Miller-Rabin素性测试算法
Pollard ρ整数分解算法

潘宇冲

绍兴市第一中学

主要内容

Outline

内容

内容

- 若$a \in \mathbb{Z}$，p是素数，那么

内容

- 若$a \in \mathbb{Z}$，p是素数，那么

$$a^{p-1} \equiv 1 \ (mod \ p)$$

证明

证明

- 首先我们证明这样一个结论:

证明

- 首先我们证明这样一个结论:
- 若p是素数，则对于任意一个小于p的正整数a，$\{a, 2a, 3a, ..., (p-1)a\}$除以$p$的余数恰好是1到$p$-1的一个排列。

证明

证明

- 使用反证法，假设结论不成立，那么就是说有2个小于p的正整数m和n使得ma和na除以p的余数相同。

证明

- 使用反证法，假设结论不成立，那么就是说有2个小于p的正整数m和n使得ma和na除以p的余数相同。
- 不妨设$n > m$，则p可以整除$a(n-m)$。

证明

- 使用反证法，假设结论不成立，那么就是说有2个小于p的正整数m和n使得ma和na除以p的余数相同。
- 不妨设$n > m$，则p可以整除$a(n-m)$。
- 因为p是素数，那么a和$n-m$中至少有一个含有因子p。

证明

- 使用反证法，假设结论不成立，那么就是说有2个小于p的正整数m和n使得ma和na除以p的余数相同。
- 不妨设$n > m$，则p可以整除$a(n-m)$。
- 因为p是素数，那么a和$n-m$中至少有一个含有因子p。
- 这显然是不可能的，因为a和$n-m$都比p小。

证明

- 使用反证法，假设结论不成立，那么就是说有2个小于p的正整数m和n使得ma和na除以p的余数相同。
- 不妨设$n > m$，则p可以整除$a(n-m)$。
- 因为p是素数，那么a和$n-m$中至少有一个含有因子p。
- 这显然是不可能的，因为a和$n-m$都比p小。
- 假设错误，原结论正确。

证明

证明

- 用同余式表述，我们证明了：

证明

- 用同余式表述，我们证明了：

$$(p-1)! \equiv \prod_{i=1}^{p-1} i * a \ (mod\ p)$$

证明

- 用同余式表述，我们证明了：

$$(p-1)! \equiv \prod_{i=1}^{p-1} i * a \ (mod\ p)$$

- 即

证明

- 用同余式表述，我们证明了：

$$(p-1)! \equiv \prod_{i=1}^{p-1} i * a \ (mod\ p)$$

- 即

$$(p-1)! \equiv (p-1)! * a^{p-1} \ (mod\ p)$$

证明

- 用同余式表述，我们证明了：

$$(p-1)! \equiv \prod_{i=1}^{p-1} i * a \ (mod\ p)$$

- 即

$$(p-1)! \equiv (p-1)! * a^{p-1} \ (mod\ p)$$

- 两边同除以$(p-1)!$，即

证明

- 用同余式表述，我们证明了：

$$(p-1)! \equiv \prod_{i=1}^{p-1} i * a \ (mod\ p)$$

- 即

$$(p-1)! \equiv (p-1)! * a^{p-1} \ (mod\ p)$$

- 两边同除以$(p-1)!$，即

$$1 \equiv a^{p-1} \ (mod\ p)$$

逆否命题

逆否命题

- 根据费马小定理的逆否命题，若$a^{p-1} \neq 1 \pmod p$，则p一定是合数。

逆命题

逆命题

- 费马小定理的逆命题的这样的:

逆命题

- 费马小定理的逆命题的这样的:
- 若p是一个正整数，$\exists a$与p互质，使得$a^{p-1} \equiv 1\ (mod\ p)$，则$p$为素数。

逆命题

- 费马小定理的逆命题的这样的:
- 若p是一个正整数，$\exists a$与p互质，使得$a^{p-1} \equiv 1 \ (mod \ p)$，则$p$为素数。
- 在大部分情况下逆命题是成立的。

伪素数

伪素数

- 人们把能整除$2^{n-1}-1$的合数n叫做伪素数。

伪素数

- 人们把能整除$2^{n-1}-1$的合数n叫做伪素数。
- 不满足$2^{n-1}\equiv 1 \pmod n$的n一定不是素数，如果满足的话则多半是素数。

伪素数

- 人们把能整除$2^{n-1}-1$的合数n叫做伪素数。
- 不满足$2^{n-1} \equiv 1 \ (mod \ n)$的$n$一定不是素数，如果满足的话则多半是素数。
- 341是第一个伪素数$2^{340} \equiv 1 \ (mod \ 341)$，但$341 = 11 * 31$。

一个基于伪素数的素性判断方法

一个基于伪素数的素性判断方法

- 制作一张伪素数表，记录某个范围内的所有伪素数，那么所有满足$2^{n-1} \equiv 1 \pmod n$且不在伪素数表中的n就是素数。

一个基于伪素数的素性判断方法

- 制作一张伪素数表，记录某个范围内的所有伪素数，那么所有满足$2^{n-1} \equiv 1 \ (mod\ n)$且不在伪素数表中的$n$就是素数。
- 我们可以用快速幂快速计算$2^{n-1}\ mod\ n$的值，用二分查找、$Hash$表、$Trie$树等方法查找伪素数表。

出错概率

出错概率

- 如果只计算2^{n-1} $mod\ n$的值，而不准备伪素数表，那么素性判断出错的概率有多少？

出错概率

- 如果只计算2^{n-1} mod n的值，而不准备伪素数表，那么素性判断出错的概率有多少？
- 统计表明，前10亿个自然数中共有50847534个素数，而伪素数有5597个。

出错概率

- 如果只计算2^{n-1} mod n的值，而不准备伪素数表，那么素性判断出错的概率有多少？
- 统计表明，前10亿个自然数中共有50847534个素数，而伪素数有5597个。
- 那么算法出错的可能性约为0.00011。

出错概率

- 如果只计算$2^{n-1}\ mod\ n$的值，而不准备伪素数表，那么素性判断出错的概率有多少？
- 统计表明，前10亿个自然数中共有50847534个素数，而伪素数有5597个。
- 那么算法出错的可能性约为0.00011。
- 这个概率太高了，如果想免去建立伪素数表的工作，我们需要改进素性判断的算法。

伪素数的扩展

伪素数的扩展

- 最简单的想法是，我们刚才只考虑了$a = 2$的情况。对于$a^{n-1}\ mod\ n$，取不同的a可能导致不同的结果。

伪素数的扩展

- 最简单的想法是，我们刚才只考虑了$a = 2$的情况。对于$a^{n-1} \ mod \ n$，取不同的a可能导致不同的结果。
- 一个合数可能在$a = 2$时通过了测试，但$a = 3$时的计算结果却排除了素数的可能。

伪素数的扩展

- 最简单的想法是，我们刚才只考虑了$a = 2$的情况。对于$a^{n-1}\ mod\ n$，取不同的a可能导致不同的结果。
- 一个合数可能在$a = 2$时通过了测试，但$a = 3$时的计算结果却排除了素数的可能。
- 于是，人们扩展了伪素数的定义：称满足$a^{n-1} \equiv 1\ (mod\ n)$的合数$n$叫做以$a$为底的伪素数。

伪素数的扩展

- 最简单的想法是，我们刚才只考虑了$a = 2$的情况。对于$a^{n-1} \bmod n$，取不同的a可能导致不同的结果。
- 一个合数可能在$a = 2$时通过了测试，但$a = 3$时的计算结果却排除了素数的可能。
- 于是，人们扩展了伪素数的定义：称满足$a^{n-1} \equiv 1 \pmod n$的合数n叫做以a为底的伪素数。
- 前10亿个自然数中同时以2和3为底的伪素数只有1272个，不到刚才的$\frac{1}{4}$。

伪素数的扩展

- 最简单的想法是，我们刚才只考虑了$a=2$的情况。对于$a^{n-1} \mod n$，取不同的a可能导致不同的结果。
- 一个合数可能在$a=2$时通过了测试，但$a=3$时的计算结果却排除了素数的可能。
- 于是，人们扩展了伪素数的定义：称满足$a^{n-1} \equiv 1 \pmod n$的合数n叫做以a为底的伪素数。
- 前10亿个自然数中同时以2和3为底的伪素数只有1272个，不到刚才的$\frac{1}{4}$。
- 即如果我们同时验证$a=2$和$a=3$两种情况，算法出错的概率降到了0.000025。

Fermat素性测试

Fermat素性测试

- 容易想到，用来测试的a越多，算法越准确。

$Fermat$素性测试

- 容易想到，用来测试的a越多，算法越准确。
- $Fermat$素性测试即为:

Fermat素性测试

- 容易想到，用来测试的a越多，算法越准确。
- Fermat素性测试即为：
- 随机选择若干个小于待测数的正整数作为底数a进行若干次测试，只要有一次没有通过测试就判定为合数。

Carmichael数

Carmichael数

- 如果考虑了所有小于n的底数a，出错概率能否降到0呢？

Carmichael数

- 如果考虑了所有小于n的底数a，出错概率能否降到0呢?
- 对于合数n，若$\forall b \in \mathbb{Z}^+$，且$b$和$n$互质，都有$b^{n-1} \equiv 1 \pmod{n}$成立，则$n$称为Carmichael数。

Carmichael数

- 如果考虑了所有小于n的底数a，出错概率能否降到0呢？
- 对于合数n，若$\forall b \in \mathbb{Z}^+$，且$b$和$n$互质，都有$b^{n-1} \equiv 1 \pmod n$成立，则$n$称为Carmichael数。
- 前3个Carmichael数是561、1105、1729。

Carmichael数

- 如果考虑了所有小于n的底数a，出错概率能否降到0呢?
- 对于合数n，若$\forall b \in \mathbb{Z}^+$，且$b$和$n$互质，都有$b^{n-1} \equiv 1 \pmod n$成立，则$n$称为Carmichael数。
- 前3个Carmichael数是561、1105、1729。
- 前10亿个自然数中，Carmichael数有600多个。

Carmichael数

- 如果考虑了所有小于n的底数a，出错概率能否降到0呢？
- 对于合数n，若$\forall b \in \mathbb{Z}^+$，且$b$和$n$互质，都有$b^{n-1} \equiv 1 \pmod n$成立，则$n$称为Carmichael数。
- 前3个Carmichael数是561、1105、1729。
- 前10亿个自然数中，Carmichael数有600多个。
- Carmichael数的存在说明，我们需要加强素性测试的算法。

Outline

内容

内容

- 若p是素数，x是小于p的正整数，且$x^2 \equiv 1 \ (mod \ p)$，那么

内容

- 若p是素数，x是小于p的正整数，且$x^2 \equiv 1 \ (mod\ p)$，那么

$$x = 1 \ or \ x = p - 1$$

证明

证明

- $x^2 \equiv 1\ (mod\ p)$相当于p能整除x^2-1。

证明

- $x^2 \equiv 1 \ (mod\ p)$相当于p能整除$x^2 - 1$。
- 即p能整除$(x+1)(x-1)$。

证明

- $x^2 \equiv 1 \ (mod\ p)$相当于p能整除$x^2 - 1$。
- 即p能整除$(x+1)(x-1)$。
- 由于p是素数，那么只可能是$x-1$能被p整除或$x+1$能被p整除。

证明

- $x^2 \equiv 1\ (mod\ p)$相当于p能整除x^2-1。
- 即p能整除$(x+1)(x-1)$。
- 由于p是素数，那么只可能是$x-1$能被p整除或$x+1$能被p整除。
- 因此$x=1$或$p-1$。

一个例子

一个例子

- 下面我们来演示一下二次探测定理如何应用到$Fermat$素性测试上。

一个例子

- 下面我们来演示一下二次探测定理如何应用到$Fermat$素性测试上。
- 前面说过341可以通过以2为底的$Fermat$测试，因为$2^{340} \equiv 1 \ (mod\ 341)$。

一个例子

- 下面我们来演示一下二次探测定理如何应用到$Fermat$素性测试上。
- 前面说过341可以通过以2为底的$Fermat$测试，因为$2^{340} \equiv 1 \ (mod\ 341)$。
- 如果341是素数，那么$2^{170} \equiv 1 \ (mod\ 341)$只可能是1或340。

一个例子

- 下面我们来演示一下二次探测定理如何应用到$Fermat$素性测试上。
- 前面说过341可以通过以2为底的$Fermat$测试，因为$2^{340} \equiv 1 \ (mod\ 341)$。
- 如果341是素数，那么$2^{170} \equiv 1 \ (mod\ 341)$只可能是1或340。
- 计算得到$2^{170} \equiv 1 \ (mod\ 341)$成立，我们继续计算$2^{85}$除以341的余数。

一个例子

- 下面我们来演示一下二次探测定理如何应用到$Fermat$素性测试上。
- 前面说过341可以通过以2为底的$Fermat$测试，因为$2^{340} \equiv 1 \ (mod\ 341)$。
- 如果341是素数，那么$2^{170} \equiv 1 \ (mod\ 341)$只可能是1或340。
- 计算得到$2^{170} \equiv 1 \ (mod\ 341)$成立，我们继续计算$2^{85}$除以341的余数。
- 我们发现$2^{85} \equiv 32 \ (mod\ 341)$。

一个例子

- 下面我们来演示一下二次探测定理如何应用到$Fermat$素性测试上。
- 前面说过341可以通过以2为底的$Fermat$测试，因为$2^{340} \equiv 1\ (mod\ 341)$。
- 如果341是素数，那么$2^{170} \equiv 1\ (mod\ 341)$只可能是1或340。
- 计算得到$2^{170} \equiv 1\ (mod\ 341)$成立，我们继续计算$2^{85}$除以341的余数。
- 我们发现$2^{85} \equiv 32\ (mod\ 341)$。
- 这一结果说明了341不是素数。

Outline

算法描述

算法描述

- 不断提取$n-1$中的因子2，把$n-1$表示成$n-1=d*2^r$，其中d是一个奇数。

算法描述

- 不断提取$n-1$中的因子2，把$n-1$表示成$n-1=d*2^r$，其中d是一个奇数。
- 我们需要计算的东西变成了a^{d*2^r}除以n的余数。

算法描述

- 不断提取$n-1$中的因子2，把$n-1$表示成$n-1=d*2^r$，其中d是一个奇数。
- 我们需要计算的东西变成了a^{d*2^r}除以n的余数。
- 若n是素数，那么$a^{d*2^r} \equiv 1 \ (mod\ n)$。

算法描述

- 不断提取$n-1$中的因子2，把$n-1$表示成$n-1=d*2^r$，其中d是一个奇数。
- 我们需要计算的东西变成了a^{d*2^r}除以n的余数。
- 若n是素数，那么$a^{d*2^r}\equiv 1\ (mod\ n)$。
- 由二次探测定理，$a^{d*2^{r-1}}\equiv 1\ (mod\ n)$或$a^{d*2^{r-1}}\equiv n-1\ (mod\ n)$。

算法描述

- 不断提取$n-1$中的因子2，把$n-1$表示成$n-1=d*2^r$，其中d是一个奇数。
- 我们需要计算的东西变成了a^{d*2^r}除以n的余数。
- 若n是素数，那么$a^{d*2^r} \equiv 1 \ (mod \ n)$。
- 由二次探测定理，$a^{d*2^{r-1}} \equiv 1 \ (mod \ n)$或$a^{d*2^{r-1}} \equiv n-1 \ (mod \ n)$。
- 若$a^{d*2^{r-1}} \equiv 1 \ (mod \ n)$，定理继续适用于$a^{d*a^{r-2}}$。

算法描述

- 不断提取$n-1$中的因子2，把$n-1$表示成$n-1=d*2^r$，其中d是一个奇数。
- 我们需要计算的东西变成了a^{d*2^r}除以n的余数。
- 若n是素数，那么$a^{d*2^r} \equiv 1 \pmod n$。
- 由二次探测定理，$a^{d*2^{r-1}} \equiv 1 \pmod n$或$a^{d*2^{r-1}} \equiv n-1 \pmod n$。
- 若$a^{d*2^{r-1}} \equiv 1 \pmod n$，定理继续适用于$a^{d*a^{r-2}}$。
- 这样不断开方下去，直到对于某个i满足$a^{d*2^i} \equiv n-1 \pmod n$，或者$n-1$中的2用完了得到$a^d \equiv 1 \pmod n$或$a^d \equiv n-1 \pmod n$。

具体实现

具体实现

- 具体实现时，我们可以将上述过程倒着做。

具体实现

- 具体实现时，我们可以将上述过程倒着做。
- 即先计算出a^d除以n的余数，然后把它平方r次。

具体实现

- 具体实现时，我们可以将上述过程倒着做。
- 即先计算出a^d除以n的余数，然后把它平方r次。
- 注意一点，一个数平方后可能会超出64位整形的范围。

具体实现

- 具体实现时，我们可以将上述过程倒着做。
- 即先计算出a^d除以n的余数，然后把它平方r次。
- 注意一点，一个数平方后可能会超出64位整形的范围。
- 这种情况使用快速乘算法计算即可。

伪代码

Algorithm 1: *Miller-Rabin Primality Test*

```
Function Witness(a, n)
    temporary ← n − 1
    while temporary is even do
        temporary ← temporary/2
    end while
    x1 ← a^temporary mod n
    while temporary ≠ n − 1 do
        x0 ← x1
        x1 ← x1 * x1 mod n
        if x1 = 1 and x0 ≠ 1 and x0 ≠ n − 1 then
            return true
        end if
    end while
    if x1 ≠ 1 then
        return true
    else
        return false
    end if
end
```

强伪素数

强伪素数

- 我们把通过以a为底的*Miller-Rabin*测试的合数称作以a为底的强伪素数。

强伪素数

- 我们把通过以a为底的*Miller-Rabin*测试的合数称作以a为底的强伪素数。
- 第一个以2为底的强伪素数为2047。

强伪素数

- 我们把通过以a为底的$Miller$-$Rabin$测试的合数称作以a为底的强伪素数。
- 第一个以2为底的强伪素数为2047。
- 第一个以2和3为底的强伪素数则大到1373653。

误判概率

误判概率

- 随机选取k个底数进行测试的$Miller$-$Rabin$算法的误判概率为4^{-k}。

误判概率

- 随机选取k个底数进行测试的$Miller$-$Rabin$算法的误判概率为4^{-k}。
- 《算法导论》中提供了误判概率的证明。

误判概率

- 随机选取k个底数进行测试的$Miller$-$Rabin$算法的误判概率为4^{-k}。
- 《算法导论》中提供了误判概率的证明。
- 通常认为，这个误判概率是可以令人接受的。

底数选取

底数选取

- 对于大数的素性判断，底数一般为随机选取。

底数选取

- 对于大数的素性判断，底数一般为随机选取。
- 当待测数不太大时，选择测试的底数有一些技巧。

底数选取

- 对于大数的素性判断，底数一般为随机选取。
- 当待测数不太大时，选择测试的底数有一些技巧。
- 下面是*Wikipedia*提供的一些底数选取方法。

底数选取

- 对于大数的素性判断，底数一般为随机选取。
- 当待测数不太大时，选择测试的底数有一些技巧。
- 下面是*Wikipedia*提供的一些底数选取方法。

- if n < 2,047, it is enough to test a = 2;
- if n < 1,373,653, it is enough to test a = 2 and 3;
- if n < 9,080,191, it is enough to test a = 31 and 73;
- if n < 25,326,001, it is enough to test a = 2, 3, and 5;
- if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;
- if n < 1,122,004,669,633, it is enough to test a = 2, 13, 23, and 1662803;
- if n < 2,152,302,898,747, it is enough to test a = 2, 3, 5, 7, and 11;
- if n < 3,474,749,660,383, it is enough to test a = 2, 3, 5, 7, 11, and 13;
- if n < 341,550,071,728,321, it is enough to test a = 2, 3, 5, 7, 11, 13, and 17;
- if n < 3,825,123,056,546,413,051, it is enough to test a = 2, 3, 5, 7, 11, 13, 17, 19, and 23.

Outline

原理

原理

- 用某种方法生成2个整数a和b。

原理

- 用某种方法生成2个整数a和b。
- 不断计算$(a-b,n)$，直到$(a-b,n)\neq 1$，则$(a-b,n)$是n的一个约数。

算法描述

算法描述

- $Pollard\ \rho$算法的主要实现方法是从某个初值x_1开始，通过一个适当的多项式进行迭代$x_i = f\left(x_{i-1}\right)\ mod\ n$，直至找到$n$ 的一个非平凡因子。

算法描述

- $Pollard\ \rho$算法的主要实现方法是从某个初值x_1开始，通过一个适当的多项式进行迭代$x_i = f\left(x_{i-1}\right)\ mod\ n$，直至找到$n$ 的一个非平凡因子。
- 经典的选择是$f(x) = x^2 + a$，其中$a \neq 0, -2\ (mod\ n)$。

算法描述

- $Pollard\ \rho$算法的主要实现方法是从某个初值x_1开始，通过一个适当的多项式进行迭代$x_i = f(x_{i-1}) \ mod\ n$，直至找到$n$ 的一个非平凡因子。
- 经典的选择是$f(x) = x^2 + a$，其中$a \neq 0, -2 \ (mod\ n)$。
- 不选择$a = 0$或$a = -2$的原因是避免当序列中某一项$x_i \equiv \pm 1 \ (mod\ n)$时，后续各项全部为1的情况。

算法描述

算法描述

- 由于$\mathbf{Z_n}$是有限的，并且序列中的每一个值仅仅取决于前一个值，所有序列最终将产生循环。

算法描述

- 由于$\mathbf{Z_n}$是有限的，并且序列中的每一个值仅仅取决于前一个值，所有序列最终将产生循环。
- 一旦运算达到一个x_i，使得对某个$j < i$有$x_i = x_j$，则处在一个循环中，并有$x_{i+1} = x_{j+1}, x_{i+2} = x_{j+2}, \ldots$。

算法描述

- 由于$\mathbf{Z_n}$是有限的，并且序列中的每一个值仅仅取决于前一个值，所有序列最终将产生循环。
- 一旦运算达到一个x_i，使得对某个$j < i$有$x_i = x_j$，则处在一个循环中，并有$x_{i+1} = x_{j+1}, x_{i+2} = x_{j+2}, \dots$。
- 这个算法取名为ρ的原因就在于$x_1, x_2, \dots, x_{j-1}$可以化成ρ的“尾”，而循环$x_j, x_{j+1}, \dots, x_i$可以画成ρ的“体”。

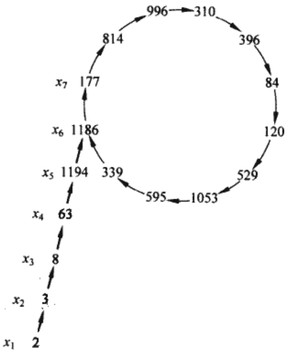
算法描述

算法描述

- 如图所示是$x_{i+1} = \left(x_i^2 - 1\right) \ mod \ 1387$所产生的值。

算法描述

- 如图所示是$x_{i+1}=\left(x_i^2-1\right)\ mod\ 1387$所产生的值。



时间复杂度

时间复杂度

- 根据生日悖论，序列出现循环的期望时间和循环的期望长度均为$\Theta(\sqrt{n})$。

时间复杂度

- 根据生日悖论，序列出现循环的期望时间和循环的期望长度均为$\Theta\left(\sqrt{n}\right)$。
- 令p是满足$\left(p, \frac{n}{p}\right) = 1$的$n$的一个非平凡因子。

时间复杂度

- 根据生日悖论，序列出现循环的期望时间和循环的期望长度均为$\Theta\left(\sqrt{n}\right)$。
- 令p是满足$\left(p, \frac{n}{p}\right) = 1$的$n$的一个非平凡因子。
- 令序列$\{x_i'\}$满足$x_i' = x_i \ mod\ p$。

时间复杂度

- 根据生日悖论，序列出现循环的期望时间和循环的期望长度均为$\Theta\left(\sqrt{n}\right)$。
- 令p是满足$\left(p, \frac{n}{p}\right) = 1$的$n$的一个非平凡因子。
- 令序列$\{x_i'\}$满足$x_i' = x_i \ mod \ p$。
- 定义$f_n(x) = f(x) \ mod \ n$。

时间复杂度

- 根据生日悖论，序列出现循环的期望时间和循环的期望长度均为$\Theta\left(\sqrt{n}\right)$。
- 令p是满足$\left(p, \frac{n}{p}\right) = 1$的$n$的一个非平凡因子。
- 令序列$\{x_i'\}$满足$x_i' = x_i \; mod \; p$。
- 定义$f_n(x) = f(x) \; mod \; n$。
- 序列从模p角度看是从模n角度看的一个较小版本。

时间复杂度

时间复杂度

- 推导数列$\{x_i'\}$的递推式。

时间复杂度

- 推导数列$\{x_i^{'}\}$的递推式。
- $x_{i+1}^{'}$

时间复杂度

- 推导数列$\{x_i^{'}\}$的递推式。
- $x_{i+1}^{'}$
- $= x_{i+1}\ mod\ p$

时间复杂度

- 推导数列$\{x_i^{'}\}$的递推式。
- $x_{i+1}^{'}$
- $= x_{i+1} \ mod \ p$
- $= f_n\left(x_i\right) \ mod \ p$

时间复杂度

- 推导数列$\{x_i^{'}\}$的递推式。
- $x_{i+1}^{'}$
- $= x_{i+1}\ mod\ p$
- $= f_n(x_i)\ \ mod\ p$
- $= \left((x_i^2 + a)\ \ mod\ n\right)\ \ mod\ p$

时间复杂度

- 推导数列$\{x_i^{'}\}$的递推式。
- $x_{i+1}^{'}$
- $= x_{i+1}\ mod\ p$
- $= f_n\left(x_i\right)\ mod\ p$
- $= \left(\left(x_i^2 + a\right)\ mod\ n\right)\ mod\ p$
- $= \left(x_i^2 + a\right)\ mod\ p$

时间复杂度

- 推导数列$\{x_i'\}$的递推式。
- x_{i+1}'
- $= x_{i+1}\ mod\ p$
- $= f_n(x_i)\ mod\ p$
- $= \left(\left(x_i^2 + a\right)\ mod\ n\right)\ mod\ p$
- $= \left(x_i^2 + a\right)\ mod\ p$
- $= \left((x_i\ mod\ p)^2 + a\right)\ mod\ p$

时间复杂度

- 推导数列$\{x_i'\}$的递推式。
- x_{i+1}'
- $= x_{i+1}\ mod\ p$
- $= f_n(x_i)\ \ mod\ p$
- $= \left(\left(x_i^2 + a\right)\ \ mod\ n\right)\ \ mod\ p$
- $= \left(x_i^2 + a\right)\ \ mod\ p$
- $= \left((x_i\ mod\ p)^2 + a\right)\ \ mod\ p$
- $= \left(\left(x_i'\right)^2 + a\right)\ \ mod\ p$

时间复杂度

- 推导数列$\{x_i^{'}\}$的递推式。
- $x_{i+1}^{'}$
- $= x_{i+1} \bmod p$
- $= f_n(x_i) \bmod p$
- $= ((x_i^2 + a) \bmod n) \bmod p$
- $= (x_i^2 + a) \bmod p$
- $= \left((x_i \bmod p)^2 + a\right) \bmod p$
- $= \left(\left(x_i^{'}\right)^2 + a\right) \bmod p$
- $= f_p\left(x_i^{'}\right)$

时间复杂度

时间复杂度

- 可以发现序列$\{x_i'\}$与序列$\{x_i\}$具有相同的递推式。

时间复杂度

- 可以发现序列$\{x_i'\}$与序列$\{x_i\}$具有相同的递推式。
- 因此根据前面的结论，序列$\{x_i'\}$在循环出现之前，预计执行的步数是$\Theta\left(\sqrt{p}\right)$。

时间复杂度

- 可以发现序列$\{x_i'\}$与序列$\{x_i\}$具有相同的递推式。
- 因此根据前面的结论，序列$\{x_i'\}$在循环出现之前，预计执行的步数是$\Theta\left(\sqrt{p}\right)$。
- 令t表示序列$\{x_i'\}$中第一个循环出现的值的下标，$u > 0$表示循环的长度。

时间复杂度

- 可以发现序列$\{x'_i\}$与序列$\{x_i\}$具有相同的递推式。
- 因此根据前面的结论，序列$\{x'_i\}$在循环出现之前，预计执行的步数是$\Theta\left(\sqrt{p}\right)$。
- 令t表示序列$\{x'_i\}$中第一个循环出现的值的下标，$u > 0$表示循环的长度。
- 也就是说，t和$u > 0$是对于所有$i \geq 0$，满足$x'_{t+i} = x'_{t+u+i}$的最小值。

时间复杂度

- 可以发现序列$\{x_i'\}$与序列$\{x_i\}$具有相同的递推式。
- 因此根据前面的结论，序列$\{x_i'\}$在循环出现之前，预计执行的步数是$\Theta\left(\sqrt{p}\right)$。
- 令t表示序列$\{x_i'\}$中第一个循环出现的值的下标，$u > 0$表示循环的长度。
- 也就是说，t和$u > 0$是对于所有$i \geq 0$，满足$x_{t+i}' = x_{t+u+i}'$的最小值。
- 由$x_{t+i}' = x_{t+u+i}'$，可知$p \mid (x_{t+u+i} - x_{t+i})$。

时间复杂度

- 可以发现序列$\{x_i'\}$与序列$\{x_i\}$具有相同的递推式。
- 因此根据前面的结论，序列$\{x_i'\}$在循环出现之前，预计执行的步数是$\Theta\left(\sqrt{p}\right)$。
- 令t表示序列$\{x_i'\}$中第一个循环出现的值的下标，$u>0$表示循环的长度。
- 也就是说，t和$u>0$是对于所有$i\geq 0$，满足$x_{t+i}'=x_{t+u+i}'$的最小值。
- 由$x_{t+i}'=x_{t+u+i}'$，可知$p\mid(x_{t+u+i}-x_{t+i})$。
- 因此，$(x_{t+u+i}-xt+i,n)>1$，即找到了n的一个非平凡因子。

时间复杂度

时间复杂度

- 由前面可知，t和u的期望值都是$\Theta\left(\sqrt{p}\right)$。

时间复杂度

- 由前面可知，t和u的期望值都是$\Theta\left(\sqrt{p}\right)$。
- 所以产生因子p所要求的期望执行步数为$\Theta\left(\sqrt{p}\right)$。

时间复杂度

- 由前面可知，t和u的期望值都是$\Theta\left(\sqrt{p}\right)$。
- 所以产生因子p所要求的期望执行步数为$\Theta\left(\sqrt{p}\right)$。
- 对于一个合数n完全分解因子，只要找出所有小于$\sqrt{n}$的素数因子就可以了。

时间复杂度

- 由前面可知，t和u的期望值都是$\Theta\left(\sqrt{p}\right)$。
- 所以产生因子p所要求的期望执行步数为$\Theta\left(\sqrt{p}\right)$。
- 对于一个合数n完全分解因子，只要找出所有小于$\sqrt{n}$的素数因子就可以了。
- 因此$Pollard\ \rho$算法的期望复杂度为$\Theta\left(\sqrt[4]{n}\right)$。

Brent判圈算法

Brent判圈算法

- 每次计算x_i，并记录x_{2^k}，使得$2^k < i$且k最大，设为y。

Brent判圈算法

- 每次计算x_i，并记录x_{2^k}，使得$2^k < i$且k最大，设为y。
- 当$x_i = y$时，说明存在一个循环且已经遍历了这个循环。

Brent判圈算法

- 每次计算x_i，并记录x_{2^k}，使得$2^k < i$且k最大，设为y。
- 当$x_i = y$时，说明存在一个循环且已经遍历了这个循环。
- 每次计算$(x_i - y, n)$，当$(x_i - y, n) > 1$时说明找到了n的一个非平凡因子。

伪代码

Algorithm 2: *Pollard's ρ Integer Factorization Algorithm*

```
Function Pollard_rho(a, n)
    i ← 1
    x ← Random(0, n − 1)
    y ← x
    k ← 2
    while true do
        i ← i + 1
        d ← GCD(y − x + n, n)
        if d > 1 and d < n then
            return d
        end if
        if i = k then
            y ← x
            k ← 2k
        end if
        x ← (x^2 + a) mod n
        if x = y then
            return false
        end if
    end while
end
```

Outline

题目大意

题目大意

- 给定两个数a和b的GCD和LCM，求a和b。

题目大意

- 给定两个数a和b的GCD和LCM，求a和b。
- 若有多组解，则输出$a+b$最小的一组。

题目大意

- 给定两个数a和b的GCD和LCM，求a和b。
- 若有多组解，则输出$a+b$最小的一组。
- GCD和LCM均在64位有符号整数范围内。

题目大意

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。
- 现在给剩余的格子填正整数。

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。
- 现在给剩余的格子填正整数。
- 对于每个格子，设这个格子里的数是X，满足:

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。
- 现在给剩余的格子填正整数。
- 对于每个格子，设这个格子里的数是X，满足:
- 如果它左边相邻有一个数Y，那么$Y \mid X$;

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。
- 现在给剩余的格子填正整数。
- 对于每个格子，设这个格子里的数是X，满足:
- 如果它左边相邻有一个数Y，那么$Y \mid X$;
- 如果它上面相邻有一个数Z，那么$Z \mid X$;

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。
- 现在给剩余的格子填正整数。
- 对于每个格子，设这个格子里的数是X，满足:
- 如果它左边相邻有一个数Y，那么$Y \mid X$;
- 如果它上面相邻有一个数Z，那么$Z \mid X$;
- 不存在和它填有相同数字的格子。

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。
- 现在给剩余的格子填正整数。
- 对于每个格子，设这个格子里的数是X，满足:
- 如果它左边相邻有一个数Y，那么$Y \mid X$;
- 如果它上面相邻有一个数Z，那么$Z \mid X$;
- 不存在和它填有相同数字的格子。
- 问是否存在一种方案可行。

题目大意

- 一个$3*3$的格子，左上角填M，右下角填N。
- 现在给剩余的格子填正整数。
- 对于每个格子，设这个格子里的数是X，满足:
-     如果它左边相邻有一个数Y，那么$Y \mid X$;
-     如果它上面相邻有一个数Z，那么$Z \mid X$;
-     不存在和它填有相同数字的格子。
- 问是否存在一种方案可行。
- $1 \leq M, N \leq 10^{17}$。

题目大意

题目大意

- 令$n = \sum_{i=1}^{k} p_i^{e_i}$，其中$p_i$是$n$的一个质因子，$e_i$是$p_i$在$n$中的最高次。

题目大意

- 令$n = \sum_{i=1}^{k} p_i^{e_i}$，其中$p_i$是$n$的一个质因子，$e_i$是$p_i$在$n$中的最高次。
- 设$f(n) = \prod_{i=1}^{k} \left(p_i^{2e_i+1} + 1\right)$。

题目大意

- 令$n=\sum_{i=1}^{k} p_i^{e_i}$，其中$p_i$是$n$的一个质因子，$e_i$是$p_i$在$n$中的最高次。
- 设$f(n)=\prod_{i=1}^{k}\left(p_i^{2e_i+1}+1\right)$。
- 设$g(n)=\sum_{i=1}^{n}\frac{n}{(n,i)}$。

题目大意

- 令$n=\sum_{i=1}^{k} p_i^{e_i}$，其中$p_i$是$n$的一个质因子，$e_i$是$p_i$在$n$中的最高次。
- 设$f(n)=\prod_{i=1}^{k}\left(p_i^{2e_i+1}+1\right)$。
- 设$g(n)=\sum_{i=1}^{n}\frac{n}{(n,i)}$。
- 求$\frac{f(n)}{g(n)}\ mod\ (10^9+7)$。

题目大意

- 令$n=\sum_{i=1}^{k} p_i^{e_i}$，其中$p_i$是$n$的一个质因子，$e_i$是$p_i$在$n$中的最高次。
- 设$f(n)=\prod_{i=1}^{k}\left(p_i^{2e_i+1}+1\right)$。
- 设$g(n)=\sum_{i=1}^{n}\frac{n}{(n,i)}$。
- 求$\frac{f(n)}{g(n)}\ mod\ (10^9+7)$。
- 测试组数$T\le 10^4$，$2\le n\le 10^{12}$。

Outline

参考文献

参考文献

1 *POJ*: http://poj.org/

参考文献

1 *POJ*: http://poj.org/

2 *SPOJ*: http://www.spoj.com/

参考文献

1 *POJ*: http://poj.org/

2 *SPOJ*: http://www.spoj.com/

3 *Wikipedia*: http://en.wikipedia.org/

参考文献

1 *POJ*: http://poj.org/

2 *SPOJ*: http://www.spoj.com/

3 *Wikipedia*: http://en.wikipedia.org/

4 *BZOJ*: http://www.lydsy.com/JudgeOnline/

参考文献

1 *POJ*: http://poj.org/

2 *SPOJ*: http://www.spoj.com/

3 *Wikipedia*: http://en.wikipedia.org/

4 *BZOJ*: http://www.lydsy.com/JudgeOnline/

5 *Matrix67's Blog*: http://www.matrix67.com/

参考文献

1 *POJ*: http://poj.org/

2 *SPOJ*: http://www.spoj.com/

3 *Wikipedia*: http://en.wikipedia.org/

4 *BZOJ*: http://www.lydsy.com/JudgeOnline/

5 *Matrix67's Blog*: http://www.matrix67.com/

6 *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to ALGORITHMS*