

**《数学实验》报告**

**实验名称   素性测试**

**学院   计算机与通信工程**

**专业班级   信安1602**

**姓名   李宇桐**

**学号   41624545**

**2018年4 月**

素性测试算法

# 摘要

关于素数的一个基本问题是如何有效地确定一个数是否是一个素数，即素性测试问题。素性测试问题不仅在数学上是一个有挑战性的问题，而且在实际应用系统中也具有十分重要的地位。例如，很多现代密码学应用通常需要确定一个几百位的素数，如果不采用一些快速有效的素性测试方法，就算使用运算速度最快的计算机也要花费很多时间。

本文探究素性检测的算法，并用matlab实现代码部分和分析比较各类算法的有效性。

分工：刘海壮用matlab实现代码部分、孙文睿用matlab分析比较各类算法的有效性、李宇桐负责撰写论文。

关键词： 素性检测 确定性检测算法 概率性检测算法

# 一、问题分析

密码学是网络信息安全的基础，而公钥密码体制是密码学的重要组成部分，其中 RSA 算法作为公钥密码体制中较为完善的算法之一，具有较高的安全性，被广泛应用于数据加、解密和数字签名技术中。但RSA算法的公开密钥和私有密钥是一对大素数，因此研究素性测试对此具有十分重要的意义。

素性检测的方法分为两大类：确定性检测算法和概率性检测算法。确定性素数方法的优点在于产生的数一定是素数；缺点是生成的素数带有一定的限制。若算法设计不当，很容易构造出的有规律的素数，致使密码分析者很容易地分析到素数的变化，直接猜测到密码系统所使用的素数。概率性方法是目前检测素数的主要算法。该方的优点是：检测伪素数速度较快、原理简单、易于编程实现，构造的伪素数没有规律性，其缺点是其检测的数具有一定的误判，所以称为伪素数。

# 二、算法实现

## 2.1确定性检测算法

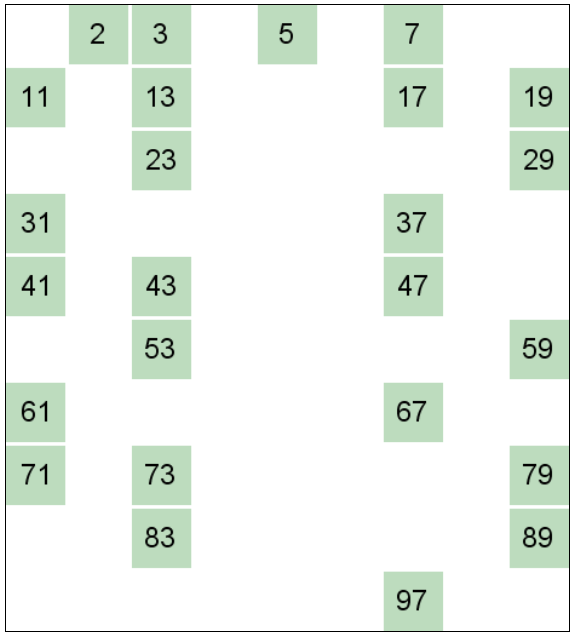
2.1.1 Eratosthenes筛法

Eratosthenes筛法是对于所有素数都有效的最古老的素数测试算法，然而它的复杂性是关于输入整数规模的幂指数关系，因此用它来测试大的素数是不合适的。

理论基础：

设正整数>1，如果对于所有的素数，均有，则为素数。

例如给定一个一百以内的数，给出要筛选数值的范围，找出以内的素数。先把2留下，2的倍数剔除；再把3留下，3的倍数剔除；接着5留下，5的倍数剔除，不断重复下去。



算法实现：

%Eratosthenes筛选法

%确定性素性检测算法

%函数返回值 对应位置为1代表为素数 0为和数

function ss=Eratosthenes(x)

size\_x=size(x);

ss=ones(size(x));

for i=1:size\_x(1)

for j=1:size\_x(2)

n=fix(sqrt(x(i,j)));%向下取整sqrt(x(i,j))

if(x(i,j)==1||x(i,j)==2)%如果是1 2直接跳过

continue;

end

for k=2:n

if(mod(x(i,j),k)==0)

ss(i,j)=0;

break;

end

end

end

end

2.1.2 AKS算法

2002年，印度理工学院的Agrawal，Kayal和Saxena等三人提出了一种多项式时间的确定性素性测试方法。

理论基础：

设为变量，，是两个正整数，，则为素数当且仅当：



算法实现：

1）主函数

function ss=AKS(n)

if(n==1)

ss=0;

elseif(n==2)

ss=1;

else

a=rprime(n);

%a=15;

x=-a;ss=1;

while(1)

q=mod\_long((x+a),n,n);

w=mod(a,n);

r=mod\_long(a,n,n);

t=mod\_long(x,n,n);

if(w~=r || q~=mod(t+w,n)||q~=mod(t+r,n))

ss=0;

break

end

if(x>n-1-a)

break

end

x=x+1;

end

end

2）rprime函数

%随机生成一个和x互质且不为1的数字

function ss=rprime(x)

while(1)

a=fix(x\*rand(1));

if(gcd(a,x)==1 && a~=1)

break

end

end

ss=a;

3）mod\_long函数

%函数功能计算x^n mod m 的值

%返回值在0-m-1之内

%输入值只能为1位（长度为1）

function ss=mod\_long(x,n,m)

if(n==1)%到底出结果

ss=mod(x,m);

elseif(n==Euller(m))%到达欧拉函数也出结果

ss=1;

else

if(mod(n,2)==0)

ss=mod\_long(mod(x^2,m),n/2,m);%函数递归n取一半 mod(x^2,n)

else

ss=mod(mod\_long(mod(x^2,m),(n-1)/2,m)\*x,m);%乘x前n-1个合并

end

end

4）Euller函数

%欧拉函数

function ss=Euller(x)

[w,p]=fprime(x);

size\_p=size(p);

ss=x;

for i=1:size\_p(2)

ss=ss\*(1-1/p(i));

end

5）fprime函数

%质因数分解

%输入一个x返回两个向量

%第一个向量中包括第二个向量中对应质因子的幂次方

%第二个向量中包含n个x的质因子(不包含1)

%例如 20=2^2\*5^1

%返回m=[2 1]ss=[2 5]

function [m,ss]=fprime(x)

k=1;

p=primes(x);

size\_p=size(p);

for i=1:size\_p(2) %遍历整个2- x-1

if(gcd(p(1,i),x)==p(1,i))

ss(1,k)=p(1,i);

j=1;

while(1)

flag\_n=gcd(p(1,i)^j,x)==p(1,i)^j;

if(flag\_n==1)

m(1,k)=j;

else

break

end

j=j+1;

end

k=k+1;

end

end

6）改进

a) mod\_long2

%和1不同此函数可以输入一个矩阵 同时返回一个矩阵对应位置为1则原矩阵中对应数

%的n次方mod m的得数最终返回值为一个0-m-1内数字

function ss=mod\_long2(x,n,m)

size\_x=size(x);

ss=zeros(size(x));

for i=1:size\_x(1)

for j=1:size\_x(2)

ss(i,j)=mod\_long(x(i,j),n,m);

end

end

b) Euller2

%将函数改造成矩阵形式

function ss=Euller2(x)

ss=zeros(size(x));

size\_x=size(x);

for i=1:size\_x(1)

for j=1:size\_x(2)

ss(i,j)=Euller(x(i,j));

end

end

c) AKS2

%将函数改造成矩阵形式

function ss=AKS2(x)

ss=zeros(size(x));

size\_x=size(x);

for i=1:size\_x(1)

for j=1:size\_x(2)

ss(i,j)=AKS(x(i,j));

end

end

## 2.2概率性检测算法

2.2.1 Fermat素性测试法

由于Eratosthenes筛法所需的时间复杂度为，因而效率极低，不能在实际中应用。17世纪，Fermat提出一个有效的素性测试方法，只花费的多项式时间。

理论基础：

1. Fermat小定理：

当为素数时，对每个，，均有：

1. Fermat素性测试
2. 随机选取整数,
3. 计算，如果，则为合数
4. 计算，如果，则为合数
5. 如果(b)、(c)均不发生，则可能为素数
6. 将(a)~(d)重复t次，如果每次得到可能为素数，则为素数的概率为

算法实现：

1. 主函数

%Fermat素性测试方法 第二个参数为精度要求

%当缺省时默认进行20次精度为1/2^20

function ss=Fermat(n,N)

if (nargin<2)

N=20;

end

if(n==2 || n==3)

ss=1;

else

if(mod(n,2)==0)

ss=0;

else

ss=1;

for t=1:N

while(1)

b=rprime(n);

if(b>=2 && b<=n-2)

break

end

end

g=gcd(b,n);

if(g~=1)

ss=0;

break

end

r=mod\_long(b,n-1,n);%要用大数mod\_long函数不然会出现溢出问题)

if(r~=1)

ss=0;

break

end

end

if(ss==1)

ss=1-0.5^t;

end

end

end

1. rprime函数

%随机生成一个和x互质且不为1的数字

function ss=rprime(x)

while(1)

a=fix(x\*rand(1));

if(gcd(a,x)==1 && a~=1)

break

end

end

ss=a;

1. mod\_long函数

%函数功能计算x^n mod m 的值

%返回值在0-m-1之内

%输入值只能为1位（长度为1）

function ss=mod\_long(x,n,m)

if(n==1)%到底出结果

ss=mod(x,m);

elseif(n==Euller(m))%到达欧拉函数也出结果

ss=1;

else

if(mod(n,2)==0)

ss=mod\_long(mod(x^2,m),n/2,m);%函数递归n取一半 mod(x^2,n)

else

ss=mod(mod\_long(mod(x^2,m),(n-1)/2,m)\*x,m);%乘x前n-1个合并

end

end

4）Euller函数

%欧拉函数

function ss=Euller(x)

[w,p]=fprime(x);

size\_p=size(p);

ss=x;

for i=1:size\_p(2)

ss=ss\*(1-1/p(i));

end

2.2.2 Lehmann素性测试法

理论基础：

1. 随机选取整数，
2. 计算，如果，则为合数
3. 计算，如果，则为合数
4. 如果(2)、(3)均不发生，则可能为素数

将(1)~(4)重复次，如果每次得到可能为素数，则为素数的概率为：

算法实现：

1. 主函数

%Lehmann素性检测

function ss=Lehmann(x,N)

if(nargin<2)

N=20;

end

if(x==2 || x==3)

ss=1;

else

if(mod(x,2)==0)

ss=0;

else

ss=1;

for i=1:N

while(1)

b=fix(x\*rand(1));

if(b>=2&&b<=x-2)

break

end

end

g=gcd(b,x);

if(g~=1)

ss=0;

break

end

r=mod\_long(b,(x-1)/2,x);

if(r==1||r==x-1)

continue

else

ss=0;

end

end

if(ss==1)

ss=1-0.5^i;

end

end

end

2） mod\_long函数

%函数功能计算x^n mod m 的值

%返回值在0-m-1之内

%输入值只能为1位（长度为1）

function ss=mod\_long(x,n,m)

if(n==1)%到底出结果

ss=mod(x,m);

elseif(n==Euller(m))%到达欧拉函数也出结果

ss=1;

else

if(mod(n,2)==0)

ss=mod\_long(mod(x^2,m),n/2,m);%函数递归n取一半 mod(x^2,n)

else

ss=mod(mod\_long(mod(x^2,m),(n-1)/2,m)\*x,m);%乘x前n-1个合并

end

end

3）Euller函数

%欧拉函数

function ss=Euller(x)

[w,p]=fprime(x);

size\_p=size(p);

ss=x;

for i=1:size\_p(2)

ss=ss\*(1-1/p(i));

end

4） 改进

%函数矩阵化

function ss=Lehmann2(x,N)

if(nargin<2)

N=20;

end

ss=zeros(size(x));

size\_x=size(x);

for i=1:size\_x(1)

for j=1:size\_x(2)

ss(i,j)=Lehmann(x(i,j),N);

end

end

2.2.3 Miller-Rabin素性测试法

Miller-Rabin素性检测算法是Fermat算法的一个变形的改进，它的理论基础是Fermat定理引申出来的。

理论基础：

1）Miller-Rabin测试：设为奇正合数，，其中，为奇整数，至多有个，满足如下条件：

或者存在，使得



2）Miller-Rabin测试法：在0与之间随机选取，计算，如果其值为，则通过测试，否则依次计算，如果得到，则也通过测试；否则为复合数

如果随机选取个，均通过测试，则为素数的概率：

算法实现：

1） 主函数

%Miller-Rabin素性测试法

function ss=MR(x,N)

if(mod(x,2)==0 && x~=2)

ss=0;

elseif(x==2)

ss=1;

else

s=1;k=1;

while(1)

if(gcd(x-1,2^k)==2^k)

s=k;

k=k+1;

else

break

end

end

m=(x-1)/2^s;

if(nargin<2)

N=20;

end

ss=1;

for t=1:N

while(1)

b=fix(x\*rand(1));

if(b>0 && b<x)

break;

end

end

flag\_bm=mod\_long(b,m,x);

if(flag\_bm==1 || flag\_bm==x-1)

continue

else

flag\_2rm=0;

for i=1:s-1

mod2rm=mod\_long(b,(2^i)\*m,x);

if(mod2rm==x-1||mod2rm==1)

flag\_2rm=1;

break

end

end

if(flag\_2rm==0)

ss=0;

break

end

end

end

if(ss==1)

ss=1-0.25^t;

end

end

2） mod\_long函数

%函数功能计算x^n mod m 的值

%返回值在0-m-1之内

%输入值只能为1位（长度为1）

function ss=mod\_long(x,n,m)

if(n==1)%到底出结果

ss=mod(x,m);

elseif(n==Euller(m))%到达欧拉函数也出结果

ss=1;

else

if(mod(n,2)==0)

ss=mod\_long(mod(x^2,m),n/2,m);%函数递归n取一半 mod(x^2,n)

else

ss=mod(mod\_long(mod(x^2,m),(n-1)/2,m)\*x,m);%乘x前n-1个合并

end

end

3）Euller函数

%欧拉函数

function ss=Euller(x)

[w,p]=fprime(x);

size\_p=size(p);

ss=x;

for i=1:size\_p(2)

ss=ss\*(1-1/p(i));

end

4）改进：

%函数矩阵化

function ss=MR2(x,N)

if(nargin<2)

N=20;

end

ss=zeros(size(x));

size\_x=size(x);

for i=1:size\_x(1)

for j=1:size\_x(2)

ss(i,j)=MR(x(i,j),N);

end

end

# 三、算法比较

研究上述确定性检测算法和概率性检测算法的有效性。通过输入一个数，求出的素数个数，并输出运算时间。

算法实现：

function A=test(n)

aa=0;ee=0;ff=0;ll=0;mrr=0;

Atime=0;Etime=0;Ftime=0;Ltime=0;MRtime=0;

for x=2:n

tic; a=AKS(x); atime=toc;

Atime=Atime+atime;

tic; e=Eratosthenes(x); etime=toc;

Etime=Etime+etime;

tic; f=Fermat(x); ftime=toc;

Ftime=Ftime+ftime;

tic; l=Lehmann(x); ltime=toc;

Ltime=Ltime+ltime;

tic; mr=MR(x); mrtime=toc;

MRtime=MRtime+mrtime;

if(a==1)

aa=aa+1;

end

if(e==1)

ee=ee+1;

end

if(f~=0)

ff=ff+1;

end

if(l~=0)

ll=ll+1;

end

if(mr~=0)

mrr=mrr+1;

end

end

A=[aa,ee,ff,ll,mrr;Atime,Etime,Ftime,Ltime,MRtime];

结果比较：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| test(5) | AKS | Eratosthenes | Fermat | Lehmann | MR |
| 素数个数 | 3.0000 | 3.0000 | 3.0000 | 3.0000 | 3.0000 |
| 运算时间 | 0.0586 | 0.0044 | 0.0074 | 0.0052 | 0.0072 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| test(26) | AKS | Eratosthenes | Fermat | Lehmann | MR |
| 素数个数 | 9.0000 | 9.0000 | 9.0000 | 9.0000 | 9.0000 |
| 运算时间 | 0.2179 | 0.0042 | 0.0287 | 0.0462 | 0.0510 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| test(74) | AKS | Eratosthenes | Fermat | Lehmann | MR |
| 素数个数 | 21.0000 | 21.0000 | 21.0000 | 21.0000 | 21.0000 |
| 运算时间 | 3.5485 | 0.0006 | 0.1293 | 0.3930 | 0.4300 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| test(99) | AKS | Eratosthenes | Fermat | Lehmann | MR |
| 素数个数 | 25.0000 | 25.0000 | 25.0000 | 25.0000 | 25.0000 |
| 运算时间 | 6.3170 | 0.0010 | 0.1943 | 0.6222 | 0.6983 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| test(150) | AKS | Eratosthenes | Fermat | Lehmann | MR |
| 素数个数 | 35.0000 | 35.0000 | 35.0000 | 35.0000 | 35.0000 |
| 运算时间 | 20.9991 | 0.0011 | 0.3942 | 1.2999 | 1.5900 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| test(200) | AKS | Eratosthenes | Fermat | Lehmann | MR |
| 素数个数 | 46.0000 | 46.0000 | 46.0000 | 46.0000 | 46.0000 |
| 运算时间 | 55.3161 | 0.0016 | 0.6436 | 2.8552 | 3.1502 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| test(500) | AKS | Eratosthenes | Fermat | Lehmann | MR |
| 素数个数 | 95.0000 | 95.0000 | 95.0000 | 95.0000 | 95.0000 |
| 运算时间 | 758.5255 | 0.0119 | 3.6584 | 17.8102 | 19.9859 |

# 四、参考文献

[1]李创成，陈文庆.基于Delphi的大素数Miller-Rabin检测方法的实现[J].湖南科技学院学报，2011,32(12):67-69

[2]程胜利，韩智强.素性检测及其实现[J].计算机与数字工程，1995,23(3-4):41-48

[3]赵勇，张益新，杨文伟. AKS算法及其在公钥加密术中的意义[J].广东工业大学学报，2004,21(3):79-82