- 随机算法的基本概念
- 随机采样问题
- 第k小元素问题的随机算法
- Sherwood随机化方法
- 判断字符串是否相等问题
- 子串匹配问题
- 求最近点对的随机算法
- 素数测试

随机算法的基本概念

- 例子
  - 判断函数 $f(x_1,x_2,\ldots x_n)$在区域 D中是否恒为0，f 很复杂，不能数学化简，如何判断就很麻烦
  - 若随机产生一个n维坐标$(r_1,r_2,\ldots r_n)\in D$，代入得$f(r_1,r_2,\ldots r_n)\neq 0$，则可判定区域D内f不恒为0
  - 若对很多个随机产生的坐标进行测试，结果次次均为0，则可说$f\neq 0$的概率是非常小
- 有不少问题，目前只有效率很差的确定性求解算法， 但用随机算法去求解，可以很快地获得相当可信的结果

随机算法的基本概念

- 随机算法
  - 随机算法是一种使用概率和统计方法在其执行过程中对于下一计算步骤作出随机选择的算法
- 随机算法的优越性
  - 对于有些问题：算法简单
  - 对于有些问题：时间复杂性低
  - 对于有些问题：同时兼有简单和时间复杂性低

随机算法的基本概念

- 随机算法分类
  - 随机数值算法
  - Monte Carlo算法
  - Las Vegas算法
  - Sherwood算法
- 随机数值算法
  - 主要用于数值问题求解
  - 算法的输出往往是近似解
  - 近似解的精确度与算法执行时间成正比

随机算法的基本概念

- 随机算法分类
  - 随机数值算法
  - Monte Carlo算法
  - Las Vegas算法
  - Sherwood算法
- Monte Carlo算法
  - 主要用于求解需要准确解的问题
  - 算法可能给出错误解
  - 获得精确解概率与算法执行时间成正比

随机算法的基本概念

- 随机算法分类
  - 随机数值算法
  - Monte Carlo算法
  - Las Vegas算法
  - Sherwood算法
- Las Vegas算法
  - 一旦找到一个解，该解一定是正确的
  - 找到解的概率与算法执行时间成正比
  - 增加对问题反复求解次数，可是求解无效的概率任意小

随机算法的基本概念

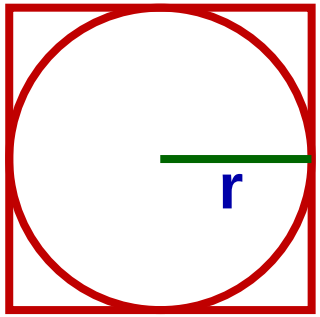
- 随机算法分类
  - 随机数值算法
  - Monte Carlo算法
  - Las Vegas算法
  - Sherwood算法
- Sherwood算法
  - 一定能够求得一个正确解
  - 确定算法的最坏与平均复杂性差别大时, 加入随机性, 即得到Sherwood算法
  - 消除最坏行为与特定实例的联系

随机算法的基本概念

- 随机算法分析的目标
  - 平均时间复杂性：时间复杂性随机变量的均值
  - 获得正确解的概率
  - 获得优化解的概率
  - 解的精确度估计

随机数值算法

- 计算π值
  - 设有一个半径为 r 的圆及其外切四边形
  - 向正方形随机地投掷n个点, 设k个点落入圆内
  - 投掷点落入圆内的概率为 $(\pi r^2)/(4r^2)= \pi/4$.
  - 用k/n逼近π/4, 即$k/n \approx \pi/4$, 于是 $\pi \approx (4k)/n$.



随机数值算法

- **计算π值**

```
k=0;
for i=1 to n do
    随机地产生四边形中的一点(x, y);
    if x²+y²≤1 then
        k=k+1;
return (4k)/n;
```

- **时间复杂性=O(n)，n是随机样本的大小**
- **解的精确度随着随机样本大小n增加而增加**

随机数值算法

- 计算积分$\int_a^b g(x)dx$
  - 强大数定律
    - 假定{s(x)}是相互独立同分布的随机变量序列，如果它们有有限的数学期望E(s(x))=a，则

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}s(x_i)=\mathrm{a}=\mathrm{E}(\mathrm{s}(\mathrm{x}))$$

  - 计算积分
    - 令f(x)=1/(b-a)为概率密度函数，a ≤ x ≤ b
    - {s(x)}为离散随机变量，期望$\mathrm{E}(s(x))=\sum s(x)f(x)$
    - {s(x)}若为连续型的，期望$\mathrm{E}\big(s(x)\big)=\int_a^b s(x)f(x)dx$
    - 令g(x)=s(x)f(x)，则期望 $\mathrm{E}\big(s(x)\big)=\int_a^b g(x)dx$

独立同分布

随机数值算法

- 计算积分$\int_a^b g(x)dx$
  - 强大数定律
    - $\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}s(x_i)=\mathrm{a}=\mathrm{E}(\mathrm{s}(\mathrm{x}))$
  - 计算积分
    - 令g(x)=s(x)f(x)，则期望 $\mathrm{E}\big(s(x)\big)=\int_a^b g(x)dx$
    - $\int_a^b g(x)dx=\mathrm{E}\big(s(x)\big)=\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}s(x_i)$
    - $=\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}g(x_i)/f(x_i)$
    - $=\lim_{n\to\infty}\frac{b-a}{n}\sum_{i=1}^{n}g(x_i)$

随机数值算法

- 计算积分$\int_a^b g(x)dx$
  - R=0;
  - **for** i=1 **to** n **do**
  - 　　随机产生[a, b]中点x;
  - 　　R=R+g(x);
  - **return** (b-a)*R/n

- **时间复杂性=O(n)，n是随机样本的大小**
- **解的精确度随着随机样本大小n增加而增加**

第k小元素(Las Vegas)

- 输入：$S=\{x_1, x_2, \ldots, x_n\}$，整数k，$1 \le k \le n$.
- 输出：S中第k小元素.

第k小元素(Las Vegas)

- 随机算法
  - 在n个数中随机的找一个数A[i]=x, 然后将其余n-1个数与x比较，分别放入三个数组中：
  - S_1(元素均 < x), S_2(元素均=x), S_3(元素均 > x)
    - 若$|S_1|\geq k$ 则调用Select(S_1,k)；
    - 若$|S_1|<k$ 但($|S_1|+|S_2|)\geq k$，则第k小元素就是x；
    - 否则有($|S_1|+|S_2|)< k$，调用Select(S_3,k-$|S_1|$-$|S_2|$)。

第k小元素(Las Vegas)

- 定理
  - 若以等概率方法在n个数中随机取数，则该算法用到的比较次数的期望值不超过4n
  - 证明：
    - 设C(n)是输入规模为n时，算法比较次数的期望值，并设取到任一个数的概率相同。假定取到第j小的数
    - 若j＞k，则需要调用Select(S_1,k)。
      - ∵$|S_1|$= j-1(因为x是第j小的数)，∴调用Select(k,S_1)的比较次数的期望值为C(j-1)。
    - 若j = k，则直接返回第j个元素，无需继续进行比较。
    - 若j＜k，则需要调用Select(S_3,k-j)(j=$|S_1|$+$|S_2|$)。
      - ∵$|S_3|$ = n-j，∴本次调用的比较次数的期望值是C(n-j)

第k小元素(Las Vegas)

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

$$=n+\frac{1}{n}\left(\sum_{i=k}^{n-1}C(i)+\left(C(n-1)+C(n-2)+\cdots+C(n-k+1)\right)\right)$$

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

由于C(i)是非减函数，即i＜j 时，总有C(i)≤C(j)，
C(n)在k=⌈n/2⌉时取得最大值

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

归纳法证明
C(n)≤4n

第k小元素(Las Vegas)

- 归纳法证明 $C(n) \le n + \frac{1}{n}(\sum_{i=n-\frac{n}{2}+1}^{n-1} C(i) + \sum_{i=\frac{n}{2}}^{n-1} C(i)) \le 4n$
  - 当n=1时，C(1) ≤4显然成立
  - 假设当n<m时，C(n) ≤4n成立
  - 证明当n=m时，

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

$$\le n + \frac{1}{n}(\sum_{i=n-\frac{n}{2}+1}^{n-1} 4i + \sum_{i=\frac{n}{2}}^{n-1} 4i) \le n + \frac{1}{n}(8\sum_{i=\frac{n}{2}}^{n-1} i) \le 4n$$

Sherwood随机化方法

- 一般过程
  - 若问题已经有平均性质较好的确定性算法，
  - 但是该算法在最坏情况下效率不高，
  - 引入一个随机数发生器(通常服从均匀分布)
  - 将一个确定性算法改成一个随机算法
  - 例如：
    - 快速排序，每次选择一个基准数，比它小的放左边，大的放右边，如此递归
    - 平均效率很好，但是最坏情况下，每次选择的基准若都是最小的，或是最大的，算法效率不高
    - 可随机预处理(洗牌)，使输入均匀分布，再运行算法

判断字符串是否相等(Monte Carlo)

- 设A处有一个长字符串x（e.g. 长度为10^6），
- B处也有一个长字符串y，
- A将x发给B，由B判断是否有x=y
  - 首先由A发一个x的长度给B，若长度不等，则x≠y
  - 若长度相等，则采用“取指纹”的方法：
    - A对x进行处理，取出x的“指纹”，将指纹发给B
    - 由B检查x的指纹是否等于y 的指纹
    - 若取k次指纹(每次指纹不同)，每次两者结果均相同，则认为x与y是相等的。
    - 随着k的增大，误判率可趋于0

判断字符串是否相等(Monte Carlo)

- 字符串X的指纹取法
  - 令字符串x的二进制编码，长度为n，则 $x < 2^n$
  - 取$f(x) \equiv x \pmod p$作为x的指纹，
    - 其中p是小于$2n^2$的素数，
    - p 二进制长度：$\log_2 p \le \lfloor \log_2(2n^2) \rfloor + 1 = O(\log_2 n)$,
    - f(x)的二进制长度≤ $\log_2 p$，即传输长度可大大缩短
      - x 的二进制是10^6位，即$n=10^6$，则 $p < 2 \times 10^{12} \approx 2^{40.8631}$
      - f(x)位数不超过41位，传输一次指纹 f 可节省约2.5万倍
      - 根据下面所做的分析，错判率小于$\frac{1}{10^6}$
      - 若取5次指纹，错判率小于$\frac{1}{10^{30}}$

判断字符串是否相等(Monte Carlo)

- 错判率分析　　$f(x) \equiv x \pmod p$作为x的指纹
  - B接到指纹f(x)后与f(y)比较，
  - 若$f(x) \neq f(y)$，当然有$x \neq y$。
  - 若$f(x)=f(y)$而$x \neq y$，此时是一个错误匹配
  - 错误匹配的概率有多大？
    - 随机取一个小于$2n^2$的素数p，则对于给定的x和y，
    - 错判率P_{fail} = $\dfrac{x \neq y，但使得f(x)=f(y)的小于2n^2的素数的个数}{小于2n^2的素数的总个数}$

判断字符串是否相等(Monte Carlo)

- 错判率分析　　f(x) ≡ x(mod p)作为x的指纹

  - 错判率P_{fail} = $\frac{x\neq y，但使得f(x)=f(y)的小于2n^2的素数的个数}{小于2n^2的素数的总个数}$

数论定理1：设π(n)是小于n的素数个数，则$\pi(n)\approx\frac{n}{\ln n}$

则小于$2n^2$的素数的总个数为：$\pi(2n^2)\approx\frac{n^2}{ln\ n}$

数论定理2：a≡b (mod p) iff p整除|a-b|

使得f(x)=f(y)的素数的个数 = 能够整除|x-y|的素数的个数

数论定理3：若 $a < 2^n$，则能整除a的素数个数不超过π(n)个

∵|x-y|<max{x,y}≤2^n-1，∴能整除|x-y|的素数个数≤ π(n)

$$错判率P_{fail} = \frac{\pi(n)}{\pi(2n^2)} = \frac{1}{n}$$

判断字符串是否相等(Monte Carlo)

数论定理1：设π(n)是小于n的素数个数，则$\pi(n)\approx\frac{n}{\ln n}$

n=500	1000	10^4	10^5	10^6	10^7	10^8	10^9
π(n)=95	168	1229	9592	78498	664579	5761445	50847478

子串匹配问题

- 给定两个字符串：
  - X = “$x_1\ x_2\ \ldots\ x_n$”
  - Y = “$y_1\ y_2\ \ldots\ y_m$”
- 判断Y是否为X的子串？

X=“maintenance”，
Y=“ten”是S的子串，Y在X中的位置为4

子串匹配问题

- 判断Y是否为X的子串？
  - 记X(j)=$x_j\, x_{j+1}\cdots x_{j+m-1}$
  - 逐一比较X(j)的指纹 f(X(j)) 与Y的指纹 f(Y)
  - f(X(j+1))可根据f(X(j))计算，故算法可很快完成
    - 不失一般性，令X和Y都是0-1串 (二进制编码)
    - $f(X(j+1)) = (x_{j+1}\cdots x_{j+m})(\text{mod } p)$
    - $= (2(x_{j+1}\cdots x_{j+m-1})+x_{j+m})(\text{mod } p)$
    - $= (2(x_{j+1}\cdots x_{j+m-1})+ 2^m x_j - 2^m x_j + x_{j+m})(\text{mod } p)$
    - $= (2(x_j x_{j+1}\cdots x_{j+m-1}) - 2^m x_j + x_{j+m})(\text{mod } p)$
    - $= (2 ((x_j x_{j+1}\cdots x_{j+m-1})\text{mod } p) - 2^m x_j + x_{j+m})(\text{mod } p)$
    - $= (2*f(X(j)) - (2^m \text{ mod } p)x_j + x_{j+m}) (\text{mod } p)$

数论定理4：
(xy+z)(mod p)
=(x(y mod p)+z)(mod p)

被余数都在p附近，与p相差很小，只需一两次加减法即可求余　[-(p-1), 2p-1]

子串匹配问题

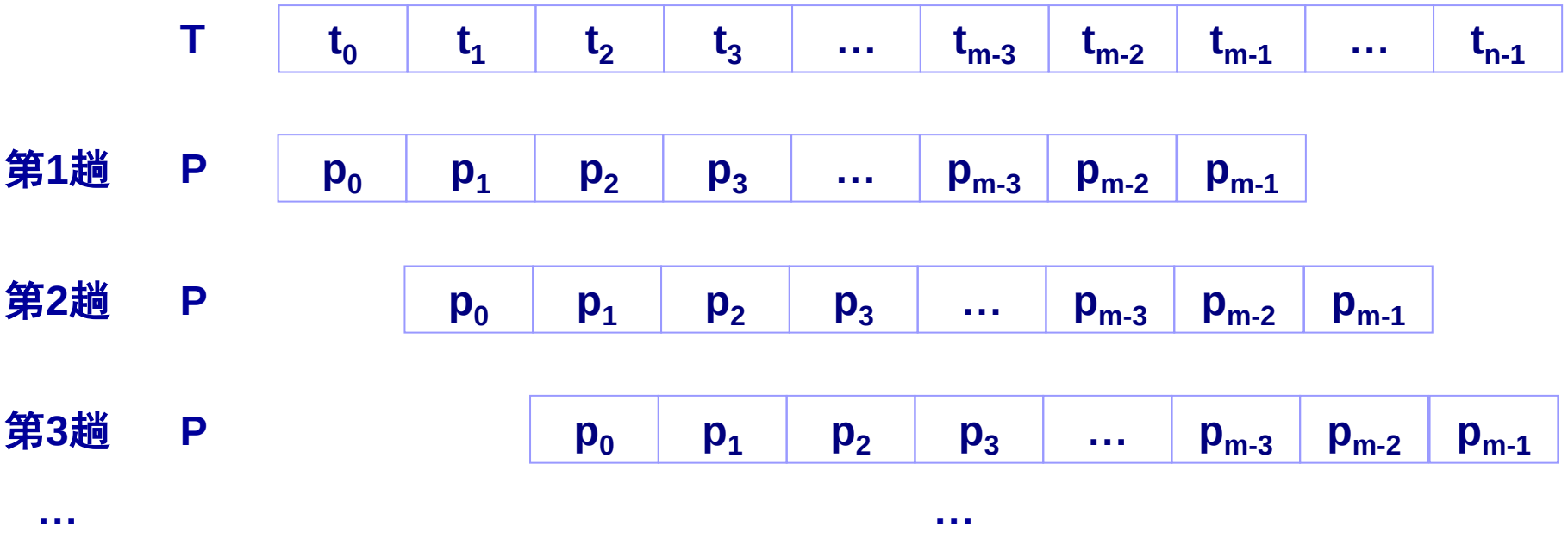
- 出错的概率
  - 当Y≠X(j)，但f(Y)=f(X(j))时产生错误
  - 而f(Y)=f(X(j)) 当且仅当p能整除|Y-X(j)|
    - 当p能整除|Y-X(j)|时，p当然也能整除
    - $Z = |Y-X(1)| \times |Y-X(2)| \times \ldots \times |Y-X(j)| \times \ldots \times |Y-X(n-m+1)|$
    - ∵$|Y-X(j)|<2^m$。
    - ∴$Z = |Y-X(1)|\ |Y-X(2)| \ldots |Y-X(n-m+1)| < (2^m)^{n-m+1} \leq 2^{mn}$

数论定理3：若 $a < 2^n$，则能整除a的素数个数不超过π(n)个

- ∴能整除Z的素数的个数不超过π(mn)个
- $P_{fail} = \dfrac{\text{x不包含y，能够整除Z的素数的个数}}{\text{小于}2mn^2\text{的素数的总个数}} = \dfrac{\pi(mn)}{\pi(2mn^2)} = \dfrac{1}{n}$

子串匹配问题

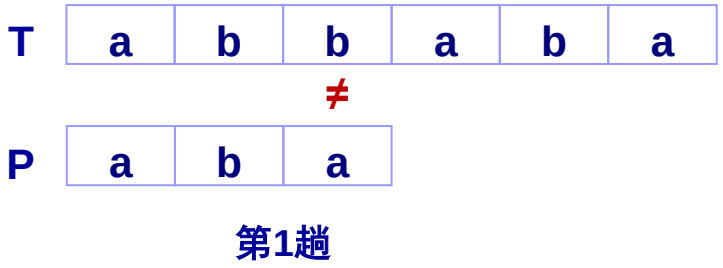
- 字符串的穷举模式匹配算法
  - 匹配失败时，目标串T回溯，模式串P从头开始



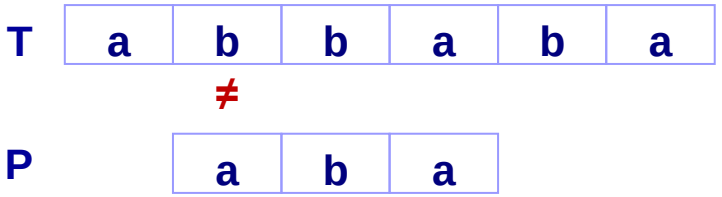
时间复杂度O(m*n)

子串匹配问题

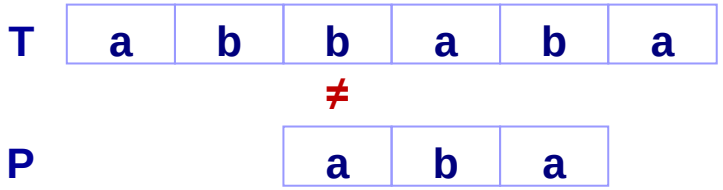
- 字符串的穷举模式匹配算法
  - 匹配失败时，目标串T回溯，模式串P从头开始



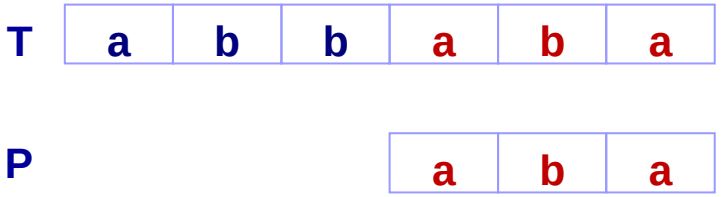
第1趟



第2趟



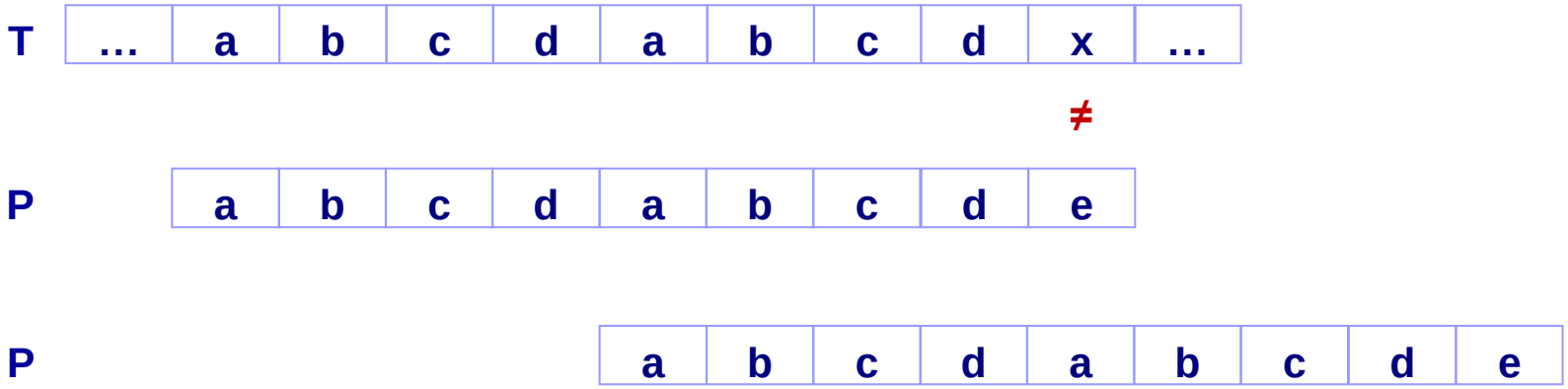
第3趟



第4趟

子串匹配问题

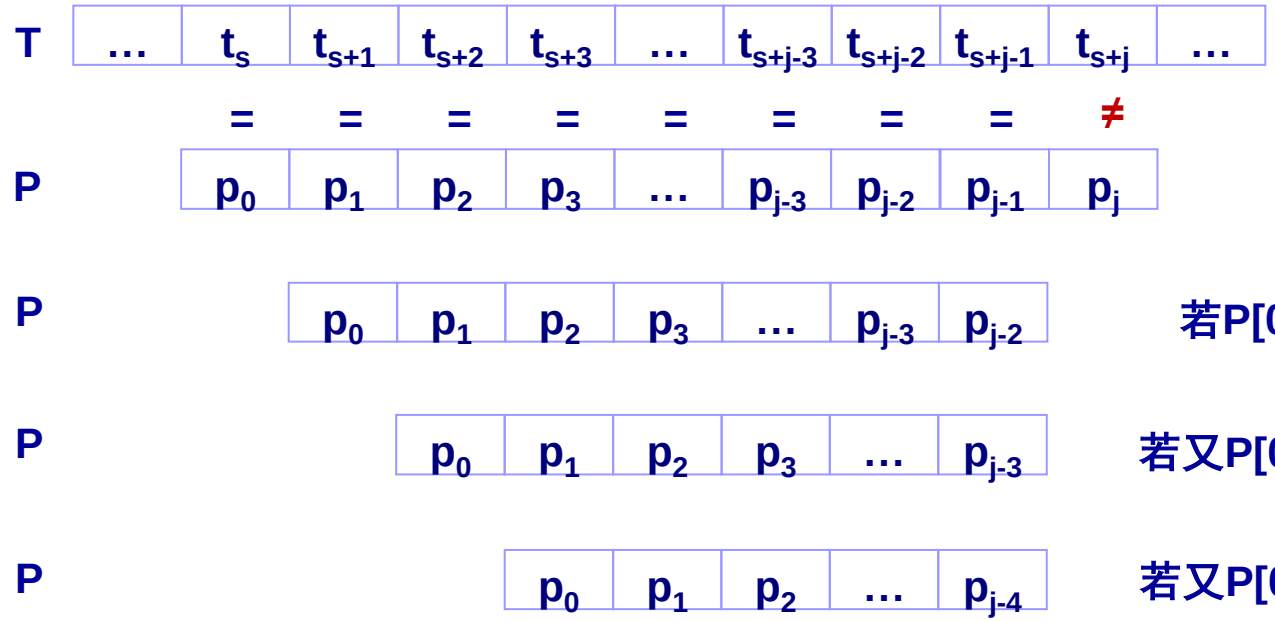
- KMP算法



例子：P中重复出现abcd，但是e和x不匹配时，
可直接向右滑动4个字符开始匹配，可少匹配4趟

子串匹配问题

- KMP算法



设T[s, s+j-1] = P[0, j-1]，
但T[s, s+j] ≠ P[0, j]

若P[0, j-2] ≠ P[1, j-1]，可少匹配1趟

若又P[0, j-3] ≠ P[2, j-1]，可少匹配2趟

若又P[0, j-4] ≠ P[3, j-1]，可少匹配3趟

类推直到前缀P[0, k+1] ≠ 后缀P[j-k-2, j-1]
但是前缀P[0, k] =后缀 P[j-k-1, j-1] 时，
可少匹配 j-k-2 趟，
相当于P直接向右滑动 j-k-2 个字符

子串匹配问题

- KMP算法
  - 对模式串P进行预处理，计算可以滑过多少个字符

$$next(j)=\begin{cases} -1, & \text{当 } j=0 \\ k+1, & \text{当 } 0\le k<j-1 \text{ 且使 } p_0p_1\cdots p_k=p_{j-k-1}p_{j-k}\cdots p_{j-1} \text{ 的最大数} \\ 0, & \text{其他情况} \end{cases}$$

next(j)直观含义：[0, j-1] 中前缀和后缀相等的最大长度
next(j)直观作用：可滑过j-next(j)位不用匹配

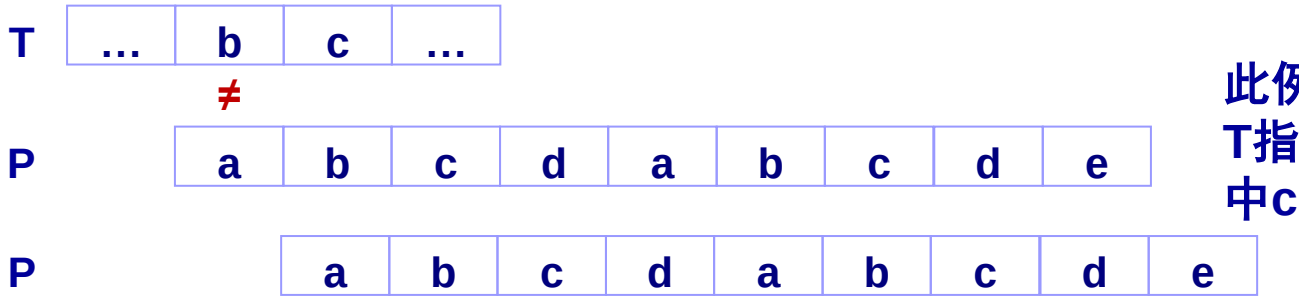
下标j	0	1	2	3	4	5	6	7	8
P	a	b	c	d	a	b	c	d	e
next(j)	-1	0	0	0	0	1	2	3	4

子串匹配问题

- KMP算法
  - 对模式串P进行预处理，计算可以滑过多少个字符

$$next(j)=\begin{cases}-1, & \text{当 } j=0\\ k+1, & \text{当 } 0\le k<j-1 \text{ 且使 } p_0p_1\cdots p_k=p_{j-k-1}p_{j-k}\cdots p_{j-1} \text{ 的最大数}\\ 0, & \text{其他情况}\end{cases}$$

next(j)=-1表示匹配失败时，T的指针加1，P的指针指向p[0]
next(j)=k+1表示匹配失败时，P的指针指向p[k+1]，
next(j)=0表示匹配失败时，P的指针指向p[0]



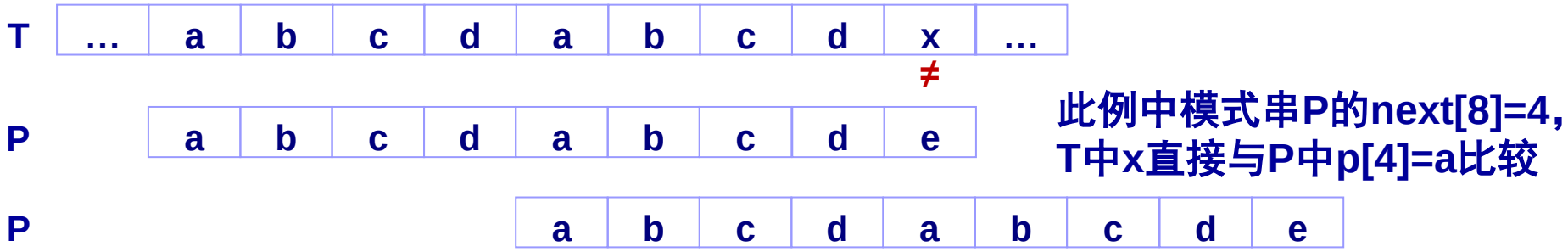
此例中模式串P的next[0]=-1，T指针加1，P指向p[0]，即T中c与P中p[0]=a进行比较

子串匹配问题

- KMP算法
  - 对模式串P进行预处理，计算可以滑过多少个字符

$$next(j)=\begin{cases} -1, & \text{当 } j=0 \\ k+1, & \text{当 } 0\le k<j-1 \text{ 且使 } p_0p_1\cdots p_k=p_{j-k-1}p_{j-k}\cdots p_{j-1} \text{ 的最大数} \\ 0, & \text{其他情况} \end{cases}$$

next(j)=-1表示匹配失败时，T的指针加1，P的指针指向p[0]
next(j)=k+1表示匹配失败时，P的指针指向p[k+1]，
next(j)=0表示匹配失败时，P的指针指向p[0]

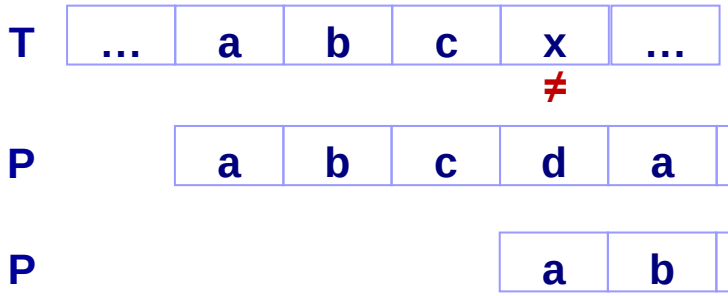


子串匹配问题

- KMP算法
  - 对模式串P进行预处理，计算可以滑过多少个字符

$$next(j)=\begin{cases}-1, & \text{当 } j=0\\ k+1, & \text{当 } 0\le k<j-1 \text{ 且使 } p_0p_1\cdots p_k=p_{j-k-1}p_{j-k}\cdots p_{j-1} \text{ 的最大数}\\ 0, & \text{其他情况}\end{cases}$$

next(j)=-1表示匹配失败时，T的指针加1，P的指针指向p[0]
next(j)=k+1表示匹配失败时，P的指针指向p[k+1]，
next(j)=0表示匹配失败时，P的指针指向p[0]



此例中模式串P的next[3]=0，
T中x直接与P中p[0]=a比较

子串匹配问题

- KMP算法
  - 对模式串P进行预处理，计算可以滑过多少个字符

$$next(j)=\begin{cases} -1, & \text{当 } j=0 \\ k+1, & \text{当 } 0\le k<j-1 \text{ 且使 } p_0p_1\cdots p_k=p_{j-k-1}p_{j-k}\cdots p_{j-1} \text{ 的最大数} \\ 0, & \text{其他情况} \end{cases}$$

可按定义直接计算next，下面介绍一种快速的计算next的方法

```
j=0;k=-1;next[0]=-1;
while(j<pLength)  {
   if(k==-1 || ch[j]==ch[k]) {
      j++;k++;next[j]=k;
   }
   else k = next[k];
}
```

假设已知next(j)=x，现在计算next(j+1)

若$p_x=p_j$，则next(j+1) = x+1 = next(j)+1

否则，设next(x)=h，(此时有p[0,h-1]=p[x-h,x-1]=p[j-h,j-1])

若$p_h=p_j$，则next(j+1) = h+1

否则，令next(h)=t, (此时有p[0,t-1]=p[h-t,h-1]=p[j-t,j-1])

继续判断是否$p_t=p_j$，直到找到或者到next(0) = -1

……

子串匹配问题

- KMP算法
  - 对模式串P进行预处理，计算可以滑过多少个字符

$$next(j)=\begin{cases}-1, & \text{当 } j=0\\ k+1, & \text{当 } 0\le k<j-1 \text{ 且使 } p_0p_1\cdots p_k=p_{j-k-1}p_{j-k}\cdots p_{j-1} \text{ 的最大数}\\ 0, & \text{其他情况}\end{cases}$$

可按定义直接计算next，下面介绍一种快速的计算next的方法

```
j=0;k=-1;next[0]=-1;
while(j<pLength)  {
   if(k==-1 || ch[j]==ch[k]) {
      j++;k++;next[j]=k;
   }
   else k = next[k];
}
```

下标j	0	1	2	3	4	5	6	7	8	9	10
P	a	b	a	b	c	a	b	a	b	a	x
next(j)	-1	0	0	1	2	0	1	2	3	4	?

≠

=

next(10)=2+1=3

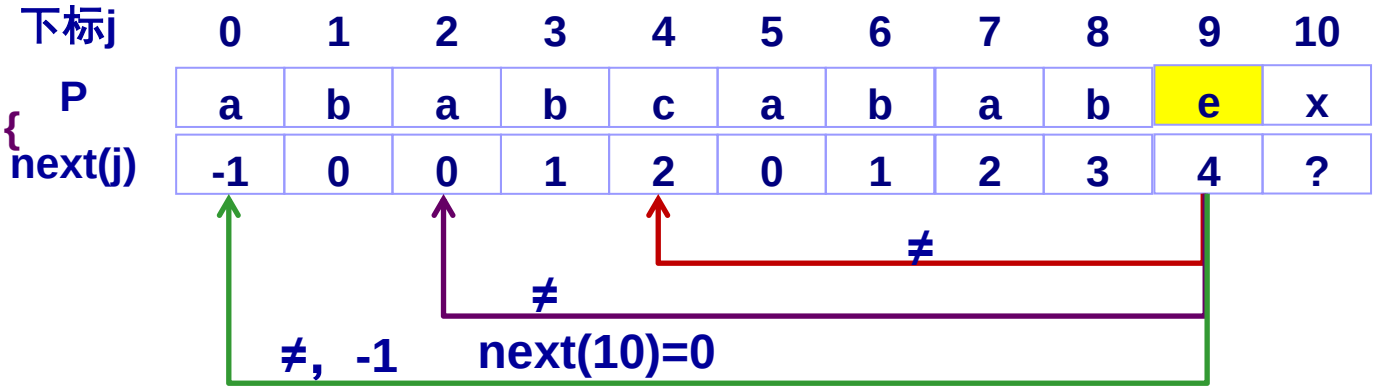
子串匹配问题

- KMP算法
  - 对模式串P进行预处理，计算可以滑过多少个字符

$$next(j)=\begin{cases} -1, & \text{当 } j=0 \\ k+1, & \text{当 } 0\le k<j-1 \text{ 且使 } p_0p_1\cdots p_k=p_{j-k-1}p_{j-k}\cdots p_{j-1} \text{ 的最大数} \\ 0, & \text{其他情况} \end{cases}$$

可按定义直接计算next，下面介绍一种快速的计算next的方法

```
j=0;k=-1;next[0]=-1;
while(j<pLength)  {
   if(k==-1 || ch[j]==ch[k]) {
      j++;k++;next[j]=k;
   }
   else k = next[k];
}
```



素数测试

- RSA加密算法用到大素数

公钥: (n,e)	n是两个大素数p和q的乘积 (p和q必须保密), e与(p-1)(q-1)互质
私钥: (n,d)	$e \times d \equiv 1 \pmod{(p-1)(q-1)}$
加密	$C = M^e \pmod{n}$
解密	$M = C^d \pmod{n}$

例：令p=3，q=11，
n=p×q=3×11=33；
(p-1)(q-1)=2×10=20；
取e=3，（3与20互质）
ed - 1 = kφ(n), 有ed + φ(n)k = 1
使用扩展欧几里德算法
即3×d≡1 mod 20，可取d=7
明文25，加密得密文16=25^3 mod 33
密文16，解密得明文25=16^7 mod 33

其中关键过程是得到两个大素数，但是当今判断一个非常大的数是否是素数并非简单的事

素数测试

- 大整数的素因子分解
  - 如n=10^{60}（比已知最大素数小很多），
  - 若算法时间复杂度为O(n)，设高速计算机
  - 每秒1亿亿次(10^{16})基本运算，则需10^{44}秒，
  - 大于10^{42}分钟，大于10^{40}小时，大于10^{38}天，
  - 大于10^{35}年！
  - O(n)时间的算法绝对不能接受！

素数测试

- 求$a^m \pmod n$的算法($m \le n$)
  - m的二进制表示为$b_k b_{k-1} \ldots b_1 b_0$ ($b_k=1$, $k \approx \log_2 m$)
  - 例：$m=41= b_k b_{k-1} \ldots b_1 b_0 = 101001_{(2)}$，($k=5$)

素数测试

- 求$a^m(\mathrm{mod}\ n)$的算法($m \le n$)
  - 求a^m可以用下述方法：
    - 从m的二进制的高位到低位，平方，遇1还要乘a
  - 例如，计算a^{41}，
  - 初始C=1
  - b_5=1：$C=C^2=1$，$\because b_5=1$，$C=a*C=a$；
  - b_5b_4=10：$C=C^2=a^2$
  - $b_5b_4b_3$=101：$C=C^2=a^4$，$\because b_3=1$，$C=a*C=a^5$
  - $b_5b_4b_3b_2$=1010：$C=C^2=a^{10}$
  - $b_5b_4b_3b_2b_1$=10100：$C=C^2=a^{20}$
  - $b_5b_4b_3b_2b_1b_0$=101001：$C=C^2=a^{40}$,$\because b_0=1$,$C=a*C=a^{41}$

素数测试

- 求$a^m \pmod n$的算法($m \le n$)
  - 求a^m可以用下述方法：
    - 从m的二进制的高位到低位，平方，遇1还要乘a
  - ∵模运算有规则：(x*y)%n =((x%n) * (y%n)) % n
  - 求$a^m \pmod n$可在求a^m过程中的每一步求模
  - EXPMOD(a,m,n)
  - C=1;
  - for j=k to 0 do
  - $C=C^2 \pmod n$
  - if b_j=1 then $C=a*C \pmod n$
  - return C

每一步求模的好处是：
C的最大值不超过n-1，
中间值不超过$\max\{(n-1)^2, a(n-1)\}$
求$a^m \pmod n$时不会占用很多空间

素数测试

- 求$a^m \pmod n$的算法($m \le n$)
  - 求a^m可以用下述方法：
    - 从m的二进制的高位到低位，平方，遇1还要乘a
  - ∵模运算有规则：(x*y)%n =((x%n) * (y%n)) % n
  - 求$a^m \pmod n$可在求a^m过程中的每一步求模
  - 例如，计算$a^5 \bmod n$
  - 初始C=1
  - b_2=1：$C=C^2 \bmod n$，∵b_2=1，$C=a*C \bmod n$；
  - b_2b_1=10：$C=C^2 \bmod n$
  - $b_2b_1b_0$=101：$C=C^2 \bmod n$，∵b_0=1，$C=a*C \bmod n$

素数测试

- **Fermat小定理**
  - 若n为素数，0<a<n，则有$a^{n-1} \equiv 1 \pmod n$
  - 即若$a^{n-1} \neq 1 \pmod n$，则n必为合数
  - 而条件$a^{n-1} \equiv 1 \pmod n$是素数的必要条件，非充分
    - n是合数，对任意a，也可能$a^{n-1} \equiv 1 \pmod n$，称之为Carmichael数，例如前几个561，1105，1729，2465，…，小于1亿只有255个Carmichael数
    - Carmichael数虽然分布很稀，但仍有无穷多个

用与n互质的a去测试，Carmichael数会漏网
a从(2,3,…,n-1)全部判断不现实

定理：设n是一个奇合数，则使得Miller-Rabin算法回答为“合数”的a（合数的证据数）在{1,2,…n-1}中至少有一半（≥(n-1)/2）

素数测试

总的时间复杂度$O(\log^3 n)$

- Miller-Rabin算法
  - 定理：设n为素数,且0<x<n，则当$x^2 \equiv 1 \pmod n$时，必有 x=1 或 x=n-1
  - 推论：当0<x<n 且 $x^2 \equiv 1 \pmod n$成立时，若x≠1且x≠n-1，则n是合数

$n-1=m*2^q$

$a^{n-1}=a^{m*2^q}$

```
O(log^3 n)   x=a^m(mod n);
q≤O(log n)   for i=1 to q do
O(log^2 n)       y = x^2(mod n);
                 if y=1 且 x≠n-1且 x≠1 then //此时满足推论
                     return “n为合数”;
                 x = y;
             if x≠1 then return “n为合数”
             else return “n为素数” //此时满足Fermat小定理
```

素数测试

- **Carmichael数561：**
  - **n-1=560=2^4*35，即有q=4，m=35。**
  - **假定选a=7，则$a^m=7^{35}\equiv241(\text{mod } 561)$。**
    - **7^{35}，7^{70}，7^{140}，7^{280}，7^{560}**
    - **mod n 后分别为：241，298，166，67，1**
    - **$x=7^{280}\equiv67(\text{mod } 561)$，**
    - **而$x^2=7^{560}\equiv1(\text{mod } 561)$，x≠1且x≠n-1，则561是合数**