第9章 随机算法

Las Vegas 型随机算法

随机快速排序

随机选择

随机 *n*后放置

Monte Carlo型随机算法

主元素测试

串相等测试

模式匹配

素数测试

随机算法的分类与局限性

随机快速排序算法

算法9.1 随机快速排序算法

输入：包含 n 个元素的数组

输出：经过排序的 n 个元素的数组

1. 若数组包含 0 或 1 个元素则返回
2. 从数组中随机选择一个元素作为枢轴元素
3. 把数组元素分为三个子数组，并且按照 A, B, C 顺序排列

   A：包含比枢轴元素小的元素；

   B：包含与枢轴元素相等的元素；

   C：包含比枢轴元素大的元素.
4. 对 A 和 C 递归地执行上述步骤.

算法分析

定理9.1 设数组含n个不同元素，对任意常数$\varepsilon>0$，随机快速排序算法的期望比较次数

$$T(n) \le 2\, n \ln n.$$

证明一：求解递推式

随机选取枢轴元素，其位于排序后第 i 位置 ($i=1,2,\ldots,n$)的概率是$1/n$，A和C的元素数分别是 i 个和 $n-i-1$个, 得

$$T(n) = (n-1) + \frac{1}{n}\sum_{i=1}^{n-1}[T(i) + T(n-i-1)]$$

解为$\Theta(n\log n)$. 可归纳证明精确上界是 $T(n) \le 2n \ln n$.

$$T(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} 2i \ln i \le (n-1) + \frac{2}{n}\int_1^n 2x \ln x dx$$

$$\le (n-1) + \frac{2}{n}(n^2 \ln n - \frac{n^2}{2} + \frac{1}{2}) \le 2n \ln n$$

北京大学

随机选择算法

算法 RandSelect(A, p, r, k)　　//从$A[p..r]$中选第k小

1. if $p=r$ then return $A[p]$
2. $i \leftarrow$ Random(p, r)
3. 以 $A[i]$ 为标准划分 A
4. $j \leftarrow$划分后小于等于 $A[i]$ 的数构成数组的大小
5. if $k \leq j$
6. 　　then return RandSelect (A, p, $p+j-1$, k)
7. 　　else return RandSelect (A, $p+j$, r, $k-j$)

时间期望值估计

假设1.. n 中每个数被选的概率相等，并且假设第 k 个数总是出现在划分后两个数组中较大的数组，算法的期望时间为

$$T(n) \le \frac{1}{n}(T(n-1)+T(n-2)+\ldots+T(\frac{n}{2}+1)$$

$$+T(\frac{n}{2})+T(\frac{n}{2}+1)+\ldots+T(n-1))+O(n) \le \frac{2}{n}\sum_{i=n/2}^{n-1}T(i)+O(n)$$

归纳证明 $T(n) \le cn$. 归纳步骤如下：假设对 $k<n$ 命题为真，

$$T(n) \le \frac{2}{n}[c\frac{n}{2}+c(\frac{n}{2}+1)+\ldots+c(n-1)]+tn = \frac{2c}{n}\frac{(\frac{n}{2}+n-1)\frac{n}{2}}{2}+tn$$

$$= \frac{3cn}{4}-\frac{c}{2}+tn \le \frac{3cn}{4}+tn = (\frac{3}{4}+\frac{t}{c})cn \le cn,$$ 取 $c \ge 4t$ 即可

n后放置的随机选择

算法BoolQueen(n)

1. $k \leftarrow 1$ // k 放皇后的行号
2. $count \leftarrow 0$ // $count$ 放好的皇后数
3. while $k \le n$ do
4.   for $i \leftarrow 1$ to n do // i 为待选列号
5.   检查 i 与前面 k-1个皇后的相容性
6.   如果相容则将 i 加入S
7.   if $S \neq \emptyset$ then
8. $j \leftarrow$ Random(1,$|S|$)
9. $x_k \leftarrow S[j]$
10. $count \leftarrow count+1$
11. $k \leftarrow k+1$
12.   else $k \leftarrow n+1$
13. return $count$

n后问题的随机算法

算法QueenLV(n)　　//重复调用随机算法BoolQueen

1. $p \leftarrow$BoolQueen(n)
2. while $p<n$ do
3. 　$p \leftarrow$BoolQueen(n)

改进算法---与回朔相结合

设 $stopVegas \le n$, 表示用QueenLV算法放置的皇后数

剩下 n - $stopVegas$ 个皇后用回朔方法放置

$stopVegas$ = 0 时是完全的回朔算法

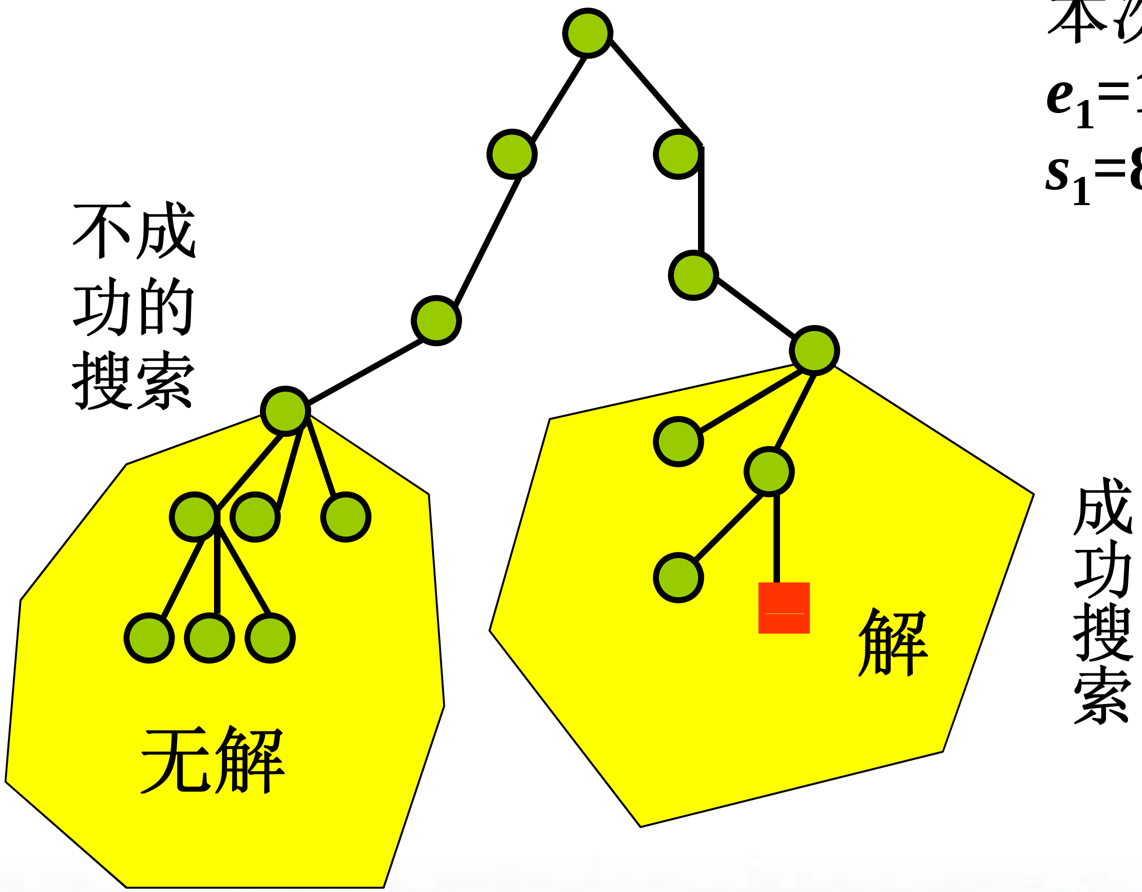
$stopVegas$ = n 时是完全的Las Vegas算法

成功搜索与不成功搜索

本次抽样：

e_1=10

s_1=8



改进算法的分析

对于不同的 *stopVegas* 值，设

- p 为算法成功概率
- s 为一次成功搜索访问的结点数的平均值
- e 为一次不成功搜索访问的结点数的平均值
- t 为算法找到一个解的平均时间

$$t = ps + (1-p)(e+t) \Rightarrow t = s + e\frac{1-p}{p}$$

n=12时的统计数据：*stopVegas* = 5时算法效率高

stopVegas	*p*	*s*	*e*	*t*
0	1.0000	262.00	-	262.00
5	0.5039	33.88	47.23	80.39
12	0.0465	13.00	10.20	222.11

北京大学

LasVegas型随机算法总结

- 特点
  - 通过修改确定性算法得到，一般将算法的某步的确定型选择变成随机选择
  - 一次运行可能得不到解；若得到解，则解一定是正确的
  - 改进途径：与确定型算法相结合
  - 有可能改进确定型算法平均情况下的时间复杂度
- 有效的 LasVegas 算法
  - 运行时间是随机变量，期望运行时间是输入的多项式且总能给出正确答案的随机算法

主元素测试

问题

主元素：出现次数超过一半以上的元素

输入：***n*** 个元素的数组 ***T***

输出：如果存在主元素则输出“**true**”，否则“**false**”

算法 Majority(T,n)

1. $i \leftarrow$ Random(1,n)
2. $x \leftarrow T(i)$
3. 计数 x 在 T 中出现的个数 k
4. **if** $k>n/2$ **then return true**
5. **else return false**

算法的正确性

若回答true：则T存在主元素，算法正确；若回答false，T仍可能存在主元素，算法可能出错. 回答正确概率大于1/2.

算法 BoolMajority(T,n)

1. if Majority(T,n) then return true
2. else return Majority(T,n)

BoolMajority 算法正确的概率为

$$p+(1-p)p=2p-p^2=1-(1-p)^2>\frac{3}{4}$$

调用 k 次Majority算法正确的概率为

$$p+(1-p)p+(1-p)^2p+\ldots+(1-p)^{k-1}p=1-(1-p)^k>1-2^{-k}$$

调用次数 k	1	2	3	4	5	6
正确概率大于	0.5	0.75	0.875	0.938	0.969	0.985

北京大学

改进途径

对于任意给定的 ε>0, 如果要使出错的概率不超过 ε，则调用次数 *k* 满足

$$\left(\tfrac{1}{2}\right)^k \le \varepsilon \Rightarrow k\log\tfrac{1}{2} \le \log\varepsilon \Rightarrow -k \le \log\varepsilon$$
$$\Rightarrow k \ge -\log\varepsilon \Rightarrow k \ge \left\lceil \log\tfrac{1}{\varepsilon} \right\rceil$$

出错概率不超过ε 的算法——**MCMajority**

算法 MCMajority(*T*,*n*,ε)

1. $k \leftarrow \lceil \log(1/\varepsilon) \rceil$
2. for $i \leftarrow 1$ to k
3.   if Majority(*T*,*n*) then return true
4. return false

串相等测试

问题：*A* 有一个长串 *x*, *B* 有长串 *y*，*A* 和 *B* 希望知道 $x=y$?

方法一：

A 将 *x* 发送给 *B*，*B* 测试 $x = y$?

发送消耗：长串占用信道资源大

方法二：

A 用 *x* 导出一个短串 $f(x)$ (fingerprints)

A 将 $f(x)$ 发送到 *B*

B 使用同样方法导出相对于 *y* 的短串 $f(y)$

B 比较 $f(x)$ 与 $f(y)$

如果 $f(x) \neq f(y)$，则 $x \neq y$;

如果 $f(x) = f(y)$，则不确定.

指纹产生

设 x 和 y 的二进制表示为正整数 $I(x), I(y)$

选择素数 p，指纹函数为

$$I_p(x) = I(x) \bmod p$$

A 传送 p 和 $I_p(x)$ 给 B. 当 p 不太大时，传送一个短串.

存在问题：

$$x = y \Rightarrow I_p(x) = I_p(y)$$

$$I_p(x) = I_p(y) \not\Rightarrow x = y$$

出错条件：固定

$$p \mid (I(x) - I(y))$$

改进算法的途径

改进方法： 随机选择素数 p 进行测试

算法 StringEqualityTest

1. 随机选择小于 M 的素数 p　　//M为正整数
2. A 发送 p 和 $I_p(x)$ 给 B
3. B 测试是否 $I_p(x) = I_p(y)$

出错的必要条件：

x 的位数等于 y 的位数

$p \mid (I(x)-I(y))$

有关素数的性质

函数 $\pi(t)$：小于 t 的不同的素数个数，

例如，$\pi(20)=8$，素数8个：2, 3, 5, 7, 11, 13, 17, 19

两个相关结果：

(1) 素数定理 $\pi(t) \approx t / \ln t$

(2) 若 $k<2^n$，n不太小，整除 k 的不同素数个数小于 $\pi(n)$

n	10^3	10^4	10^5	10^6	10^7
$\pi(n)$	168	1229	9592	78498	664579
$t/\ln t$	145	1086	8686	72382	620421
比值	1.159	1.132	1.104	1.085	1.071

$k < 2^{10} =1024$，整除 k 的素数个数 $< \pi(10) = 4$,

例如 $k = 984$，整除 984 的素数只有 2, 3, 41

出错概率估计

n: x 和 y 的二进制表示的位数

$x, y \le 2^n$

若选择 $M \ge 2n^2$, 一次测试的出错概率估计:

$$\frac{|\{p \mid p\text{是小于}2^n\text{的素数，且 } p \text{ 整除 } I(x)-I(y)\}|}{\pi(M)}$$

$$\le \frac{\pi(n)}{\pi(M)} \approx \frac{n/\ln n}{2n^2/\ln(2n^2)} \approx \frac{n/\ln n}{2n^2/2\ln n} \le \frac{1}{n}$$

改进算法

重复执行 j 次，每次随机选择小于M 的素数

算法 StringTest

输入：x, y，n位二进制数

输出：“Yes” (如果$x = y$)； 或者“No” (如果 $x \neq y$)

1. for $i \leftarrow 1$ to j
2. 随机选择小于M 的素数 p // M为正整数
3. A 发送 p 和 $I_p(x)$ 给 B
4. B 测试
5. if $I_p(x) \neq I_p(y)$
6. then return “No”
7. return”Yes”

算法分析

令 $j = \lceil \log\log n \rceil$, 则算法出错的概率

$$\left(\frac{1}{n}\right)^j \le \frac{1}{n^{\lceil \log\log n \rceil}}$$

实例： x 和 y 是1000000位二进制数

$n = 10^6$

$M = 2 \cdot 10^{12} = 2^{40.8631}$

素数 p 的二进制表示至多 $\lfloor \log M \rfloor + 1 = 41$ 位

$I_p(x)$ 的位数至多 $\lfloor \log(p-1) \rfloor + 1 \le \lfloor \log M \rfloor + 1 = 41$

总共传送 82 位

模式匹配

问题：输入二进制串 $X = x_1 x_2 \ldots x_n$，$Y = y_1 y_2 \ldots y_m$，$m \le n$

输出：若Y在 X 中，Y 出现的第一个位置；否则为 **“0”**

算法一：顺序比较

初始 Y 与 X 的首元素对齐，依次从前到后比较 X 与 Y 的元素. 如果 X 与 Y 的所有元素都相等，输出 Y 的首位置 j；否则将 Y 的位置向后移动一个字符，重复原来过程.

运行时间：$O(mn)$

算法二：利用有限状态自动机的模式匹配算法

(Knuth,Morris,Pratt), Introduction to Algorithms

运行时间：$O(m+n)$

随机算法

算法三　利用串比较的随机算法

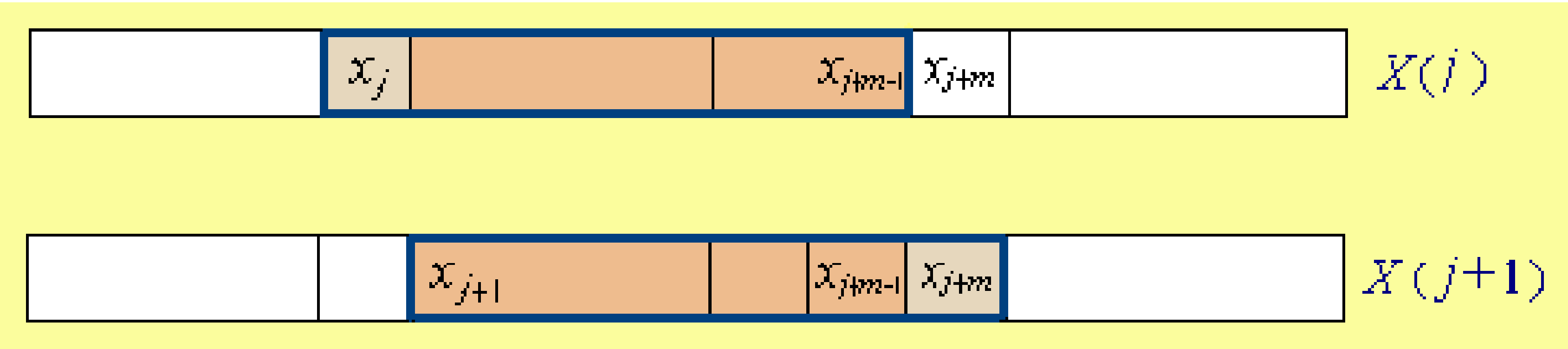
设计思想：设 $X(j) = x_j x_{j+1} \ldots x_{j+m-1}$，把 $X(j)$ $(j=1,2, \ldots, n-m+1)$ 与Y 逐个字符的比较，改成对指纹 $I_p(X(j))$与 $I_p(Y)$ 的比较

$X(j)$与$X(j+1)$的关系

$$X(j) = 2^{m-1}x_j + \underline{2^{m-2}x_{j+1} + \ldots + 2x_{j+m-2} + x_{j+m-1}}$$

$$X(j+1) = \underline{2^{m-1}x_{j+1} + 2^{m-2}x_{j+2} + \ldots + 2x_{j+m-1}} + x_{j+m}$$

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



算法三的关键技术

由 $I_p(X(j))$ 求 $I_p(X(j+1))$ 的公式

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

$$W_p = 2^m (\bmod p)$$

该公式说明：由 $I_p(X(j))$ 计算 $I_p(X(j+1))$ 仅需要常数时间

公式的证明：

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

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

令 $W_p = 2^m (\bmod p)$

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

算法

算法 PatternMaching

输入：串 X 和 Y, $|X|=n$, $|Y|=m$, $m \le n$

输出：如果 Y 在 X 中，Y 出现的第一位置；否则为“0”

1. 从小于 M 的素数集合中随机选择素数 p
2. $j \leftarrow 1$
3. $W_p \leftarrow 2^m \pmod p$
4. $I_p(X(j)) \leftarrow I(X(j)) \pmod p$
5. $I_p(Y) \leftarrow I(Y) \pmod p$
6. while $j \le n-m+1$ do
7. if $I_p(X(j)) = I_p(Y)$ then return j
8. $I_p(X(j+1)) \leftarrow (2I_p(X(j)) - W_p x_j + x_{j+m}) \pmod p$
9. $j \leftarrow j+1$
10. return 0

算法分析

时间复杂度为 $O(m+n)$

W_p, $I_p(Y)$, $I_p(X(1))$ 计算 $O(m)$ 时间

从 $I_p(X(j))$ 计算 $I_p(X(j+1))$ 总共需要$O(n)$时间

出错条件：

$$Y \neq X(j) \wedge I_p(Y) = I_p(X(j))$$

$$\Leftrightarrow p \,\Big|\, \prod_{\{j \mid Y \neq X(j)\}} |I(Y) - I(X(j))|$$

出错概率：乘积大小不超过 $(2^m)^n$，整除它的素数个数不超过 $\pi(mn)$，选 $M = 2mn^2$，则出错概率不超过

$$\frac{\pi(mn)}{\pi(M)} \approx \frac{mn/\ln(mn)}{2mn^2/\ln(mn^2)} = \frac{\ln(mn^2)}{2n\ln(mn)} < \frac{\ln(mn)^2}{2n\ln(mn)} = \frac{1}{n}$$

素数测试

- 求 x 的 m 次幂
- 求 a 的模 n 的 m 次幂
- **Fermart**小定理
- 测试算法分析

求 x 的 m 次幂

输入：x为实数

$m = d_k d_{k-1} \ldots d_1 d_0$ 为二进制自然数

输出：x^m

算法 Exp(x,m)

```
y←1;
for j←k  downto  0  do
    y←y² ;
    if  d_j=1  then  y←xy
return y
```

实例

x^{101}: $d_2=1, d_1=0, d_0=1$

$y=1$

$j=2$	$j=1$	$j=0$
$y=1$	$y=x^2$	$y=x^4$
$y=x$		$y=x^5$

北京大学

a模n的m次幂

输入：$a, m, n \in Z^+, m \le n$,

$m = b_k b_{k-1} \ldots b_1 b_0$为二进制自然数

输出：$a^m \pmod n$

算法ExpMod(a, m,n)

1. $c \leftarrow 1$
2. for $j \leftarrow k$ downto 0 do
3. $\quad c \leftarrow c^2 \pmod n$
4. $\quad$ if $b_j=1$ then $c \leftarrow ac \pmod n$
5. return c

$T(n)=O(k\log^2 n)=O(\log^3 n)$ 以位乘作为基本运算

Fermart小定理:测试原理

定理1： 如果 n为素数，则对所有的正整数 $a \neq 0 \pmod n$有

$$a^{n-1} \equiv 1 \pmod n$$

素数测试原理：检测 $2^{n-1} \equiv 1 \pmod n$. 如是，输出“素数”否则输出“合数”

算法 Ptest1(n)

输入：奇整数n, n>5

输出： “prime” 或者 “composite”

1. if ExpMod(2,n–1,n) =1 then return prime
2. else return composite

问题：算法 Ptest1只对a=2进行测试. 如果 n为合数且算法输出“素数”，则称 n 为基2的伪素数. 例如341.

改进算法(一)

改进算法：随机选取2--n-1中的数，进行测试. 如取a=3，3^{340}(mod 341) ≡ 56，341不是素数.

算法Ptest2(n)

1. a←Random(2, n-2)
2. if Expmod(a, n-1, n)=1 then return prime
3. else return composite

- **Fermat** 小定理是必要条件，不是充分条件，满足该条件的也可能是合数. 对所有与 n 互素的正整数 a 都满足条件的合数 n 称为 **Carmichael 数**， 如 561，1105，1729，2465 等. Carmichael 数非常少，小于 10^8 的只有 255 个.
- 如果 n为合数，但不是Carmichael数，算法Ptest2 测试 n 为合数的概率至少为1/2.

素数的另一个必要条件

定理2 如果n为素数，则方程$x^2 \equiv 1(\mathrm{mod}\ n)$的根只有两个，即 $x = 1$，$x = -1$（或 $x = n-1$）.

证明 $x^2(\mathrm{mod}\ n) \equiv 1$

$$\Leftrightarrow x^2-1 \equiv 0(\mathrm{mod} n)$$

$$\Leftrightarrow (x+1)(x-1) \equiv 0(\mathrm{mod} n)$$

$$\Leftrightarrow x+1 \equiv 0 \text{ 或 } x-1 \equiv 0 \quad \text{（域中没有零因子）}$$

$$\Leftrightarrow x = n-1 \text{ 或 } x=1$$

称 $x \neq \pm 1$ 的根为非平凡的.

判别方法：如果方程有非平凡的根，则 n 为合数.

例如：$x^2(\mathrm{mod}\ 12) \equiv 1 \Leftrightarrow x = 1$ 或 $x = 11$ 或 $x = 5$ 或 $x=7$

结论：由于5 和 7 是非平凡的根，12是合数

北京大学

测试方法

设n为奇素数，存在q, m使得 $n-1=2^q m, (q\geq 1)$.

构造序列：

$$a^m \pmod n, a^{2m} \pmod n, a^{4m} \pmod n, \ldots, a^{2^q m} \pmod n$$

其最后一项为 $a^{n-1} \pmod n$, 而且每一项是前面一项的平方.

测试方法：

1. 对于任意 i（$i = 0,1,\ldots, q-1$）, 判断

$$a^{2^i m} \pmod n$$

   是否为 1 和 n-1, 且它的后一项是否为1.
2. 如果其后项为1，但本项不等于 1 和 $n-1$, 则它就是非平凡的根，从而知道n不是素数.
3. 随机选择 $a\in\{2,3,\ldots, n-1\}$, 进行上述测试.

实例

例如 n=561, n–1=560=$2^4 \cdot 35$，假设 a=7，构造的序列为

$$7^{35}(\bmod 561) = 241,$$

$$7^{2^1 35}(\bmod 561) = 7^{70}(\bmod 561) = 298,$$

$$7^{2^2 35}(\bmod 561) = 7^{140}(\bmod 561) = 166,$$

$$7^{2^3 35}(\bmod 561) = 7^{280}(\bmod 561) = 67,$$

$$7^{2^4 35}(\bmod 561) = 7^{560}(\bmod 561) = 1$$

第 5 项为 1，但是第 4 项等于 67，它既不等于 1 也不等于 560，是个非平凡的根，因此可以判定 n 为合数.

根据这个思想设计的计算机算法称为 **Miller-Rabin 算法**，它随机选择正整数 $a\in\{2,3,\ldots, n-1\}$，然后进行上述测试.

算法子过程(一)

算法 findq-m(n)　//找 q, m 使得 n-1=$2^q m$

1. $q \leftarrow 0$; $m \leftarrow n-1$
2. **repeat**
3. 　$m \leftarrow m/2$
4. 　$q \leftarrow q+1$
5. **until** m是奇数

运行时间：$O(\log n)$

算法子过程（二）

算法 test(n,q,m) //检测序列是否存在非平凡的根

1. $a \leftarrow$Random(2,n-1)
2. $x_0 \leftarrow$ExpMod(a,m,n) //$x_0=a^m$(mod$n$), $O(\log^3 n)$
3. for $i \leftarrow 1$ to q do //$q=O(\log n)$
4. $x_i \leftarrow x_{i-1}^2$ (mod n) //$O(\log^2 n)$
5. if $x_i=1$ and $x_{i-1}\neq 1$ and $x_{i-1}\neq n-1$
6. then return composite
7. if $x_q \neq 1$ then return composite 8. return prime

性能分析：

- 1 次测试运行时间 $O(\log^3 n)$
- 可证明1次测试出错的概率至多 1/2. 重复运行k次，可将出错概率降到至多2^{-k}.

北京大學

Miller-Rabin算法

令$k=\lceil \log n \rceil$, 出错的概率小于等于$2^{-k}\le 1/n$. 即算法给出正确答案的概率为$1-1/n$. 换句话说，如果n为素数，则算法输出素数. 如果n为合数，则算法以$1-1/n$的概率输出“合数”.

算法 PremalityTest(n) //$n\ge 5$，奇整数

1. findq-m(n)
2. $k \leftarrow \lceil \log n \rceil$
3. for $i \leftarrow 1$ to k //重复执行log n次
4. test(n, q, m)

时间：$T(n)=O(\log^4 n)$ //按位乘统计

北京大学

Las Vegas型与 Monte Carlo型随机算法

- **LasVegas型随机算法**
  - 如果得到解，总是给出正确的结果，区别只在于运行时间的长短.
  - 拉斯维加斯型随机算法的运行时间本身是一个随机变量
  - 期望运行时间是输入规模的多项式且总是给出正确答案的随机算法称为有效的拉斯维加斯型算法.
- **Monte Carlo型随机算法**
  - 这种算法有时会给出错误的答案.
  - 其运行时间和出错概率都是随机变量，通常需要分析算法的出错概率.
  - 多项式时间内运行且出错概率不超过1/3的随机算法称为有效的蒙特卡洛型算法

单侧错误和双侧错误

- 弃真型单侧错误
  - 当算法宣布接受时，结果一定是对的
  - 当算法宣布拒绝时，结果有可能是错的.
  - 例如主元素测试算法
- 取伪型单侧错误
  - 当算法宣布拒绝时，结果一定是对的
  - 而当算法宣布接受时，结果有可能是错的
  - 例如素数测试
- 双侧错误
  - 在所有的输入上同时出现上述两种不同的错误

随机算法的分类与局限性

- 拉斯维加斯型随机算法
  - 零错误概率多项式时间算法(有效的)，ZPP
- 蒙特卡洛型随机算法
  - 错误概率有界的有效算法(多项式时间)，BPP
  - 弃真型单侧错误概率有界的有效算法，RP
  - 取伪型单侧错误概率有界的有效算法，coRP

$$P \subseteq ZPP = RP \cap coRP \subseteq BPP$$

- 随机算法的局限性
  - 错误概率有界的多项式时间随机算法不太可能解决NP完全问题

北京大學

第10章 处理难解问题的策略

- 对问题施加限制
  固定参数算法
- 改进指数时间算法
  3SAT、指数时间假设
- 启发式方法
  启发式方法、随机化策略、重启策略、模拟退火
- 平均情形的复杂性
  ***G*(*n*,*p*)**、哈密顿回路、**DistNP**完全
- 难解算例的生成

北京大学

对问题施加限制

- **SAT问题**

  二元可满足性（**2SAT**）属于**P**

  HornSAT：输入限制为霍恩公式（析取式中正文字，即不带否定号的变量）至多出现一次，属于**P**

- 图的问题

问题	P	NPC
VC	$D \leq 2$	$D \geq 3$
HC	2	3
顶点三着色	3	4
反馈顶点集	2	3
团	给定D	任意

固定参数算法

- 通常把优化问题转化为判定问题时，都会在输入中引入一个参数，这是固定参数算法中参数的来源之一.
- 输入中带有一个参数 k，当输入规模为 n 时运行时间为 $O(f(k)n^c)$ 的算法，这里的 $f(k)$ 是与 n 无关的函数，c 是与 n 和 k 都无关的常数.
- 例

  VC：给定图 G, 正整数 K(不超过 G 的顶点数)，问是否存在不超过 K 的顶点覆盖？

  固定常数 k，输入为 (G, k). 穷举所有 k 元 顶点子集，看看是否存在顶点覆盖. 算法复杂度大约是

$$O(knC_n^k)=O(kn^{k+1})$$

  存在 $O(2^k kn)$ 的算法.

改进的指数时间算法

- O^*：表示忽略了多项式因子. 如$O^*(2^n)=O(n^{O(1)}2^n)$
- 当一个问题的蛮力算法为$O^*(2^n)$时间时，对任何满足$1 < c < 2$的常数c，时间复杂度为$O^*(c^n)$的指数时间算法称为非平凡的指数时间算法，或改进的指数时间算法.
- 可证明在$O^*(1.8393^n)$时间内正确求解3SAT，截止到2010年底 最好结果：$O^*(1.321^n)$时间的随机算法和$O^*(1.439^n)$时间的确定型算法.
- 任意色数的图的顶点着色问题都有$O^*(3^n)$的算法. 背包问题有比$O^*(2^{n/2})$更好的算法. 货郎问题也有比$O^*(2^n)$更好的算法.

其他处理难解问题的策略

- 启发式方法（**Heuristics**）：目前无法从理论上给出任何性能保证，但在实践中效果良好，就把这类方法统称为启发式方法（**Heuristics**）.

  常用的启发式方法主要包括：回溯与分支限界法、局部搜索法（随机化策略、重启策略、模拟退火）、遗传算法等.

- 平均情况下的复杂度

  有些**NP**完全问题在平均复杂性度量下是易解的，哈密顿回路问题的平均情况下对图$G(n,1/2)$有$O(n^3)$时间的算法.

- 难解算例生成: 确定紧的实例
- 基于统计物理的消息传递算法