

程序设计竞赛中的数论

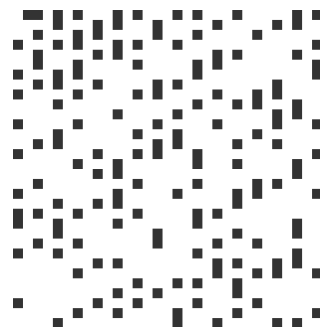
基础篇

作者：陈昱翔

组织：AHU ACM-ICPC Lab

时间：December 4, 2019

版本：1.0



Euclid marched into the wonderful world of Number Theory alone. — Millay, 1923

目 录

写在前面	1	第 4 章 习题	74
1 数的整除性	2	5 积性函数	75
1.1 最小公倍数	2	5.1 因子次幂和函数	75
1.2 线性方程定理	3	5.2 欧拉函数	76
第 1 章 习题	6	5.3 莫比乌斯函数	80
2 素数	7	5.4 莫比乌斯反演	87
2.1 质因数分解	7	5.5 积性函数与狄利克雷卷积	87
2.2 区间素数筛	9	5.6 积性函数前缀和	92
2.3 区间质因数分解	12	5.7 杜教筛	96
2.4 大区间素数筛与质因数分解	14	5.8 拓展埃氏筛	102
2.5 梅森素数与完全数	17	第 5 章 习题	129
第 2 章 习题	19	6 不定方程	130
3 同余	20	6.1 佩尔方程	130
3.1 同余式与同余方程	20	6.2 其它不定方程	130
3.2 快速乘与快速幂	23	第 6 章 习题	133
3.3 费马小定理与逆元	24	7 其他	134
3.4 中国剩余定理	27	7.1 勾股数组	134
3.5 欧拉公式与欧拉降幂	30	7.2 平方数之和、圆上整点	134
3.6 素性测试	35	7.3 循环节问题	141
3.7 Pollard_Rho 质因数分解	39	7.4 DFS Similar	146
3.8 离散对数	44	7.5 FFT 与 NTT	152
3.9 原根	47	第 7 章 习题	164
第 3 章 习题	51	A 常用的表	167
4 剩余	53	A.1 梅森素数表	167
4.1 二次剩余	53	A.2 卡米歇尔数	167
4.2 二次互反律	56	A.3 常见的素数及其原根	167
4.3 求解二次剩余	59	A.4 一些公式	170
4.4 求解 N 次剩余	64		

写在前面

参加 ACM-ICPC 已经快三年了，想着留下些什么，于是决定将数论方面的一些知识整理成册，供后人参考。

这本册子的目的是降低参赛同学们收集数论相关知识的时间成本，对知识点的介绍尽量做到详细且通俗易懂。书中会对所有出现的题目提供 solution，一些经典的定理、结论还会提供证明。

在册子完成过程中，AHU ICPC-Lab 的同学们提出了许多宝贵意见，在此向他们表示衷心的感谢。此外，本书使用了 ElegantBook 模板 (<https://github.com/ElegantLaTeX/>)，减少了许多排版方面的工作，在此也表示感谢！

由于本人水平所限，书中一定存在不少错误，欢迎同学们在 GitHub 上提出 issues 或者邮件联系我。

联系方式：

- GitHub 网址: <https://github.com/Feynman1999/number-theory-tutorial-in-Programming-Competition>
- 邮件: chenyx.cs@gmail.com

本书的 Tex 源码开源在 GitHub，未经本人许可不可用于商业用途。

陈昱翔
2019 年 12 月

第 1 章 数的整除性

本章内容提要

□ 欧几里得算法

□ 拓展欧几里得算法

1.1 最小公倍数

两个数 a 与 b (不全为零) 的最大公因数是整除它们两个的最大数, 记为 $\gcd(a, b)$ 。如果 $\gcd(a, b) = 1$, 我们称 a 与 b 互质。

求两个数最大公因数的最有效的方法是欧几里得算法, 其核心操作是辗转相除, 先看一个例子。

例 1.1 欧几里得算法求 $\gcd(28, 93)$ 的例子。

第一步用 93 除以 28 得商为 3, 余数为 7, 记作下面式子:

$$93 = 3 * 28 + 7$$

第二步用上一步的除数作为新的被除数, 上一步的余数作为新的除数, 即:

$$28 = 4 * 7 + 0$$

发现此时余数为 0, 算法不再继续。欧几里得算法指出当得到余数 0 时, 除数 (上一步的余数) 就是最初两个数的最大公因数。所以 $\gcd(28, 93) = 7$ 。

定理 1.1. 欧几里得算法

要计算两个整数 a 与 b 的 GCD, 先令 $r_{-1} = a$ 且 $r_0 = b$, 然后计算相继的商和余数:
 $r_{i-1} = q_{i+1} * r_i + r_{i+1}$ ($i = 0, 1, 2, \dots$), 直到某余数 $r_{n+1} = 0$, 最后的非零余数 r_n 就是 a 与 b 的最大公因数。

证明 考虑一般情形, 有

$$r_{-1} = q_1 * r_0 + r_1$$

$$r_0 = q_2 * r_1 + r_2$$

$$r_1 = q_3 * r_2 + r_3$$

$$r_2 = q_4 * r_3 + r_4$$

\dots

$$r_{n-3} = q_{n-1} * r_{n-2} + r_{n-1}$$

$$r_{n-2} = q_n * r_{n-1} + r_n$$

$$r_{n-1} = q_{n+1} * r_n + 0$$

欧几里得算法说，最后的 r_n 就是 gcd，那么我们先证明 r_n 一定是 $a(r_{-1})$ 和 $b(r_0)$ 的因子。

由最后一行知， r_n 整除 r_{n-1} ，于是由倒数第二行知 r_n 整除 r_{n-2} ，依次类推， r_n 整除 r_0 和 r_{-1} 。

下面再证明 r_n 是 a 与 b 的最大公因数。假设 d 是 a 与 b 的任意公因数，则由上面式子第一行知 d 整除 r_1 ，于是由第二行知 d 整除 r_2 ，依次类推， d 整除 r_n ，即 r_n 是最大的公因数。

最后，还要说明一定存在 r_{n+1} 为 0，易知，每一次余数都是严格递减的，所以余数最后总能到 0。那自然会问，要多少步呢？实际上，欧几里得算法的步数至多是 b 的位数的 7 倍。


```
1 //欧几里得算法求解gcd
long long gcd(long long a,long long b)
3 {
    if(b==0) return a;
5     return gcd(b,a%b);
}
```

code01/gcd.cpp

说到 GCD，往往还会提到最小公倍数 (LCM)，有如下式子

$$lcm(a,b) * gcd(a,b) = a * b$$

$$lcm(S/a,S/b) = S/gcd(a,b)$$

 **注意** 另外一种计算两个数最大公约数的算法，叫做 Stein 算法。这种算法是针对欧几里得算法在对大整数进行运算时，需要试商导致增加运算时间的缺陷而提出的改进算法。感兴趣的可以自行查阅。

1.2 线性方程定理

已知两个整数 a, b ，现在我们观察由 a 的倍数加上 b 的倍数得到的所有可能整数，也就是说考察 $ax + by$ 得到的所有整数，其中 x, y 可以是任何整数。

例如取 $a = 42$, $b = 30$ ，随意列出一些 x, y ，会发现得到的数都可以被 6 整除。这其实很显然，因为 42 和 30 都能被 6 整除。更一般的，显然形如 $ax + by$ 的每个数被 $gcd(a, b)$ 整除，因为 $gcd(a, b)$ 整除 a 与 b 。

接着问题又来了， $ax + by = gcd(a, b)$ 是否一定有整数解呢？答案是肯定的。证明是利用拓展欧几里得算法，不停的将余数用 a 和 b 表示，最后的 $gcd(a, b)$ 一定可以用 a, b 表示。也称欧几里得回代法。

例 1.2 欧几里得回代法解 $60x + 22y = gcd(60, 22)$

首先写出欧几里得算法计算 $gcd(60, 22)$ 的过程：

$$60 = 2 * 22 + 16$$

$$22 = 1 * 16 + 6$$

$$16 = 2 * 6 + 4$$

$$6 = 1 * 4 + 2$$

$$4 = 2 * 2 + 0$$


这表明 $\gcd = 2$ 。下面关键来了，我们想用 a, b 表示倒数第二行 $6 = 1 * 4 + 2$ 的那个 2，就从第一行表示 a, b 回代，如表 1.1：

表 1.1: 欧几里得回代法示例

原式	带入过程
$a = 2 * b + 16$	$16 = a - 2 * b$
$b = 1 * 16 + 6$	$6 = b - 1 * 16 = -a + 3b$
$16 = 2 * 6 + 4$	$4 = 16 - 2 * 6 = 3a - 8b$
$6 = 1 * 4 + 2$	$2 = 6 - 1 * 4 = -4a + 11b$

于是得到 $x = -4, y = 11$ 。

这样一行行进行，将陆续得到形如最新余数 $=a$ 的倍数 $+b$ 的倍数的等式，最终得到非零余数 \gcd ，就给出了方程的解。

 **注意** $\gcd(a, b)$ 是否是形如 $ax + by$ 的最小正整数呢？是的。因为 $\gcd(a, b) \leq \min(a, b)$ 。

既然一个线性方程 $ax + by = \gcd(a, b)$ 总有整数解 x, y ，下面的问题是**如何表述线性方程的所有解**。

先考虑 $\gcd(a, b) = 1$ 的情况（即 a, b 互质，其他情况可以转换为互质），假设 x_1, y_1 是方程 $ax + by = 1$ 的一个解。通过 x_1 加上 b 的倍数和 y_1 减去 a 的相同倍数，可得到其他解。即对于任何整数 k ，我们得到新解 $(x_1 + kb, y_1 - ka)$ 。（带入即可验证）

证明 $(x_1 + kb, y_1 - ka)$ 可以给出方程的所有解。

假设 (x_1, y_1) 与 (x_2, y_2) 是方程 $ax + by = 1$ 的两个解，即 $ax_1 + by_1 = 1$ 且 $ax_2 + by_2 = 1$ ，我们用 y_2 乘以第一个方程，用 y_1 乘以第二个方程，再相减消去 b ，整理后得到 $ax_1y_2 - ax_2y_1 = y_2 - y_1$ 。类似的，用 x_2 乘以第一个方程，用 x_1 乘以第二个方程，再相减便得到 $bx_2y_1 - bx_1y_2 = x_2 - x_1$ 。令 $k = x_2y_1 - x_1y_2$ ，则得到 $x_2 = x_1 + kb, y_2 = y_1 - ka$ 。得证。

再考虑 $\gcd(a, b) > 1$ 的情况，为简便，令 $g = \gcd(a, b)$ ，由前面“欧几里得回代法”知方程 $ax + by = g$ 至少有一个解 (x_1, y_1) 。而 g 整除 a, b ，故 (x_1, y_1) 是简单方程 $\frac{a}{g}x + \frac{b}{g}y = 1$ 的解。于是由前面证明，通过式子 $x_1 + k * \frac{b}{g}, y_1 - k * \frac{a}{g}$ 改变 k 的值可得到其他解。这就完成了对方程 $ax + by = g$ 解的描述，我们把它概括为下面定理。

定理 1.2. 线性方程定理

设 a 与 b 是非零整数， $g = \gcd(a, b)$ 。方程 $ax + by = g$ 总是有一个整数解 (x_1, y_1) （也称贝祖定理），它可由前面叙述的欧几里得回带法得到。则方程的每一个解可由 $(x_1 + k * \frac{b}{g}, y_1 - k * \frac{a}{g})$ 得到，其中 k 可为任意整数。



现在我们想怎么用代码实现上面说到的回代法。

如果用上面的思路，我们似乎需要事先知道欧几里得的结果，然后根据结果去算系数。但是一般用计算机求解 \gcd ，是递归算法，就是说我们需要逆推（上面过程的逆过程），这样才能写出递归代码。

假设当前我们要处理的是求出 a 和 b 的最大公约数，并要求出 x 和 y 使得 $a * x + b * y = g$ 。而我们已经求出了下一个状态： b 和 $a \% b$ 的最大公约数 (欧几里得算法核心, $a = b, b = a \% b$)，并且求出了一组 x_1 和 y_1 使得： $b * x_1 + (a \% b) * y_1 = g$ 。那么这两个相邻的状态之间存在一种关系可以求解，我们知道 $a \% b = a - (a/b) * b$ ，那么较末状态 (先开始递归计算的) 有：

$$b * x_1 + (a \% b) * y_1 = g$$

$$b * x_1 + (a - (a/b) * b) * y_1 = g$$

$$b * x_1 + a * y_1 - (a/b) * b * y_1 = g$$

$$a * y_1 + b * (x_1 - a/b * y_1) = g$$

对比较先的状态 (后递归到的，要求的)，要求一组 x 和 y 使得： $a * x + b * y = g$ ，比较对应项系数得到

$$x = y_1$$

$$y = x_1 - a/b * y_1$$

这就是递归的关键，我们把它称为拓展欧几里得算法。之所以叫拓展，是因为它相比欧几里得算法多了一小点，但能解决模线性方程、逆元、同余方程等许多经典问题 (后面会介绍)。

递归的终止条件为 $b = 0$ ，这时候 $x = 1, y = 0, a = gcd$ 。

```

/*
2 拓展欧几里得算法 (a,b非零)
   求解ax+by=gcd(a,b)的一个解(x1,y1)
4 则通解为(x1+k*b/g,y1-k*a/g)

6 如果求解ax+by=c
   输入a,b后 下面模板会求出
8 ax+by=+-gcd(a,b)的一个解(x1,y1)
   正负号虽然不确定 (和扩欧写法有关) 但等式是成立的
10 ax+by=c 的一个解为(x2,y2)=c/g*(x1,y1)
   则通解为(x2+k*b/g,y2-k*a/g)
12 */
#include<bits/stdc++.h>
14 using namespace std;
   typedef long long ll;
16 ll e_gcd(ll a,ll b,ll &x,ll &y)
{
18     if(b==0){
           x=1;
20     y=0;

```

```
    return a;
22 }
    ll ans=e_gcd(b,a%b,x,y);
24 ll temp=x;
    x=y;
26 y=temp-a/b*y;
    return ans;
28 }
int main()
30 {
    ll a,b,x,y;
32 cin>>a>>b;
    ll gcd=e_gcd(a,b,x,y);
34 cout<<"gcd:"<<gcd<<' '<<"x1:"<<x<<' '<<"y1:"<<y<<endl;
    cout<<"x的最小正整数解"<<(x%(b/gcd)+b/gcd)%(b/gcd)<<endl;
36 return 0;
}
```

code01/exgcd.cpp

第 1 章 习题

1. 拓展欧几里得算法的时间复杂度是多少?
2. 青蛙的约会 POJ1061
3. Utawarerumono ac.nowcoder.com/acm/contest/201/C
4. SGU140 多元线性同余方程

第 2 章 素数

本章内容提要

- 质因数分解
- 筛选素数
- 大区间问题
- 梅森素数
- 完全数
- 因数之和函数

2.1 质因数分解

定义 2.1. 素数

素数 p 是大于 1 且它的因数只有 1 和 p 的整数。

性质 令 p 是素数，假设 p 整除乘积 ab ，则 p 整除 a 或 p 整除 b (或者 p 既整除 a 也整除 b)。

定理 2.1. 无穷多素数定理

存在无穷多个素数。

证明 证明的思路是，假设我们现在有一个素数表，只要说明如何利用这些素数找出新的不在表中的素数，这样迭代下去，就表明必有无穷多个素数。设素数表一开始为 p_1, p_2, \dots, p_n ，令 $A = p_1 p_2 \dots p_n + 1$ ，设 q 是某个整除 A 的素数，且设其是最小的那个，那么 q 一定不在最初的素数表中，因为如果 $q = p_i$ ，则 q 能整除 1，显然错误。于是我们可以无限扩充素数表。

定理 2.2. 素数定理

当 x 很大时，小于 x 的素数的个数近似等于 $x/\ln(x)$ ，即

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{x/\ln(x)} = 1$$

其中 $\pi(x)$ 为素数计数函数， $\pi(x) = \#\{\text{prime } p : p \leq x\}$ 。

定理 2.3. 素数整除性质

假设素数 p 整除乘积 $a_1 a_2 \dots a_r$ ，则 p 整除 a_1, a_2, \dots, a_r 中至少一个因数。

证明 使用上面的性质可以方便的证明该定理。

下面将用素数整除性质证明每个正整数可唯一分解成素数的乘积（算术基本定理）。

定理 2.4. 算术基本定理

每个整数 $n \geq 2$ 可唯一分解成素数乘积 $n = p_1 p_2 \dots p_r$ 。

证明 算术基本定理包含两个断言：

断言 1: 数 n 可以以某种方式分解成素数的乘积;

断言 2: 仅有一种这样的因数分解方式 (因数重排除外);

- 对于断言 1 的证明用到归纳证明法 (当 $N+1$ 是合数时, 其必然可以分解成 $N+1 = n_1 * n_2$, 而前面已经归纳完毕, 即 n_1, n_2 一定可以分解成素数, 因此得证)。

- 对于断言 2, 可以由定理 2.4 证得。

下面考虑如何进行质因数分解。

简单直接的方法是试除 2, 3, 4, 5, ... 分解 n , 但效率较低。考虑一个整数 n , 如果 n 本身不是素数, 则必有整除 n 的素数 $p \leq \sqrt{n}$ 。于是我们可以遍历 $2 \sim \sqrt{n}$ 进行试除, 时间复杂度为 $O(\sqrt{n})$ 。如果提前预处理出所有素数, 则可每次只试除素数, 时间复杂度为 $O(\frac{\sqrt{n}}{\ln \sqrt{n}})$ 。

```

1 //O(sqrt n)
  const int maxn=100;
3 int a[maxn]; //质因子
  int b[maxn]; //质因子的指数
5 int tot; //1~tot
  void factor(int n)
7 {
    int now=n;
9    tot=0;
    for(int i=2; i*i<=now; ++i) if(now%i==0){ //时间稍紧后使用素数ans[i]代替i
11        a[++tot]=i;
        b[tot]=0;
13        while(now%i==0){
            ++b[tot];
15            now/=i;
        }
17    }
    if(now>1){ //一个合数n必定有小于等于sqrt{n}的质因子 将其全部除尽后 若不为1
        //则为大于sqrt{n}的质因子 且仅有一个
19        a[++tot]=now;
        b[tot]=1;
21    }
  }
23 int main()
  {
25    int n;
    cin>>n;
27    factor(n); //n>1
    for(int i=1; i<=tot; ++i){
29        if(i==tot) cout<<a[i]<<'^'<<b[i]<<endl;
        else cout<<a[i]<<'^'<<b[i]<<" * ";
    }
  }

```

```

31     }
    }

```

code02/factor.cpp

2.2 区间素数筛

这里区间素数筛指的是给定一个上限 n ，要求出区间 $[1, n]$ 中所有的素数。如果采用直接遍历每个数并判断是不是素数的方法，时间复杂度为 $O(n^{\frac{3}{2}})$ ，若 $n > 10^6$ ，则不能在 $1s$ 的时间内求解。下面介绍的两种筛法，均使用“筛选”的思想，时间复杂度分别为 $O(n \log(\log n))$ 和 $O(n)$ ，在比赛中经常使用。

2.2.1 埃氏筛法

考虑如果一个数 n 如果是合数(非素数)，那么它一定有小于等于 \sqrt{n} 的质因子，也即 n 是这个质因子的倍数。那么我们可以从小到大 ($2 \sim n$) 依次遍历每个数，并存在一些标记记录每个数是否是合数，当遍历到某个数时，如果当前数没有被标记为合数(就一定是素数了)，就把当前数的所有倍数标记为合数。代码如下：

```

//埃氏筛法 O(nloglogn)
2 const int maxn=1e6+10;
  bool valid[maxn];
4 int ans[maxn]; //存素数
  void getprime(int n,int &tot,int ans[])
6 {
    tot=0;
8   for(int i=2;i<=n;++i) valid[i]=true;
    for(int i=2;i<=n;++i){
10      if(valid[i]){
          if(n/i<i) break; //防止下面爆int
12      for(int j=i*i;j<=n;j+=i) valid[j]=false;
      }
14  }
    for(int i=2;i<=n;++i) if(valid[i]) ans[tot++]=i;
16 }

18 int main()
  {
20   int Count; //素数个数
    getprime(1e6,Count,ans);
22   cout<<Count<<endl;
    return 0;
24 }

```

code02/aishi.cpp

注 代码中有两个关键的地方：

- 第 10 行，只有当是素数时，才进行筛选，因为使用合数筛是多余的（该合数的质因子已经筛过了）；
- 第 12 行，从 $i * i$ 开始筛，因为 $(i - 1) * i, (i - 2) * i, \dots, 2 * i$ 已经被筛过了。

考虑埃氏筛法的时间复杂度，由于每次只考虑素数去筛，即为：

$$n \sum_{p \leq n} \frac{1}{p}, \quad p \text{ is prime}$$

上式可近似认为是 $O(n \log \log n)$ ，感兴趣的可以阅读 [Sieve_of_Eratosthenes#/Algorithm_complexity](#)

观察埃氏筛法，是否有重复被筛去的合数呢？有的，就是一些质数的较大的公倍数，比如 12，当 $i=2$ 时（从 4 开始）会筛 12；当 $i=3$ 时（从 9 开始）还会筛 12。

2.2.2 欧拉线性筛法

欧拉筛的思想是保证每个合数都被其最小的质因子筛去，这样就不会像埃氏筛那样重复筛某个数。

首先其还是筛选法，考虑用什么筛的：对于每个 i ，乘以小于 i 的每个素数，直到能整除就停止，这样能保证每个数只被筛去一次。

这个整除停止很关键，假设没有这个 `break`，就会有很多重复。

比如 $i = 6$ ，取质数 3，筛去了 18； $i = 9$ ，取质数 2，也筛去了 18。

再比如 $i = 6$ ，取质数 5，筛去了 30； $i = 10$ ，取质数 3，筛去了 30； $i = 15$ ，取质数 2，筛去了 30。

如果加上 `break`，则筛去 18 的是 $9 * 2$ ，筛去 30 的是 $15 * 2$ 。为什么 18 不会被 $6 * 3$ 筛去？因为 $6 \% 2 = 0$ 提前 `break` 了。即若有一个素数 $p | i$ ，则我用 i 和后面更大的素数去筛选，不如用 p 和更大的 i 去筛选，这样保证了筛掉每个数的是其最小质因子。

```
//素数筛法 O(n)
2 const int maxn=1e6+10;
   bool valid[maxn];
4 int ans[maxn];
   void getprime(int n,int &tot,int ans[])
6 {
    tot=0;
8   memset(valid,true,sizeof(valid));
   for(int i=2;i<=n;++i){
10      if(valid[i]){
          tot++;
12      ans[tot]=i;
      }
   }
```

```

14      //下面的主角是小于等于i的每个质数
      for(int j=1;(j<=tot) && (i*ans[j]<=n);++j){
16          valid[i*ans[j]]=false;
          if(i%ans[j]==0) break;//如果整除就break;
18      }
      }
20 }

```

code02/euler.cpp

由于每个数只会被筛去一次，时间复杂度是 $O(n)$ 。

2.2.3 区间数的最大、最小质因子

上面介绍的埃氏筛法和欧拉筛法经过一点改动就可以用来求区间数的最大质因子和最小质因子。

对于区间最大质因子，稍微增加点埃氏筛的复杂度，从 $2*i$ 而不是 $i*i$ 开始筛：

```

const int maxn=1e5+10;
2 int ans[maxn];//数的最大质因子 素数就是自己
void get_prime(int n)
4 {
    for(int i=2;i<=n;++i) ans[i]=i;
6    for(int i=2;i<=n;++i){
        if(ans[i]==i){//素数
8            for(int j=i*2;j<=n;j+=i)//最后筛去的即为最大质因子
                ans[j]=i;
10        }
    }
12 }

```

code02/maxprime.cpp

对于区间最小质因子，只需修改欧拉筛中标记数组即可，时间复杂度不变：

```

const int maxn=1e6+10;
2 int min_prime[maxn];
  int tot;
4 int prime[maxn];
  //每个数最小的质因子
6 void getprime(int n)
  {
8     tot=0;
    for(int i=2;i<=n;++i){
10        if(min_prime[i]==0){
            prime[++tot]=i;

```

```

12         min_prime[i]=i;
        }
14         for(int j=1;j<=tot && prime[j]*i<=n;++j){
            min_prime[prime[j]*i]=prime[j];
16             if(i%prime[j]==0) break;
        }
18     }
}

```

code02/minprime.cpp

2.3 区间质因数分解

这里区间质因数分解指的是给定一个上限 n ，要求出区间 $[1, n]$ 中所有数质因数分解的结果。可以用埃氏法先求出所有数的所有质因子；或者使用欧拉法求出区间数的最小质因子再利用区间信息求解。

2.3.1 埃氏法

先 $O(n \log n)$ 求出每个数的所有质因子，再 $O(n \log n)$ 直接除求出质因子对应的指数：

```

1 //注意多个vector push速度很慢
  //若无需记录 尽量不用vector
3 const int maxn=1e6+10;
  vector<int> ans[maxn]; //质因子
5 vector<int> bns[maxn]; //对应指数
  //先预处理每个数有哪些质因数
7 void init1(int n) //处理1000005个数本机跑了1s左右
{
9     for(int i=2;i<=n;++i){
        if(ans[i].size()==0){ //说明是素数
11         for(int j=i;j<=n;j+=i){
            ans[j].push_back(i);
13         }
        }
15     }
}
17 void init2(int n)
{
19     for(int i=2;i<=n;++i){
        int tp=i;
21         for(int j=0;j<ans[i].size();++j){
            bns[i].push_back(0);
23         while(tp%ans[i][j]==0)

```

```

25         {
                bns[i][j]++;
                tp/=ans[i][j];
27         }
        }
29    }
    }
31 int main()
    {
33     int n;
        cin>>n;
35     init1(n);
        init2(n);
37 }

```

code02/interaishi.cpp

2.3.2 欧拉法

先 $O(n)$ 欧拉筛求出每个数的最小质因子，再 $O(n\log n)$ 利用区间信息求出对应的指数。区间信息指的是，对于每个数如果我们都知道其最小质因子，则可以先将当前数的最小质因子除完，剩下的数的最小质因子我们还是知道的，于是继续除，直到 1 为止。

```

1  const int maxn=1e6+10;
    vector<int> ans[maxn];
3  vector<int> bns[maxn];
    int min_prime[maxn];
5  int tot;
    int prime[maxn];
7  void init1(int n)//预处理每个数最小的质因子
    {
9      tot=0;
        for(int i=2;i<=n;++i){
11         if(min_prime[i]==0){
                prime[++tot]=i;
13         min_prime[i]=i;
            }
15         for(int j=1;j<=tot && prime[j]*i<=n;++j){
                min_prime[prime[j]*i]=prime[j];
17         if(i%prime[j]==0) break;
            }
19     }
    }
21 void init2(int n)

```

```

{
23     for(int i=2;i<=n;++i){
        int tp=i;
25         while(tp!=1)
            {
27             int num=0;
                int tem=min_prime[tp];
29             ans[i].push_back(tem);
                while(tp%tem==0)
31                 {
                    num++;
33                     tp/=tem;
                }
35             bns[i].push_back(num);
        }
37     }
}

```

code02/intereuler.cpp

2.4 大区间素数筛与质因数分解

大区间指的是咱们考虑的区间不再是 $[1, n]$ ，而是 $[L, R]$ ，这里 $R \leq 2^{40}$ ， $R - L \leq 10^6$ 。

2.4.1 大区间素数筛

对于一个合数 n ，一定有小于等于 \sqrt{n} 的质因子，利用这一点，我们依然可以使用埃氏筛的思想。先预处理出 $[1, \sqrt{R}]$ 区间内的所有素数，然后拿这些素数去做筛选即可。

时间复杂度为 $O(N + l \log l)$ ，其中 N 为 \sqrt{R} ， R 是区间右端点，即需要预处理的素数范围； l 为区间长度 $R - L$ 。

```

//R<= 2^{32}
2 const int maxn=(1<<16)+10;
    bool valid[maxn];
4 int ans[maxn];
    int tot;
6 void get_prime(int n)
    {
8         tot=0;
            for(int i=2;i<=n;++i) valid[i]=true;
10        for(int i=2;i<=n;++i){
            if(valid[i]){
12                ans[++tot]=i;
            }
        }
    }

```



```

    }
14     for(int j=1;j<=tot && ans[j]*i<=n;++j){
        valid[ans[j]*i]=false;
16         if(i%ans[j]==0) break;
    }
18 }
}
20 //区间长度1e6
bool vis[maxn<<4]; //用于被筛 起平移标记作用
22 int main()
{
24     //求区间素数 闭区间
    ios::sync_with_stdio(false);
26     get_prime(1<<16);
    ll L,R;
28     while(cin>>L>>R)
    {
30         for(int i=1;i<=(R-L+1);++i){vis[i]=true;} // -L+1编号 即L-> 1, L+1
        ->2,...
        if(L==1) vis[1]=false; //1不是素数
32         //下面用素数筛
        for(int i=1;i<=tot;i++) //tot 是预处理的所有素数的个数
34             for(ll j=max(ans[i], (L-1)/ans[i]+1); j<=R/ans[i]; j++) //当前素数的j倍
                vis[ans[i]*j-L+1]=false; //ans[i]*j是合数
36 // (L-1)/ans[i]+1 找的是从L开始第一个ans[i]的倍数是其多少倍
        //考虑这里为什么当L较小时 从ans[i]倍ans[i]开始筛 (当L较大时自然从(L-1)/ans[i]+1倍
        开始)
38 //能从ans[i]*ans[i]开始的条件是 ans[i]的ans[i-1]倍、ans[i-2]倍...已经筛过了(如果
        在L,R区间里)
        //那当i=i-1、i-2...时不不是吗 和埃筛一样
40         vector<ll> tep; //答案
        for(int i=1;i<=(R-L+1);++i){
42             if(vis[i]) tep.push_back(L+i-1);
        }
44         cout<<tep.size()<<endl;
    }
46     return 0;
}

```

code02/large-inter-prime.cpp

2.4.2 大区间质因数分解

和上面类似，先埃筛求出小于等于 \sqrt{R} 的所有质因子，然后对于区间内的每个数直接除即可。

因为一个合数 n 大于 \sqrt{n} 的质因子最多只有 1 个，因此将已知的质因子全部除尽后，若不为 1，就为那个大于 \sqrt{n} 的质因子，对应指数为 1。

时间复杂度为 $O(N + l \log l)$ ，其中 N 为 \sqrt{R} ， R 是区间右端点，即需要预处理的素数范围； l 为区间长度 $R - L$ 。

```

1 const int maxn=(1<<16)+10;
  bool valid[maxn];
3 int prime[maxn];
  int tot;
5 void get_prime(int n)
{
7     tot=0;
    for(int i=2;i<=n;++i) valid[i]=true;
9     for(int i=2;i<=n;++i){
        if(valid[i]==true){
11         prime[++tot]=i;
        }
13         for(int j=1;j<=tot && prime[j]*i<=n;++j){
            valid[prime[j]*i]=false;
15             if(i%prime[j]==0) break;
        }
17     }
}
19 //<<2是为了1e6区间长度
vector<int> ans[maxn<<2]; //区间每个数的质因子    0 对应 L
21 vector<int> bns[maxn<<2]; //质因子对应的指数
int main()
23 {
    get_prime(1<<16);
25     int L,R;
    cin>>L>>R;
27     for(int i=1;i<=tot;++i){
        for(int j=(L-1)/prime[i]+1;j<=R/prime[i];++j){
29             ans[j*prime[i]-L].push_back(prime[i]);
        }
31     }
    //L存在0这里
33     for(int i=0;i<R-L+1;++i){
        if(ans[i].size()==0){

```

```

35         ans[i].push_back(i), bns[i].push_back(1);
           continue;
37     }
           int x=i+L; //加上偏移 还原
39     for(int j=0; j<ans[i].size(); ++j){
           int tp=ans[i][j];
41         int num=0;
           while(x%tp==0){
43             x/=tp;
             num++;
45         }
           bns[i].push_back(num);
47     }
           if(x!=1) ans[i].push_back(x), bns[i].push_back(1);
49     }
           return 0;
51 }

```

code02/large-inter-factor.cpp

2.5 梅森素数与完全数

梅森素数与完全数在程序设计竞赛中不是很常见，这里简要介绍一下。

2.5.1 梅森素数

现在我们来研究形如 $a^n - 1 (n \geq 2)$ 的素数。例如 31 就是这样的数，因为 $31 = 2^5 - 1$ 。由于 $a - 1$ 始终是 $a^n - 1$ 的因子。为啥呢？因为使用几何级求和公式：（展开即可证明该公式）

$$x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + \dots + x^2 + x + 1)$$

所以若 $a^n - 1$ 为素数，则 a 一定等于 2。反之显然不成立。

再进一步考虑 n ，假设 n 为合数，即 $n = mk$ ，则 $2^n = 2^{mk} = (2^m)^k$ 。使用 $x = 2^m$ 的几何级数公式得：

$$2^n - 1 = (2^m)^k - 1 = (2^m - 1)((2^m)^{k-1} + (2^m)^{k-2} + \dots + (2^m)^2 + (2^m) + 1)$$

这证明了若 n 是合数，则 $2^n - 1$ 也是合数。综上有以下命题：

命题 2.1

如果对整数 $a \geq 2$ 与 $n \geq 2$ ， $a^n - 1$ 是素数，则 a 必等于 2 且 n 一定是素数。

形如 $2^p - 1$ 的素数叫做**梅森素数**，已知的梅森素数在<https://www.mersenne.org/primes/>中可以找到，目前已经找到了 51 个（截至 2019.09.03）。发现更多的梅森素数没有太大的数学意义，数学上更令人感兴趣的是下面的问题，其答案尚未知晓：

问题 存在无穷多个梅森素数吗？

2.5.2 完全数

定义 2.2. 完全数

完全数是等于其真因数之和的数。真因数指的是小于自己的因数 (即除去自己)。

定理 2.5. 欧几里得完全数公式

如果 $2^p - 1$ 是素数, 则 $2^{p-1}(2^p - 1)$ 是完全数。

证明 将 $2^{p-1}(2^p - 1)$ 所有真因数直接写出求和即证。

这里的 $(2^p - 1)$ 不就是梅森素数吗？也就是说只要求得一个梅森素数, 就得到一个完全数。

那欧几里得完全数公式是否表示了所有完全数呢？千年之后的欧拉证明了欧几里得完全数公式至少给出了所有偶完全数, 称为欧拉完全数定理。

在介绍和证明欧拉完全数定理之前, 我们先来介绍一个函数 $\sigma(n)$ 。

$\sigma(n) = n$ 的所有因数之和 (包括 1 和 n)。

例如 $\sigma(6) = 1 + 2 + 3 + 6 = 12$ 。下面试着写出其一般公式, $\sigma(p) = p + 1$, $\sigma(p^k) = 1 + p + p^2 + \dots + p^k = \frac{p^{k+1}-1}{p-1}$ 。

如果你用程序打个表, 也许会发现 $\sigma(mn)$ 时常会等于 $\sigma(m)\sigma(n)$, 实际上这在 m 和 n 互质时成立。

定理 2.6. σ 函数公式

- 如果 p 是素数, $k \geq 1$, 则 $\sigma(p^k) = 1 + p + p^2 + \dots + p^k = \frac{p^{k+1}-1}{p-1}$;
- 如果 $\gcd(m, n) = 1$, 则 $\sigma(mn) = \sigma(m)\sigma(n)$ 。

σ 函数在程序设计竞赛中也时常出现 (或者其类似函数, 如因子个数函数), 后面章节还会说到它, 现在我们用它来辅助证明欧拉完全数定理。

显然, σ 函数可以和完全数产生联系, 如果 $\sigma(n) = 2n$, 则 n 恰好是完全数。

定理 2.7. 欧拉完全数定理

如果 n 是偶完全数, 则 n 一定是 $2^{p-1}(2^p - 1)$ 的形式, 其中 $2^p - 1$ 是梅森素数。

证明 假设 n 是偶完全数, n 是偶数则说明可将它分解成 $n = 2^k m$, $k \geq 1$ & m is odd

则 $\sigma(n) = \sigma(2^k m) = \sigma(2^k)\sigma(m) = (2^{k+1} - 1)\sigma(m)$ 。又 n 是完全数, 则 $\sigma(n) = 2n = 2^{k+1}m$ 。

因此可以得到

$$2^{k+1}m = (2^{k+1} - 1)\sigma(m)$$

由上式可知, 由于 $(2^{k+1} - 1)$ 一定是奇数, 所以 $2^{k+1} | \sigma(m)$, 于是设 $\sigma(m) = c * 2^{k+1}$, 带入上式有:

$$2^{k+1}m = (2^{k+1} - 1) * c * 2^{k+1}$$

$$m = (2^{k+1} - 1) * c$$

现在我们有二个式子, $m = (2^{k+1} - 1) * c$ 和 $\sigma(m) = c * 2^{k+1}$ 。

实际上这里 c 只能等于 1, 下面我们来证明 $c = 1$ 。

假设 $c > 1$, 则 $m = (2^{k+1} - 1) * c$ 至少被不同的数 $1, c, m$ 所整除, 因此有

$$\sigma(m) \geq 1 + c + m = 1 + c + (2^{k+1} - 1) * c$$

显然, 这是错误的 (会得出 $0 \geq 1$), 因此 $c = 1$, 所以 $m = 2^{k+1} - 1$, $\sigma(m) = 2^{k+1} = m + 1$, 所以 m 是素数。

现在我们已经证明了, 如果 n 是偶完全数, 则

$$n = 2^k(2^{k+1} - 1), \quad \text{where } (2^{k+1} - 1) \text{ is prime}$$

由于 $2^{k+1} - 1$ 是素数, 则 $(k + 1)$ 一定是素数, (上节已说明)。令 $k + 1 = p$, 则 $n = 2^{p-1}(2^p - 1)$, 其中 $2^p - 1$ 是梅森素数。

欧拉完全数定理证毕。

 **注意** 关于完全数的几个结论:

- 目前还没找到奇完全数
- 每个完全数的全部因数倒数之和都是 2
- 除了 6 以外的完全数, 每个都可以表示成连续奇数的立方和
- 每个完全数都可以表示成 2 的一些连续正整数次幂之和
- 完全数都是以 6、8 结尾

问题 存在奇完全数吗?

第 2 章 习题

1. 区间数的最大质因子是否有线性时间的算法?
2. 大区间数的最小质因子是否有线性时间的算法?
3. 试着写出求单点 $\sigma(n)$ 以及区间 $[1, n]$ 所有 σ 值的代码。

第3章 同余

内容提要

- | | |
|-------------------------------------|---------------------------------------|
| <input type="checkbox"/> 同余方程 | <input type="checkbox"/> 卡米歇尔数 |
| <input type="checkbox"/> 快速乘、快速幂 | <input type="checkbox"/> Miller_Rabin |
| <input type="checkbox"/> 逆元、线性求区间逆元 | <input type="checkbox"/> Pollard_Rho |
| <input type="checkbox"/> 中国剩余定理 | <input type="checkbox"/> 离散对数 |
| <input type="checkbox"/> 欧拉降幂 | <input type="checkbox"/> 原根 |

3.1 同余式与同余方程

3.1.1 同余式

整除性是很好的性质，这在最大公因数、模线性方程和素数分解中均得到了体现。而同余式提供了一种描述整除性质的简便方式。

定义 3.1. 同余

如果 $m|(a-b)$ ，我们就说 a 与 b 模 m 同余并记之为 $a \equiv b(\text{mod } m)$ 。

特别地， $a \equiv a \% m(\text{mod } m)$ 。

数 m 叫做同余式的模。具有相同模的同余式在许多方面表现得很像通常的等式。例如：

若 $a_1 \equiv b_1(\text{mod } m)$ ， $a_2 \equiv b_2(\text{mod } m)$ 则 $a_1 \pm a_2 \equiv b_1 \pm b_2(\text{mod } m)$ ， $a_1 a_2 \equiv b_1 b_2(\text{mod } m)$

但 $ac \equiv bc(\text{mod } m)$ 时，未必有 $a \equiv b(\text{mod } m)$ ，只有 $\text{gcd}(c, m) = 1$ ，才可以消去 c 。

3.1.2 同余方程

如果同余式含有未知数，我们考虑如何求解。

首先考虑“穷举法”，要解模 m 同余式，可让每个变量试取 $0, 1, 2, \dots, m-1$ 。例如，解同余式 $x^2 + 2x - 1 \equiv 0(\text{mod } 7)$ ，就去试 $x = 0$ ， $x = 1$ ，...， $x = 6$ ，这样可以求出两个解 $x \equiv 2(\text{mod } 7)$ 和 $x \equiv 3(\text{mod } 7)$ 当然还有其他解，但新的解与 2 或 3 是同余的，如 9 和 10。当我们说“求同余式的所有解时”，是指求所有不同余的解，即相互不同余的所有解。

许多同余式是没有解的，如 $x^2 \equiv 3(\text{mod } 10)$ 。

下面考虑如何求解同余式 $ax \equiv c(\text{mod } m)$ 。

例 3.1 解同余式 $18x \equiv 8(\text{mod } 22)$

等价于求 $22 \mid (18x - 8)$ ，即求 $18x - 22y = 8$ 。

解这类方程的问题在第一章中研究过。

对于 $ax \equiv c(\text{mod } m)$ ，其有解当且仅当线性方程 $ax - my = c$ 有解。

由前可知，线性方程 $ax - my = c$ 有解的充分必要条件是 $\gcd(a, m) \mid c$ 。

且方程 $au + mv = g$ 一定有解，设一个解为 (u_0, v_0) ，则有

$$a \frac{cu_0}{g} + m \frac{cv_0}{g} = c$$

这说明 $x_0 = \frac{cu_0}{g}$ 是同余式 $ax \equiv c(\text{mod } m)$ 的一个解，通解为 $x = x_0 + k \cdot \frac{m}{g}$ 。

由于相差 m 的倍数的任何两个解认为是相同的，所有恰好有 g 个不同的解，这些解通过取 $k = 0, 1, 2, \dots, g-1$ 而得到。将上述过程概述为定理：

定理 3.1. 线性同余式定理 $ax \equiv c(\text{mod } m)$

设 a, c 与 m 是整数， $m \geq 1$ ，且设 $g = \gcd(a, m)$ 。

(a) 如果 $g \nmid c$ ，则同余式 $ax \equiv c(\text{mod } m)$ 没有解

(b) 如果 $g \mid c$ ，则同余式 $ax \equiv c(\text{mod } m)$ 恰好有 g 个不同的解。要求这些解，首先求线性方程 $au + mv = g$ 的一个解 (u_0, v_0) (欧几里得回带法，在计算机上递归求得，称为扩展欧几里得算法)。则 $x_0 = (\frac{cu_0}{g} \% \frac{m}{g} + \frac{m}{g}) \% \frac{m}{g}$ 是 $ax \equiv c(\text{mod } m)$ 的解，不同余解的完全集由

$$x \equiv x_0 + k \cdot \frac{m}{g} (\text{mod } m), \quad k = 0, 1, 2, \dots, g-1$$

给出。

例如，同余式 $943x \equiv 381(\text{mod } 2576)$ 无解，这是因为 $\gcd(943, 2576) \nmid 381$ 。

另一方面，同余式 $893x \equiv 266(\text{mod } 2432)$ 有 19 个解，因为 $\gcd(893, 2432) = 19$ ， $19 \mid 266$ 这个 19 即为不同余解的个数。

下面解方程 $893u - 2432v = 19$ ，使用欧几里得回带法可以求得解 $(u, v) = (79, 29)$ ，乘以 $266/19=14$ 得方程 $893x - 2432y = 266$ 的解 $(x, y) = (1106, 406)$ ，即 1106 是同余式方程的一个解，这样的互不同余的解共有 19 个。1106 加上 $2432/19=128$ 的倍数 ($\%2432$) 就可得到完全解集。

注 线性同余式定理最重要的情形是 $\gcd(a, m) = 1$ ，在这种情形下，同余式恰好有一个解。

```
1 #include<bits/stdc++.h>
   using namespace std;
3 typedef long long ll;
   //拓展欧几里得模板
5 ll e_gcd(ll a,ll b,ll &x,ll &y)
   {
7     if(b==0){
           x=1;
9         y=0;
           return a;
11    }
```

```

    ll ans=e_gcd(b,a%b,x,y);
13    ll temp=x;
    x=y;
15    y=temp-a/b*y;
    return ans;
17 }
    //ax同余c(mod m) 输出[0,m) 中的解
19 vector<ll> mod_equation(ll a,ll c,ll m){
    ll u,v;
21    ll d=e_gcd(a,m,u,v);
    vector<ll>ans;
23    ans.clear();
    if(c%d==0){
25        u=(u*(c/d));
        u=(u%(m/d) + (m/d))%(m/d); //最小正整数解
27        ans.push_back(u);
        for(ll k=1;k<d;++k) //mod n意义下有d(gcd)个解
29            ans.push_back((ans[0]+k*(m/d))%m);
    }
31    return ans;
}
33 vector<long long>ans;
int main()
35 {
    ll a,c,mod;
37    cin>>a>>c>>mod;
    ans=mod_equation(a,c,mod);
39    if(!ans.size()) cout<<"无解"<<endl;
    else{
41        for(int i=0;i<ans.size();++i){
            cout<<ans[i]<<' ';
43        }
    }
45    return 0;
}

```

code03/modequation.cpp

对于非线性的同余式，其解“不是很确定”。

我们熟悉的是对于一个 d 次实系数多项式的实根不超过 d 个，这个结论对于同余式并不成立。

例如 $x^2 + x \equiv 0 \pmod{6}$ 有 4 个模 6 不同的根：0,2,3,5。

但是，当 p 为素数时，这个结论依然成立：

定理 3.2. 模 p 多项式根定理

设 p 为素数， $f(x) = a_0x^d + a_1x^{d-1} + \dots + a_d$ 是次数为 $d \geq 1$ 的整系数多项式，且 p 不整除 a_0 ，则同余式 $f(x) \equiv 0 \pmod{p}$ 最多有 d 个模 p 不同余的解。



3.2 快速乘与快速幂

快速乘和快速幂作为工具经常在程序设计竞赛中遇见。

3.2.1 快速乘

在 $C++$ 中，变量最多只能表示到 $2^{64} - 1$ 这么大，所以如果我们要计算 $a * b \% c$ ，而 a, b 都是接近表示上限的数，这个时候就需要快速乘。即将 b 按二进制位分解分别加上，时间复杂度为 $O(\log(b))$ 。

```
1 ll mod_mul(ll a, ll b, ll c){ //a*b %c 乘法改加法 防止超long long
    ll res=0;
3    a=a%c;
    assert(b>=0);
5    while(b)
    {
7        if(b&1) res=(res+a)%c;
        b>>=1;
9        a=(a+a)%c;
    }
11    return res;
}
```

code03/fastmul.cpp


3.2.2 $O(1)$ 快速乘

上面的“快速乘”时间是 $O(\log b)$ 的，如果是两个 63 位数相乘（结果 long long 存不下），则还有一个巧妙的方法可以简单解决这个问题：

```
inline ll mod_mul(ll a, ll b, ll p){
2    a%=p, b%=p;
    ll c=a*b-(ll)((long double)a*b/p+0.5)*p;
4    return c<0?c+p:c;
}
```

code03/o1fastmul.cpp

首先使用浮点运算来得到 $\lfloor \frac{a*b}{p} \rfloor$ 的值, 显然 $a*b - d*p$ 中的两个乘法都有可能会溢出, 但是没关系, 因为可以知道其差是一个 64bit 可容纳的正整数, 那么溢出部分的差仅可能为 0 或者 1, 最后特判处理一下即可。


 **注意** 当然, 如果编译器支持 `int128` 的话, 可以直接用 128 位数。

3.2.3 快速幂

现在我们要计算 $a^b \% c$, 如果乘 b 次, 时间复杂度太高, 考虑将 b 按照二进制分解, 每一位分别计算并乘在一起即可。时间复杂度为 $O(\log(b))$ 。相比于快速乘, 只是加法变了乘法。

```
1 ll fast_exp(ll a,ll b,ll c){
    ll res=1;
3   a=a%c;
    assert(b>=0);
5   while(b>0)
    {
7       if(b&1) res=(a*res)%c;
        b=b>>1;
9       a=(a*a)%c;
    }
11  return res;
}
```

code03/fastexp.cpp

 **注意** 快速幂可以使用不同的进制, 以及在特定问题下做一些预处理进行记忆化, 可以节省时间。

如2019icpc 南昌网络赛 The Nth Item。

3.3 费马小定理与逆元

前面我们讨论了关于同余式、同余方程的一些性质, 小结一下, 互质这个条件很重要。

- 对于 $ac \equiv bc(mod\ m)$, 如果 $gcd(c,m) = 1$, 则可以消去 c , 得到 $a \equiv b(mod\ m)$ 。
- 对于同余方程 $ax \equiv c(mod\ m)$, 若 $gcd(a,m) = 1$, 同余式恰好有一个解。

对于式子 $ax \equiv c(mod\ m)$, 令 $c = 1$, 得 $ax \equiv 1(mod\ m)$, 若 $gcd(a,m) = 1$, 则方程有唯一解 x_0 , 我们称 x_0 为 a 在模 m 意义下的逆元, 常记作 a^{-1} 。

逆元可以用来说明一些事情, 比如如果 $ac \equiv bc(mod\ m)$, 若 $gcd(c,m) = 1$, 则存在 c^{-1} 使得 $cc^{-1} \equiv 1(mod\ m)$ 。所以可以对 $ac \equiv bc(mod\ m)$ 两边同时乘以 c^{-1} , 得到 $a \equiv b(mod\ m)$ 。

如何求解逆元呢? 拓展欧几里得即可, 因为就是一个同余方程:

```
//拓展欧几里得
```

```

2 ll e_gcd(ll a,ll b,ll &x,ll &y)
{
4     if(b==0){x=1;y=0;return a;}
    ll ans=e_gcd(b,a%b,x,y);
6     ll temp=x; x=y; y=temp-a/b*y;
    return ans;
8 }

10 //求逆元 已知gcd(a,mod)=1
ll inv(ll a,ll mod)
12 {
    ll ans,tmp;
14     e_gcd(a,mod,ans,tmp);
    return (ans+mod)%mod;
16 }

```

code03/inverse.cpp

若模数是素数，我们还可以用费马小定理求解。

定理 3.3. 费马小定理

设 p 是素数， a 是任意整数且 $p \nmid a$ ，则 $a^{p-1} \equiv 1 \pmod{p}$ 。

在证明费马小定理之前，先来看一个引理。

引理 3.1. 为证明费马小定理做准备

设 p 是素数， a 是任何整数且 $p \nmid a$ 则数

$$a, 2a, 3a, \dots, (p-1)a \pmod{p}$$

与数

$$1, 2, 3, \dots, (p-1) \pmod{p}$$

相同，尽管它们的次序不同。

证明 数列 $a, 2a, 3a, \dots, (p-1)a$ ，包含 $p-1$ 个数，显然没有一个数被 p 整除，假设从中取出两个数 ja 和 ka 是关于 p 同余的，即 $p \mid (j-k)a$ ，又 p 是素数且 p 不整除 a ，所以 p 整除 $(j-k)$ 。但是 $|j-k| < p-1$ ，所以 $j-k=0$ ，即 $j=k$ 。

这表明，这 $p-1$ 个数模 p 不同，由于任何数 \pmod{p} 仅有 $p-1$ 个不同的非零值，证毕。

证明 费马小定理的证明。


利用该引理3.1，即可完成对费马小定理3.3的证明，将上面引理列到的两组数相乘，可得到

$$a^{p-1} \cdot (p-1)! \equiv (p-1)! \pmod{p}$$

由于 $(p-1)!$ 与 p 互质（显然除了1没有其它公共因子了），可以消去它（本节开头有提到这个性质），则 $a^{p-1} \equiv 1 \pmod{p}$ 。

有了费马小定理, 若模数 m 是质数, 则数 a 关于 m 的逆元就是 a^{m-2} , 因为 $a * a^{m-2} \equiv 1 \pmod{p}$ 。使用快速幂直接计算即可。

使用费马小定理还可以进行素数测试, 后面小节会提到。

 **注意** 如果要求 $1, 2, 3, \dots, n$ 所有数的逆元, 除了每个单独求之外, 实际上还有线性的做法。

设 $p = ki + b$, 那么 $ki + b \equiv 0 \pmod{p}$ 。

两边同乘以 $i^{-1}b^{-1}$ 得到 $kb^{-1} + i^{-1} \equiv 0 \pmod{p}$, $i^{-1} \equiv -kb^{-1}$, 于是有:

$$i^{-1} \equiv -\lfloor \frac{p}{i} \rfloor * (p \% i)^{-1} \pmod{p}$$

通常在前面加上 p , 变为正数:

$$i^{-1} \equiv (p - \lfloor \frac{p}{i} \rfloor) * (p \% i)^{-1} \pmod{p}$$

代码: 预处理 $1 \sim n$ 的逆元、阶乘和阶乘逆元。时间复杂度为线性。

```

1 typedef long long ll;
2 const int mod=10007;
   ll inv[mod+10];
4 ll Fac[mod+10];
   ll Fac_inv[mod+10];
6 void init()
   {
8     inv[1]=1;
       Fac[0]=1;
10    Fac_inv[0]=1;
       for(int i=1;i<mod;++i){
12        if(i!=1) inv[i]=(mod-mod/i)*(inv[mod%i]))%mod;
           Fac[i]=Fac[i-1]*i%mod;
14        Fac_inv[i]=Fac_inv[i-1]*inv[i]%mod;
       }
16 }

```

code03/linearinverse.cpp

3.4 中国剩余定理

定理 3.4. 中国剩余定理

设 m 与 n 是整数, $\gcd(m, n) = 1$, b 与 c 是任意整数, 则同余式组 $x \equiv b \pmod{m}$ 与 $x \equiv c \pmod{n}$ 恰有一个解 $0 \leq x < mn$.



证明 对于第一个同余式 $x \equiv b \pmod{m}$, 其解由形如 $x = my + b$ 的所有数组成。将其带入第二个方程可得 $my \equiv c - b \pmod{n}$, 已知 $\gcd(m, n) = 1$, 由线性同余式定理知其恰有一个解 y_1 , $0 \leq y_1 < n$, 则 $x_1 = my_1 + b$ 给出了原来同余式组的解, 这是在 $[0, mn)$ 之间的唯一解。

上面只考虑了两个同余式, 如果有多个呢?

问题 求出方程组 $x \equiv a_i \pmod{m_i} (0 \leq i < n)$ 的解 x , 其中 $m_0, m_1, m_2, m_3, \dots, m_{n-1}$ 两两互质。

解 令 $M_i = \prod_{j \neq i} m_j$ 则有 $(M_i, m_i) = 1$

故存在 p_i, q_i , 使得 $M_i * p_i + m_i * q_i = 1$

令 $e_i = M_i p_i$, p_i 即为 M_i 模 m_i 下的逆元。

则有

$$e_i \equiv \begin{cases} 0 \pmod{m_j}, & j \neq i \\ 1 \pmod{m_j}, & j = i \end{cases}$$

故 $e_0 a_0 + e_1 a_1 + e_2 a_2 + \dots + e_{n-1} a_{n-1}$ 是方程的一个解。

由中国剩余定理知, $[0 \sim \prod_{i=0}^{n-1} m_i]$ 中必有一解, 将上式模 $\prod_{i=0}^{n-1} m_i$ 即可。

时间复杂度 $O(n \log m)$, 其中 n 表示有 n 个方程。

```
typedef long long ll;
2 const int maxn=101; //方程个数
  //拓展欧几里得模板
4 ll e_gcd(ll a, ll b, ll &x, ll &y)
{
6     if(b==0){
        x=1;
8         y=0;
        return a;
10    }
    ll ans=e_gcd(b, a%b, x, y);
12    ll temp=x;
    x=y;
14    y=temp-a/b*y;
    return ans;
16 }
  //x同余a mod m
18 //a为方程右值 m为方程的模 n为方程数
```

```

//下标从0开始
20 ll CRT(ll a[],ll m[],int n){
    ll M=1;
22     for(int i=0;i<n;++i) M*=m[i];
    ll ret=0;
24     for(int i=0;i<n;++i){
        ll x,y;
26         ll tm=M/m[i];
        ll tmp=e_gcd(tm,m[i],x,y); //tm 和 m[i] 互质 只有一个解
28         //cout<<x<<endl;
        ret=(ret+tm*x*a[i])%M; //这里x可能为负值 但不影响结果
30     }
    return (ret+M)%M;
32 }
ll a[maxn];
34 ll m[maxn];
int n;
36 int main()
{
38     ios::sync_with_stdio(false);
    cin>>n;
40     for(int i=0;i<n;++i)
        cin>>a[i]>>m[i];
42     cout<<"一个解为:"<<CRT(a,m,n)<<endl;
    return 0;
44 }

```

code03/crt.cpp

问题 如果这些方程的模数不互质呢？（上面互质的方法只是一种巧妙的构造）

一样可以求解，每一次我们将两个方程合并为一个方程，具体来说，假设目前的前两个方程为 $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$ 。将第一个带入第二个可得到 $km_1 \equiv a_2 - a_1 \pmod{m_2}$ ，解这个同余方程，求出 k 的最小正整数解 k_0 ，那么 $k = \frac{m_2}{\gcd}y + k_0$ ，带入 $x = km_1 + a_1$ 中可以得到 $x = \frac{m_1 m_2}{\gcd}y + m_1 k_0 + a_1$ ，也即将前两个方程合并为了一个： $x \equiv m_1 k_0 + a_1 \pmod{\frac{m_1 m_2}{\gcd}}$ ，这样迭代下去，最后的那个方程即为所求。

时间复杂度为 $n \log m$ ，其中 n 是方程个数。代码中的 x_0 即为上面的 k_0 。

```

typedef long long ll;
2 typedef __int128 lll;
lll e_gcd(lll a,lll b,lll &x,lll &y)
4 {
    if(b==0){
6         x=1;
        y=0;

```

```

8     return a;
    }
10    ll ans=e_gcd(b,a%b,x,y);
    ll temp=x;
12    x=y;
    y=temp-a/b*y;
14    return ans;
}
16 //x = a (mod m)  index form 0
bool crt_flag;
18 const int maxn = 1e5+10;
    ll a[maxn],m[maxn];
20 ll lcm;
ll EXCRT(ll a[], ll m[], int n){//n 是方程数量 时间复杂度: nlogm
22    ll m1 = m[0], a1 = a[0], m2, a2, k1, k2, x0, gcd, c;
    a1 = a1%m1;
24    lcm = m1;//lcm 是最小公倍数
    for(int i=1;i<n;i++){
26        m2 = m[i], a2 = a[i];
        a2 = a2%m2;
28        c = a2 - a1;
        gcd = e_gcd(m1, m2, k1, k2);// solve m1*k1 + m2*k2 = gcd(m1,m2)
30        lcm = m2/gcd*m1;
        if(c%gcd){
32            crt_flag = 1;//无解
            return 0;//注意0也不一定无解 要先看crt_flag
34        }
        x0 = c/gcd*k1;
36        ll mod = m2/gcd;// 应该一定为正数
        x0 = (x0%mod+mod)%mod;//最小非负整数解
38        a1 = (a1+m1*x0%lcm)%lcm;
        m1 = lcm;
40    }
    return (a1%lcm+lcm)%lcm;
42 }
int main()
44 {
    crt_flag = false;
46    int n;    cin>>n;
    for(int i=0;i<n;++i) cin>>m[i]>>a[i];
48    ll ans = EXCRT(a,m, n);
    if(crt_flag) cout<<"no solution"<<endl;
50    else cout<<ans<<endl;

```

```

return 0;
52 }

```

code03/excrt.cpp

3.5 欧拉公式与欧拉降幂

3.5.1 欧拉公式

费马小定理很漂亮 $a^{p-1} \equiv 1 \pmod{p}$ ，但限制 p 是素数且 $p \nmid a$ 。如果 p 是合数，即使 a, p 互质，结论也不正确了。那是否有 $a^{???} \equiv 1 \pmod{m}$ 成立的指数呢？带规律的那种。首先，如果 a 的某个幂模 m 余 1，则 a 和 m 必互质（可由线性方程定理证明）。

这再次提醒我们观察与 m 互素的数的集合：

$$a : 1 \leq a \leq m, \quad \gcd(a, m) = 1$$

在 $1 \sim m$ 之间与 m 互质的整数个数是个重要的量，我们赋予这个量一个名称：

定义 3.2. 欧拉函数 φ

$$\varphi(m) = \{a : 1 \leq a \leq m, \quad \gcd(a, m) = 1\}$$

注意 p 是素数时，每个整数 $1 \leq a < p$ 都与 p 互素，所以对于素数 p 有公式

$$\varphi(p) = p - 1$$

我们设法模仿费马小定理的证明。例如，假设要求 7 的幂次模 10 余 1，不取所有 $1 \sim 9$ ，而是恰好取与 10 互素的数，它们是

$$1, 3, 7, 9$$

如果用 7 去乘每个数可得

$$7 \cdot 1 \equiv 7 \pmod{10} \quad 7 \cdot 3 \equiv 1 \pmod{10} \quad 7 \cdot 7 \equiv 9 \pmod{10} \quad 7 \cdot 9 \equiv 3 \pmod{10}$$

得到的 4 个数是之前的 4 个数的重排！如果将它们乘起来就得到相同的乘积

$$(7 \cdot 1)(7 \cdot 3)(7 \cdot 7)(7 \cdot 9) \equiv 1 \cdot 3 \cdot 7 \cdot 9 \pmod{10}$$

$$7^4(1 \cdot 3 \cdot 7 \cdot 9) \equiv 1 \cdot 3 \cdot 7 \cdot 9 \pmod{10}$$

由于 $1 \cdot 3 \cdot 7 \cdot 9$ 与 10 是互质的，因此可以消去，所以得到 $7^4 \equiv 1 \pmod{10}$ ，这个形式和费马小定理很像了！

考虑这里的指数 4 和费马小定理中的 $p - 1$ 的共同点，都是 $1 \sim m$ 中与 m 互质的数的个数，即欧拉函数 $\varphi(m)$ 。

定理 3.5. 欧拉公式

如果 $\gcd(a, m) = 1$ ，则

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

引理 3.2. 为证明欧拉公式做准备

如果 $\gcd(a, m) = 1$, 则数列

$$b_1a, b_2a, b_3a, \dots, b_{\varphi(m)}a \pmod{m}$$

与数列

$$b_1, b_2, b_3, \dots, b_{\varphi(m)} \pmod{m}$$

相同, 尽管它们可能次序不同, b_i 表示小于 m 且与 m 互质的数。



证明 引理的证明。

- 注意到 b_i 和 a 均与 m 是互质的, 则 $b_i a$ 也与 m 互质。又因为所有与 m 互质的数 $\%m$ 后依然与 m 互质 (如果 $x \cdot km$ 与 m 不互质, 则 x 与 m 也不互质了), 所以数列 $b_1a, b_2a, b_3a, \dots, b_{\varphi(m)}a \pmod{m}$ 同余于数列 $b_1, b_2, b_3, \dots, b_{\varphi(m)} \pmod{m}$ 中的某一个数 (因为就这 $\varphi(m)$ 个和 m 互质)。又每个数列有 $\varphi(m)$ 个数, 因此, 如果能进一步证明第一个数列中的数对于模 m 互不相同, 就可得出两个数列 (重排后) 相同。
- 从第一个数列中任选两个数, 假设它们是同余的, 那么意味着 $m \mid a(b_i - b_j)$, 由于 a, m 是互质的, 因而有 $m \mid b_i - b_j$, 又 b_i, b_j 在 1 与 m 之间, 这说明 $b_i = b_j$, 即第一个数列中的数模 m 是不同的。引理证毕。

证明 欧拉公式的证明。

利用该引理 3.2, 即可完成对 3.5 欧拉公式的证明, 由引理知第一个数列中数的乘积等于第二个数列中数的乘积:

$$(b_1a) \cdot (b_2a) \cdot (b_3a) \cdot \dots \cdot (b_{\varphi(m)}a) \equiv b_1 \cdot b_2 \cdot b_3 \cdot \dots \cdot b_{\varphi(m)} \pmod{m}$$

左边提出 $\varphi(m)$ 个 a 得到 $a^{\varphi(m)}B \equiv B \pmod{m}$, 其中 $B = b_1b_2b_3 \cdot \dots \cdot b_{\varphi(m)}$ 。

由于每个 b 与 m 都是互质的, 因此 B 与 m 也是互质的, 因此 B 可以消去, 于是得到

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

证毕。

关于欧拉函数 ϕ , 后面还会遇到。下一节让我们先看一下欧拉公式的一个应用。

3.5.2 欧拉降幂

如何计算 $5^{1000000000000000} \pmod{12830603}$? (实际上会有很多零, 比如 10^5 个, 这里为了说明问题简写)

如果 12830603 是素数, 则直接使用费马小定理, 可以将指数除以 $p-1$, 将余数作为幂计算即可。

但 $12830603 = 3571 \cdot 3593$, 不是素数。

但, $\gcd(5, 12830603) = 1$, 因此由欧拉公式知 $5^{\varphi(12830603)} \equiv 1 \pmod{12830603}$ 。计算得到 $\varphi(12830603) = 12823440$, 因此只要把 100..000 除以 12823440 的余数作为指数即可。注意这里要求 5 和 12830603 互质。

如果底数和模数不互质呢? 有广义欧拉降幂公式, 总结如下:



定理 3.6. 广义欧拉降幂公式

$$a^b \equiv \begin{cases} a^{b \% \phi(p)} & \gcd(a, p) = 1 \\ a^b & \gcd(a, p) \neq 1, b \leq \phi(p) \\ a^{b \% \phi(p) + \phi(p)} & \gcd(a, p) \neq 1, b > \phi(p) \end{cases} \pmod{p}$$



证明 第一行和第二行的式子之前已经说明，下面证明 $b > \phi(p)$ 的情况。

设 $b = A * \phi(p) + C$ ，其中 $A \geq 1, 0 \leq C < \phi(p)$ 。

那么我们要证明的就是 $a^{A * \phi(p) + C} \equiv a^{C} \pmod{p}$ 。

如果我们能证明 $a^{A * \phi(p)} \equiv 1 \pmod{p}$ ，则上式也就成立。

即证 $a^{2 * \phi(p)} \equiv 1 \pmod{p}$ ，移项即证

$$p \mid a^{\phi(p)}(a^{\phi(p)} - 1)$$

(这里 p 不一定是素数)

假设

$$\left(\frac{p}{(p, a^{\phi(p)})}, a\right) = 1$$

那么根据欧拉定理，

$$a^{\phi(p)} = a^{k * \phi\left(\frac{p}{(p, a^{\phi(p)})}\right)} \equiv [a^{\phi\left(\frac{p}{(p, a^{\phi(p)})}\right)}]^k \equiv 1, \pmod{\frac{p}{(p, a^{\phi(p)})}}$$

其中 $k \geq 1$ ，移项可得 $\frac{p}{(p, a^{\phi(p)})} \mid (a^{\phi(p)} - 1)$ 。两边同时乘 $(p, a^{\phi(p)})$ 可得 $p \mid (p, a^{\phi(p)}) * (a^{\phi(p)} - 1)$ ，于是也就证明了 $p \mid a^{\phi(p)}(a^{\phi(p)} - 1)$ 。证毕。

但上面的假设还没有证明，实际上这个假设是一定成立的，下面证明。

对 a 和 p 进行质因数分解，

$$a = p_1^{a_1} * p_2^{a_2} * \dots * p_{t_1}^{a_{t_1}} * q_1^{b_1} * q_2^{b_2} * \dots * q_{t_2}^{b_{t_2}}$$

$$p = p_1^{c_1} * p_2^{c_2} * \dots * p_{t_1}^{c_{t_1}} * r_1^{d_1} * r_2^{d_2} * \dots * r_{t_3}^{d_{t_3}}$$

则 $(a, p) = p_1^{\min(a_1, c_1)} * p_2^{\min(a_2, c_2)} * \dots * p_{t_1}^{\min(a_{t_1}, c_{t_1})}$,

$$(a^{\phi(p)}, p) = p_1^{\min(a_1 * \phi(p), c_1)} * p_2^{\min(a_2 * \phi(p), c_2)} * \dots * p_{t_1}^{\min(a_{t_1} * \phi(p), c_{t_1})}。$$

我们分析一下 $a_i * \phi(p)$ ， $a_i * \phi(p) \geq a_i * p_i^{c_i-1} * (p_i - 1) \geq p_i^{c_i-1} * (p_i - 1) \geq p_i^{c_i-1} \geq c_i$ 。(其中 p_i 是 p 的因子)。

于是有 $(a^{\phi(p)}, p) = p_1^{\min(a_1 * \phi(p), c_1)} * p_2^{\min(a_2 * \phi(p), c_2)} * \dots * p_{t_1}^{\min(a_{t_1} * \phi(p), c_{t_1})} = p_1^{c_1} * p_2^{c_2} * \dots * p_{t_1}^{c_{t_1}}$ 。

于是

$$\left(\frac{p}{(p, a^{\phi(p)})}, a\right) = 1$$

证毕。

实际上，广义欧拉降幂公式说明的是 $a^b \% c$ 循环节的问题，

这里<https://math.stackexchange.com/questions/653682> 有相关的讨论， $\phi(c)$ 不一定是最小的循环节长度。

例 3.2 2019 南京网络赛 B.superlog，题目经简化后即求 $a^{a^{\dots}} \% m$ 的值，其中 \dots 共有 b 个

a , 数据范围是 $a, b, m \leq 10^6$ 。

由于 m 的范围已知, 发现只有很少个形如 $a^{a^{\dots}}$ 的式子的真值会比 m 小, 提前预处理一下即可, 这也是比较常见的做法, 因为指数套指数增长的非常快。

代码如下, 时间复杂度为 $O(T * \log(m) * \sqrt{m})$ 。

```

11 vis[8][5];
2 11 phi(11 x)
   {
4     if(x==1) return 1;
       11 res=x;
6     for(int i=2;i*i<=x;++i){
           if(x%i==0){
8               res-=res/i;
                   do{
10                      x/=i;
                          }while(x%i==0);
12         }
           }
14     if(x>1) res-=res/x;
       return res;
16 }
11 fast_exp(11 a,11 b,11 c){
18     11 res=1;
       a=a%c;
20     assert(b>=0);
       while(b>0)
22     {
           if(b&1) res=a*res%c;
24           b=b>>1;
           a=a*a%c;
26     }
       return res;
28 }
11 solve(11 a, 11 b, 11 m)
30 {
       if(b==1) return a%m;
32     if(m==1) return 0;
       11 phim = phi(m);
34     //cout<<"phim: "<<phim<<endl;
       if(a<=7 && b<5 && vis[a][b] && vis[a][b]<=phim) return fast_exp(a, solve(a,
           b-1, phim), m);
36     return fast_exp(a, solve(a, b-1, phim) + phim, m);
}

```

```

38 void init()
{
40     for(int i=3;i<=7;++i){
        vis[i][2] = pow(i,i);
42         vis[i][2] = pow(i,i);
        vis[i][2] = pow(i,i);
44     }
    vis[2][2] = 1LL<<2;
46     vis[2][3] = 1LL<<(vis[2][2]);
    vis[2][4] = 1LL<<(vis[2][3]);
48 }
int main()
50 {
    init();
52     int t;
    cin>>t;
54     while(t-->0)
    {
56         ll a,b,m;
        cin>>a>>b>>m;
58         if(a==1 || b==0){
            cout<<1%m<<endl;
60             continue;
        }
62         cout<<solve(a,b,m)<<endl;
    }
64     return 0;
}

```

code03/superlog.cpp

3.5.3 威尔逊定理

在欧拉公式的证明中，我们记 $B = b_1 b_2 b_3 \dots b_{\phi(m)}$ ，即 B 为小于 m 且与 m 互质的数的乘积。那么我们能给出一个结论， $B \equiv 1(\text{mod } m)$ 或 $B \equiv m-1(\text{mod } m)$ 。那 m 的模式是怎样的？（即何时为 1，何时为 $m-1$ ）

定理 3.7. 威尔逊定理

将 m 质因数分解，若其形如 $2^2, p^k, 2 * p^k$ 中的一种，则 $B \equiv m-1(\text{mod } m)$ ，否则 $B \equiv 1(\text{mod } m)$ ，其中 p 为奇素数。

特殊地，若 m 为素数，则 $B = (m-1)!$ ，有 $(m-1)! \equiv m-1(\text{mod } m)$ ，反过来也成立。

威尔逊定理给出了判定一个自然数是否为素数的充分必要条件，但是由于阶乘是

呈爆炸增长的，其结论对于实际操作意义不大。实际（比赛）中，更常用的是一个叫 *Miller_Rabin* 的测试，下面就来看一下如何做素性测试。

3.6 素性测试

素数是整数中优美的一部分，怎么判别一个数是不是素数呢？费马小定理告诉我们 $a^p \equiv a \pmod{p}$ ，首先，不满足上式的数，一定不是素数。但就算你尝试了许多 a 之后发现都满足上式，也不能断言 p 就是素数！

确实，对于大部分小的合数 n ，你会发现选取的大部分 $a (a < n)$ ，都不满足费马小定理。但是！还是有一些合数，比如 $561 = 3 * 11 * 17$ ，对所有的 $0 \leq a < 561$ ，发现都满足费马小定理！

下面证明一下，要证明 $a^{561} \equiv a \pmod{561}$ ，只要证明

$$a^{561} \equiv a \pmod{3}, \quad a^{561} \equiv a \pmod{11}, \quad a^{561} \equiv a \pmod{17}$$

为什么呢？因为若第一个式子成立，则 $3 \mid a^{561} - a$ ，同理 11, 17 也整除，由于 3, 11, 17 互质，则 561 也整除。下面就来证明，对于第一个式子，若 3 能整除 a ，则显然成立；否则，由于 $a^2 \equiv 1 \pmod{3}$ ，则 $a^{561} = a^{2 \cdot 280 + 1} = (a^2)^{280} * a \equiv 1 * a \equiv a \pmod{3}$ ，后面两个式子同理。

561 很特殊！类似这样的数，称为卡米歇尔数。<http://oeis.org/A002997/b002997.txt>

3.6.1 卡米歇尔数

定义 3.3. 卡米歇尔数

卡米歇尔数是这样的合数 n ，即对每个整数 $1 \leq a < n$ ，都有

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

关于卡米歇尔数，存在一些猜想：

1. 卡米歇尔数都是奇数；
2. 卡米歇尔数是不同素数的乘积。

证明 对 1 的证明。

由于 $a^n \equiv a \pmod{n}$ ，令 $a = n - 1$ ，则可得 $(-1)^n \equiv -1 \pmod{n}$ 则 n 是奇数。

证明 对 2 的证明。

假设卡米歇尔数的任意一个素因子（次幂）为 $p^{(e+1)}$ ，下面我们努力证明 $e = 0$ 。

将 p^e 带入 $a^n \equiv a \pmod{n}$ ，得到 $p^{en} \equiv p^e \pmod{n}$ ，所以 $n \mid p^{en} - p^e$ ，又 $p^{e+1} \mid n$ ，所以 $p^{e+1} \mid p^{en} - p^e$ ，从而 $e = 0$ 。证毕。

定理 3.8. 卡米歇尔数的考塞特判别法

设 n 是合数，则 n 是卡米歇尔数当且仅当它是奇数，且整除 n 的每个素数 p 满足下述两个条件：（充分必要）

- p^2 不整除 n

- $p-1$ 整除 $n-1$ (实际上整除更小的数 $\frac{n}{p}-1$)



利用考塞特判别法，我们可以在 $O(\sqrt{n})$ 的时间内处理单个数的判别以及 $O(T * n)$ 求区间 carmichael 数，其中 T 表示大常数。

下面的代码求出 $\leq 10^7$ 的所有 carmichael 数。

```

1 typedef long long ll;
   const int maxn=1e7+10;
3 bool valid[maxn];
   int ans[maxn/10];
5 int res[maxn]; //每个数的最小质因子
   void getprime(int n,int &tot)
7 {
   tot=0;
9   memset(valid,true,sizeof(valid));
   for(int i=2;i<=n;++i){
11      if(valid[i]){
          tot++;
13      ans[tot]=i;
          res[i]=i;
15      }
          for(int j=1;(j<=tot) && (i*ans[j]<=n);++j){
17      int tp=i*ans[j];
          valid[tp]=false;
19      res[tp]=ans[j]; //记录每个数的最小质因子
          if(i%ans[j]==0) break;
21      }
   }
23 }
   bool check(int x)
25 {
   if (x%2==0) return false;
27   if (valid[x]) return false;
   //对x进行分解 要求每个素因子的指数均为1且p-1 | x-1
29   int tp=x;
   while(tp!=1) //是合数
31   {
       int minp=res[tp];
33       if((x-1)%(minp-1)) return false;
       tp=tp/minp;
35       if(tp%minp==0) return false; //p指数>1
   }
37   return true;

```

```

}
39 vector<int> carmichael;
   int main()
41 {
       int tot;
43     getprime(1e7,tot);
       for(int i=2;i<=1e7;++i) if(check(i)) carmichael.push_back(i);
45     for(auto k:carmichael) cout<<k<<endl;
       cout<<carmichael.size()<<"个"<<endl;
47     return 0;
   }

```

code03/carmichael.cpp

1994 年, W.R.Alford 等人证明了卡米歇尔数有无穷多个。

卡米歇尔数的存在, 使得我们需要一个更好的检验合数的办法。

3.6.2 Miller_Rabin 测试

合数的 miller_rabin 测试是基于以下事实的:

定理 3.9. 素数的一个性质

设 p 是奇素数, 记 $p-1 = 2^k q$, q 是奇数, 设 a 是不被 p 整除的任何数, 则下述两个条件之一一定成立: (但满足条件之一的数不一定是素数, 卡米歇尔数)

- a^q 模 p 余 1
- 数 $a^q, a^{2q}, a^{2^2q}, \dots, a^{2^{k-1}q}$ 之一模 p 余 -1

证明 费马小定理告诉我们 $a^{p-1} \equiv 1 \pmod{p}$ 。这意味着对于数表 $a^q, a^{2q}, a^{2^2q}, \dots, a^{2^{k-1}q}, a^{2^kq}$, 最后一个数模 p 余 1, 且表中的每个数是前一个数的平方。因此下述两种可能之一必成立:

- 表中的第一个数模 p 余 1
- 表中的一些数模 p 不余 1, 但是, 当平方时它就模 p 余 1, 所以该数是 $-1 \pmod{p}$, 即表包含 $-1 \pmod{p}$

证毕。

定理 3.10. miller_rabin 测试

设 n 是奇素数, 记 $n-1 = 2^k q$, q 是奇数, 对不被 n 整除的某个 a , 如果下述两个条件都成立, 则 n 一定是合数: (但有不成立的也不一定是素数, 于是可以做多次...)

- $a^q \not\equiv 1 \pmod{n}$
- 对所有的 $i = 0, 1, 2, \dots, k-1$, $a^{2^i q} \not\equiv -1 \pmod{n}$

和费马小定理测试相比, miller_rabin 测试不存在“卡米歇尔型数”, 因为可以保证, 如果 n 是奇合数, 则 $1 \sim n-1$ 之间至少有约 75% 的数可作为 miller_rabin 的证据。即这些数作为 a 时, 可以说明其合数性。

换句话说, 每个合数都有许多证据来说明它的合数性。

例如随机选取 100 个 a 的值，若实验发现其中均没有 n 的 miller_rabin 证据 (两个条件一直都有不成立的)，则 n 是合数的概率小于 0.25^{100} 。

时间复杂度： $O(T * \log n * \log n)$ ，其中 T 为测试次数，(如果 $n > 10^9$ ，需要使用快速乘，认为是两个 \log)。

```

typedef long long ll;
2 //ll mod_mul(ll a,ll b,ll c){快速乘
//    ll res=0;
4 //    a=a%c;
//    while(b)
6 //    {
//        if(b&1) res=(res+a)%c;
8 //        b>>=1;
//        a=(a+a)%c;
10 //    }
//    return res;
12 //}
ll mod_mul(ll a, ll b, ll c){或者int128
14     return a*b%c;
}
16 ll fast_exp(ll a,ll b,ll c){
    ll res=1;
18     a=a%c;
    while(b)
20     {
        if(b&1) res=mod_mul(res,a,c);
22         b>>=1;
        a=mod_mul(a,a,c);
24     }
    return res;
26 }
bool test(ll n,ll a)//false表示为合数
28 {
    ll d = n-1;
30     if(!(n&1)) return false;
    while(!(d&1)) d>>=1;//将d分解为奇数 至少有一个2因子，所以d!=n-1
32     ll t=fast_exp(a,d,n);
    while(d!=n-1 && t!=1 && t!=n-1){
34         t=mod_mul(t,t,n);//平方 使用快速乘 因为t,n 1e18 或者int128
        d<<=1;
36     }
    return ((t==n-1) || (d&1) ==1 );//两个条件都不成立则一定是合数；第二个条件d为
    奇数 即t一开始为1 (mod n)

```



```

38     //若两个有一个成立，且多次，对于合数来说，可能性极小，所以可以认为是素数
    }
40 bool is_prime(ll n){
    if(n<2) return false;
42     if(n==2) return true;
    srand(time(0));
44     int test_num = 10;
    for(int i=0;i<test_num;++i) { //test test_num times
46         ll tpa=1LL*rand()*rand()%(n-2)+2; //取 [2,n-1] 随机数
        if(!test(n,tpa)) return false; //找到证据说明是合数
48     }
    return true; //如果上面所有测试都是true
50 }

```

code03/miller.cpp

3.7 Pollard_Rho 质因数分解

Pollard_Rho 算法是一种用于质因数分解的算法，对于一个待分解的数 N ，假设 N 的最小的质因数为 $p(p \neq N)$ ，那么 Pollard_Rho 算法能够在 $O(\sqrt{p} * \alpha(N))$ 的期望时间复杂度内将 N 分解为两个不是 1 的数的乘积，其中 $\alpha(N)$ 是求解两个数的 gcd 的时间复杂度，而且其不需要额外的空间！下面我们就来看一下它是怎么工作的。

我们之前质因数分解的方法就是去拿 $1 \sim \sqrt{N}$ 的所有数去试除，那如果 N 是 10^{18} ，就顶不住了。其实，可以随机去做这件事，如果运气好的话，就可以将一个数分解为两个数相乘 ($\frac{1}{\sqrt{N}}$ 的概率)。有没有更好的方法，使得猜中其因数的概率增大呢？

我们先来考虑这样一个问题：在 $[1, 1000]$ 里随机选择一个数，是 23 的概率是多少？显然是 $\frac{1}{1000}$ ，那如果我们选择两个数 i, j ， $|i - j| = 23$ 的概率是多少？答案大概是 $\frac{1}{500}$ 。这给了我们一点启发，这种“多点采样”的方法，是否能提高选中目标的概率？著名的生日悖论就是这种思想！

假如说班上有 k 个人，如果找到一个人的生日是 2 月 3 日，这个概率比较低。但是如果单纯想找到两个生日相同的人，这个概率就会很高！由高中的数学知识知道， k 个人生日互不相同，其概率为： $p = \frac{365}{365} * \frac{364}{365} * \frac{363}{365} * \dots * \frac{365-k+1}{365}$ ，故生日有重复的现象的概率是 $P(k) = 1 - \prod_{i=1}^k \frac{365-i+1}{365}$ 。当 $P(k) \geq \frac{1}{2}$ 时，解得 k 大概只需要 23。当 k 取到 60 时， $P(k) \approx 0.9999$ 。这可能和我们的直觉不符，因此被称为悖论。

现在回到因数分解的问题上，由于一定有 $\gcd(k, n) | n$ ，如果我们通过一些组合选取适当的 k ，有 $\gcd(k, n) > 1$ ，就找到了一个 n 的因数！这样的 k 的数量还是蛮多的，假设 n 有若干个质因子，则每个质因子的倍数都是可行的（简单一想，至少得有 $O(\sqrt{n})$ 个吧）。于是，我们可以选取一组数 x_1, x_2, \dots, x_n ，若有 $\gcd(|x_i - x_j|, N) > 1$ ，则 $\gcd(|x_i - x_j|, N)$ 是 N 的一个因子。有前人在论文中指出，要使概率接近 1，需要选取的数的个数大约是 $O(N^{\frac{1}{4}})$ 。但是如果要用 $O(n^2)$ 枚举两两计算，还不如直接 \sqrt{N} 暴力。

我们不妨考虑构造一个伪随机数序列，然后取相邻的两项做差来求 gcd ，这样时间复杂度就降了下来。为了生成一串优秀的随机数，Pollard 设计了这样一个函数： $f(x) = (x^2 + c) \bmod N$ ，其中 c 是一个随机的常数。之所以叫伪随机数，是因为这个序列里可能会含有循环，比如取 $c = 7, N = 20, x_1 = 1$ ，序列为 $1, 8, 11, 8, 11, 8, \dots$ 。循环节的产生很自然，在模 N 的意义下，函数的值域为 $0, 1, 2, \dots, N-1$ ，只要有一次数列的值之前出现过，那么序列就会开始循环。如果画出这个序列，并让循环的地方指向之前，其轨迹很像一个希腊字母 ρ 。（算法名字的由来）

为了方便地判断环的存在，可以用快慢指针。

考虑有一个单链表，其中可能有一个环，也就是某个节点的 `next` 指向的是链表中在它之前的节点，这样在链表的尾部会形成环，如图 3.1。现在有两个指针都指向开始节点 A ，然后向右移动，一个一次移动一个节点，另外一个一次移动两个节点，假设相遇在 P 点，那么可以证明一定有 $P - B$ 的长度等于 $A - B$ 。对于上面的伪随机数序列，有 $A - P$ (不包含 P) 为一个完整的循环节。于是我们可以设置两个变量，一个变量每次向后迭代一次，而另一个每次向后迭代计算两次，在这两个变量相等之前，取它们的差用于求 GCD 。

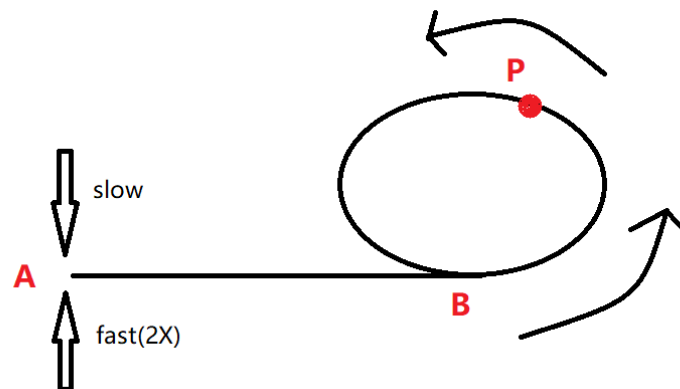


图 3.1: Fast-and-slow-pointer

代码如下：

```

1 int f(int x,int c,int n)
2 {
3     return (x*x+c)%n;
4 }
5 int pollard_rho(int N)
6 {
7     int c=rand()%(N-1)+1;
8     int t=f(0,c,N),r=f(f(0,c,N),c,N); //两倍
9     while(t!=r)
10    {
11        int d=gcd(abs(t-r),N);
12        if(d>1) return d;
    }

```

```

    t=f(t,c,N),r=f(f(r,c,N),c,N);
14 }
    return N;//没有找到,重新调整参数c
16 }

```

code03/pollard-rho.cpp

由于求 GCD 大概会有一个常数,考虑进一步优化。显然如果我们能求得 $gcd(ab, N) > 1$, 那么也是找到了一个因子, 只要 a, b 中某一个有 N 的因子即可。所以我们可以先不急着想做 GCD , 而是做一系列的 $|t-s|$ 的连乘, 到一定次数再做 GCD 。为了不溢出, 我们使连乘的结果 $mod\ N$, 由欧几里得算法知答案不变。

```

#include<bits/stdc++.h>
2 using namespace std;
    typedef __int128 lll;
4 typedef long long ll;
    ll f(lll x, ll c, ll n){ return (x*x+c)%n; }
6 ll gcd(ll a, ll b){
    if(b==0) return a;
8     return gcd(b, a%b);
}
10 ll pollard_rho(ll N){
    if(N<=1) return -1;
12     ll s=0,t=0,c=1LL*rand()*rand()%(N-1)+1;
    int goal, stp;
14     lll val = 1;
    int up = (1<<6)-1;
16     for(goal = 1;;goal<=1,s=t,val=1)//维护区间
    {
18         for(stp=1;stp<=goal;++stp){
            t = f(t,c,N);
20             val = val * abs(t-s) % N;//int 128 或者快读乘
            if(!(stp&up))
22             {
                ll d = gcd(val, N);
24                 if(d>1) return d;
            }
26         }
        ll d = gcd(val, N);
28         if(d>1) return d;
    }
30 }
int main(){
32     cout<<pollard_rho(91)<<endl;

```

```

    return 0;
34 }

```

code03/pollard-rho-with-multi.cpp

pollard-rho 算法往往会和 miller-rabin 同时使用，比如章节题目5.。其要判断一个数是否是素数，若不是则输出其最大质因子。

代码如下，可作为模板，时间复杂度 $O(T * \alpha * (N^{\frac{1}{4}} + \log^2(N)))$ 。其中 T 表示数据组数， α 表示一个不小的常数。

```

#include<bits/stdc++.h>
2 using namespace std;
  typedef __int128 lll;
4 typedef long long ll;
  ll f(lll x, ll c, ll n){ return (x*x+c)%n; }
6 ll gcd(ll a, ll b){
    if(b==0) return a;
8     return gcd(b, a%b);
  }
10 ll mod_mul(lll a, ll b, ll c){return a*b%c;}
  ll fast_exp(ll a,ll b,ll c){
12     ll res=1;a=a%c;
    while(b)
14     {
        if(b&1) res=mod_mul(res,a,c);
16         b>>=1; a=mod_mul(a,a,c);
    }
18     return res;
  }
20 bool test(ll n,ll a)//false表示为合数
  {
22     ll d = n-1;
    if(!(n&1)) return false;
24     while(!(d&1)) d>>=1;//将d分解为奇数 至少有一个2因子，所以d!=n-1
    ll t=fast_exp(a,d,n);
26     while(d!=n-1 && t!=1 && t!=n-1){
        t=mod_mul(t,t,n);//平方 使用快速乘 因为t,n 1e18 或者int128
28         d<<=1;
    }
30     return ((t==n-1) || (d&1) ==1 );//两个条件都不成立则一定是合数；第二个条件d为
    奇数 即t一开始为1 (mod n)
    //若两个有一个成立，且多次，对于合数来说，可能性极小，所以可以认为是素数
32 }

bool is_prime(ll n){

```

```

34     if(n<2) return false;
        if(n==2) return true;
36     srand(time(0));
        int test_num = 10;
38     for(int i=0;i<test_num;++i) { //test test_num times
            ll tpa=1LL*rand()*rand()%(n-2)+2; //取[2,n-1]随机数
40         if(!test(n,tpa)) return false; //找到证据说明是合数
        }
42     return true; //如果上面所有测试都是true
    }
44 ll pollard_rho(ll N){
        assert(N!=1);
46     ll s=0,t=0,c=1LL*rand()*rand()%(N-1)+1;
        int goal, stp;
48     ll val = 1;
        int up = (1<<7)-1;
50     for(goal = 1;;goal<=1,s=t,val=1) //维护区间
        {
52         for(stp=1;stp<=goal;++stp){
            t = f(t,c,N);
54             val = val * abs(t-s) % N; //int 128 或者快读乘
            if(!(stp&up))
56             {
                ll d = gcd(val, N);
58                 if(d>1) return d;
            }
60         }
        ll d = gcd(val, N);
62         if(d>1) return d;
    }
64 }
    set<ll> factors; //所有质因子
66 //ll max_factor; //最大质因子
    void solve(ll x)
68 {
        if(x==1) return ;
70         //if(x<=max_factor) return ; 发现这样剪枝并不快多少 2333
        if(is_prime(x)){
72             //max_factor = max(max_factor, x);
            factors.insert(x);
74             return ;
        }
76     ll y = pollard_rho(x);

```

```

    while(y>=x) y = pollard_rho(x);
88 solve(y),solve(x/y);
}
80 int main(){
    int t;
82 cin>>t;
    while(t--){
84 {
        //max_factor = 0;
86 factors.clear();
        ll n;
88 cin>>n;
        if(is_prime(n)){cout<<"Prime"<<endl;}
90 else{
            solve(n);
92 if(n>1) //cout<<max_factor<<endl;
                cout<<*factors.rbegin()<<endl;
94 }
        }
96 return 0;
}

```

code03/luogu-p4718.cpp

3.8 离散对数

现在我们来思考这样一个问题：

问题 给定 a, b, m ，求 $a^x \equiv b \pmod{m}$ 的解 x 。

解

我们设 $x = A \lceil \sqrt{m} \rceil - B$ ，其中 $0 \leq B < \lceil \sqrt{m} \rceil$ ， $0 < A \leq \lceil \sqrt{m} \rceil + 1$ ，这样化简后的方程是

$$a^{A \lceil \sqrt{m} \rceil} \equiv b \cdot a^B \pmod{m}$$

由于 A 和 B 值域都是 $O(\sqrt{m})$ 级别的，所以可以先计算右边部分的值，存入 *Hash* 表，然后从小到大枚举 A 计算左边的值，在 *Hash* 表中查找。（当然，可以这样做的原因是一定存在 a 的逆元）即只要 $\gcd(a, m) = 1$ ，上面的方法就是有效的。所以当 m 是质数时，用这种方法 (Baby Step Giant Step, bsgs) 即可。

当 m 不是质数时，我们要求解的是 $a^x + km = b$ ，设 $g = \gcd(a, m)$ ，如果 g 不整除 b ，则无解，否则方程两边同除以 g ，得到 $\frac{a}{g}a^{x-1} + k\frac{m}{g} = \frac{b}{g}$ 。这样便消去了 m 的一个因子，得到方程

$$\frac{a}{g}a^{x-1} \equiv \frac{b}{g} \pmod{\frac{m}{g}}$$

令 $m' = \frac{m}{g}$, $b' = \frac{b}{g}(\frac{a}{g})^{-1}$, 得到

$$a^{x'} \equiv b' \pmod{m'}$$

于是可以递归, 得到的解加 1 即 $x = x' + 1$ 为原方程的解。

但是, 进行一次这样的操作, 新的方程不一定可以用 bsgs 求解, 所以会进行多次。

如果中途出现 $b' = 1$ 则 $x' = 0$ 。

时间复杂度 $O(\sqrt{m} \log \sqrt{m})$, 手写二分比 unordered_map 会快一点。

```

1 //SPOJ - MOD
  //sqrt(m)*常数
3 //用二分常数比map小一点
  ///a^x = b (mod m) solve x    1<=a,b,m<=10^9
5 #include<bits/stdc++.h>
  using namespace std;
7 typedef long long ll;
  ll fast_exp(ll a,ll b,ll c){
9     ll res=1;a=a%c;assert(b>=0);
    while(b>0)
11     {
        if(b&1) res=(a*res)%c;
13         b=b>>1;a=(a*a)%c;
    }
15     return res;
  }
17 ll gcd(ll a,ll b){if(b==0) return a;return gcd(b,a%b);}
  struct pli{
19     ll first;
    int second;
21     pli(){}
    pli (ll x_,int y_){
23         first=x_;
        second=y_;
25     }
    bool operator < (const pli &b) const {
27         if(first==b.first) return second>b.second;//因为second更大的解更小, 所以
        >
        return first<b.first;
29     }
  };
31 ll bsgs(ll a,ll b,ll m)
  {
33     a%=m,b%=m;
    if(b==1) return 0;

```

```

35     int cnt=0;
        ll t=1;
37     for(ll g=gcd(a,m);g!=1;g=gcd(a,m)){
            if(b % g) return -1;//no solution
39         m/=g , b/=g , t = t * (a / g) % m;//记录下要算逆元的值
            ++cnt;
41         if(b==t) return cnt;
        }
43     int M=int(sqrt(m)+1);//这里的m不等于形式参数中的值了
        ll base=b;
45     //     std::unordered_map<ll,int> hash;
        //     for(int i=0;i!=M;++i){
47         //         hash[base]=i;//存的是大的编号，所以可以保证最小解
        //         base=base*a%m;
49     //     }
        //     base=fast_exp(a,M,m);//必要时要用快速乘
51     //     ll now=t;
        //     for(int i=1;i<=M+1;++i){
53         //         now=now*base%m;//这里乘在左边了 相当于右边乘逆元
        //         if(hash.count(now)) return i*M-hash[now]+cnt;
55     //     }

        pli hash[int(1e5)];//注意再大可能会爆内存
57     for(int i=0;i!=M;++i){
            hash[i]=pli(base,i);
59         base=base*a%m;
        }
61     sort(hash,hash+M);//默认以first
        base=fast_exp(a,M,m);
63     ll now=t;
        for(int i=1;i<=M+1;++i){
65         now=now*base%m;
            //注意下面M+10 这样可以保证解最小
67         int id=lower_bound(hash, hash+M, pli(now,M+10))-hash;//默认是first
            assert(id>=0 &&id<=M);
69         if(id!=M && hash[id].first==now) return i*M-hash[id].second+cnt;//减去
            的编号second越大越好
        }
71     return -1;
    }
73 int main()
    {
75     ll a,b,m;
        while(cin>>a>>m>>b,m)

```



```

77 {
    if(m==1) assert(b==0);
79 ll ans=bsgs(a,b,m);
    if(ans==-1) cout<<"No Solution"<<endl;
81 else{
        cout<<ans<<endl;
83        assert(fast_exp(a,ans,m)==b%m);
    }
85 }
    return 0;
87 }

```

code03/bsgs.cpp

注意到代码中有一个技巧，不用把每一步的逆元实际求出来，放到式子左边乘起来就行。查表时，把初值设置为这个数 $*a^{\lceil \sqrt{m} \rceil}$ 即可。

3.9 原根

3.9.1 幂模 p 与原根

如果 a 和 p 互素，费马小定理告诉我们， $a^{p-1} \equiv 1 \pmod{p}$ ，那么这个指数 $p-1$ 是唯一的使得结果为 1 的吗？我们选择一些 a 和 p 来看一下，对于 $a=3, p=7$ ，指数只有为 6 时才取到 1，如表 3.1。

表 3.1: $a=3, p=7$

$3^1 \equiv 3 \pmod{7}$	$3^2 \equiv 2 \pmod{7}$	$3^3 \equiv 6 \pmod{7}$
$3^4 \equiv 4 \pmod{7}$	$3^5 \equiv 5 \pmod{7}$	$3^6 \equiv 1 \pmod{7}$

再列多一点，如表 3.2，从表中可以看出，似乎有这样的性质：

- 对于任何底数，最小指数 e 整除 $p-1$ ；
- 总有一些底数，指数需要到 $p-1$ 。

为了方便，我们定义 a 模 p 的阶指 $e_p(a) = [\min e \text{ s.t. } a^e \equiv 1 \pmod{p}]$ ，其中 a 和 p 互质。另外，规定 $e_p(a) \geq 1$ ，显然 $e_p(a) \leq p-1$ 。

定理 3.11. 次数整除性质

设 a 是不被素数 p 整除的整数，假设 $a^n \equiv 1 \pmod{p}$ ，则次数 $e_p(a)$ 整除 n ，特别地 $e_p(a)$ 总整除 $p-1$ 。

证明 次数 $e_p(a)$ 的定义告诉我们

$$a^{e_p(a)} \equiv 1 \pmod{p}$$

假设 $a^n \equiv 1 \pmod{p}$ ，设 $G = \gcd(e_p(a), n)$ ，并设 (u, v) 是方程 $e_p(a)u - nv = G$ 的正整数解（可知

表 3.2: 不同底数对应的最小指数

$p = 5$	$p = 7$	$p = 11$
$1^1 \equiv 1 \pmod{5}$	$1^1 \equiv 1 \pmod{7}$	$1^1 \equiv 1 \pmod{11}$
$2^4 \equiv 1 \pmod{5}$	$2^3 \equiv 1 \pmod{7}$	$2^{10} \equiv 1 \pmod{11}$
$3^4 \equiv 1 \pmod{5}$	$3^6 \equiv 1 \pmod{7}$	$3^5 \equiv 1 \pmod{11}$
$4^2 \equiv 1 \pmod{5}$	$4^3 \equiv 1 \pmod{7}$	$4^5 \equiv 1 \pmod{11}$
	$5^6 \equiv 1 \pmod{7}$	$5^5 \equiv 1 \pmod{11}$
	$6^2 \equiv 1 \pmod{7}$	$6^{10} \equiv 1 \pmod{11}$
		$7^{10} \equiv 1 \pmod{11}$
		$8^{10} \equiv 1 \pmod{11}$
		$9^5 \equiv 1 \pmod{11}$
		$10^2 \equiv 1 \pmod{11}$

一定有解)。现在有两种不同的方法计算 $a^{e_p(a)u}$:

$$a^{e_p(a)u} = (a^{e_p(a)})^u \equiv 1 \pmod{p}$$

$$a^{e_p(a)u} = a^{nv+G} \equiv a^G \pmod{p}$$

这表明 $a^G \equiv 1 \pmod{p}$, 所以必有 $e_p(a) \leq G$ 。

另一方面 $G \mid e_p(a)$, 所以 $G = e_p(a)$, $e_p(a) \mid n$, 证毕。

现在我们的一个猜想得到了证明, 来看另外一个: $e_p(a) = p - 1$ 的底数 a 有什么规律。

如果 a 是这样的数, 则幂

$$a^1, a^2, a^3, \dots, a^{p-2}, a^{p-1} \pmod{p}$$

必须都是模 p 不同的。[如果幂不是全不相同, 则对某指数 $1 \leq i < j \leq p - 1$ 有 $a^i \equiv a^j \pmod{p}$, 由于 a, p 互质, 则有 $1 \equiv a^{j-i} \pmod{p}$, 由于 $j - i < p - 1$, 则矛盾]。

这 $p - 1$ 个不同的数取遍了 $[1, p - 1]$ 。

这样的数很重要, 为了方便, 我们称这样的数 g 为模数 p 的原根, 即 $e_p(g) = p - 1$ 。

回顾前面的那张表 3.2, 5 的原根是 2, 3; 7 的原根是 3, 5; 11 的原根是 2, 6, 7, 8。可见原根可以不止一个, 那到底有多少个? 所有素数都有原根吗?

定理 3.12. 原根定理

每个素数 p 都有原根。更精确地, 有恰好 $\varphi(p - 1)$ 个原根。



神奇! 尽管原根定理没有指出求 p 的原根的具体方法, 但一旦能求得一个原根, 就可以容易得求出其他原根(后面介绍)。求原根的常用方法是从小到大枚举正整数 $a=2, 3, 5, 6, \dots$ 。因为原根的分布比较广。(思考为什么没有 4)。为啥没 4? 因为 $4^{\frac{p-1}{2}} \equiv 2^{p-1} \equiv 1 \pmod{p}$, 即 4 一定不是原根。(可见 2 的高次幂都不是)

证明 原根定理的证明。

对 $1 \sim p - 1$ 之间的每个数 a , $e_p(a) \mid (p - 1)$, 所以, 对整数 $p - 1$ 的每个因子 d , 我们可能会问, 有多少个 $e_p(a)$ 等于 d , 由于要用, 我们记这个数为 $\psi(d)$, $1 \leq a < p$ 。 $\psi(p - 1)$ 即为模数 p

的原根个数。

设 n 是整除 $p-1$ 的任何数, 比如说 $p-1 = nk$, 则可将多项式 $X^{p-1} - 1$ 分解成 $X^{nk} - 1 = (X^n)^k - 1 = (X^n - 1)((X^n)^{k-1} + (X^n)^{k-2} + \dots + (X^n)^2 + X^n + 1)$

我们数一下这些多项式模 p 有多少个根。

首先, $X^{p-1} - 1 \equiv 0 \pmod{p}$ 恰好有 $p-1$ 个解, 即 $[1, p-1]$ 这些。

另一方面, $(X^n - 1) \equiv 0 \pmod{p}$ 至多有 n 个解, 且

$$(X^n)^{k-1} + (X^n)^{k-2} + \dots + (X^n)^2 + X^n + 1 \equiv 0 \pmod{p}$$

至多有 $nk - n$ 个解。

因此我们得到:

$$\underbrace{X^{p-1} - 1}_{\text{模 } p \text{ 恰好有 } p-1=nk \text{ 个根}} = \underbrace{(X^n - 1)}_{\text{模 } p \text{ 至多有 } n \text{ 个根}} \underbrace{\left((X^n)^{k-1} + (X^n)^{k-2} + \dots + X^n + 1 \right)}_{\text{模 } p \text{ 至多有 } nk-n \text{ 个根}}$$

图 3.2: 根的情况

上式要成立, $X^n - 1$ 模 p 恰有 n 个根。这就证明了下述重要事实:

如果 $n \mid p-1$, 则同余式 $X^n - 1 \equiv 0 \pmod{p}$ 恰好有 n 个根满足 $0 \leq X < p$ 。

现在再用另一种计数方法计数这个同余式的解的个数。如果 $X = a$ 是一个解, 则 $a^n \equiv 1 \pmod{p}$, 因此 $e_p(a) \mid n$, 即 $e_p(a)$ 对应 n 的一个因子 d 。从这个角度, $X^n - 1 \equiv 0 \pmod{p}$ 解 ($0 \leq X < p$) 的个数等于:

$$\sum_{d \mid n} \psi(d) = \psi(d_1) + \psi(d_2) + \dots + \psi(d_r)$$

综合上面两个角度, 我们得到一个结论:

$$\text{若 } n \text{ 整除 } p-1, \text{ 则 } \sum_{d \mid n} \psi(d) = n$$

这个公式和欧拉函数的形式一样! 首先, $\varphi(1) = 1$, 而 $\psi(1) = 1$, 下面证明当 $n = q$ 为素数时, $\psi(n) = \varphi(n)$ 。q 的因数是 1 和 q, 所以 $\psi(q) + \psi(1) = q = \varphi(q) + \varphi(1)$, 由于 $\psi(1) = \varphi(1)$, 则 $\psi(q) = \varphi(q)$ 。n = q² 和 n = q₁q₂ 也可以类似证明。更正式地, 可以给出归纳证明, 这里不展示了。证毕。

综上, 我们已证明对整除 $p-1$ 的每个整数 n , 恰好有 $\varphi(n)$ 个底数 a 使得 $e_p(a) = n$, 取 $n = p-1$, 则 $\varphi(p-1)$ 为原根个数。显然, 每个素数至少有一个原根。

求原根, 时间复杂度: 对于单个 p , $O(\sqrt{p} + T * \log n * \log n)$; 对于区间问题, 可以做到 $O(n \log n + \frac{n}{\log n} * T * \log n * \log n)$, 其中 T 表示平均意义下每个素数最小原根。除以 $\log n$ 是因为素数的分布。

```
1 //求一个素数的原根
  #include<bits/stdc++.h>
3 using namespace std;
  typedef long long ll;
5 vector<ll> a;//p-1的所有质因子
```

```
11 fast_pow(ll a,ll b,ll c){
7   ll res=1;
   a=a%c;
9   while(b)
   {
11      if(b&1) res=(res*a)%c;
          b=b>>1;
13      a=(a*a)%c;
   }
15   return res;
}
17 bool g_test(ll g,ll p){
   for(ll i=0;i<a.size();++i)
19      if(fast_pow(g,(p-1)/a[i],p)==1)
          return 0;
21   return 1;
}
23 ll primitive_root(ll p)
{
25   a.clear();
   ll tmp=p-1;
27   for(ll i=2;i<=tmp/i;++i)
       if(tmp%i==0){
29       a.push_back(i);
           while(tmp%i==0) tmp/=i;
31   }
   if(tmp!=1) a.push_back(tmp);
33   long long g=1;
   while(1)
35   {
       if(g_test(g,p)) return g;
37       ++g;
   }
39 }
int main()
41 {
   ios::sync_with_stdio(false);
43   cout<<primitive_root(1e9+9)<<endl;
   return 0;
45 }
```

code03/primitive-root.cpp

3.9.2 原根与指标

我们大概知道原根是啥了，那指标是什么呢？对于模数 13，2 是它的原根，那么 $2^x \bmod 13$ 会取遍 $[1, 12]$ 里所有的数， $x \in [1, 12]$ 。而指标函数 I 就是从余数到指数的双射函数。比如 $2^4 \equiv 3 \pmod{13}$ ，则 $I(3) = 4$ 。

定理 3.13. 指标法则

指标满足下述法则：

- $I(ab) \equiv I(a) + I(b) \pmod{p-1}$ 乘积法则
- $I(a^k) \equiv k * I(a) \pmod{p-1}$ 幂法则



注意 模数是 $p-1$ 。

证明 $g^{I(ab)} \equiv ab \equiv g^{I(a)}g^{I(b)} \equiv g^{I(a)+I(b)} \pmod{p}$

这意味着 $g^{I(ab)-I(a)-I(b)} \equiv 1 \pmod{p}$ ，又 g 是原根，则 $I(ab) - I(a) - I(b)$ 是 $p-1$ 的倍数。所以乘积法则得证。幂法则同理。

利用指标这个工具，可以方便地解一些高次同余方程。

问题 求同余式 $3x^{30} \equiv 4 \pmod{37}$ 。

解 使用乘积法则和幂法则：

$$I(3x^{30}) = I(4)$$

$$I(3) + 30I(x) \equiv I(4) \pmod{36}$$

$$26 + 30I(x) \equiv 2 \pmod{36}$$

$$30I(x) \equiv -24 \equiv 12 \pmod{36}$$

对于最后一个式子，是一个同余方程，由于 $\gcd(30, 36) = 6 \mid 12$ ，则其有解，且有 6 个不同余解。我们求得

$$I(x) \equiv 4, 10, 16, 22, 28, 34 \pmod{36}$$

再查双射表得到对应的值，有 6 个解 16, 25, 9, 21, 12, 28。

可以看到指标法的优点在于将幂运算转为乘法，将乘法转为加法。这一点和对数函数很像：

$$\log(ab) = \log(a) + \log(b)$$

$$\log(a^k) = k \log(a)$$

类似这一题的同余式称为剩余问题，下一章我们就来研究一下。这一章就到这。

第 3 章 习题

1. Strange Way to Express Integers POJ2891 同余方程组，模数不一定互质

2. Exponential 欧拉降幂
3. Problem about GCD 威尔逊定理的应用
4. Zball in Tina Town 威尔逊定理 素性测试
5. 【模板】Pollard-Rho 算法 素性测试 Pollard_Rho 质因数分解
6. MOD - Power Modulo Inverted 离散对数
7. discrete logarithm problem 模数特殊的离散对数问题
8. 对任何正整数 k , 求 $1^k + 2^k + 3^k + \dots + (p-1)^k \pmod p$ 的值。ans:0 solution
9. (2017 四川省赛 K.2017) 给你 n (不超过 200w) 个数, 和一个数 r , 问你有多少种方案, 使得你取出某个子集, 能够让它们的乘积 $\pmod{2017}$ 等于 r 。由于方案数众多, 最后只需要输出答案的奇偶性即可。 原根

第 4 章 剩余

内容提要

- ☐ 二次互反律
- ☐ 二次剩余
- ☐ 勒让德符号
- ☐ N 次剩余

第三章中我们已经知道了如何解线性同余式，现在让我们来考虑更高次的同余方程，首先来看下二次同余方程。

4.1 二次剩余

在发现规律前，人们总要先做一些实验。取模数为 7，取遍底数：

$0^2 \equiv 0(\text{mod}7)$

$1^2 \equiv 1(\text{mod}7)$

$2^2 \equiv 4(\text{mod}7)$

$3^2 \equiv 2(\text{mod}7)$

$4^2 \equiv 2(\text{mod}7)$

$5^2 \equiv 4(\text{mod}7)$

$6^2 \equiv 1(\text{mod}7)$

我们可以看到余数并不会包含所有底数，而且不包含的还挺多。
再取多一点模数，如图4.1。



图 4.1: 二次剩余的模式

可以看到上下的对称性，即数 b 的平方剩余与数 $p-b$ 的平方剩余是模 p 相同的。这

一点也比较好证明：

$$p^2 + b^2 - 2pb = (p - b)^2 \equiv b^2 \pmod{p}$$

因此，若要列出模 p 的所有（非零）平方剩余，只需要计算出其中的一半：

$$1^2 \pmod{p}, 2^2 \pmod{p}, \dots, \left(\frac{p-1}{2}\right)^2 \pmod{p}$$

那如何快速判断一个数是否是二次剩余呢？在探索之前，我们先明确一下定义：

定义 4.1. 二次剩余与二次非剩余

- 与一个平方数模 p 同余的非零数称为模 p 的二次剩余
- 不与任何一个平方数模 p 同余的数称为模 p 的二次非剩余
- 将二次剩余简记为 QR ，二次非剩余简记为 NR
- 与 0 模 p 同余的数既不是 QR ，也不是 NR

定理 4.1. 二次剩余性质

设 p 为一个奇素数，则恰有 $\frac{p-1}{2}$ 个模 p 的二次剩余，且恰有 $\frac{p-1}{2}$ 个模 p 的二次非剩余。

证明 由前面的结论知道，只要证明 $1^2, 2^2, \dots, \left(\frac{p-1}{2}\right)^2 \pmod{p}$ 是两两不同的。

假设 b_1, b_2 是 $[1, \frac{p-1}{2}]$ 之间的数，且满足 $b_1^2 \equiv b_2^2 \pmod{p}$ 。

我们要证明 $b_1 = b_2$ 。

由于 $b_1^2 \equiv b_2^2 \pmod{p}$ ，得到 $p \mid (b_1^2 - b_2^2) = (b_1 - b_2)(b_1 + b_2)$ ，

然而 $b_1 + b_2$ 显然不能被 p 整除，所以 $b_1 - b_2 = 0$ ，证毕。

QR 与 NR 有什么关系呢？一个不难想到的结论是 $QR * QR = QR$ 。（等于号表示仍为 QR ）因为平方数乘平方数仍为平方数，所以两个二次剩余乘积模 p 仍为二次剩余。那么其他的组合呢？经过一些小的表观察，可以得到：

$$QR * QR = QR, \quad QR * NR = NR, \quad NR * NR = QR$$

在验证后面两个关系之前，我们先来看下原根与二次剩余的关系，原根是分析一些问题时好用的工具，也许能帮助我们证明。

设 g 是模 p 的一个原根，那么 g 的幂：

$$g, g^2, g^3, \dots, g^{p-1} \quad (4.1)$$

可以给出 p 的所有非零剩余，即 $[1, p-1]$ 。其中一半是二次剩余，一半是二次非剩余。如何确定哪些是 QR ，哪些是 NR 呢？

显然 g 的每个偶次幂一定是一个 QR ，即 g^{2k} 。

注意到在式子 4.1 中恰有一半是偶次幂，所以 g 的偶次幂给出了所有的二次剩余。而剩下的奇次幂必定是二次非剩余。

同时，也可以用指标来描述，二次剩余是指标 $I(a)$ 为偶数的那些数 a ；二次非剩余

是指标 $I(a)$ 为奇数的那些数 a 。利用二次剩余与二次非剩余的这种指标性质，可以很简单地证明二次剩余的乘法法则。

定理 4.2. 二次剩余乘法法则--表达方式 1

设 p 为素数，则

- 两个模 p 的二次剩余的积是二次剩余
- 二次剩余与二次非剩余的积是二次非剩余
- 两个二次非剩余的积是二次剩余

即

$$QR * QR = QR, \quad QR * NR = NR, \quad NR * NR = QR$$



证明 对与 p ($p > 2$) 互素的任意两个数 a, b ，由指标的乘积法则知 $I(ab) \equiv I(a) + I(b) \pmod{p-1}$ ，从而有 $I(ab) \equiv I(a) + I(b) \pmod{2}$ 。后面的证明就很自然了。可以讨论定理4.2中的三种情况。

对于定理4.2，你肯定会想到 QR, NR 和 $+1, -1$ 的性质类似。许多年前，勒让德 (Adrien-Marie Legendre) 也想到了，而且还引入了一种符号：

定义 4.2. 勒让德符号

a 模 p 的勒让德符号是

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is Quadratic residue} \\ -1 & \text{else} \end{cases}$$



利用勒让德符号，二次剩余的乘法法则可用一个公式表示。

定理 4.3. 二次剩余乘法法则--表达方式 2

设 p 为素数，则

$$\left(\frac{a}{p}\right) \left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right)$$



勒让德符号使计算可以更直观，比如

$$\left(\frac{75}{97}\right) = \left(\frac{3 \cdot 5 \cdot 5}{97}\right) = \left(\frac{3}{97}\right) \left(\frac{5}{97}\right) \left(\frac{5}{97}\right)$$

而

$$\left(\frac{5}{97}\right) \left(\frac{5}{97}\right) = 1$$

所以

$$\left(\frac{75}{97}\right) = \left(\frac{3}{97}\right) = 1 \quad (3 \text{ is } QR)$$

这里能够判断出 3 是模 97 的一个 QR 有些幸运 (10^2)，我们似乎还没有回答如何快速计算一个数是否是二次剩余，即如何快速计算勒让德符号。

4.2 二次互反律

通过前一节的讨论，我们清楚了对于任何一个奇素数， $[1, p-1]$ 有一半是二次剩余。现在先换个角度，考虑对于一个数 a ，看看对于哪些 p ，这个数是 QR 。

我们先令 $a = -1$ ，看对于哪些素数 p ，同余式 $x^2 \equiv -1 \pmod{p}$ 有解。或者说，对哪些素数， $\left(\frac{-1}{p}\right) = 1$ 。同样，通过列出小的数据，可以看出，若 p 与 1 模 4 同余，则 -1 似乎是 p 的 QR ；若 p 与 3 模 4 同余，则 -1 似乎是 NR 。

用来证明这个猜想的工具称为“费马小定理的平方根”，即考虑 $A = a^{\frac{p-1}{2}} \pmod{p}$ 值为多少？

定理 4.4. 欧拉准则

设 p 为素数，则

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}$$

证明 令 $A = a^{\frac{p-1}{2}}$ ，显然 $A^2 \equiv 1 \pmod{p}$ ，因此 p 整除 $(A-1)(A+1)$ 。从而 p 要么整除 $(A-1)$ 要么整除 $(A+1)$ ，因此 A 要么和 1 模 p 同余，要么和 -1 。

当 a 是 QR 时，则 $a \equiv g^{2k} \pmod{p}$ ，则 $a^{\frac{p-1}{2}} \equiv (g^{p-1})^k \equiv 1 \pmod{p}$ ；当 a 是 NR 时，则 $a \equiv g^{2k+1} \pmod{p}$ ，则 $a^{\frac{p-1}{2}} \equiv g^{\frac{p-1}{2}} \pmod{p}$ 。由前面讨论知道， $a^{\frac{p-1}{2}}$ 要么和 1 模 p 同余，要么和 -1 。这里由于 g 是原根，则和 1 模 p 同余的最小次幂只能是 $p-1$ 。所以这里 $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$ 。证毕。

有了欧拉准则，就可以轻松的判断 -1 是不是 p 的二次剩余了：

定理 4.5. 二次互反律—part one

设 p 为素数，若 p 与 1 模 4 同余，则 -1 是 p 的 QR ；若 p 与 3 模 4 同余，则 -1 是 NR 。

用勒让德符号表示如下：

$$\left(\frac{-1}{p}\right) = \begin{cases} 1 & \text{if } p \equiv 1 \pmod{4} \\ -1 & \text{if } p \equiv 3 \pmod{4} \end{cases}$$

证明 带入欧拉准则即证。

下面考虑 $a = 2$ 的情况。如果直接使用欧拉准则，即 $2^{\frac{p-1}{2}} \pmod{p}$ ，似乎不能看出结果是 1 还是 -1 。高斯提出了一个方法，可以简单地知道 $2^{\frac{p-1}{2}} \pmod{p}$ 的值是 -1 还是 1，其结论和二次互反律 part one 一样简单：

定理 4.6. 二次互反律—part two

$$\left(\frac{2}{p}\right) = \begin{cases} 1 & \text{if } p \equiv 1 \text{ or } 7 \pmod{8} \\ -1 & \text{if } p \equiv 3 \text{ or } 5 \pmod{8} \end{cases}$$

证明 利用欧拉准则知，需要寻找 $2^{\frac{p-1}{2}} \pmod{p}$ 结果的模式。 p 是一个素数，令 $P = \frac{p-1}{2}$ ，从偶数

2, 4, 6, ..., $p-1$ 开始, 将它们相乘, 并从每个数中提出 2 因子, 可得

$$2 * 4 * 6 * \cdots * (p-1) = 2^P * P!$$

然后再对 2, 4, 6, ..., $p-1$ 进行模 p 化简, 使其全部落在 $[-P, P]$ 之间, 乘起来。比较这两个乘积, 可得

$$2^P * P! \equiv (-1)^{\text{Number of negative signs}} * P! \pmod{p}$$

负号的个数是指对 2, 4, 6, ..., $p-1$ 进行模 p 化简后落在 $[-P, -1]$ 之间的个数。消去 $P!$, 得

$$2^{\frac{p-1}{2}} \equiv (-1)^{\text{Number of negative signs}} \pmod{p}$$

于是当负数的个数为偶数时, 2 是 p 的二次剩余。而这里负数的个数和 $p \bmod 8$ 相关, 具体如定理中所示。

现在总结一下。对于一个给定的数 a , 我们要确定哪些素数 p 以 a 为二次剩余。上面解决了 $a = -1$ 和 $a = 2$ 时的问题。这个时候我们可以通过查看 $p \% m$ 的一些结果得出 a 是否是 QR 或 NR , 且 m 较小, 为 4 和 8。下面要解决的是其他 a 值的勒让德符号 $\left(\frac{a}{p}\right)$ 的计算问题。(当然, 你可以直接使用欧拉准则去计算, 但下面的方法还是很值得知晓)

例如, 假设要计算 $\left(\frac{70}{p}\right)$, 由前面的二次剩余乘法法则知, $\left(\frac{70}{p}\right) = \left(\frac{2}{p}\right)\left(\frac{5}{p}\right)\left(\frac{7}{p}\right)$, 怎么计算 $\left(\frac{5}{p}\right)$ 和 $\left(\frac{7}{p}\right)$ 呢?

概括来说, 怎么计算 $\left(\frac{q}{p}\right)$ 呢? 其中 q 也为素数。因为由乘法法则知, 素数可以解决后, 整数都可以解决。

通过打表观察后 (此处略去 500 字.....), 可以得到对于一些 p, q , 有 $\left(\frac{q}{p}\right) = \left(\frac{p}{q}\right)$; 但有些不是, 但不是的竟然都是 $\left(\frac{q}{p}\right) = -\left(\frac{p}{q}\right)$ 。这其中有没有模式呢? 有的。

定理 4.7. 二次互反律

$$\left(\frac{-1}{p}\right) = \begin{cases} 1 & \text{if } p \equiv 1 \pmod{4} \\ -1 & \text{if } p \equiv 3 \pmod{4} \end{cases}$$

$$\left(\frac{2}{p}\right) = \begin{cases} 1 & \text{if } p \equiv 1 \text{ or } 7 \pmod{8} \\ -1 & \text{if } p \equiv 3 \text{ or } 5 \pmod{8} \end{cases}$$

$$\left(\frac{q}{p}\right) = \begin{cases} \left(\frac{p}{q}\right) & \text{if } p \equiv 1 \pmod{4} \text{ or } q \equiv 1 \pmod{4} \\ -\left(\frac{p}{q}\right) & \text{if } p \equiv 3 \pmod{4} \text{ and } q \equiv 3 \pmod{4} \end{cases}$$



证明 前两部分已经证明, 最后一个部分感兴趣的读者可以自行查阅资料。

二次互反律不仅很美, 也很实用。它使我们可以翻转勒让德符号 $\left(\frac{q}{p}\right)$, 用 $\pm\left(\frac{p}{q}\right)$ 来替代它, 然后可以模 q 化简 p , 并不断重复此过程, 这样会使 p, q 急剧下降。

所以, 计算勒让德符号的困难之处不是二次互反律的使用, 而是对数字的因式分解。如果不分解, 继续做下去, 这样答案是否正确呢?

正确！也就是说对于 $\left(\frac{a}{p}\right)$ ，之前是 p, q 为素数，现在是对任意的正奇数 a 和 b ，可以给勒让德符号 $\left(\frac{a}{b}\right)$ 指定一个值，反复使用广义二次互反律来计算结果。这种广义勒让德符号常称作雅克比符号。


定理 4.8. 广义二次互反律

设 a, b 为正奇数，则

$$\left(\frac{-1}{b}\right) = \begin{cases} 1 & \text{if } b \equiv 1 \pmod{4} \\ -1 & \text{if } b \equiv 3 \pmod{4} \end{cases}$$

$$\left(\frac{2}{b}\right) = \begin{cases} 1 & \text{if } b \equiv 1 \text{ or } 7 \pmod{8} \\ -1 & \text{if } b \equiv 3 \text{ or } 5 \pmod{8} \end{cases}$$

$$\left(\frac{a}{b}\right) = \begin{cases} \left(\frac{b}{a}\right) & \text{if } a \equiv 1 \pmod{4} \text{ or } b \equiv 1 \pmod{4} \\ -\left(\frac{b}{a}\right) & \text{if } a \equiv 3 \pmod{4} \text{ and } b \equiv 3 \pmod{4} \end{cases}$$

 **注意** 只允许当 a 是正奇数的时候翻转 a, b ，这一点极为重要。当 a 是偶数时，可以分解出因子 2；当 a 是负数时，可以分解出因子 -1 。

使用二次互反律求解勒让德符号，时间复杂度 $O(\log b)$ 。

当然也可以直接用欧拉准则，快速幂即可。

```

1 //poj 1808
  //p>2
3 typedef long long ll;
  ll a,p;//求解勒让德符号(a/p)
5 ll Legendre(ll a,ll p)
  {
7     a=a%p;
    if(a<0) a+=p;
9     int num=0;//将a的2次幂提出 保证是奇数
    while(a%2==0){
11         a>>=1;
        num++;
13     }
    ll ans1,ans2=1;//ans2是2的次幂的答案,默认为1
15     if(a==1) ans1=1;
    else if(a==2){
17         if(p%8==1||p%8==7) ans1=1;
        else ans1=-1;
19     }
    else{
21         if(p%4==3 && a%4==3) ans1=-Legendre(p,a);
        else ans1=Legendre(p,a);
    }
  }

```

```

23     }
    if(num%2 &&(p%8==3||p%8==5)) ans2=-1;
25     return ans1*ans2;
    }
27 int main()
    {
29     int t;
    cin>>t;
31     int T=0;
    while(t-->0)
33     {
        T++;
35     cin>>a>>p;
        cout<<"Scenario #"<<T<<": "<<endl<<Legendre(a,p)<<endl<<endl;
37     }
    return 0;
39 }

```

code04/Legendre.cpp

现在我们可以快速计算勒让德符号了（两种方法）。但知道有解还是不够的，往往我们想知道解是什么。

4.3 求解二次剩余

Cipolla's algorithm 是求解二次剩余的经典方法。

定理 4.9. Cipolla's algorithm

现有同余式 $x^2 \equiv n \pmod{p}$ ，其中 $x, n \in \mathcal{F}_p$ ， p 是奇素数， \mathcal{F}_p 是有 p 个元素的有限域： $\{0, 1, \dots, p-1\}$ 。

1. 寻找一个 $a \in \mathcal{F}_p$ ，使得 $a^2 - n$ 是非二次剩余。（随机寻找即可，期望随机次数为 2）
2. 在域 $\mathcal{F}_{p^2} = \mathcal{F}_p(\sqrt{a^2 - n})$ 中计算 $x = (a + \sqrt{a^2 - n})^{(p+1)/2}$ 即为满足 $x^2 = n$ 的一个解。



在证明前，我们先来看一个例子。注意第二步骤之前所有元素都是在域 \mathcal{F}_{13} 中，第二步中在域 \mathcal{F}_{13^2} 中。

寻找 $x^2 = 10$ 的所有解。

在执行算法前，你可以先用欧拉准则或者二次互反律计算一下 $\left(\frac{10}{13}\right)$ 是否为 1，若不是 1，则无解；若是 1，执行下面算法。

1. 随机寻找一个 $a \in \mathcal{F}_{13}$ ，使得 $a^2 - 10$ 是非二次剩余。假如随机到 $a = 2$ ，则 $a^2 - 10$ 为 7， $\left(\frac{7}{13}\right) = -1$ ，满足要求。

2. 计算 $x = (a + \sqrt{a^2 - n})^{(p+1)/2} = (2 + \sqrt{-6})^7$ 。

$$(2 + \sqrt{-6})^2 = 4 + 4\sqrt{-6} - 6 = -2 + 4\sqrt{-6}$$

$$(2 + \sqrt{-6})^4 = (-2 + 4\sqrt{-6})^2 = -1 - 3\sqrt{-6}$$

$$(2 + \sqrt{-6})^6 = (-2 + 4\sqrt{-6})(-1 - 3\sqrt{-6}) = 9 + 2\sqrt{-6}$$

$$(2 + \sqrt{-6})^7 = (9 + 2\sqrt{-6})(2 + \sqrt{-6}) = 6$$

所以 $x = 6$ 是一个解，当然 $(-6) \bmod 13 = 7$ 也是一个解。

证明 Cipolla's algorithm 正确性的证明。

Part I 证明 $\mathcal{F}_{p^2} = \mathcal{F}_p(\sqrt{a^2 - n}) = \{x + y\sqrt{a^2 - n} : x, y \in \mathcal{F}_p\}$ 确实是一个域。

为了记号方便，令 $\omega = \sqrt{a^2 - n}$ ，由于 $a^2 - n$ 是非二次剩余，所以在 \mathcal{F}_p 中是没有平方根的。这里 ω 可以类比复数域中的 i 。

\mathcal{F}_{p^2} 中的加法被定义为：

$$(x_1 + y_1\omega) + (x_2 + y_2\omega) = (x_1 + x_2) + (y_1 + y_2)\omega$$

乘法被定义为：

$$(x_1 + y_1\omega)(x_2 + y_2\omega) = x_1x_2 + x_1y_2\omega + y_1x_2\omega + y_1y_2\omega^2 = (x_1x_2 + y_1y_2(a^2 - n)) + (x_1y_2 + y_1x_2)\omega$$

可交换、可结合、可分配都比较显然。加法幺元是 $0 + 0\omega$ ，乘法幺元是 $1 + 0\omega$ 。下面证明加法和乘法逆元的存在。显然 $x + y\omega$ 的加法逆元是 $-x - y\omega$ 。对于乘法，记 $\alpha = x_1 + y_1\omega$ ， $\alpha^{-1} = x_2 + y_2\omega$ ，则有：

$$(x_1 + y_1\omega)(x_2 + y_2\omega) = (x_1x_2 + y_1y_2(a^2 - n)) + (x_1y_2 + y_1x_2)\omega = 1$$

通过对应系数可得两个方程：

$$\begin{cases} x_1x_2 + y_1y_2(a^2 - n) = 1 \\ x_1y_2 + y_1x_2 = 0 \end{cases}$$

解得：

$$\begin{aligned} x_2 &= -y_1^{-1}x_1(y_1(a^2 - n) - x_1^2y_1^{-1})^{-1} \\ y_2 &= (y_1(a^2 - n) - x_1^2y_1^{-1})^{-1} \end{aligned}$$

也就是说 x_2, y_2 确实存在，因为其表达式中的元素均在 \mathcal{F}_p 中。

Part I 证毕， \mathcal{F}_{p^2} 确实是一个域。

Part II 证明对于域中的每个元素，有 $x + y\omega \in \mathcal{F}_{p^2} : (x + y\omega)^p = x - y\omega$ 。

首先在模 p 意义下有 $(a + b)^p = a^p + b^p$ ， $x^p = x$ ， $\omega^{p-1} = (\omega^2)^{\frac{p-1}{2}} = -1$ ($\omega^2 = a^2 - n$ 是非二次剩余，欧拉准则)。所以

$$(x + y\omega)^p = x^p + y^p\omega^p = x - y\omega$$

Part II 证毕。

Part III 证明若 $x_0 = (a + \omega)^{\frac{p+1}{2}} \in \mathcal{F}_{p^2}$ ，则有 $x_0^2 = n \in \mathcal{F}_p$ 。

$$x_0^2 = (a + \omega)^{p+1} = (a + \omega)(a + \omega)^p = (a + \omega)(a - \omega) = a^2 - \omega^2 = a^2 - (a^2 - n) = n$$

注意上面的计算都发生在域 \mathcal{F}_{p^2} 中, 所以 $x_0 \in \mathcal{F}_{p^2}$ 。由拉格朗日定理, 在任何域上, n 阶多项式最多有 n 个根。在 \mathcal{F}_p 上, $x^2 - n$ 有两个根, 这些根在 \mathcal{F}_{p^2} 上同样也是根。所以在 \mathcal{F}_{p^2} 上 $x^2 - n$ 的两个根 $x_0, -x_0$ 也是 \mathcal{F}_p 上 $x^2 - n$ 的根。

Part III 证毕。

更多资料可以查看https://en.wikipedia.org/wiki/Cipolla%27s_algorithm

时间复杂度: $O(\log p)$, 常数较大, 因为域 \mathcal{F}_{p^2} 中运算取模较多。

模数为素数。

```

1 typedef long long ll;
   #define random(a,b) (rand()%(b-a+1)+a) // [a,b]
3 ll mod;
   ll w2;//a^2-n = omega^2    random find a s.t. a^2-n is non-quad-res
5 ll fast_exp(ll a, ll b, ll c){
   ll res=1;
7   a=a%c;
   while(b)
9   {
       if(b&1) res=(res*a)%c;
11      b>>=1;
       a=(a*a)%c;
13   }
   return res;
15 }
   //p>2
17 //x^2 = n (mod p)
   bool isqr(ll n, ll p)
19 {
       if(fast_exp(n, (p-1)/2, p)!=1) return 0;//欧拉准则
21   return true;
   }
23 struct FP2{//Field p^2
   ll x,y;
25   FP2 operator * (FP2 tmp)
   {
27       FP2 ans;
       ans.x = (this->x * tmp.x %mod + this->y * tmp.y %mod * w2 %mod) %mod;
29       ans.y = (this->x * tmp.y %mod + this->y * tmp.x %mod) %mod;
       return ans;
31   }

```

```

    FP2 operator ^ (ll b)
33 {
    FP2 ans;
35     ans.x = 1;
    ans.y = 0; // 幺元
37     FP2 a = *this;
    while(b)
39     {
        if(b&1) ans = ans*a;
41         b>>=1;
        a = a*a;
43     }
    return ans;
45 }
};
47 //return the smaller solution -1 means no solution
//solve  $x^2 = n \pmod{p}$ 
49 ll solve(ll n, ll p)
{
51     mod = p;
    n=(n%p+p)%p;
53     if(n==0) return 0; //这里考虑了解x=0的情况 注意[1,p-1]中解成对出现 而0只有一个解
    if(p==2) return n%p;
55     if(!isqr(n,p)) return -1; //no solution
    ll x;
57     if(p%4==3) x=fast_exp(n, (p+1)/4, p); //直接求解
    else{//Cipolla's algorithm
59         //random find a s.t.  $a^2-n$  is non-quad-res
        ll a;
61         srand(time(0));
        while(1)
63         {
            a = random(0,p-1);
65             w2 = ((a*a-n)%p+p)%p;
            if(!isqr(w2,p)) break; //have found
67         }
        FP2 ans;
69         ans.x = a;
        ans.y = 1;
71         ans = ans ^ ((p+1)/2);
        assert(ans.y==0); //FP^2 Field -> FP Field
73         x = ans.x;

```



```

    }
75   if(2*x>=p) x = p-x; //取较小解
    return x;
77 }
int main()
79 {
    int t;
81   cin>>t;
    while(t-->0)
83   {
        ll n,p;
85     cin>>n>>p;
        ll ans = solve(n,p);
87     if(ans==-1) cout<<"Hola!"<<endl; //no solution
        else{
89         assert(ans>=0 && ans<p && ans<p-ans);
            if(ans%p == (p-ans)%p) cout<<ans%p<<endl; //解为0时只有1个解
91         else cout<<ans<<" "<<p-ans<<endl;
        }
93     }
    return 0;
95 }

```

code04/quad-res.cpp

如果模数是质数幂呢？

设 $q = p^k$ ，考虑求解 $x^2 \equiv n \pmod{q}$ 。当 $n \equiv 0 \pmod{p^k}$ 时，解为 $x \equiv 0 \pmod{p^{\lceil k/2 \rceil}}$ 。否则可令 $n = p^r a$, $p \nmid a$, $0 \leq r < k$ ，有解时需要 r 为偶数，令 $x = p^{r/2} x'$ ，则 $x'^2 \equiv a \pmod{p^{k-r}}$ 。

所以只要考虑 $x^2 \equiv a \pmod{q}$ ，且 $p \nmid a$ 的情况即可。下面分 $p = 2$ 和 $p > 2$ 两种情况讨论。

1. 模 2 的幂。 $x^2 \equiv a \pmod{q}$, $q = 2^k$, $2 \nmid a$ 。 $q = 4$ 时，有解当且仅当 $a \equiv 1 \pmod{4}$ ，解为 $x \equiv 1, 3 \pmod{4}$ 。

下面考虑 $q \geq 8$ 的情况。由于任意奇数的平方模 8 余 1，于是需有 $a \equiv 1 \pmod{8}$ 。此时， $x^2 \equiv a \pmod{q}$ 恰有 4 个解 $\pm x_k, \pm (q/2 - x_k)$ 。

证明 $q = 8$ 时，4 个解分别为 $x \equiv \pm 1, \pm 3 \pmod{8}$ 。下面进行归纳。设 $x^2 \equiv a \pmod{2^k}$ 的 4 个解为 $\pm x_k, \pm (2^{k-1} - x_k)$ ，则

$$\begin{aligned}
 & x_k^2 - (2^{k-1} - x_k)^2 \\
 &= (2x_k - 2^{k-1}) 2^{k-1} \\
 &\equiv x_k 2^k \pmod{2^{k+1}} \\
 &\equiv 2^k \pmod{2^{k+1}}
 \end{aligned}$$

于是 $x_k, 2^{k-1} - x_k$ 中恰有一个可以作为 x_{k+1} , 使得 $x_{k+1}^2 \equiv a \pmod{2^{k+1}}$ 。又由 $(2^k - x_{k+1})^2 \equiv x_{k+1}^2 \pmod{2^{k+1}}$ 即得另外两个解。证毕。

2. 模奇素数的幂。 $x^2 \equiv a \pmod{q}, q = p^k, p \nmid a$ 。当且仅当 a 是 p 的二次剩余时, 上式有解。有解时恰有两个解 $\pm x_k$ 。

证明 使用归纳法。若 $x_k^2 \equiv a \pmod{p^k}$, 设

$$\begin{aligned} x_{k+1} &= x_k + tp^k \pmod{p^{k+1}}, t = 0, 1, \dots, p-1 \\ x_{k+1}^2 &= x_k^2 + t^2 p^{2k} + 2x_k t p^k \\ &\equiv x_k^2 + 2x_k t p^k \pmod{p^{k+1}} \\ &\equiv a \pmod{p^{k+1}} \end{aligned}$$

于是

$$2x_k t \equiv (a - x_k^2)/p^k \pmod{p}$$

解出 t 即可得到对应的 x_{k+1} 。证毕。

如果模数是合数, 将模数分解后分别计算, 再用中国剩余定理合并。

参考资料: [jcvb](#) 二次剩余相关

4.4 求解 N 次剩余

问题 给定 N, a, p , 求出 $x^N \equiv a \pmod{p}$ 在模 p 意义下的所有解 (其中 p 是素数)。

解 如果能找到原根 g , 则 $\{1, 2, \dots, p-1\}$ 与 $\{g^1, g^2, \dots, g^{p-1}\}$ 之间就建立了双射关系。

令 $g^y = x, g^t = a, x^N \equiv a \pmod{p}$, 则有

$$g^{y*N} \equiv g^t \pmod{p}$$

因为 p 是素数, 所以方程左右都不会为 0。原问题转化为:

$$N * y \equiv t \pmod{p-1}$$

由于 N, p 已知, 则上式为解模线性方程。

而 t 的值, 由 $g^t \equiv a \pmod{p}$, 用解离散对数的方法求出。

输入: $1 \leq a, N < p \leq 10^9, p$ 为素数。

时间复杂度 $O(\sqrt{p} \log(\sqrt{p}))$

```
1 //51Nod - 1038
  //N次剩余 0(\sqrt{p}*大常数)
3 typedef long long ll;
  struct pli{
5     ll first;
    int second;
7     pli(){}
    pli (ll x_,int y_){
```

```

9         first=x_;
          second=y_;
11     }
    bool operator < (const pli &b) const {
13         if(first==b.first) return second>b.second; //因为second更大的解更小, 所以
            重载>
            return first<b.first;
15     }
};
17 //ll mod_mul(ll a,ll b,ll c){//a*b %c 乘法改加法 防止超long long
    //    ll res=0;
19    //    a=a%c;
    //    assert(b>=0);
21    //    while(b)
    //    {
23    //        if(b&1) res=(res+a)%c;
    //        b>>=1;
25    //        a=(a+a)%c;
    //    }
27    //    return res;
    //}
29 ll fast_exp(ll a,ll b,ll c){
    ll res=1;
31    a=a%c;
    assert(b>=0);
33    while(b>0)
    {
35        if(b&1) res=(a*res)%c;
        b=b>>1;
37        a=(a*a)%c;
    }
39    return res;
}
41 vector<ll> a; //p-1的所有质因子
bool g_test(ll g,ll p){
43     for(ll i=0;i<a.size();++i)
        if(fast_exp(g,(p-1)/a[i],p)==1) //非素数时, 要将p-1换为\varphi(p)
45         return 0;
        return 1;
47 }
ll primitive_root(ll p)
49 {
    a.clear();

```

```

51     ll tmp=p-1; //非素数时, 要将p-1换为\varphi(p)
    for(ll i=2; i<=tmp/i; ++i)
53         if(tmp%i==0){
            a.push_back(i);
55             while(tmp%i==0) tmp/=i;
        }
57     if(tmp!=1) a.push_back(tmp);
    long long g=1;
59     while(1)
    {
61         if(g_test(g,p)) return g;
            ++g;
63     }
}
65 ll gcd(ll a, ll b){
    if(b==0) return a;
67     return gcd(b, a%b);
}
69 //a^x \equiv b (mod m)
ll bsgs(ll a, ll b, ll m)
71 {
    a%=m, b%=m;
73     if(b==1) return 0;
    int cnt=0;
75     ll t=1;
    for(ll g=gcd(a,m); g!=1; g=gcd(a,m)){
77         if(b % g) return -1;
            m/=g, b/=g, t = t * a / g % m; //记录下要算逆元的值
79         ++cnt;
            if(b==t) return cnt;
81     }
    //cout<<a<<" "<<b<<" "<<m<<endl;
83     //bsgs
    int M=int(sqrt(m)+1); //这里的m不等于参数中的值了
85     ll base=b;
    //     std::unordered_map<ll, int> hash;
87     //     for(int i=0; i!=M; ++i){
        //         hash[base]=i; //存的是大的编号
89     //         base=base*a%m;
        //     }
91     //     base=fast_exp(a, M, m); //必要时要用快速乘
        //     ll now=t;
93     //     for(int i=1; i<=M+1; ++i){

```

```

//      now=now*base%m;//这里乘在左边了 相当于右边乘逆元
95 //      if(hash.count(now)) return i*M-hash[now]+cnt;
//  }
97  pli hash[int(1e5)];
    for(int i=0;i!=M;++i){
99      hash[i]=pli(base,i);
        base=base*a%m;
101  }
    sort(hash,hash+M);//默认以first
103  base=fast_exp(a,M,m);
    ll now=t;
105  for(int i=1;i<=M+1;++i){
        now=now*base%m;
107      //找大于等于now的值
        //注意下面M+10 这样可以保证解最小,若设为-1,则找不到解了
109      int id=lower_bound(hash,hash+M,pli(now,M+10))-hash;//默认是first
        assert(id>=0 &&id<=M);
111      if(id!=M && hash[id].first==now) return i*M-hash[id].second+cnt;//减去
        的编号second越大越好
    }
113  return -1;
}
115 ll e_gcd(ll a,ll b,ll &x,ll &y)
{
117     if(b==0){
        x=1;
119     y=0;
        return a;
121     }
    ll ans=e_gcd(b,a%b,x,y);
123     ll temp=x;
    x=y;
125     y=temp-a/b*y;
    return ans;
127 }
vector<int> residue(int p,int N,int a){
129     int g=primitive_root(p);
    ll t=bsgs(g,a,p);
131     vector<int> ans;
    if(a==0){
133         ans.push_back(0);
        return ans;
135     }

```

```

    if(t==1) return ans;
137 //解不定方程
    ll A=N,B=p-1,C=t,x,y;
139 ll d=e_gcd(A,B,x,y);
    if(C % d !=0) return ans;
141 x=x*(C/d)%B;
    ll delta=B/d;
143 for(int i=0;i<d;++i){//一共d组解
        x=((x+delta)%B+B)%B;
145 ans.push_back((int)fast_exp(g,x,p));
    }
147 sort(ans.begin(),ans.end());
    //unique的作用是“去掉”容器中相邻元素的重复元素（不要求数组有序），
149 //它会把重复的元素添加到容器末尾（所以数组大小并没有改变），而返回值是去重之后的尾
    地址
    ans.erase(unique(ans.begin(),ans.end()),ans.end());
151 return ans;
}
153 int main()
{
155 ios::sync_with_stdio(false);
    int t,p,N,A;
157 //x^N=A (mod p)
    cin>>t;
159 while(t-->0)
    {
161 cin>>p>>N>>A;
        vector<int> ans=residue(p,N,A);
163 if(ans.size()){
            for(int i=0;i<ans.size()-1;++i){
165 cout<<ans[i]<<" ";
            }
167 cout<<*ans.rbegin()<<endl;
        }
169 else cout<<"No Solution"<<endl;
    }
171 return 0;
}

```

code04/N-res.cpp

如果模数是非素数呢？

模数非素数：时间和素数一样，甚至更低，因为分解质因数处理的时候规模较小，最后中国剩余定理合并。

```

#define Mo 1000007
2 int test,mo,a,b,g,cnt,size,hcnt,re,phi,now,x,y,G;
  int p[20],ans[40000],Ans[40000],tans[40000];
4 int acnt,Acnt,tacnt;
  int prime[5000],head[Mo];
6 bool vis[31760];
  struct hmap{
8     int num,k,next;
  }h[40000];
10 inline int read(){
    int ret=0;char c=getchar();
12    while((c>='9')||(c<='0'))c=getchar();
    while((c>='0')&&(c<='9'))ret=(ret<<1)+(ret<<3)+c-'0',c=getchar();
14    return ret;
  }
16 inline int Pow(int x,int k){
    int ans=1;
18    for(;k;k>>=1,x=1LL*x*x%mo)if(k&1)ans=1LL*ans*x%mo;
    return ans;
20 }
  inline int exgcd(int a,int b,int &x,int &y){
22    if(a==0){y=1;x=0;return b;}
    int tmp=b/a,d=exgcd(b%a,a,y,x);
24    x-=tmp*y;
    return d;
26 }
  inline int find(int x){
28    for(int i=2;;i++)if(i%x){
        bool ok=true;
30        for(int j=1;j<=cnt;j++)if(Pow(i,phi/p[j])==1){ok=false;break;}
        if(ok)return i;
32    }
  }
34 void ins(int k,int x){
    int key=x%Mo;
36    h[++hcnt]=(hmap){x,k,head[key]};
    head[key]=hcnt;
38 }
  inline int query(int x){
40    int key=x%Mo;
    for(int i=head[key];i;i=h[i].next)if(h[i].num==x)return h[i].k;
42    return -1;
  }

```

```

    }
44 void solve(int p){
    if(p%G!=0)return;
46     else{
        p/=G;
48         int tmp=Pow(g,phi/G),w=Pow(g,1LL*x*p%(phi/G));
        for(int i=0;i<G;i++,w=1LL*w*tmp%mo) Ans[++Acnt]=w;
50     }
    }
52 void bsgs(int x){
    size=int(sqrt(phi)+0.0000001);int w=1,gi=Pow(g,phi-size);
54     for(int i=0;i<size;i++){
        ins(i,w);
56         w=1LL*w*g%mo;
    }
58     w=x;
    for(int i=0;i<=phi;i+=size){
60         int tmp=query(w);
        if(tmp!=-1){
62             w=1;hcnt=0;
            for(int j=0;j<size;j++)head[w%Mo]=0,w=1LL*w*g%mo;
64             solve(i+tmp);
            return;
66         }
        w=1LL*w*gi%mo;
68     }
    }
70 void work(int a,int b,int pr,int s){
    mo=1;
72     for(int i=1;i<=s;i++)mo*=pr;b%=mo;
    if(b%pr==0){
74         if(b==0)
            for(int i=1,S=pr;i<=s;i++,S*=pr)if(i*a>=s){
76                 for(int j=0;j<mo/S;j++)Ans[++Acnt]=j*S;
                break;
78             }
        else{
80             int ts=0;while(b%pr==0)b/=pr,ts++;
            if(ts%a==0){
82                 work(a,b,pr,s-ts);int S=1;
                for(int j=1;j<=ts/a;j++)S*=pr;
84                 for(int j=1;j<=Acnt;j++)Ans[j]*=S;
            }
        }
    }
}

```



```

86     }
        return;
88     }
    phi=mo/pr*(pr-1);a%=phi;
90     if((pr==2)&&(s>2)){
        phi/=2;
92         if(b%4==3){
            if(a%2==0)return;
94             g=5;b=mo-b;G=exgcd(a,phi,x,y);if(x<0)x+=phi/G;
            bsgs(b);
96             tacnt=0;
            for(int i=1;i<=Acnt;i++)if(Ans[i]%4==1)tans[++tacnt]=Ans[i];
98             Acnt=tacnt;
            for(int i=1;i<=Acnt;i++)Ans[i]=mo-tans[i];
100         }else{
            g=5;G=exgcd(a,phi,x,y);if(x<0)x+=phi/G;
102             bsgs(b);
            if(a%2==0){
104                 int tmp=Acnt;
                for(int i=1;i<=tmp;i++)Ans[++Acnt]=mo-Ans[i];
106             }
        }
        return;
    }
110     int tmp=pr-1;cnt=0;
    for(int i=1;prime[i]*prime[i]<=tmp;i++)if(tmp%prime[i]==0){
112         p[++cnt]=prime[i];
        while(tmp%prime[i]==0)tmp/=prime[i];
114     }
    if(tmp>1)p[++cnt]=tmp;
116     if(s>2)p[++cnt]=pr;
    g=find(pr);G=exgcd(a,phi,x,y);if(x<0)x+=phi/G;
118     bsgs(b);
}
120 void merge(int Pow){
    tacnt=0;
122     exgcd(now,Pow,x,y);
    for(int i=1;i<=acnt;i++)
124         for(int j=1;j<=Acnt;j++)tans[++tacnt]=(1LL*x*(Ans[j]-ans[i])%Pow+Pow)%
        Pow*now+ans[i];
    acnt=tacnt;for(int i=1;i<=tacnt;i++)ans[i]=tans[i];now*=Pow;
126 }
int main(){

```

```

128     for(int i=2;i<=31700;i++){
            if(!vis[i])prime[++cnt]=i;
130     for(int j=1;(j<=cnt)&&(prime[j]*i<=31700);j++){
            int k=prime[j]*i;
132     vis[k]=true;
            if(i%prime[j]==0)break;
134     }
    }
136     test=read();
    while(test--){
138         mo=read();a=read();b=read();size=int(sqrt(mo)+0.0000001);
        //x^a = b(mod mo)
140         now=1;acnt=1;ans[1]=0;
        int tmp=mo;
142         for(int i=1;(prime[i]*prime[i]<=tmp)&&(acnt);i++)if(tmp%prime[i]==0){
            int s=0,S=1;
144             while(tmp%prime[i]==0)tmp/=prime[i],s++,S*=prime[i];
            Acnt=0;
146             work(a,b,prime[i],s);merge(S);
        }
148         if((tmp>1)&&(acnt))Acnt=0,work(a,b,tmp,1),merge(tmp);
        if(acnt){
150             sort(ans+1,ans+1+acnt);
            for(int i=1;i<=acnt;i++)printf("%d ",ans[i]);
152             puts("");
        }else puts("No Solution");
154     }
}

```

code04/N-res-notprime.cpp

例 4.1 已知 $x^{2^{30}+3} \bmod n = c$, 给定 c, n , 其中 n 是两个相邻素数 ($p \in [10^5, 10^9]$) 的乘积。求解 x , 题目保证在模意义下只有一个解。10⁵ 组测试数据。(CCPC2018-Final Mr. Panda and Kakin)

解 注意到这里 $2^{30} + 3$ 和 $\phi(n)$ 一定互质, 所以可以直接求 $2^{30} + 3$ 模 $\phi(n)$ 下的逆元, 最后两边做逆元次方即得答案。

注意要使用 $O(1)$ 快速乘, 否则超时。

```

1 //快速乘
    inline ll mod_mul(ll a,ll b,ll c){return (a*b-(ll)((long double)a*b/c)*c+c)%c;}
3 //拓展欧几里得
    ll e_gcd(ll a,ll b,ll &x,ll &y)
5 {

```

```
    if(b==0){x=1;y=0;return a;}
7   ll ans=e_gcd(b,a%b,x,y);
    ll temp=x; x=y; y=temp-a/b*y;
9   return ans;
}
11 //求逆元 已知gcd(a,mod)=1
    ll inv(ll a,ll mod)
13 {
    ll ans,tmp;
15   e_gcd(a,mod,ans,tmp);
    return (ans+mod)%mod;
17 }
ll fast_exp(ll a,ll b,ll c){
19   ll res=1;
    a=a%c;
21   assert(b>=0);
    while(b>0)
23   {
        if(b&1) res=mod_mul(res,a,c);
25         b=b>>1;
        a=mod_mul(a, a, c);
27     }
    return res;
29 }
int main(void)
31 {
    //freopen("in.txt","r",stdin);
33   int t;
    cin>>t;
35   ll n,c;
    ll zhi = (1<<30)+3;
37   for(int tcase=1;tcase<=t;++tcase){
        cout<<"Case " <<tcase<<": ";
39         cin>>n>>c;
        ll p,q;
41         ll M = sqrt(n)+1e1;
        while(1)
43         {
            if(n%M==0){
45                 p = M;
                q = n / M;
47                 break;
            }
        }
    }
```

```
49         M--;  
        }  
51         ll ni = inv(zhi, (p-1)*(q-1));  
         ll ans = fast_exp(c, ni, n);  
53         cout<<ans<<endl;  
        }  
55     return 0;  
}
```

code04/Kakin.cpp

第 4 章 习题

1. 二次剩余 洛谷 P5491 模板
2. 任意模数 N 次剩余 51nod 1123 模板

第 5 章 积性函数

内容提要

- 因子次幂和函数
- 欧拉函数
- 莫比乌斯函数
- 莫比乌斯反演
- 积性函数与狄利克雷卷积
- 积性函数前缀和
- 杜教筛
- 拓展埃氏筛

5.1 因子次幂和函数

定义 5.1. 因子次幂和函数

定义 t 次幂 σ 函数 $\sigma_t(n) = \sum_{d|n} d^t$

两个特殊情况：

1. $t = 0$, $\sigma_0(n)$ 是因子个数函数，计算方法是质因子分解后指数 +1 相乘；
2. $t = 1$, $\sigma_1(n)$ 是因子和函数，计算方法是：

$$\prod \frac{p_i^{n_i+1} - 1}{p_i - 1}$$

他们都是积性函数。

使用欧拉筛预处理前 maxn 项因子和函数的值 (因子个数函数类似)。时间复杂度 $O(n)$ 。

`help[maxn]` 存每个数最小质因子的指数次方，如 12 存 2^2 ，18 存 2^1 ，32 存 2^5 。


```
11 g[maxn]; // g(n) = \sum_{i|n} i
2 int ans[maxn/10];
   int help[maxn]; // 存每个数最小质因子~指数的值 如12存2^2 18存2^1 32存2^5
4 bool valid[maxn];
   int tot;
6 void get_prime(int n)
   {
8     memset(valid, true, sizeof(valid));
       tot=0;
10    g[1]=1;
       for(int i=2; i<=n; ++i){
12        if(valid[i]){
           ans[++tot]=i;
14           g[i]=i+1;
           help[i]=i;
```

```

16     }
    for(int j=1;j<=tot && ans[j]*i<=n;++j){
18         valid[ans[j]*i]=false;
        if(i%ans[j]==0){
20             help[i*ans[j]]=help[i]*ans[j];
            g[i*ans[j]]=g[i]*ans[j]+g[i/help[i]];
22             break;
        }
24         else{
            help[i*ans[j]]=ans[j];
26             g[i*ans[j]]=g[i]*g[ans[j]];
        }
28     }
    }
30 }

```

code05/Sum-function-of-factors.cpp

 **注意** 对因子和函数 $g(n)$ 再求前缀和可得到函数 $f(n) = \sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor * i$ 。也即 $g(n) = f(n) - f(n-1)$ 。类似的，对因子个数函数再求前缀和可得到函数 $f(n) = \sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor$ 。

因此可以在 $O(\sqrt{n})$ 的时间内求得这类函数 (因子和、因子个数) 的前缀和，这类分块的思想是杜教筛的基础。

5.2 欧拉函数

定义 5.2. 欧拉函数

欧拉函数 $\varphi(n)$ ，表示小于等于 n 的数中与 n 互质的数的数目。

即 $\varphi(n) = \sum_{i=1}^n [(n, i) = 1] \cdot 1$ 。

相关性质：

- $\varphi(1) = 1$ ， $\varphi(x)$ 是偶数 where $x > 2$;
- 若 n 是素数 p 的 k 次幂， $\varphi(n) = p^k - p^{k-1} = (p-1)p^{k-1}$;
- 若 m, n 互质， $\varphi(mn) = \varphi(m)\varphi(n)$;
- 当 n 为奇数时，有 $\varphi(n) = \varphi(2n)$ ；(可以理解为此时 2 是 $2n$ 的最小质因子)
- $\varphi(n) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * \dots * (1 - \frac{1}{p_k})$ ， p_1, p_2, \dots, p_k 是 n 的质因子；
- $n = \sum_{d|n} \varphi(d)$ ，质因数分解乘起来即可证明；
- $\sum_{i=1}^n [(n, i) = 1] \cdot i = \frac{n \cdot \varphi(n) + [n=1]}{2}$;
- $\varphi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$ ；(由 $n = \sum_{d|n} \varphi(d)$ 莫比乌斯反演即得)

几个式子总结：

- $\sum_{d|n} \varphi(d) = n$ ； $\varphi * id = I$

- $\sum_{d|n} \varphi\left(\frac{n}{d}\right) \cdot d = \sum_{gcd(i,n)=1} i$ ($1 \leq i \leq n$) = $[0, n]$ 中任选两个数 a, b , 且 $a * b$ 是 n 的倍数的方案数: $\varphi * I$
- $\sum_{d|n} \sum_{w|d} \varphi\left(\frac{d}{w}\right) \cdot w = n$ 乘以 (n 的因子数目)。 $\varphi * I * id = I * I$

求解欧拉函数单值

若时间卡的比较紧，先筛选下素数。

```

1 //0 sqrt(p) 求解欧拉函数
2 //若时间卡的比较紧 先筛选下素数
11 phi(11 x)
4 {
    if(x==1) return 1;
6     ll res=x;
    for(int i=2;i*i<=x;++i){
8         if(x%i==0){
                res-=res/i;
10                do{
                        x/=i;
12                }while(x%i==0);
        }
14    }
    if(x>1) res-=res/x;
16    return res;
}

```

code05/euler.cpp

欧拉函数线性筛

p 为 N 的素因子，若 $p^2 | N$ ，则 $\varphi(N) = \varphi\left(\frac{N}{p}\right) * p$ ；否则 $\varphi(N) = \varphi\left(\frac{N}{p}\right) * (p - 1)$ 。做素数筛时顺带处理即可。

```

1 //欧拉筛法 O(n)
    const int maxn=1e7+10;
3 bool valid[maxn];
    int phi[maxn];
5 int ans[maxn];
    void getprime(int n,int &tot,int ans[])
7 {
        tot=0;
9        phi[1]=1;
        memset(valid,true,sizeof(valid));
11       for(int i=2;i<=n;++i){
                if(valid[i]){

```

```

13         tot++;
           ans[tot]=i;
15         phi[i]=i-1;//i为素数
           }
17         //下面的主角是小于等于i的每个质数
           for(int j=1;(j<=tot) && (i*ans[j]<=n);++j){
19             valid[i*ans[j]]=false;
               if(i%ans[j]==0){//ans[j]是i的素因子
21                 phi[i*ans[j]]=phi[i]*ans[j];
                   break;//如果整除就break;
23             }
               else phi[i*ans[j]]=phi[i]*(ans[j]-1);
25         }
           }
27 }
int main()
29 {
    int Count;//素数个数
31     getprime(1e7,Count,ans);
        cout<<Count<<endl;
33     return 0;
}

```

code05/euler-linear.cpp

下面来看一些例题。

例 5.1 求 $\sum gcd(i,N) (1 \leq i \leq N)$ 。 [HYSBZ - 2705]

解 枚举哪些数 d 作为 i 和 N 的 gcd ，则 $d = gcd(i,N)$ ，即 $1 = gcd(i/d, N/d)$ 。记答案为 $f(N)$ ，有 $f(N) = \sum_{d|N} \varphi(\frac{N}{d}) * d$ 为答案。

同时 $f(n)$ 还表示 $[0,n]$ 中任选两个数 a,b ，且 $a \cdot b$ 是 n 的倍数的方案数。(下面例 2)

由于是积性函数，素数次幂欧拉值可以 $O(1)$ 得到，只剩下质因子分解的时间。总时间复杂度 $O(\sqrt{n})$ 。

例 5.2 定义 $f(n) =$ 选两个 $[0,n]$ 的整数 a,b ，且 ab 不是 n 的倍数的方案数。求 $g(n) = \sum_{d|n} f(d)$ 。数据组数 $1 \leq T \leq 20000$, $1 \leq n \leq 10^9$ 。 [HDU - 5528] icpc2015 长春 B

解 我们考虑 $f(n)$ 怎么计算， $f(n)$ 等于 n^2 减去 ab 是 n 的倍数的方案数。

令 $h(n)$ 表示 ab 是 n 的倍数的方案数，则 $h(n) = \sum_{d|n} \varphi(\frac{n}{d}) \cdot d$ 。如何理解呢？我们去枚举 $gcd(a,n)$ 的值，可知这个值会等于 n 的某个因子，因此我们枚举 n 的因子，则当

$\gcd(a, n) = d$ 时, 即 $\gcd(\frac{a}{d}, \frac{n}{d}) = 1$, a 有 $\varphi(\frac{N}{d})$ 种取值, 由于 a 只有 N 的因子 d , 则 b 要包含 N/d 这个因子, 那么有多少 b 满足呢? 即 $N/N/d = d$ 个。因此 $h(n) = \sum_{d|n} \varphi(\frac{n}{d}) \cdot d$ 。

所以所求为 $g(n) = \sum_{d|n} f(d) = \sum_{d|n} d^2 - \sum_{d|n} h(d)$ 。

对于第二项, 质因子分解后, 则 $\sum_{d|p^k} h(d) = \sum_{d|p^k} \sum_{d'|d} \varphi(\frac{d}{d'}) d'$ 。

对于 p^k 的因子只有 $k+1$ 个, 即 \log 级别, 因此总共只有质因子分解的时间。

总时间复杂度 $O(T * (\frac{\sqrt{n}}{\log(n)} + \log^2 n))$, 注意这里质因子分解时用素数去筛 (先预处理出素数), 否则 TLE 。

本题还有一种思路: 我们知道 $\sum_{d|n} \varphi(d) = n$, 其卷积形式为 $\varphi * id = id * \varphi = I$, 其中 I 为恒等函数, id 为单位函数。本题我们主要求解的是 $\sum_{d|n} h(n) = \sum_{d|n} \sum_{w|d} \varphi(\frac{d}{w}) \cdot w = \sum_{d|n} (\varphi * I)(d) = \varphi * I * id = (\varphi * id) * I = I * I = \sum_{d|n} I(d) \cdot I(\frac{n}{d}) = \sum_{d|n} d \cdot \frac{n}{d} = \sum_{d|n} n = n \cdot \sum_{d|n} 1 = n$ 乘以 (n 的因子数目)。



注意 这里卷积操作 $*$ 指的是狄利克雷卷积, 后面会介绍。

例 5.3 给出 T 组 N, M , 求出 $\sum_{i=1}^N \sum_{j=1}^M \gcd(i, j)$ 的值。 $N, M \leq 1e6, T \leq 1e3$ 。

解 考虑 $n = \sum_{d|n} \varphi(d)$, 则 $\sum_{i=1}^N \sum_{j=1}^M \gcd(i, j) = \sum_{i=1}^N \sum_{j=1}^M \sum_{d|\gcd(i, j)} \varphi(d)$ 。

可以对于每个 d , 去考虑 i, j 的贡献, 则有 $\sum_{i=1}^N \sum_{j=1}^M \sum_{d|\gcd(i, j)} \varphi(d) = \sum_d \varphi(d) * (\sum_{1 \leq i \leq N \text{ and } d|i} \sum_{1 \leq j \leq M \text{ and } d|j} 1) = \sum_d \varphi(d) * \lfloor \frac{N}{d} \rfloor \lfloor \frac{M}{d} \rfloor$ 。

如果直接暴力上式子, 时间复杂度是 $O(\min(N, M))$ 。

由于出现了下取整符号, 考虑进一步优化: 分块块。

$\lfloor \frac{N}{d} \rfloor$ 和 $\lfloor \frac{M}{d} \rfloor$ 在一起分块的区间最多只有 $2 \lfloor \sqrt{n} \rfloor + 2 \lfloor \sqrt{m} \rfloor$ 段。

总时间复杂度 $O(1e6 + T * (\sqrt{n} + \sqrt{m}))$

```

typedef long long ll;
2 const int maxn=1e6+10;
  bool valid[maxn];
4 int ans[maxn];
  ll sum[maxn];
6 int phi[maxn];
  int tot;
8 void get_prime(int n)
{
10     memset(valid, true, sizeof(valid));
    tot=0;
12     phi[1]=1;
    for(int i=2; i<=n; ++i){
14         if(valid[i]){
            tot++;
16             ans[tot]=i;
            phi[i]=i-1;
18         }
    }
}

```

```

    for(int j=1;j<=tot && ans[j]*i<=n;++j){
20         valid[ans[j]*i]=false;
        if(i%ans[j]==0){
22             phi[i*ans[j]]=phi[i]*ans[j];
            break;
24         }
        else phi[i*ans[j]]=phi[i]*(ans[j]-1);
26     }
    }
28     for(int i=1;i<=1e6;++i){
        sum[i]=sum[i-1]+phi[i];
30     }
}
32 ll solve(ll up1,ll up2)
{
34     ll ans=0;
    ll up=min(up1,up2);
36     for(ll l=1,r;l<=up;l=r+1){
        r=min(up1/(up1/l),up2/(up2/l)); //这里取最小的右边界
38         ans+=(sum[r]-sum[l-1])*(up1/l)*(up2/l);
        //cout<<"l r:"<<l<<" "<<r<<endl;
40     }
    return ans;
42 }
int main()
44 {
    get_prime(1e6);
46     //cout<<phi[6]<<endl;
    //cout<<tot<<endl;
48     ll n,m;
    cin>>n>>m;
50     //cout<<n<<" "<<m<<endl;
    cout<<solve(n,m)<<endl;
52     return 0;
}

```

code05/euler-example3.cpp

5.3 莫比乌斯函数

莫比乌斯函数 $\mu(n)$ 是做莫比乌斯反演的时候一个很重要的系数。



定义 5.3. 莫比乌斯函数

$$\mu(x) = \begin{cases} 1, & x = 1 \\ (-1)^k, & x = p_1 p_2 \cdots p_k \\ 0, & x = \text{others} \end{cases}$$


相关性质:

- $[n = 1] = \sum_{d|n} \mu(d)$, 将 $\mu(d)$ 看作是容斥的系数即可证明。也可以记作 $e = \mu * id$, 其中 e 为元函数, 即 μ 在狄利克雷卷积的乘法中与单位函数 id 互为逆元。

证明 设 $n^* = p_1 \cdots p_k$ 可知只有当 $d|n^*$ 时, μ 值有贡献, 所以 $n > 1$ 时有:

$$\sum_{d|n} \mu(d) = \sum_{d|n^*} \mu(d) = \sum_{i=0}^k C_k^i (-1)^i = (1-1)^k = 0$$

- $\varphi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$, 也写作 $\sum_{d|n} \frac{\mu(d)}{d} = \frac{\phi(n)}{n}$, 由 $n = \sum_{d|n} \varphi(d)$ 莫比乌斯反演即得。

 **注意** 后面会介绍莫比乌斯反演。

求莫比乌斯函数单值

$O(\sqrt{n}/\log \sqrt{n})$

```

1 short getmob(ll a){
    ll x=a;
3   int cnt=0,now=0;
    for(int j=1;1LL*ans[j]*ans[j]<=x && j<=tot;j++){
5       now=0;
        if(x%ans[j]==0){
7           while(x%ans[j]==0){
                now++;
9               if(now>1) return 0;
                x/=ans[j];
11            }
            cnt++;
13        }
    }
15   if(x!=1) cnt++;
    return (cnt&1)?-1:1;
17 }

```

code05/mobiwusi.cpp

莫比乌斯函数线性筛 $O(n)$

```

1 const int maxn=1e7+10;

```

```

bool valid[maxn];
3 short mu[maxn];
  int ans[maxn/10];
5 int tot;
void get_prime(int n)
7 {
    tot=0;
9    mu[1]=1;
    memset(valid,true,sizeof(valid));
11   for(int i=2;i<=n;++i){
        if(valid[i]){
13             ans[++tot]=i;
            mu[i]=-1;
15         }
        for(int j=1;j<=tot&&ans[j]*i<=n;j++){
17             valid[i*ans[j]]=false;
            if(i%ans[j]==0){
19                 mu[i*ans[j]]=0;
                    break;
21             }
            else{
23                 mu[i*ans[j]]=-mu[i];
            }
25         }
    }
27 }

```

code05/mobi-linear.cpp

例 5.4 求 $1 \leq x, y \leq N$ 且 $Gcd(x, y)$ 为素数的数对 (x, y) 有多少对。 $N \leq 1e7$ 。 (bzoj2818)

解

1. 方法一。

枚举小于 N 的素数，由于 x, y 可以交换，不妨设 $x < y$ ，则求 $gcd(x, y) = p$ 等价于 $gcd(x/p, y/p) = 1$ 。枚举素数，对于一个素数，再枚举 $y = p, 2p, \dots, kp$ ， x 的方案数即求一个欧拉函数的前缀和。于是先预处理出欧拉函数前缀。

时间复杂度：预处理 $O(1e7)$ ，每个询问是 $O(N/\log N)$ 。

2. 方法二。

枚举素数 p ，令 $n = \lfloor \frac{N}{p} \rfloor$ ，则求

$$\sum_{i=1}^n \sum_{j=1}^n [gcd(i, j) = 1]$$

(其实到这一步直接用欧拉函数就行了, 即法 1, 下面用莫比乌斯函数) 即求

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{d|\gcd(i,j)} \mu(d)$$

即

$$\sum_{d=1}^n \mu(d) * \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} 1$$

即

$$\sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{n}{d} \rfloor$$

由于我们要枚举素数 p , 即这里的 $n = N/p$, 如果我们纯暴力计算上面和式 (先筛出 μ), 时间复杂度是 $O(\sum_{i=1}^N \frac{N}{i}, \text{ where } i \text{ is prime}) = O(N \log N / \log N) \approx O(N)$ 。

这样复杂度比方法一是要差的。因为方法一是 $O(N/\log N)$ 。考虑可不可以进一步优化?

对于一个 n , 上面和式中, $\lfloor \frac{n}{d} \rfloor$ 只有 \sqrt{n} 种取值, 即一段 d 对应的 $\lfloor \frac{n}{d} \rfloor$ 值是相同的, 因此只要有连续一段的 μ 的和值 (预处理前缀和), 即可 $O(\sqrt{n})$ 求解上面和式。

因此时间复杂度是 (对于每个询问): $O(\sum_{i=1}^n \sqrt{\frac{N}{i}}, \text{ where } i \text{ is prime}) = O(N/\log N)$ 。

时间复杂度同方法一。

本题是一道经典的题目, 使用了常见的 φ, μ 两种函数求解。

$O(N/\log N)$ 不是很优秀, 考虑能不能进一步优化。

3. 方法三。 (bzoj2820)

即现在要求 $\sum_p \sum_{d=1}^{\frac{N}{p}} \mu(d) \lfloor \frac{\frac{N}{p}}{d} \rfloor \lfloor \frac{\frac{N}{p}}{d} \rfloor = \sum_p \sum_{d=1}^{\frac{N}{p}} \mu(d) \lfloor \frac{N}{pd} \rfloor \lfloor \frac{N}{pd} \rfloor$ 。

考虑枚举 $D = pd$, 则原式等于 $\sum_D \lfloor \frac{N}{D} \rfloor \lfloor \frac{N}{D} \rfloor \sum_{p|D} \mu(\frac{D}{p})$, where p is prime。

令 $g(n) = \sum_{p|n} \mu(\frac{n}{p})$, where p is prime, 可以预处理出其前缀和, 因此可以分块块。

时间复杂度: $O(1e7) + T * O(\sqrt{N})$ 。

```

1 //O(1e7+T*\sqrt{n})
  typedef long long ll;
3 const int maxn=1e7+10;
  int ans[maxn/10];
5 int mu[maxn];
  ll sum[maxn];
7 bool valid[maxn];
  int g[maxn]; //sum_{p|n} mu(n/p)
9 int tot;
  void get_prime(int n)
11 {
    tot=0;
13    mu[1]=1;
    g[1]=0;
15    memset(valid,true,sizeof(valid));

```

```

17     for(int i=2;i<=n;++i){
18         if(valid[i]){
19             ans[++tot]=i;
20             mu[i]=-1;
21             g[i]=1;
22         }
23         for(int j=1;j<=tot && ans[j]*i<=n;++j){
24             valid[i*ans[j]]=false;
25             if(i%ans[j]==0){
26                 mu[i*ans[j]]=0;
27                 g[i*ans[j]]=mu[i]; //妙啊 展开细算即可 发现只有最小质因子对应的那个非0
28                 break;
29             }
30             else{
31                 mu[i*ans[j]]=-mu[i];
32                 g[i*ans[j]]=mu[i]-g[i]; //妙啊 展开细算即可
33             }
34         }
35     }
36     for(int i=1;i<=n;++i){
37         sum[i]=sum[i-1]+g[i];
38     }
39 ll solve(int up)
40 {
41     ll ans=0;
42     for(int l=1,r;l<=up;l=r+1){
43         r=up/(up/l);
44         // l r
45         ll tp=up/l;
46         ans+=tp*tp*(sum[r]-sum[l-1]);
47     }
48     return ans;
49 }
50
51 int main()
52 {
53     get_prime(1e7);
54     ios::sync_with_stdio(false);
55     int n;
56     while(cin>>n) cout<<solve(n)<<endl;
57     return 0;
58 }

```

code05/mobi-bzoj2820.cpp

例 5.5 给出 T 组 N, M , 求出 $\sum_{i=1}^N \sum_{j=1}^M lcm(i, j)$ 的值。 $N, M \leq 1e7, T \leq 1e3$ 。 [HYSBZ - 2154]

解 如果求 gcd , 直接用欧拉函数经典公式换掉分块即可。

而对于 lcm , 原式等价于 $\sum_{i=1}^N \sum_{j=1}^M \frac{ij}{gcd(i, j)}$ 。

我们考虑枚举 $d = gcd(i, j)$, 则有原式 $= \sum_d \sum_{i=1}^{\lfloor \frac{N}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{M}{d} \rfloor} \frac{di \cdot dj}{d}$, where i, j is co-prime $= \sum_d d \sum_{i=1}^{\lfloor \frac{N}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{M}{d} \rfloor} [gcd(i, j) == 1] ij$ 。

为了方便, 我们定义函数 $f(n, m) = \sum_{i=1}^n \sum_{j=1}^m [gcd(i, j) == 1] ij$, 这个式子很经典, 经常出题。

出现了熟悉的 $[== 1]$, 我们用 μ 换掉, 则得 $f(n, m) = \sum_{i=1}^n \sum_{j=1}^m \sum_{d' | gcd(i, j)} \mu(d') ij$ 。

如果从正面考虑 (枚举 i, j), 复杂度不行, 考虑枚举 d' , 则 $f(n, m) = \sum_{d'} \mu(d') \cdot (d' + 2d' + \dots + \lfloor \frac{n}{d'} \rfloor d') \cdot (d' + 2d' + \dots + \lfloor \frac{m}{d'} \rfloor d') = \sum_{d'} \mu(d') \cdot (\sum_{i=1}^{\lfloor \frac{n}{d'} \rfloor} id' \sum_{j=1}^{\lfloor \frac{m}{d'} \rfloor} jd') = \frac{1}{4} \sum_{d'} \mu(d') d'^2 \cdot \lfloor \frac{n}{d'} \rfloor (\lfloor \frac{n}{d'} \rfloor + 1) \cdot \lfloor \frac{m}{d'} \rfloor (\lfloor \frac{m}{d'} \rfloor + 1)$ 。

有了 $f(n, m)$ 的式子, 原式 $= \frac{1}{4} \sum_d d \cdot \sum_{d'} \mu(d') d'^2 \cdot \lfloor \frac{\lfloor \frac{N}{d} \rfloor}{d'} \rfloor (\lfloor \frac{\lfloor \frac{N}{d} \rfloor}{d'} \rfloor + 1) \cdot \lfloor \frac{\lfloor \frac{M}{d} \rfloor}{d'} \rfloor (\lfloor \frac{\lfloor \frac{M}{d} \rfloor}{d'} \rfloor + 1) = \frac{1}{4} \sum_d d \cdot \sum_{d'} \mu(d') d'^2 \cdot \lfloor \frac{N}{dd'} \rfloor (\lfloor \frac{N}{dd'} \rfloor + 1) \cdot \lfloor \frac{M}{dd'} \rfloor (\lfloor \frac{M}{dd'} \rfloor + 1)$ 。

这个式子直接去做的话, 分两次块块, 时间复杂度是 $O(n^{\frac{3}{4}})$ 。

考虑这里 d 和 d' 的特殊性, 由于 $d \cdot d' \leq \min(N, M)$, 我们可以枚举 $D = d \cdot d'$ 的值, 则原式 $= \frac{1}{4} \sum_D \lfloor \frac{N}{D} \rfloor (\lfloor \frac{N}{D} \rfloor + 1) \cdot \lfloor \frac{M}{D} \rfloor (\lfloor \frac{M}{D} \rfloor + 1) \cdot (D \cdot \sum_{d|D} \mu(d) \cdot d)$ 。

设函数 $g(n) = n \sum_{d|n} \mu(d) d$, 可知 $g(n)$ 是积性函数, 且可以线性时间筛出。

因此只要一次分块块即可, 对于连续的一块, 由维护的 $g(n)$ 的前缀和可 $O(1)$ 得到连续的一段 $g(D)$ 的值。

时间复杂度: 预处理 $O(1e7)$, 每个询问是 $O(\sqrt{N} + \sqrt{M})$ 。

注: $g(n) = n \sum_{d|n} d \mu(d)$ 是一个积性函数, 可以用欧拉筛搞出。注意 μ 很好, 当有大于 2 次幂时贡献为 0 了。

```
1 //先筛g(n)=n* sum_{d|n}d*\mu(d)
   typedef long long ll;
3 const int mod=20101009;
   const int ni4=15075757;
5 const int maxn=1e7+10;
   int ans[maxn];
7 bool valid[maxn];
   ll g[maxn]; //g(n)=n* sum_{d|n}d*\mu(d)
9 ll sum[maxn];
   int tot;
11 void get_prime(int n)
   {
```

```

13     memset(valid,true,sizeof(valid));
    //mu[1]=1;
15     g[1]=1;
    tot=0;
17     for(int i=2;i<=n;++i){
        if(valid[i]){
19             ans[++tot]=i;
            g[i]=1LL*i*(1-i)%mod;
21         }
        for(int j=1;j<=tot && ans[j]*i<=n;++j){
23             valid[i*ans[j]]=false;
            if(i%ans[j]==0){
25                 g[i*ans[j]]=g[i]*ans[j]%mod;
                break;
27             }
            else{
29                 g[i*ans[j]]=g[i]*g[ans[j]]%mod;
            }
31         }
    }
33     for(int i=1;i<=n;++i) sum[i]=sum[i-1]+g[i]; //这里不模了 不会爆long long 下面再模
}
35 ll solve(ll up1,ll up2){
    ll ans=0;
37     ll up=min(up1,up2);
    for(ll l=1,r;l<=up;l=r+1){
39         r=min(up1/(up1/l),up2/(up2/l));
        ll tp1=up1/l;
41         tp1=tp1*(tp1+1)%mod;
        ll tp2=up2/l;
43         tp2=tp2*(tp2+1)%mod;
        ans=(ans+tp1*tp2%mod*((sum[r]-sum[l-1])%mod)%mod)%mod;
45     }
    if(ans<0) ans=ans+mod; //不要忘了
47     return ans*ni4%mod;
}
49 int main()
{
51     ios::sync_with_stdio(false);
    get_prime(1e7+5);
53     ll n,m;
    cin>>n>>m;

```



```

55     cout<<solve(n,m)<<endl;
        return 0;
57 }

```

code05/mobi-HYSBZ2154.cpp

5.4 莫比乌斯反演

定义 5.4. 莫比乌斯反演的一种形式

设 F 是 f 的因子和型函数，即

$$F(n) = \sum_{d|n} f(d)$$

那么对于所有的正整数 n ，都有下面式子成立

$$f(n) = \sum_{d|n} \mu(d) F\left(\frac{n}{d}\right)$$

即已知 $F(n)$ ，我们可以求出 $f(n)$ 。



通过欧拉函数的莫比乌斯反演进一步理解容斥与反演：

我们知道欧拉函数有： $n = \sum_{d|n} \varphi(d)$ ，由莫比乌斯反演可得 $\varphi(n) = \sum_{d|n} \mu(d) \cdot \frac{n}{d}$ 。

我们知道欧拉函数的意义是小于 n 的数中与 n 互质的数的个数，那为啥这个 μ 乘一乘、加一加就能表示欧拉函数了呢？

就是容斥啦，我们考虑 $[1, n]$ 这 n 个数，我们要找的是些数 k ，有 $\gcd(k, n) = 1$ ，那显然可能还会有一些数 k ，有 $\gcd(k, n) > 1$ ，那我们要从 n 个数中，减去 $\gcd > 1$ 的那些数 k ，这些数有多少呢？

考虑将 n 质因子分解得到质因子们 p_1, p_2, \dots, p_r ，显然有 $\frac{n}{p_1}$ 个 p_1 的倍数 $\gcd > 1$ ，同理 $\frac{n}{p_2}, \frac{n}{p_3}, \dots, \frac{n}{p_r}$ ，将这些个数减去；

但会有重复，比如有一个数 $p_1 p_2$ ，在两个地方都减去了，要再加回来，同理所有的两素数都是这样，所以加上 $\frac{n}{p_1 p_2}$ 等等两个素数作为分母；

再考虑有一个数 $p_1 p_2 p_3$ ，考虑一个素数时减去了 3 次，两个时又加上了 3 次，因此没有减去这样的数，所以要减去 $\frac{n}{p_1 p_2 p_3}$ 等等三个素数作为分母；

这个就是容斥原理。

再观察 $\varphi(n) = \sum_{d|n} \mu(d) \cdot \frac{n}{d}$ ，由于 μ 的定义，这里有贡献的 d 恰好取遍了一个素数、两个素数……的容斥情况，而当 $d = 1$ 时， $\mu(1) = 1$ ，即全体个数 n 。

5.5 积性函数与狄利克雷卷积

5.5.1 积性函数定义

数论函数：若 $f(x)$ 的定义域为正整数域，值域为复数域，即 $f: \mathbb{Z}^+ \rightarrow \mathbb{C}$ ，则称 $f(x)$ 为数论函数。

积性函数：若 $f(x)$ 为数论函数， $f(1) = 1$ ，且 $\gcd(m, n) = 1$ 时 $f(x)$ 对所有整数 m 与 n 满足乘法公式 $f(mn) = f(m)f(n)$ ，则称 $f(x)$ 为积性函数。

完全积性函数：若 $f(x)$ 为数论函数，且对任意正整数 m, n ，都有 $f(mn) = f(m)f(n)$ ，则称其为完全积性函数。

5.5.2 常见的积性函数

- 元函数 $e(n) = [n = 1]$ ，狄利克雷卷积中的乘法单元，完全积性
- 单位函数 $id(n) = 1$ ，完全积性
- 恒等函数 $I(n) = n$ ，完全积性
- 幂函数 $id^k(n) = n^k$ ，完全积性
- 欧拉函数 $\varphi(x)$
- 莫比乌斯函数 $\mu(x)$
- t 次幂 σ 函数 $\sigma_t(n) = \sum_{d|n} d^t$
- 因子个数函数 $\sigma_0(n)$
- 因子和函数 $\sigma_1(n)$
- 刘维尔函数 $\lambda(n)$

定义为将 n 分解后， $n = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$ ，令 $\lambda(n) = (-1)^{k_1 + k_2 + \dots + k_r}$ 。 $\lambda(1) = 1$ 。令

$\Omega(n) = k_1 + k_2 + \dots + k_r$ 则 $\lambda(n) = (-1)^{\Omega(n)}$ 。



注意 1. 刘维尔函数的狄利克雷逆变换是莫比乌斯函数的绝对值。

2. 刘维尔函数与莫比乌斯函数存在下面式子：

$$\begin{cases} \sum_{d|n} \lambda(d) \mu\left(\frac{n}{d}\right) = 1, & n = k^2 \\ \sum_{d|n} \lambda(d) \mu\left(\frac{n}{d}\right) = 0, & n \neq k^2 \end{cases}$$

- 使用刘维尔函数定义新的函数 $G(n)$ ，

$$G(n) = \lambda(d_1) + \lambda(d_2) + \dots + \lambda(d_r) = \sum_{d|n} \lambda(d)$$

$G(n)$ 也是积性函数。其满足下面的性质：

$$G(n) = \sum_{d|n} \lambda(d) = \begin{cases} 1 & \text{if } n \text{ is a perfect square,} \\ 0 & \text{otherwise.} \end{cases}$$

5.5.3 不是很常见的积性函数

- $g(n) = n \sum_{d|n} d \mu(d)$ 是积性函数。

可以用欧拉筛搞出。注意 μ 很好，当有大于 2 次幂时贡献为 0 了；(5.3 节中的例题用到了)

- $f(n) = n \sum_{d|n} d \varphi(d) = -n + 2 * \sum_{i=1}^n \text{LCM}(n, i)$ 是积性函数。

其 $f(p^{k+1}) = p * f(p^k) + p^{3(k+1)} - p^{3k+2} = p^3 f(p^k) - p^{k+1}(p-1)$ ，注意有减法，因此可能要维护一下每个数的最小质因子的指数，也可以 $O(n)$ 预处理。(5.5.6 节中的例题用到了)

5.5.4 “作用在因子上”

定义函数 $F(n) = \sum_{d|n} f(d)$, 若 $f(x)$ 是积性函数, 则 $F(x)$ 也是积性函数; 反之, 若 $F(x)$ 为积性函数, 则 $f(x)$ 也是积性函数。

证明 $f(x)$ 是积性函数, 对于 $\gcd(m, n) = 1$, 要证明 $F(mn) = F(m)F(n)$ 。设: d_1, d_2, \dots, d_r 是 n 的因数, e_1, e_2, \dots, e_s 是 m 的因数。

m, n 互质, 则 mn 的因子就是上面 r 个因子和 s 个因子两两相乘得到的, 共 $r * s$ 个。

又 d_i 和 e_j 互质, 则 $f(d_i e_j) = f(d_i) f(e_j)$, 则

$$\begin{aligned} F(mn) &= f(d_1 e_1) + f(d_1 e_2) + \dots + f(d_1 e_s) + \dots + f(d_r) f(e_1) + \dots + f(d_r) f(e_s) \\ &= [f(d_1) + f(d_2) + \dots + f(d_r)] * [f(e_1) + f(e_2) + \dots + f(e_s)] \\ &= F(n) * F(m) \end{aligned}$$

证毕。

反之, 当 $F(n)$ 为积性函数时, 证明类似。

5.5.5 狄利克雷卷积

定义 5.5. 狄利克雷卷积

设 f, g 为两个数论函数, 则满足 $h(n) = \sum_{d|n} f(d)g(\frac{n}{d})$ 的函数称为 f 与 g 的狄利克雷卷积函数。也可以理解为 $h(n) = \sum_{ij=n} f(i)g(j)$ 。

- 两个积性函数的卷积仍为积性函数;
- 狄利克雷卷积满足交换律和结合律, 对加法满足分配律。



注意 多个函数的狄利克雷卷积类似。 $\sum_{ijkl\dots z=n} f_1(i)f_2(j)f_3(k)\dots f_z(z)$ 。

常见的卷积:

- $\varphi * id = I$
- $\mu * id = e$

5.5.6 一些题目

例 5.6 定义 $f_0(n)$ = 满足 $pq = n$ 且 $\gcd(p, q) = 1$ 的二元组 (p, q) 个数。定义 $f_{r+1}(n) = \sum_{uv=n} \frac{f_r(u) + f_r(v)}{2}$ 。若干组询问, 每次给出 r, n , 询问 $f_r(n)$ 的值, 对 $10^9 + 7$ 取模。询问次数 $1 \leq q \leq 10^6$, $0 \leq r \leq 10^6$, $1 \leq n \leq 10^6$ 。 **cf-757E**

解 满足 $pq = n$ 且 $\gcd(p, q) = 1$ 的二元组个数是多少呢?

显然是 $2^{w(n)}$ 个, 其中 $w(n)$ 是 n 的不同的素因子的数目。

那么 $f_{r+1}(n) = \sum_{uv=n} \frac{f_r(u) + f_r(v)}{2}$ 如何计算呢?

$$f_{r+1}(n) = \sum_{uv=n} \frac{f_r(u) + f_r(v)}{2} = \sum_{d|n} f_r(d)。$$

由于 $r \leq 1e6$, 询问又有 $1e6$, 那我们不可能暴力搞。

我们知道 $f_r(n)$ 是积性函数, 所以问题转为求解 $f_r(p^k)$ 。

由于 $f_r(p^k) = \sum_{i=0}^{k-1} f_{r-1}(p^i)$, 且对任意 p , $k \geq 1$ 时, 有 $f_0(p^k) = 2$, 因此 $f_r(p^k)$ 的值和 p 无关。预处理所有的 r, k 组合即可。 k 是 \log 级别的。

时间复杂度: $O(r \log n + T * (\frac{\sqrt{n}}{\log 10^3}))$

```

1 typedef long long ll;
  const int mod=1e9+7;
3 const int maxn=1e6+10;
  ll help[maxn][21]; // r k 给定时 不同p结果相同
5 bool valid[1010];
  int ans[500];
7 int tot;
  void init()
9 {
    help[0][0]=1;
11   for(int k=1;k<=20;++k) help[0][k]=2;
    for(int r=1;r<=int(1e6);++r){
13       help[r][0]=help[r-1][0];
        for(int k=1;k<=20;++k){
15           help[r][k]=(help[r][k-1]+help[r-1][k])%mod;
        }
17   }
  }
19 void get_prime(int n)
  {
21     tot=0;
    memset(valid,true,sizeof(valid));
23     for(int i=2;i<=n;++i){
        if(valid[i]){
25         ans[++tot]=i;
        }
27         for(int j=1;ans[j]*i<=n;++j){
            valid[ans[j]*i]=false;
29             if(i%ans[j]==0){
                break;
31             }
        }
33     }
  }
35 int main()
  {
37     int t;
    scanf("%d",&t);
39     //cout<<clock()<<endl;

```

```

init();
41 get_prime(1010);
    //cout<<tot<<endl;
43 //cout<<clock()<<endl;
    //cout<<help[1][1]<<" "<<help[1][2]<<endl;
45 while(t--){
    {
47     int r,n;
        scanf("%d%d",&r,&n);
49     ll res=1;
        for(int j=1;ans[j]*ans[j]<=n && j<=tot;++j){
51         int i=ans[j];
            if(n%i==0){
53                 int k=0;
                    while(n%i==0){
55                         n/=i;
                            k++;
57                     }
                        //cout<<i<<" "<<k<<endl;
59                         res=(res*help[r][k])%mod;
                    }
61             }
                if(n>1){
63                     res=(res*help[r][1])%mod;
                        //cout<<n<<endl;
65                 }
                    printf("%lld\n",res);
67             }
                return 0;
69     }

```

code05/cf757E.cpp

例 5.7 T 个 N , 依次计算 $\sum_{a=1}^N \sum_{b=1}^N LCM(a,b)$ $N, T \leq 10^6$

解 前面我们解过 $\sum_{a=1}^N \sum_{b=1}^M LCM(a,b)$, 复杂度可以是 $T * O(\sqrt{N} + \sqrt{M})$ 。显然解这题是不够的。

考虑本题, 如果直接令一个函数 $f(N) = \sum_{a=1}^N \sum_{b=1}^N LCM(a,b)$, 其是不是积性函数呢?

可惜不是, 但这个函数 $f(n) = -n + 2 * \sum_{i=1}^n LCM(n,i)$ 是积性函数。

为啥长这样... 其实这样很美, 因为 $\sum_{i=1}^N f(i)$ 即为本题所求。

但 $f(n)$ 目前这个样子还是不知道怎么搞, 我们化简一下。

设 $\gcd(n, i) = d$, 则 $f(n) = -n + 2 \sum_{i=1}^n \frac{ni}{d} = -n + 2n \sum_{d|n} \sum_{i \leq n \text{ \& } \gcd(i, n)=d} \frac{i}{d} = -n + 2n \sum_{d|n} \sum_{k \leq \frac{n}{d} \text{ \& } \gcd(k, \frac{n}{d})=1} k = -n + 2n \sum_{d|n} \sum_{k \leq \frac{n}{d} \text{ \& } \gcd(k, d)=1} k = -n + 2n((\sum_{d|n} \sum_{d>1} \frac{d\varphi(d)}{2}) + 1) = n \sum_{d|n} d\varphi(d)$ 。

即 $f(n) = -n + 2 * \sum_{i=1}^n LCM(n, i) = n \sum_{d|n} d \cdot \varphi(d)$ 是积性函数。

所以, 只要筛出这个积性函数, 就可以 $O(1)$ 回答一个询问。

如何去筛这个函数呢? 考虑一个数 p^k , 若它新出现一个质因子变为 $p^k q$, 则函数值 $f(p^k q) = f(p^k) f(q)$, 若从 p^k 变为 p^{k+1} , 则 $f(p^{k+1}) = p * f(p^k) + p^{3(k+1)} - p^{3k+2} = p^3 f(p^k) - p^{k+1}(p-1)$, 后面的部分是 $p\varphi(p^{k+1})$, 因此欧拉筛搞一搞即可 (因为有减法, 而不仅仅是乘法, 可能要维护一下每个数最小质因数指数幂后的数)。

时间复杂度: $O(1e6 + T)$

5.6 积性函数前缀和

一些式子:

•

$$\left\lceil \frac{n}{ab} \right\rceil = \left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil, \quad \left\lfloor \frac{n}{ab} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor$$

• $\left\lceil \frac{n}{i} \right\rceil$ 只有 $O(\sqrt{n})$ 种取值。

$\left\lceil \frac{n}{i} \right\rceil$ 为什么只有 $O(\sqrt{n})$ 种取值呢? 若 $i < \sqrt{n}$ 则由于 i 只有 \sqrt{n} 种取值, 所以 $\left\lceil \frac{n}{i} \right\rceil$ 只有 \sqrt{n} 种取值; 若 $i > \sqrt{n}$, 则 $\left\lceil \frac{n}{i} \right\rceil < \sqrt{n}$, 所以 $\left\lceil \frac{n}{i} \right\rceil$ 最多也只有 \sqrt{n} 种取值。

一些常用复杂度计算:

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

•

$$\sum_{i=1}^{n^x} \sqrt{\frac{n}{i}} = O(n^{(\frac{1}{2} + \frac{1}{2}x)})$$

例如 $\sum_{i=1}^{\sqrt{n}} \sqrt{\frac{n}{i}} = O(n^{\frac{3}{4}})$, $\sum_{i=1}^n \sqrt{\frac{n}{i}} = O(n)$, $\sum_{i=1}^{n^{\frac{1}{3}}} \sqrt{\frac{n}{i}} = O(n^{\frac{2}{3}})$ 。

前缀和:

给出一个积性函数 $f(x)$, 求 $\sum_{i=1}^n f(i)$ 。

通常, $f(p^k)$ 可以比较容易的由 $f(p^{k-1})$ 等值递推出来, 其他项可以直接由积性函数的性质由 $f(x) = f(d) * f(\frac{x}{d})$ 得到。因此很多积性函数都可以在欧拉筛的过程中顺便递推出前 n 项的值, 时间复杂度 $O(n)$ 。


5.6.1 一些题目

例 5.8 设 $\sigma(n)$ 为因子和函数, 现在要求解 $\sum_{i=1}^n \sigma(i)$ 。

解 一种简单的思考方法, 考虑函数 $f(n) = \sum_{i=1}^n i * \left\lfloor \frac{n}{i} \right\rfloor$ 。

发现 $f(n) - f(n-1) = \sum_{i=1}^n i * (\lfloor \frac{n}{i} \rfloor - \lfloor \frac{n-1}{i} \rfloor) = \sigma(n)$ 。

因此 $f(n)$ 为我们所求，分块即可 $O(\sqrt{n})$ 。

 **注意** 因此我们也可以线性筛出函数 $f(1\dots n) = \sum_{i=1}^n i * \lfloor \frac{n}{i} \rfloor$ 。

例 5.9 线性筛出函数 $f(n) = \sum_{i=1}^n \lceil \frac{n}{i} \rceil$ ，即求 $f(1), f(2), \dots, f(n)$ 。

解 注意下取整和上取整的不同，下取整的差分减一对应下标减一的因子个数函数，即 $f(n) - f(n-1) = \tau(n-1) + 1$ 。

因此先预处理出因子个数函数，再移位 +1，最后求前缀和。

```
1 void init_f(int n)
2 {
3     //f(n)=sum_{i=1}^n \ceil(n/i)
4     // sigma_0后移一位并加1 对应f
5     //处理完之后全部加1并后移一位 再做一次前缀和就是f
6     f[0]=0;
7     for(int i=1;i<=n;++i){
8         f[i]=(sigma_0[i-1]+1);
9     }
10    for(int i=1;i<=n;++i){
11        f[i]=(f[i]+f[i-1])%mod;
12    }
13 }
```

code05/Factor-number-function-prefix.cpp

例 5.10 求 $ans = \sum_{i=1}^n \sum_{j=1}^i (n \bmod(i \times j)) \quad 1 \leq n \leq 10^{11}$ **18 四川省赛 G**

解 解法一

首先写作 $\sum_{i=1}^n \sum_{j=1}^i (n - \lfloor \frac{n}{ij} \rfloor * ij) = \sum_{i=1}^n \sum_{j=1}^i n - \sum_{i=1}^n \sum_{j=1}^i \lfloor \frac{n}{ij} \rfloor * ij$

对于 $\sum_{i=1}^n \sum_{j=1}^i n$ 不用说了，对于 $\sum_{i=1}^n \sum_{j=1}^i \lfloor \frac{n}{ij} \rfloor * ij = \sum_{i=1}^n i * \sum_{j=1}^i \lfloor \frac{\frac{n}{i}}{j} \rfloor * j$

对于第二维，所求是 $j \in [1, i]$ ，我们这里求 $[1, n]$ ，由对称性可知 $[1, i]$ 的两倍减去 $[i = j]$ 的情况即为所求。

所以现在目标是 $\sum_{i=1}^n i * \sum_{j=1}^n \lfloor \frac{\frac{n}{i}}{j} \rfloor * j$ 。

由于 $\lfloor \frac{n}{i} \rfloor$ 只有 $O(\sqrt{n})$ 种取值，我们枚举 $\lfloor \frac{n}{i} \rfloor$ 的值，对应一段 i 区间，可知对于这一段 i ，第二维值是相同的。

而对于第二维的计算相同，都是分块思想。所以写一个 solve 函数用于求解 $f(n) = \sum_{i=1}^n \lfloor \frac{n}{i} \rfloor * i$ 即可。

时间复杂度： $O(\sum_{i=1}^{\sqrt{n}} \sqrt{\frac{n}{i}}) = O(n^{\frac{3}{4}})$ （只有 $O(\sqrt{n})$ 种取值）

这种方法会 TLE，1e11 大概要跑 37s。

解法二（常见套路）

定义 $g(n) = f(n) - f(n-1) = \sum_{i=1}^n i * (\lfloor \frac{n}{i} \rfloor - \lfloor \frac{n-1}{i} \rfloor) = \sum_{d|n} d$ 。

对于 $g(n)$ ，即一个数的因子和，我们可以 $O(n)$ 求解出其前 n 项，对于本题 $n \leq 1e11$ ，我们处理出 g 的前 $n^{\frac{2}{3}}$ 项，然后求个前缀和就得到了 $f(n)$ ，这一部分的时间复杂度是 $O(n^{\frac{2}{3}})$ 。

所以对于 $\sum_{i=1}^n i * \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} j$ ，当 $\lfloor \frac{n}{i} \rfloor < n^{\frac{2}{3}}$ 时，可以 $O(1)$ 得到第二层结果。当 $\lfloor \frac{n}{i} \rfloor > n^{\frac{2}{3}}$ 时，使用原先的方法求解，这一部分的时间复杂度是 $O(\sum_{i=1}^{n^{\frac{1}{3}}} \sqrt{\frac{n}{i}}) = O(n^{\frac{2}{3}})$ 。

时间复杂度： $O(n^{\frac{2}{3}})$ ， $1e11$ 大概要跑 5s

```

1 typedef long long ll;
  typedef __int128 lll;
3 const int mod=1e9+7;
  inline lll cal(lll l, lll r)
5 {
    return (l+r)*(r-l+1)/2;
7 }
  inline lll solve(ll up) //solve \sum_{i=1}^n up/i *i;
9 //显然只有i<=up时有贡献
  {
11 //    num++;
    //    if(num%10000==0) cout<<clock()<<endl;
13    lll res=0;
    for(ll l=1, r; l<=up; l=r+1){
15        r=up/(up/l);
        res=(res+up/l*cal(l, r));
17    }
    return res;
19 }
  inline void write(__int128 x)
21 {
    if(x<0)
23    {
        putchar('-');
25        x=-x;
    }
27    if(x>9) write(x/10);
    putchar(x%10+'0');
29 }
  //lll help1[maxn]; //solve f(n/1) f(n/2) f(n/3) f(n/\sqrt{n})
31 //lll help2[maxn]; //solve 1 2 3 \sqrt{n}
  const int maxn=21550000;
33 ll g[maxn]; //n^(2/3) g(n)=\sum_{i|n} i
  lll f[maxn]; //sum_{i=1}^n [n/i]*i

```



```
35 int ans[maxn/10];
   int help[maxn]; //存每个数最小质因子~指数 如12存2^2 18存2^1 32存2^5
37 bool valid[maxn];
   int tot;
39 void get_prime(int n)
   {
41     memset(valid,true,sizeof(valid));
       tot=0;
43     g[1]=1;help[1]=1;
       for(int i=2;i<=n;++i){
45         if(valid[i]){
               ans[++tot]=i;
47             g[i]=i+1;
               help[i]=i;
49         }
           for(int j=1;j<=tot && ans[j]*i<=n;++j){
51             valid[ans[j]*i]=false;
               if(i%ans[j]==0){
53                 help[i*ans[j]]=help[i]*ans[j];
                   g[i*ans[j]]=g[i]*ans[j]+g[i/help[i]];
55                 break;
               }
57             else{
                   help[i*ans[j]]=ans[j];
59                 g[i*ans[j]]=g[i]*g[ans[j]];
               }
61         }
       }
63 }
   int main()
65 {
       get_prime(maxn);
67       f[0]=0;
       for(int i=1;i<maxn;++i) f[i]=f[i-1]+g[i];
69       //cout<<clock()<<endl;
       //freopen("in.txt","r",stdin);
71       int t;
       cin>>t;
73       while(t--){
           {
75               ll n;
                   cin>>n;
77               lll ans1=0;
```

```

79     for(ll l=1,r;l<=n;l=r+1){
        r=n/(n/l);
        ll tp=n/l;
81         if(tp<maxn) ans1+=f[tp]*cal(l,r);
        else ans1+=solve(tp)*cal(l,r);
83     }
    ll ans2=0; //i=j
85     for(ll i=1;i*i<=n;++i){
        ll tp=i*i;
87         ans2+=n/tp*tp;
    }
89     ans1+=ans2;
    assert(ans1%2==0);
91     ans1/=2;
    ans1=((ll)n)*n*(n+1)/2-ans1;
93     write(ans1);
    cout<<endl;
95 }
    return 0;
97 }

```

code05/18sichuanG.cpp

5.7 杜教筛

设 $S(n) = \sum_{i=1}^n f(i)$ ，下面将求解 $S(n)$ 的过程一般化：

任意数论函数 g ，设 $h = f * g$ (狄利克雷卷积)，有 $\sum_{i=1}^n h(i) = \sum_{i=1}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$ (改变计数方式易得)

从右式分出一个 $i = 1$ ，移项可得 $g(1)S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$

如果我们可以 $O(\sqrt{n})$ 计算 $\sum_{i=1}^n h(i)$ ， $O(1)$ 计算 g 的前缀和，就可以快速把问题递归为同类子问题，时间复杂度为 $O(\sum_{i=1}^{\sqrt{n}} \sqrt{\frac{n}{i}}) = O(n^{\frac{3}{4}})$ (具体计算这里略去)

如果 f 有一些比较好的性质，比如是积性函数，我们可以用欧拉筛求出前 $n^{\frac{2}{3}}$ 项，更后面的项再递归，时间复杂度为 $O(n^{\frac{2}{3}})$ 。

5.7.1 一些要注意的地方

- 会卡 map;
- 一个记忆化的技巧是，由于我们要求解的始终是 $S(n)$ ，因此我们可以开一个数组，比如 $p[]$ ， $p[i]$ 的地方存储 $S(\frac{n}{i})$ 的值，显然由于我们预处理了 $i < n^{\frac{2}{3}}$ 的情况，则 p 数组的大小只有 $n^{\frac{1}{3}}$ 大小；
- 如果出现 $\sum_{i=1}^N \sum_{d|i}$ 的形式要求解，直接对后面杜教筛可能不太好搞，因为找不到合适的狄利克雷卷积或者不好找。我们可以尝试调换贡献的枚举 (这里枚举 i 是 d

的多少倍), 变换为 $\sum_{i=1}^N \sum_{d=1}^{\lfloor \frac{N}{i} \rfloor} f(d)$ 的形式, 对后面一个求 $\sum f$ 的问题做杜教筛, 而前面不影响复杂度 (分 $i < n^{\frac{1}{3}}$ 和 $>$ 考虑)。如:

要求函数 $f(i) = i \sum_{d|i} d\varphi(d)$ 的前缀和,

$$\begin{aligned}\sum_{i=1}^N f(i) &= \sum_{i=1}^N i \sum_{d|i} d\varphi(d) \\ &= \sum_{d=1}^N d\varphi(d) \sum_{k=1}^{\frac{N}{d}} kd \\ &= \sum_{d=1}^N d^2\varphi(d) \sum_{k=1}^{\frac{N}{d}} k \\ &= \sum_{k=1}^N k \sum_{d=1}^{\lfloor \frac{N}{k} \rfloor} \phi(d)d^2 \\ &= \sum_{i=1}^N i \sum_{d=1}^{\lfloor \frac{N}{i} \rfloor} \phi(d)d^2 \quad (\text{replace symbol } k \text{ with } i)\end{aligned}$$

如果 $f(x) = \varphi(x)x^2$, $S(n) = \sum_{i=1}^n f(i)$ 可以做杜教筛, 则上面的式子也行, 不影响时间复杂度 $O(n^{\frac{2}{3}})$ 。

以及下面的第二、三个例子 5.7.2 也是类似的:

- $f(x) = \varphi(x)x^k$ 都是可以搞的, 即 $h(x) = x^{k+1}$, $g(x) = x^k$ 。

5.7.2 一些题目

例 5.11 求 $\sum_{i=1}^n \varphi(i)$ 和 $\sum_{i=1}^n \mu(i)$, $n \leq 1e9$ 多组

解

- 考虑有没有函数 g , 这个函数易于计算前缀和, 且和 φ 狄利克雷卷积后的函数也较易于计算前缀和。对于欧拉函数, 这是显然存在的, 即 $g(n) = id(n)$, $h(n) = n$, $I = \varphi * id$ 。设 $S(n) = \sum_{i=1}^n \varphi(i)$, $g(n) = 1$ 。则 $g(1)S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$ 。带入 g, h 的值后, 有 $S(n) = \sum_{i=1}^n i - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ 。用欧拉筛求出欧拉函数的前 $n^{\frac{2}{3}}$, 求前缀和得到 S 的前 $n^{\frac{2}{3}}$ 项。 $\lfloor \frac{n}{i} \rfloor$ 较大时递归计算, 总时间复杂度 $O(n^{\frac{2}{3}})$ 。
- 对于 $(Mertens) = \sum_{i=1}^n \mu(i)$, 同样的, 我们寻找函数 g , 有 $h = \mu * g$, 其中 g 可以 $O(1)$ 计算前缀和。

我们知道 μ 的一个性质是 $\sum_{d|n} \mu(d) = [n == 1]$, 它的卷积形式是 $e = \mu * id$ 。

因此 $g(1)S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$, 化简后得 $S(n) = 1 - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ 。

```
1 using namespace std;
   typedef long long ll;
3 int debug_num=0;
   const int maxn=5e6+10;
5 bool valid[maxn];
   int ans[maxn/10];
7 ll phi[maxn];
   int mu[maxn];
9 const int maxm=(1LL<<32)/maxn;
   ll help1[maxn];
11 int help2[maxn];
```

```
bool vis[maxm];
13 int tot, up, m;
void get_prime(int n)
15 {
    memset(valid, true, sizeof(valid));
17     tot=0;
    mu[1]=phi[1]=1;
19     for(int i=2; i<=n; ++i){
        if(valid[i]){
21             ans[++tot]=i;
            mu[i]=-1;
23             phi[i]=i-1;
        }
25         for(int j=1; j<=tot && ans[j]*i<=n; ++j){
            int tp=ans[j]*i;
27             valid[tp]=false;
            if(i%ans[j]==0){
29                 mu[tp]=0;
                phi[tp]=phi[i]*ans[j];
31                 break;
            }
33             else{
                mu[tp]=-mu[i];
35                 phi[tp]=phi[i]*(ans[j]-1);
            }
37         }
    }
39     for(int i=1; i<=n; ++i){
        phi[i]+=phi[i-1]; mu[i]+=mu[i-1];
41     }
43 ll get_phi(ll n)
    {
45         return (n<=up) ? phi[n] : help1[m/n] ;
    }
47 ll get_mu(ll n)
    {
49         return (n<=up) ? mu[n] : help2[m/n] ;
    }
51 void solve(ll n)
    {
53         int t;
        if(n<=up || vis[t=m/n]) return ;
```

```

55     vis[t]=true;
        help1[t]=n*(n+1)/2; help2[t]=1;
57     for(ll l=2,r;l<=n;l=r+1){
            r=n/(n/l);
59         solve(n/r);
            help1[t]-=(r-l+1)*get_phi(n/r);
61         help2[t]-=(r-l+1)*get_mu(n/r);
        }
63 }
int main()
65 {
    //freopen("in.txt","r",stdin);
67     up=maxn-10;
    get_prime(up);
69     //cout<<clock()<<endl;
    int t;
71     cin>>t;
    while(t-->0)
73     {
        int n;
75         cin>>n;
        m=n;
77         if(n<=up) cout<<phi[n]<<" "<<mu[n]<<endl;
        else{
79             memset(vis,false,sizeof(vis));
            solve(n);
81             cout<<help1[1]<<" "<<help2[1]<<endl;
        }
83     }
    return 0;
85 }

```

code05/varphi-mu.cpp

例 5.12 求 $(51 \bmod 1237)$

$$\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j)$$

解

$$ans = 2 \sum_{i=1}^n \sum_{j=1}^i \gcd(i, j) - \frac{n(n+1)}{2} = 2 \sum_{i=1}^n \sum_{d|i} \varphi\left(\frac{i}{d}\right) d - \frac{n(n+1)}{2}$$

$$= 2 \sum_{d=1}^n d \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} \varphi(k) - \frac{n(n+1)}{2}$$

这里对第一维分块不影响复杂度，只要对第二维做杜教筛即可， $I = \varphi * id$ 。

例 5.13 求 $51 \text{nod } 1238$

$$ans = \sum_{i=1}^n \sum_{j=1}^n LCM(i, j) \quad n \leq 10^{10}$$

解

$$\begin{aligned} ans &= \sum_{i=1}^n \sum_{j=1}^n [i, j] \\ &= 2 \sum_{i=1}^n \sum_{j=1}^i [i, j] - \frac{n(n+1)}{2} \\ \text{Let } s(n) &= \sum_{i=1}^n \sum_{j=1}^i [i, j], \quad f(n) = \sum_{i=1}^n [i, n] \end{aligned}$$

$$\begin{aligned} f(n) &= \sum_{i=1}^n [i, n] \\ &= \sum_{i=1}^n \frac{in}{(i, n)} \\ &= n \sum_{i=1}^n \frac{i}{(i, n)} \\ &= n \sum_{d|n} \sum_{i=1}^n [(i, n) = d] \frac{i}{d} \\ &= n \sum_{d|n} \sum_{i=1}^{\frac{n}{d}} [(i, \frac{n}{d}) = 1] i \\ &= n \sum_{d|n} \sum_{i=1}^d [(i, d) = 1] i \quad (\text{due to the symmetry of the factor}) \\ &= n \sum_{d|n} \frac{\phi(d)d + [d=1]}{2} \\ &= n \frac{1 + \sum_{d|n} \phi(d)d}{2} \end{aligned}$$

$$\begin{aligned}
s(n) &= \sum_{i=1}^n f(i) \\
&= \frac{\sum_{i=1}^n i(1 + \sum_{d|i} \phi(d)d)}{2} \\
&= \frac{\sum_{i=1}^n i + \sum_{i=1}^n i \sum_{d|i} \phi(d)d}{2} \\
&= \frac{\frac{n(n+1)}{2} + \sum_{i=1}^n i \sum_{d|i} \phi(d)d}{2} \\
&= \frac{\frac{n(n+1)}{2} + \sum_{d=1}^n \phi(d)d \sum_{d|i} i}{2} \\
&= \frac{\frac{n(n+1)}{2} + \sum_{d=1}^n \phi(d)d^2 \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} i}{2} \\
&= \frac{\frac{n(n+1)}{2} + \sum_{i=1}^n i \sum_{d=1}^{\lfloor \frac{n}{i} \rfloor} \phi(d)d^2}{2}
\end{aligned}$$

$$\begin{aligned}
ans &= 2s(n) - \frac{n(n+1)}{2} \\
&= \sum_{i=1}^n i \sum_{d=1}^{\lfloor \frac{n}{i} \rfloor} \phi(d)d^2
\end{aligned}$$

可以看到杜教筛的形式，这里对第一维分块不影响复杂度，只要对第二维做杜教筛即可。

Here's how to make a Dujiao sieve for $\phi(d)d^2$

$$\begin{aligned}
\text{Let } f(d) &= \phi(d)d^2, S(n) = \sum_{d=1}^n f(d) \\
n &= \sum_{d|n} \phi(d) \\
n^3 &= \sum_{d|n} \phi(d)n^2 \\
&= \sum_{d|n} \phi(d)d^2 \left(\frac{n}{d}\right)^2 \quad \text{Now you can see that it's } n^2 \text{ and } n^3 \\
&= \sum_{d|n} h(d) \left(\frac{n}{d}\right)^2 \\
\text{So } \sum_{i=1}^n i^3 &= \sum_{i=1}^n \sum_{d|i} f(d) \left(\frac{i}{d}\right)^2
\end{aligned}$$

$$\begin{aligned}
&= \sum_{d=1}^n f(d) \sum_{d|i} \left(\frac{i}{d}\right)^2 \\
&= \sum_{d=1}^n f(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} i^2 \\
&= \sum_{i=1}^n i^2 \sum_{d=1}^{\lfloor \frac{n}{i} \rfloor} f(d) \\
&= \sum_{i=1}^n i^2 S(\lfloor \frac{n}{i} \rfloor) \\
S(n) &= \sum_{i=1}^n i^3 - \sum_{i=2}^n i^2 S(\lfloor \frac{n}{i} \rfloor)
\end{aligned}$$

5.8 拓展埃氏筛

拓展埃氏筛 (Extended Eratosthenes Sieve) 似乎最早由 min25 引入竞赛圈 (因此也常被称为 min25 筛), 可以在低于线性的时间内求得积性函数 (一些非积性函数也可以) 的前缀和。由于其方法简单且灵活性高, 近年来经常在算法竞赛中出现, 下面就来介绍一下 EES。

首先要介绍的是在 EES 中要使用到的一个前置算法, 叫做 The Meissel, Lehmer, Lagarias, Miller, Odlyzko Method (MLLMO Method), 论文 [1] 中对其进行了系统介绍, 有兴趣的可以继续阅读。

5.8.1 MLLMO Method

MLLMO Method 要解决的是这样一个问题: 求解

$$\sum_{i=2}^n [i \in Prime] F(i)$$

其中 $F(x) = x^k$, 即求解

$$\sum_{i=2}^n [i \in Prime] i^k$$

设 P_j 表示第 j 个质数, $P_1 = 2, P_2 = 3 \dots$, 特殊地 P_0 可以认为是 0。

MLLMO Method 用动态规划的思想解决这个问题:

设

$$dp(n, j) = \sum_{i=2}^n i^k [i \in Prime \text{ or } \min(p) > P_j, \text{ where } p|i \text{ and } p \in Prime]$$

也就是说 i 是质数, 或者合数 i 的最小质因子大于 P_j 时对 $dp(n, j)$ 有贡献。

那 $dp(n, 0)$ 即为 $\sum_{i=2}^n i^k$ 。下面考虑如何进行转移。

1. 假设 $P_j^2 > n, P_{j-1}^2 \leq n$ (临界情况)。

那么对于 $dp(n, j)$, $2 \sim n$ 中不存在一个合数, 其最小质因子 $> P_j$, 即 $dp(n, j)$ 的贡献全部来自于 $[2, n]$ 中的质数。

对于 $dp(n, j-1)$, $2 \sim n$ 中也不存在一个合数, 其最小质因子 $> P_{j-1}$, 即 $dp(n, j-1)$ 的贡献也全部来自于 $[2, n]$ 中的质数。

即 $dp(n, j) = dp(n, j-1)$, 对于更大的 j' , 可知贡献还是不变, 即 $dp(n, j') = dp(n, j)$ 。所以我们可以得到一个转移式:

$$dp(n, j) = dp(n, j-1), \quad \text{when } P_j^2 > n$$

2. 下面考虑 $P_j^2 \leq n$ 时的情况。

这个时候 $dp(n, j-1)$ 有非质数的贡献, 而 $dp(n, j)$, 如果考虑临界情况, 即 $P_{j+1}^2 > n$, 没有非质数的贡献。也就是说 $P_j^2 \leq n$ 时, 从 $j-1$ 到 j 的贡献变少了, 即 $dp(n, j) = dp(n, j-1) - X$ 。考虑这个 X 是什么。

显然 X 就是最小质因子是 P_j 的那些合数所造成的贡献。由于这样的合数, 每个都有 P_j 作为质因子, 我们将 P_j^k 提出, 考虑剩下的部分, 即 $dp(n, j) = dp(n, j-1) - P_j^k * X'$ 。

X' 是什么呢? X' 是最小质因子大于等于 P_j 的数 (可以是质数) 所造成的贡献, 并且这些数要 $\leq \left\lfloor \frac{n}{P_j} \right\rfloor$ 。于是 $X' = dp\left(\left\lfloor \frac{n}{P_j} \right\rfloor, j-1\right) - dp(P_j-1, j-1)$ 。


为什么呢? 我们对 X' 造成贡献的数分成合数和质数。其中合数的贡献完全对应了式子中的被减数中的合数贡献 (不多不少刚刚好); 而对于质数, 我们需要计数的是大于等于 P_j 的那些, 而被减数中计算了 $[2, \left\lfloor \frac{n}{P_j} \right\rfloor]$ 中所有的, 于是将多算的质数减去, 即减去 $dp(P_j-1, j-1)$ 。($dp(P_j-1, j-1)$ 的贡献全来自于 $[2, P_j-1]$ 中的质数)

总结一下, 转移式如下:

$$dp(n, j) = \begin{cases} dp(n, j-1) & P_j^2 > n \\ dp(n, j-1) - P_j^k * [dp\left(\left\lfloor \frac{n}{P_j} \right\rfloor, j-1\right) - dp(P_j-1, j-1)] & P_j^2 \leq n \end{cases}$$

那我们要求解的最终答案就是 $dp(n, j)$, 其中 $P_j^2 \leq n$, $P_{j+1}^2 > n$ 。

由素数定理知, j 的量级是 $O(\frac{\sqrt{n}}{\log \sqrt{n}})$ 。

 **注意** 考虑求 $dp(n, j)$ 单点的时间复杂度是多少? 以及如何进行优化。后面我们会看到如何求 $2 * \sqrt{n}$ 个 dp 值用于 EES 。

5.8.2 Extended Eratosthenes Sieve

问题 给出一个积性函数 f , 且 $f(p)$ 为关于 p 的多项式, $p \in Prime$ 。求 $S(n) = \sum_{i=1}^n f(i)$, $n = 10^{12}$ 。


解 $\forall 2 \leq i \leq n$, 我们可以将 i 分为两类:

1. 第一类数: 最大质因子的幂次 $=1$, 则其次大质因子 $< \sqrt{n}$;
2. 第二类数: 最大质因子的幂次 >1 , 则其最大质因子 $\leq \sqrt{n}$ 。

EES 算法流程如下:

- 初始化 $S(n) = f(1)$, 记 $M = \lfloor \sqrt{n} \rfloor$;

- 枚举那些所有质因子均 $\leq M$ 的数 k , 设其最大质因子为 L , 则
 $S(n)+ = f(k) * \sum_{L < p \leq \frac{n}{k}} f(p)$, p is prime, 此时每个 $k \cdot p$ 都对应第一类数, 且能覆盖所有第一类数;
- 枚举时, 若 k 的最大质因子次幂 > 1 , $S(n)+ = f(k)$, 此时 k 就是一个第二类数, 且能覆盖所有第二类数。


 **注意** 具体实现时采用 dfs。此步骤其实算是 EES 的第二步, 第一步需要做一些预处理, 即使用上面提到的 *MLLMO Method*, 具体预处理啥呢? 往下看。

几点说明:

- dfs 时需要注意, 如果对于当前枚举的基数 now (一开始为 1), 有 $now * p * p > n$, 则不调用 $now' = now * p$ 的 dfs (因为 $now' * newp > n$, where $newp > p$), 同时也不继续枚举当前素数的指数, 因为没有贡献了。
- 如果我们可以 $O(1)$ 地求出 $\sum_{L < p \leq \frac{n}{k}} f(p)$, 那么上面过程的时间复杂度是 $\Theta(n^{1-\omega})$, 但是对于 $n \leq 10^{13}$ 这样的数据范围还是很快的。感兴趣的可以阅读 2018 年集训队论文《一些特殊的数论函数求和问题》---朱震霆。
- 设 $g(i) = \sum_{2 \leq p \leq i} f(p)$, $p \in Prime$, 现在问题只剩下了如何 $O(1)$ 求 $\sum_{L < p \leq \frac{n}{k}} f(p) = g(\lfloor \frac{n}{k} \rfloor) - g(L)$ 。由于 $\lfloor \frac{n}{k} \rfloor$ 只有 $O(\sqrt{n})$ 种, $L \leq \sqrt{n}$ 也只有 $O(\sqrt{n})$ 种, 因此我们只需要计算 g 的 $O(\sqrt{n})$ 项。在题设里提到了 $f(p)$ 是一个关于 p 的多项式, 即 $f(p) = \sum a_i p^{k_i}$, 我们对于每个 i , 假设 $f(p) = p^{k_i}$, 最后乘上系数 a_i 再累加就可以得到 ans 。
 因此现在的问题就是求 $\sum_{2 \leq p \leq i} p^k$, $p \in Prime$, 其中 i 分别取 $2, 3, \dots, M, \frac{N}{M}, \frac{N}{M-1}, \dots, \frac{N}{2}, \frac{N}{1}$ 。
 $M = \lfloor \sqrt{n} \rfloor$, 除法是下取整。那这个问题就可以用上面说到的 *MLLMO Method*。算法流程如下。

使用 *MLLMO Method* 求解 $O(\sqrt{N})$ 个 g 值:

- 记 $2, 3, \dots, M, \frac{N}{M}, \frac{N}{M-1}, \dots, \frac{N}{2}, \frac{N}{1}$ 为集合 NS 。对于集合 NS 中的每个数 i , 初始化 $Map[i] = \sum_{2 \leq j \leq i} j^k$ 。当 k 较小时, 对于每个 $Map[i]$ 可以由公式 $O(1)$ 求出。// 相当于 $dp(i, 0)$
- for $p = 2, 3, 5, 7, \dots$ (不超过 M 的所有素数, 升序): // 相当于枚举 dp 中的 j 为 $1, 2, 3, \dots$
 for 集合 NS 中的每个元素 i (降序):
 if $p * p \leq i$:
 $Map[i] -= (Map[i/p] - Map[p-1]) * p^k$
 else break
 end for
 end for

 **注意** 这里滚动掉了一维 dp 数组, 因此第二维要降序枚举 i 。时间复杂度为 $O(\frac{n^{\frac{3}{4}}}{\log n})$ 。这一步预处理可看成 *EES* 的第一步。

至此, *EES* 介绍完了, 总时间复杂度为 $O(\frac{n^{\frac{3}{4}}}{\log n}) + \Theta(n^{1-\omega})$ 。

下面来看一些例题。

例 5.14 输入一个数 N , $2 \leq N \leq 10^{10}$, 求 $S(n) \bmod 1e9 + 7$ 。其中 $S(n) = \sum_{i=1}^n \phi(i)$ 。
(51nod-1239, 欧拉函数前缀和)

```

1 //f(pq) = f(p)f(q)
  //f(p) = p-1
3 //f(p^k) = p^k - p^{k-1} = p^{k-1}*(p-1)    so f(p^k) = f(p^{k-1})*p when k>1
  //need p^0 and p^1
5 typedef long long ll;
  const int mod=1e9+7;
7 const int ni2=500000004;
  inline int add(const int x, const int v) { return x + v >= mod ? x + v - mod :
    x + v; }
9 inline int dec(const int x, const int v) { return x - v < 0 ? x - v + mod : x -
  v; }
  inline int ff(ll x){
11   x%=mod;
    return x*(x+1)%mod*ni2%mod;
13 }
  ll n,M;
15 vector<int> pre[2],hou[2],primes;

17 int dfs(ll res, int last, ll f){
    int g=dec((res > M ? hou[1][n/res] : pre[1][res]), pre[1][primes[last]-1]);
19   //g(n/k) - g(L)  L is largest prime in k    ;  g(L) = g(primes[last] - 1 )
    int ret= f*g%mod;
21   for(int i=last;i<(int) primes.size();++i){
        int p = primes[i];
23     if((ll)p*p > res) break;
        for(ll q=p,nres=res,nf=f*(p-1)%mod;q*p<=res;q*=p){//nf开始为指数是1的贡
          献 nres无需修改
25         ret = add(ret, dfs(nres/=p, i+1, nf));//枚举更大的数
          nf = nf*p%mod;//继续枚举当前素数, 指数大于1时, 指数每加1, nf=nf*p;
27         ret =add(ret,nf);
        }
29   }
    return ret;
31 }

ll solve(ll n){
33   M=sqrt(n);
    for(int i=0;i<2;++i){
35     pre[i].clear();pre[i].resize(M+1);
      hou[i].clear();hou[i].resize(M+1);

```

```

37     }
    primes.clear();primes.reserve(M+1);
39     for(int i=1;i<=M;++i){
        pre[0][i]=i-1;
41         hou[0][i]=(n/i-1)%mod;
        pre[1][i]=dec(ff(i),1);;
43         hou[1][i]=dec(ff(n/i),1);
    }
45     for(int p=2;p<=M;++p){
        if(pre[0][p]==pre[0][p-1]) continue;
47         primes.push_back(p);
        const ll q=(ll)p*p,m=n/p;
49         const int pnt0=pre[0][p-1], pnt1=pre[1][p-1];
        const int mid=M/p;
51         const int End=min((ll)M, n/q);
        for(int i=1;i<=mid;++i){
53             hou[0][i]=dec(hou[0][i],dec(hou[0][i*p],pnt0));
            hou[1][i]=dec(hou[1][i],dec(hou[1][i*p],pnt1)*(ll)p%mod);
55         }
        for(int i=mid+1;i<=End;++i){
57             hou[0][i]=dec(hou[0][i],dec(pre[0][m/i],pnt0));
            hou[1][i]=dec(hou[1][i],dec(pre[1][m/i],pnt1)*(ll)p%mod);
59         }
        for(int i=M;i>=q;--i){
61             pre[0][i]=dec(pre[0][i],dec(pre[0][i/p],pnt0));
            pre[1][i]=dec(pre[1][i],dec(pre[1][i/p],pnt1)*(ll)p%mod);
63         }
    }
65     primes.push_back(M+1);
    for (int i = 1; i <= M; i++) {
67         pre[1][i] = dec(pre[1][i], pre[0][i]); //p-1
        hou[1][i] = dec(hou[1][i], hou[0][i]);
69     }
    return n>1 ? add(dfs(n,0,1),1) : 1;
71 }//
int main(){
73     cin>>n;  cout<<solve(n)<<endl;
}

```

code05/51nod1239.cpp

例 5.15 输入两个数 a, b , $2 \leq a \leq b \leq 10^{10}$, 求 $S(b) - S(a-1) \bmod 1e9 + 7$, 即区间值。其中 $S(n) = \sum_{i=1}^n \mu(i)$ 。(51nod-1244, 莫比乌斯函数前缀和)

```

//f(pq) = f(p)f(q)
2 //f(p) = -1
  //f(p^k) = 0 when k>1
4 //need p^0
  typedef long long ll;
6 ll n,M;
  vector<ll> pre,hou,primes;
8 ll dfs(ll res, int last, ll f){
    ll g=(res > M ? hou[n/res] : pre[res])-pre[primes[last]-1];
10    //g(n/k) - g(L)  L is largest prime in k ; g(L) = g(primes[last] - 1 )
    ll ret= f*g;
12    //cout<<"now: "<<n/res<<"  f:"<<f<<"  g: "<<g<<"  ret: "<<ret<<endl;
    for(int i=last;i<(int) primes.size();++i){
14        int p = primes[i];
        if((ll)p*p > res) break;
16        for(ll q=p,nres=res,nf=f*(-1); q*p<=res ;q*=p){//nf开始为指数是1的贡献
            nres无需修改
                ret += dfs(nres/=p, i+1, nf);//枚举更大的数
18            //指数大于1时, 无贡献 直接break
                break;
20        }
    }
22    return ret;
}

24 ll solve(ll n){
    M=sqrt(n);
26    pre.clear();pre.resize(M+1);
    hou.clear();hou.resize(M+1);
28    primes.clear();primes.reserve(M+1);
    for(int i=1;i<=M;++i){
30        pre[i]=i-1;
        hou[i]=(n/i-1);
32    }
    for(int p=2;p<=M;++p){
34        if(pre[p]==pre[p-1]) continue;
        primes.push_back(p);
36        const ll q=(ll)p*p,m=n/p;
        const int pnt0=pre[p-1];
38        const int mid=M/p;
        const int End=min((ll)M, n/q);
40        for(int i=1;i<=mid;++i) hou[i]=hou[i]-(hou[i*p]-pnt0);
        for(int i=mid+1;i<=End;++i) hou[i]=hou[i]-(pre[m/i]-pnt0);
    }
}

```

```

42     for(int i=M;i>=q;--i) pre[i]=pre[i]-(pre[i/p]-pnt0);
    }
44     primes.push_back(M+1);
    for (int i = 1; i <= M; i++) {
46         pre[i] = -pre[i]; //-p^0
        hou[i] = -hou[i];
48     }
    return n>1 ? dfs(n,0,1)+1 : 1;
50 }//
int main(){
52     cin>>n;
    n--;
54     ll ansa = solve(n);
    cin>>n;
56     ll ansb = solve(n);
    cout<<ansb - ansa<<endl;
58 }

```

code05/51nod1244.cpp

例 5.16 定义 $\sigma(n) = n$ 的因子数, 求 $\sum_{i=1}^n \sigma(i^k) \bmod 2^{64}$ 。输入两个数 n, k ; $n, k \leq 10^{10}$ 。
(SPOJ DIVCNTK)

```

//f(pq) = f(p)f(q)
2 //f(p) = k+1
//f(p^e) = ek+1    so f(p^e) = f(p^{e-1}) + k
4 //need p^0
typedef unsigned long long u64;
6 u64 n,M,k;
vector<u64> pre,hou;
8 vector<int> primes;
u64 dfs(u64 res, int last, u64 f){
10     u64 g=(res > M ? hou[n/res] : pre[res])-pre[primes[last]-1];
    u64 ret= f*g;
12     for(int i=last;i<(int) primes.size();++i){
        int p = primes[i];
14         if((u64)p*p > res) break;
        for(u64 q=p,nres=res,nf=f*(k+1); q*p<=res ;q*=p){//nf开始为指数是1的贡献
            nres无需修改
16             ret += dfs(nres/=p, i+1, nf); //枚举更大的数
            nf = nf + f*k;
18             ret += nf;
        }
    }

```

```

20     }
    return ret;
22 }
u64 solve(u64 n){
24     M=sqrt(n);
    pre.clear();pre.resize(M+1);
26     hou.clear();hou.resize(M+1);
    primes.clear();primes.reserve(M+1);
28     for(int i=1;i<=M;++i){
        pre[i]=i-1;
30         hou[i]=(n/i-1);
    }
32     for(int p=2;p<=M;++p){
        if(pre[p]==pre[p-1]) continue;
34         primes.push_back(p);
        const u64 q=(u64)p*p,m=n/p;
36         const int pnt0=pre[p-1];//由于是素数个数 所以可以用int 不过这里要注意
        const int mid=M/p;
38         const int End=min(M, n/q);
        for(int i=1;i<=mid;++i) hou[i]=hou[i]-(hou[i*p]-pnt0);
40         for(int i=mid+1;i<=End;++i) hou[i]=hou[i]-(pre[m/i]-pnt0);
        for(int i=M;i>=q;--i) pre[i]=pre[i]-(pre[i/p]-pnt0);
42     }
    primes.push_back(M+1);
44     for (int i = 1; i <= M; i++) {
        pre[i] = pre[i]*(k+1);
46         hou[i] = hou[i]*(k+1);
    }
48     return n>1 ? dfs(n,0,1)+1 : 1;
}//
50 int main(){
    int t; cin>>t;
52     while(t--){
        cin>>n>>k;
54         cout<<solve(n)<<endl;
    }
56 }

```

code05/DIVCNTK.cpp

例 5.17 定义 $f(n) = n$ 的最小质因子，求 $\sum_{i=1}^n f(i) \bmod 2^{64}$ ， $1 \leq N \leq 1234567891011$ 。
(SPOJ APS2)

```

//f(pq) = f(p)  p<q
2 //f(p) = p
  //f(p^e) = p
4 //need p^0 p^1
  typedef unsigned long long u64;
6 u64 n,M;
  vector<u64> pre[2],hou[2];
8 vector<int> primes;
  inline u64 ff(u64 x){
10     if(x&1) return (x+1)/2*x;
        return x/2*(x+1);
12 }
  u64 dfs(u64 res, int last, u64 f){
14     u64 g,ans;
        if(last==0){//第一次进入dfs 统计所有素数的和
16         g=(res > M ? hou[1][n/res] : pre[1][res])-pre[1][primes[last]-1];
            ans = g;
18     }
        else{
20         g=(res > M ? hou[0][n/res] : pre[0][res])-pre[0][primes[last]-1];//否则
            拿当前数的f值直接乘以素数个数即可
            ans = f*g;
22     }
        for(int i=last;i<(int) primes.size();++i){
24             int p = primes[i];
                if((u64)p*p > res) break;
26             for(u64 q=p,nres=res,nf=(last==0? p: f); q*p<=res ;q*=p){//nf开始为指数
                是1的贡献 nres无需修改
                    ans += dfs(nres/=p, i+1, nf);//枚举更大的数
28                 ans += nf;
            }
30     }
        return ans;
32 }
  u64 solve(u64 n){
34     M=sqrt(n);
        for(int i=0;i<2;++i){
36             pre[i].clear();pre[i].resize(M+1);
                hou[i].clear();hou[i].resize(M+1);
38         }
        primes.clear();primes.reserve(M+1);
40     for(int i=1;i<=M;++i){
        pre[0][i]=i-1;
    }

```



```

42     hou[0][i]=(n/i)-1;
        pre[1][i]=ff(i)-1;
44     hou[1][i]=ff(n/i)-1;
    }
46     for(int p=2;p<=M;++p){
        if(pre[0][p]==pre[0][p-1]) continue;
48     primes.push_back(p);
        const u64 q=(u64)p*p, m=n/p;
50     const u64 pnt0=pre[0][p-1], pnt1=pre[1][p-1];
        const int mid=M/p;
52     const int End=min(M, n/q);
        for(int i=1;i<=mid;++i){
54         hou[0][i]=hou[0][i]-(hou[0][i*p]-pnt0);
            hou[1][i]=hou[1][i]-(hou[1][i*p]-pnt1)*p;
56     }
        for(int i=mid+1;i<=End;++i){
58         hou[0][i]=hou[0][i]-(pre[0][m/i]-pnt0);
            hou[1][i]=hou[1][i]-(pre[1][m/i]-pnt1)*p;
60     }
        for(int i=M;i>=q;--i){
62         pre[0][i]=pre[0][i]-(pre[0][i/p]-pnt0);
            pre[1][i]=pre[1][i]-(pre[1][i/p]-pnt1)*p;
64     }
    }
66     primes.push_back(M+1);
    return n>1 ? dfs(n,0,0) : 0;
68 }//
int main(){
70     int t; cin>>t;
    while(t--){
72         cin>>n;
        cout<<solve(n)<<endl;
74     }
}

```

code05/APS2.cpp

 **注意** 同样，也可以求最大质因子，[projecteuler-642](#)。说明了 EES 对非积性函数的可行性。

例 5.18 输入一个数 N ，输出第 N 个素数。 $1 \leq N \leq 10^9$ 。时间限制：2.6s。(SPOJ NTHPRIME)

解 使用二分答案 + MLLMO Method，时间复杂度为 $O(\frac{n^{\frac{3}{4}}}{\log n} * \log n)$ 。由于常数比较大，可以固定二分次数上限，剩下的部分用区间素数筛法或者 *miller-rabin*。

下面的代码使用的是 *miller-rabin*。实际上区间素数筛速度更快， $1e7$ 长度的可筛

区间，只要不到 1s 的时间，而 *miller - rabin* 常数比较大。

```

1 #pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
   typedef long long ll;
3 const int maxn = 200;
   bool valid[maxn];
5 int primes[maxn];
   int tot;
7 void getprime(int n, int ans[])
   {
9     tot=0;
       memset(valid,true,sizeof(valid));
11    for(int i=2;i<=n;++i){
           if(valid[i]){tot++; ans[tot]=i;}
13        for(int j=1;(j<=tot) && (i*ans[j]<=n);++j){
               valid[i*ans[j]]=false; if(i%ans[j]==0) break;
15        }
       }
17 }

ll mod_mul(const ll & a,const ll & b,const ll & c){
19     __int128 tp = 1; return tp*a*b%c;
   }

21 ll fast_exp(ll a,ll b,ll c){
       ll res=1; a=a%c;
23     while(b){
           if(b&1) res=mod_mul(res,a,c);
25         b>>=1; a=mod_mul(a,a,c);
       }
27     return res;
   }

29 bool test(ll n,ll a,ll d)
   { //d=n-1
31     if(!(n&1)) return false;
       while(!(d&1)) d>>=1; //将d分解为奇数 至少有一个2因子，所以d!=n-1
33     ll t=fast_exp(a,d,n);
       while(d!=n-1 && t!=1 && t!=n-1){
35         t=mod_mul(t,t,n);
           d<<=1;
37     }

       return ((t==n-1) || (d&1) ==1 ); //两个条件都不成立则一定是合数；若两个有一个
       成立，且多次，对于合数来说，可能性极小，所以可以认为是素数
39 }

bool is_prime(ll n){

```

```

41     if(n<2) return false;
    for(int i=1;i<=tot;++i){
43         if(n==primes[i]) return true;
        if(n%primes[i]==0) return false;
45     }
    ll a[10];
47     srand(time(0));
    for(int i=0;i<6;++i) { //测试6次
49         a[i]=rand()%(n-2)+2; //取[2,n-1]随机数
        if(!test(n,a[i],n-1)) return false; //找到证据说明是合数
51     }
    return true; //如果上面所有测试都是true
53 }
ll solve(ll n)
55 {
    vector<ll> pre, hou;
57     vector<int> primes;
    vector<double> inv;
59     int M=sqrt(n);
    pre.resize(M+1+sqrt(M)+1);
61     hou.resize(M+1);
    inv.resize(M+1);
63     primes.reserve(M+1);
    for(int i=1;i<=M;++i){
65         pre[i]=i-1;
        hou[i]=(n/i-1);
67         inv[i]=1./i;
    }
69     for(int p=2;p<=M;++p){
        if(pre[p]==pre[p-1]) continue;
71         primes.push_back(p);
        const ll q=(ll)p*p,m=n/p;
73         const int pnt0=pre[p-1];
        const int mid=M/p;
75         const int End=min((ll)M, n/q);
        for(int i=1;i<=mid;++i) hou[i]-=hou[i*p]-pnt0;
77         for(int i=mid+1;i<=End;++i){
            int j = m*inv[i] + 1e-6;
79             hou[i]-=pre[j]-pnt0;
        }
81         for(int i=M/p;i>=p;--i){
            for(int j=p-1;j>=0;--j)
83                 pre[i*p+j]-=pre[i]-pnt0;


```

```

    }
85 }
    return hou[1];
87 }
ll nthprime(ll n)
89 {
    ll l=2-1, r=n*23;//问题区间[2,x] 由于是左开右闭 所以l为2 -1
91 //23 是对于1e9测试出来的 因为输入1e9 答案是22801763489
    int num = 0;
93 ll res;
    while(r-l>1)
95 {
        ll mid=(l+r)/2;
97 res = solve(mid);
        if(res>=n){
99             r=mid;
                if(num>=16) break;
101         }
        else l=mid;
103 num++;
    }
105 //cout<<clock()<<endl;
    for(;;r--){//当前r的答案是res 即[1,r]中素数个数为res
107         if(is_prime(r)){
                if(res==n) return r;
109             else res--; //res>n
        }
111     }
    }
113 int main()
    {
115     getprime(100, primes);//100以内的素数 用于加速miller-rabin
        ll n;
117     cin>>n;// 1<= n <= 10^9
        //n=1e9;
119     cout<<nthprime(n)<<endl;
        return 0;
121 }

```

code05/nthprime.cpp

 **注意** 考虑这题有没有更快的算法，时间主要花在二分时多次 *MLLMO* 上！如果能从数学上找到一个估计函数，先大致估计一下结果，然后再微调，就只要做一次 *MLLMO*。下面尝试寻找这样的估计函数进行优化。

论文 [2] 中给出了这样一组上下界：

$$\pi(x) \geq \frac{x}{\ln x} \left(1 + \frac{1}{\ln x} + \frac{2}{\ln^2 x} \right) \quad \text{for } x \geq 88783$$

$$\pi(x) \leq \frac{x}{\ln x} \left(1 + \frac{1}{\ln x} + \frac{2.334}{\ln^2 x} \right) \quad \text{for } x \geq 2953652287$$

其中 $\pi(x)$ 表示 $1 \sim x$ 中素数的个数。这样对于输入所给的 $\pi(x)$ ，我们分别用两个估计函数解出 x (看做等号)，即可得到一个估计的范围 $[L, R]$ 。然后只需要对 L 这一点做一次 *MLLMO*，对区间 $[L, R]$ 做大区间素数筛，最后遍历一遍统计即可。至于如何解这两个方程，二分即可。

对于单组询问 $n, n \leq 10^9$ ，本机测试不超过 150ms。

```
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
2 ll solve(ll n)
{
4     vector<ll> pre, hou;
    vector<int> primes;
6     vector<double> inv;
    int M=sqrt(n);
8     pre.resize(M+1+sqrt(M)+1);
    hou.resize(M+1);
10    inv.resize(M+1);
    primes.reserve(M+1);
12    for(int i=1;i<=M;++i){
        pre[i]=i-1;
14        hou[i]=(n/i-1);
        inv[i]=1./i;
16    }
    for(int p=2;p<=M;++p){
18        if(pre[p]==pre[p-1]) continue;
        primes.push_back(p);
20        const ll q=(ll)p*p,m=n/p;
        const int pnt0=pre[p-1];
22        const int mid=M/p;
        const int End=min((ll)M, n/q);
24        for(int i=1;i<=mid;++i) hou[i]-=hou[i*p]-pnt0;
        for(int i=mid+1;i<=End;++i){
26            int j = m*inv[i] + 1e-6;
            hou[i]-=pre[j]-pnt0;
28        }
        for(int i=M/p;i>=p;--i){
30            for(int j=p-1;j>=0;--j)
```

```

        pre[i*p+j]-=pre[i]-pnt0;
32     }
    }
34     return hou[1];
}
36 double upper_func(ll x)//1~x中素数的个数估计 上界
{
38     if(x >= 2953652287){
        double lgx = log((double)x);
40         double lgx2 = lgx * lgx;
        return x/lgx*(1+1/lgx+2.334/lgx2);
42     }
    else return solve(x);
44 }
double lower_func(ll x)
46 {
    if(x >= 88783){
48         double lgx = log((double)x);
        double lgx2 = lgx * lgx;
50         return x/lgx*(1+1/lgx+2.0/lgx2);
    }
52     else return solve(x);
}
54 const int maxn=2e5;//2e5 平方之后肯定 > 1e9*23 + 1.5*1e7
bool valid[maxn+10];
56 ll ans[maxn/5];
bool vis[maxn*75];//用于被筛 1.5*1e7 够用了
58 int tot;
void get_prime(int n)
60 {
    tot=0;
62     for(int i=2;i<=n;++i) valid[i]=true;
    for(int i=2;i<=n;++i){
64         if(valid[i]) ans[++tot]=i;
        for(int j=1;j<=tot && ans[j]*i<=n;++j){
66             valid[ans[j]*i]=false;
            if(i%ans[j]==0) break;
68         }
    }
70 }
ll nthprime(ll n)
72 {
    ll l,r,up,low;

```

```

74     l=2-1, r=1e9*23;//1e9 *23就够了
    while(r-l>1)
76     {
        ll mid = (l+r)/2;
78         if(lower_func(mid)>=n) r = mid;
        else l = mid;
80     }
    up = r;//up比实际值大
82     l = 2-1;
    while(r-l>1)
84     {
        ll mid=(l+r)/2;
86         if(upper_func(mid)>=n) r=mid;
        else l=mid;
88     }
    low = r;
90     //cout<<low<<" "<<up<<endl;
    if(up==low) return low;
92     else{
        get_prime(maxn);
94         ll L,R;
        L = low;
96         R = up;
        assert(R-L+1 < maxn*75);
98         for(int i=1;i<=(R-L+1);++i) vis[i]=true;
        assert(L!=1);//应该不会出现L为1的情况
100        for(int i=1;i<=tot;i++)
            for(ll j=max(ans[i],(L-1)/ans[i]+1);j<=R/ans[i];j++)//当前素数的j倍
102                vis[ans[i]*j-L+1]=false;//ans[i]*j是合数

104        //先MLLMO Method 求出low点的值
        ll now = solve(low);
106        if(now == n && vis[1]) return low;
        else{
108            for(int i=2;i<=(R-L+1);++i){
                if(vis[i]){
110                    now+=1;
                    if(now==n) return L+i-1;
112                }
            }
114        }
    }
116 }

```

```

int main()
118 {
    ll n;cin>>n;// 1<= n <= 10^9
120    cout<<nthprime(n)<<endl;
}

```

code05/nthprime2.cpp

例 5.19 print the last prime P such that $\sum_{i=2}^P i$, where i is prime $= S$ 。时间限制：3s。

The lonely line of input contains an integer S . $0 < P \leq 10^{12}$.

(SPOJ Sum of primes (reverse mode))

解 直接想法,和上一题一开始一样,直接二分答案,使用 \log 次 $MLLMO$,但由于 $MLLMO$ 是求素数和而不是素数个数,所以多了乘法以及要使用 `__int128`,导致常数更大了。这种思路肯定是过不了。

考虑寻找两个函数对答案上下界进行估计,记 $S(x)$ 为 $1 \sim x$ 中所有素数的和。论文 [3] 中给出了这样一组上下界:

$$S(x) > \frac{x^2}{2\log x} + \frac{x^2}{4\log^2 x} + \frac{x^2}{4\log^3 x} + \frac{1.2x^2}{8\log^4 x} \quad \text{for } x \geq 905238547$$

$$S(x) < \frac{x^2}{2\log x} + \frac{x^2}{4\log^2 x} + \frac{x^2}{4\log^3 x} + \frac{5.3x^2}{8\log^4 x} \quad \text{for } x \geq 110118925$$

然后就是和上一题一样的流程。对于 $P = 10^{12}$ 的极限情况,估计的范围长度最坏在 $2.5 * 1e7$ 以内,可以区间素数筛。

```

#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
2 inline __int128 ff(__int128 x){return (x&1) ? (x+1)/2*x : x/2*(x+1) ;}
__int128 scanf128(string s)
4 {
    __int128 res = 0;
6     for(int i=0;i<s.length();++i){
        res*=10;
8         res+=s[i]-'0';
    }
10    return res;
}
12 void print128(__int128 x)
{
14     vector<int> vec;
    while(x){
16         int tp = x%10;
        vec.push_back(tp);
18         x/=10;
    }
}

```



```

20     for(int i=vec.size()-1;i>=0;--i){
        cout<<vec[i];
22     }
        cout<<endl;
24 }
    __int128 pre[1000010 + 1010], hou[1000010];
26 //+1010很关键 用于后面除法变乘法时方便不用if 1010 = (n)^0.25
    double inv[1000010];
28 int primes[100010];
    __int128 S;
30 __int128 solve(ll n){
    int M=sqrt(n);
32 int tot=0;
    while (ll(M+1)*(M+1) <= n) M++;
34 for(int i=1;i<=M;++i){
        pre[i]=ff(i)-1;
36        hou[i]=ff(n/i)-1;
        inv[i] = 1./i;
38    }
    for(int p=2;p<=M;++p){
40        if(pre[p]==pre[p-1]) continue;
        primes[++tot] = p;
42        const ll q=(ll)p*p, pnt=pre[p-1];
        const int mid=M/p;
44        const int End=min((ll)M, n/q);
        for(int i=1;i<=mid;++i) hou[i]-=(hou[i*p]-pnt)*p;
46        const ll m=n/p;
        for(int i=mid+1;i<=End;++i){
48            int j = m*inv[i] + 1e-6;
            hou[i]-=(pre[j]-pnt)*p;
50        }
        for(int i=M/p; i>=p ; --i)
52            for(int j=p-1; j>=0;--j){
                pre[i*p+j]-=(pre[i]-pnt)*p;
54            }
        }
56    return hou[1];
}
58 double upper_func(__int128 x)
{
60    if(x >= 110118925){
        double x2 = x*x;
62        double lgx = log((double)x);

```

```

        double lgx2 = lgx * lgx;
64      double lgx3 = lgx2 * lgx;
        double lgx4 = lgx3 * lgx;
66      return x2/(2*lgx) + x2/(4*lgx2) + x2/(4*lgx3) + 5.3*x2/(8*lgx4);
    }
68    else return solve(x);
}
70 double lower_func(__int128 x)
{
72    if(x >= 905238547){
        double x2 = x*x;
74        double lgx = log((double)x);
        double lgx2 = lgx * lgx;
76        double lgx3 = lgx2 * lgx;
        double lgx4 = lgx3 * lgx;
78        return x2/(2*lgx) + x2/(4*lgx2) + x2/(4*lgx3) + 1.2*x2/(8*lgx4);
    }
80    else return solve(x);
}
82 const int maxn=(1<<20); //1e6+x 平方之后肯定够用了
bool valid[maxn+10];
84 ll ans[maxn/5];
bool vis[maxn*25]; //用于被筛 2.5*1e7 够用了
86 int tot;
void get_prime(int n)
88 {
    tot=0;
90    for(int i=2;i<=n;++i) valid[i]=true;
    for(int i=2;i<=n;++i){
92        if(valid[i]) ans[++tot]=i;
        for(int j=1;j<=tot && ans[j]*i<=n;++j){
94            valid[ans[j]*i]=false;
            if(i%ans[j]==0) break;
96        }
    }
98 }
int main()
100 {
    ios::sync_with_stdio(false);
102    string str;
    cin>>str;
104    S = scanf128(str);
    ll l,r,up,low;

```

```

106     l = 0, r = 1e12 + 1e10;
        while(r-l>1)
108     {
            ll mid=(l+r)/2;
110         if(lower_func(mid)>=S) r=mid;
            else l=mid;
112     }
        up = r; //up比实际值大
114     l = 0;
        while(r-l>1)
116     {
            ll mid=(l+r)/2;
118         if(upper_func(mid)>=S) r=mid;
            else l=mid;
120     }
        low = r;
122     if(up==low) cout<<low<<endl;
        else{
124         get_prime(maxn);
            ll L,R;
126         L = low;
            R = up;
128         assert(R-L+1 < maxn*25);
            //cout<<L<<" "<<R<<endl;
130         for(int i=1;i<=(R-L+1);++i){//-L+1 来编号
                vis[i]=true;
132         }
            assert(L!=1); //应该不会出现L为1的情况
134         for(int i=1;i<=tot;i++)
                for(ll j=max(ans[i], (L-1)/ans[i]+1); j<=R/ans[i]; j++) //当前素数的j倍
136                 vis[ans[i]*j-L+1]=false; //ans[i]*j是合数

138         //先MLLM0 Method 求出low点的值
            __int128 now = solve(low);
140         if(now == S && vis[1]) cout<<low<<endl;
            else{
142                 for(int i=2;i<=(R-L+1);++i){
                        if(vis[i]){
144                                now+=L+i-1;
                                    if(now==S){
146                                            cout<<L+i-1<<endl;
                                                break;
148                                }

```

150
152
154

```

    }
}
}
return 0;
}
```

code05/sum-of-primes.cpp



注意 总结一下，2s 内可解决的问题：

表 5.1: 关于素数问题的总结

问题	范围	方法
求 $1 \sim n$ 中素数个数 (给一个素数 p ，判断是第几个素数)	$n = 1e12$	<i>MLLMO</i>
求 $1 \sim n$ 中素数的和	$n = 1e12$	<i>MLLMO</i>
求第 n 个素数	$n = 1e9$	$\pi(x)$ 估计 + <i>MLLMO</i>
给一个 S ，保证是素数的前缀和， 求组成 S 的最后一个素数 p	$p = 1e12$	$S(x)$ 估计 + <i>MLLMO</i>

例 5.20 For $n = p_1^{k_1} p_2^{k_2} \dots p_m^{k_m}$, define $f(n) = k_1 + k_2 + \dots + k_m$, please calculate $\sum_{i=1}^n f(i!) \% 998244353$ 。
输入一个 n , $1 \leq n \leq 10^{10}$ 。(2019 徐州网络赛 H)

解 注意函数 $f(x)$ 虽然不满足积性，但是也有很好的性质： $f(pq) = f(p) + f(q)$ 。这样所求的式子可以化简为 $\sum_{i=1}^n g(i)$ ，其中 $g(i) = (n - i + 1) * f(i)$ 。 n 的量级是 $1e10$ ，考虑如何 *EES*。

dfs 时参数要有 f (即当前数 k 的 $f(k)$ 值)， k (当前的数)。具体记贡献如下：

- 对于最大质因子指数为 1 的贡献： $\sum_p [n - kp_i + 1][f(k) + f(p)] = [f(k) + 1] * [\sum (n + 1) * 1 - k * \sum (p_i)]$ 。也就是说我们需要 *MLLMO* 预处理素数个数前缀和以及素数前缀和。
- 对于最大质因子指数 > 1 的数 (比如数 i) 的贡献：直接就用 $g(i) = f(i) * (n - i + 1)$ 计算。其中 i 和 $f(i)$ 边乘边维护下即可 (分别乘 p 和加 1)。

```

const int mod=998244353;
2 const int ni2=499122177;
ll n,M;
4 ll nmod;
vector<int> pre[2],hou[2],primes;
6 inline int add(const int x, const int v) {
    return x + v >= mod ? x + v - mod : x + v;
8 }
```

```

inline int dec(const int x, const int v) {
10     return x - v < 0 ? x - v + mod : x - v;
}
12 //这里res是n/枚举的数
int dfs(ll res, int last, ll f, ll k){// k is now number
14     //最大质因子是prime[last-1] 但将1放在外面值显然一样
    int t1 = nmod*dec((res > M ? hou[0][n/res] : pre[0][res]),pre[0][primes[
        last]-1])%mod;
16     int t2 = k%mod*dec((res > M ? hou[1][n/res] : pre[1][res]),pre[1][primes[
        last]-1])%mod;
    int ret= (f+1)*dec(t1,t2)%mod;
18     for(int i=last;i<(int) primes.size();++i){
        int p = primes[i];
20         if((ll)p*p > res) break;
        for(ll q=p,nres=res,nk=k,nf=f+1;q*p<=res;q*=p){//nf需修改
22             ret = add(ret,dfs(nres/=p,i+1,nf,nk*=p));//枚举更大的数
            nf+=1;//继续枚举当前素数
24             ret =add(ret,nf*dec(nmod, k*q*p%mod)%mod);//指数大于1时, 记上贡献
        }
26     }
    return ret;
28 }

inline int ff(ll x){
30     x%=mod;
    return x*(x+1)%mod*ni2%mod;
32 }

int solve(ll n){
34     M=sqrt(n);
    for(int i=0;i<2;++i){
36         pre[i].clear();pre[i].resize(M+1);
        hou[i].clear();hou[i].resize(M+1);
38     }
    primes.clear();primes.reserve(M+1);
40     for(int i=1;i<=M;++i){
        pre[0][i]=i-1;
42         hou[0][i]=(n/i-1)%mod;
        pre[1][i]=dec(ff(i),1);;
44         hou[1][i]=dec(ff(n/i),1);
    }
46     for(int p=2;p<=M;++p){
        if(pre[0][p]==pre[0][p-1]) continue;
48         primes.push_back(p);
        const ll q=(ll)p*p,m=n/p;

```

```

50     const int pnt0=pre[0][p-1],pnt1=pre[1][p-1];
       const int mid=M/p;
52     const int End=min((ll)M,n/q);
       for(int i=1;i<=mid;++i){
54         hou[0][i]=dec(hou[0][i],dec(hou[0][i*p],pnt0));
         hou[1][i]=dec(hou[1][i],dec(hou[1][i*p],pnt1)*(ll)p%mod);
56     }
       for(int i=mid+1;i<=End;++i){
58         hou[0][i]=dec(hou[0][i],dec(pre[0][m/i],pnt0));
         hou[1][i]=dec(hou[1][i],dec(pre[1][m/i],pnt1)*(ll)p%mod);
60     }
       for(int i=M;i>=q;--i){
62         pre[0][i]=dec(pre[0][i],dec(pre[0][i/p],pnt0));
         pre[1][i]=dec(pre[1][i],dec(pre[1][i/p],pnt1)*(ll)p%mod);
64     }
       }
66     //cout<<clock()<<endl;
       primes.push_back(M+1);
68     return n>1 ? add(dfs(n,0,0,1),0) : 0;
}
70 int main()
{
72     ios::sync_with_stdio(false);
       cin>>n;
74     nmod = (n+1)%mod;
       cout<<solve(n)<<endl;
76     return 0;
}

```

code05/function.cpp

例 5.21 $f(n, k)$ is the number of way to select k numbers $a_i, a_i > 1$ and $\prod_{i=1}^k a_i = n$. solve $\sum_{i=1}^n f(i, k)$ after mod $1e9+7$. Note that if $n=6, 6=2*3$ and $6=3*2$ are different way.

Input one line contains two integers n, k ($1 \leq n \leq 2^{30}, 1 \leq k \leq 30$).

(2019ccpc 湖南全国邀请赛 F.Neko and function)

解 考虑 f 函数如何计算，发现不是积性函数而且不好计算。考虑另外一个函数 g ，和 f 的区别只是限制条件 $a_i > 1$ 变为 $a_i \geq 1$ ，这个时候考虑 $g(n, k)$ 如何求解以及 f 和 g 之间的关系。

- 考虑 f 和 g 之间的关系，显然 g 的情况是不小于 f 的，考虑多的部分，其实就是有 $a_i = 1$ 的情况。自然想到枚举哪个位置放 1，剩下的用 g 继续表示，即 $f(n, k) = g(n, k) - C_k^1 * g(n, k-1)$ 。但发现其实多减去了，因为后面那个部分统计显然有重复，

重复的就是 $C_k^2 * g(n, k-2) - C_k^3 * g(n, k-3) + \dots$

也就是要容斥一下, 即 $f(n, k) = \sum_{i=0}^k (-1)^i * C_k^i * g(n, k-i)$ 。

由于 k 只有 30, 这个时候祈祷 g 是积性函数, ees 就结束了。确实是。

- 考虑函数 $g(n, k)$, 将 n 质因子分解, 可知不同质因子贡献是满足积性的。而对于 $g(p^e, k)$, 对应于 e 个球, k 个盒子, 球没有区别, 位置 (盒子) 有区别的放球模型 (可以有空盒)。即 $g(p^e, k) = C_{e+k-1}^{k-1}$ ($k-1$ 个插板, $e+k-1$ 个位置)。

最多 30 次 ees 即可。注意特判 $n=1$ 的情况。

```

1 #pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
  typedef long long ll;
3 int pre[100010], hou[100010];
  double inv[100010];
5 int primes[100010];
  ll ni[40], Fac[40], Fac_inv[40];
7 const int mod = 1e9+7;
  inline int add(const int x, const int v) {return x + v >= mod ? x + v - mod : x
    + v;}
9 inline int dec(const int x, const int v) {return x - v < 0 ? x - v + mod : x -
  v;}
  int n, nowk, M, tot;
11 int dfs(int res, int last, int f){
  int g = dec(res > M ? hou[n/res] : pre[res], pre[primes[last]-1]);
13 int ans = 1LL*g*f%mod*nowk%mod;
  for(int i=last; i<tot; ++i){
15 int p = primes[i];
    if((ll)p*p > res) break;
17 int num=1;
    for(ll q=p, nres=res, nf=1LL*f*nowk%mod; q*p<=res; q*=p){
19 ans = add(ans, dfs(nres/=p, i+1, nf));
      nf = nf*(num + nowk)%mod*ni[num+1]%mod;
21 num++;
      ans = add(ans, nf);
23 }
    }
25 return ans;
  }
27 int solve(int n){
  M=sqrt(n);
29 tot=0;
  while ((ll)(M+1)*(M+1) <= n) M++;
31 for(int i=1; i<=M; ++i){
  pre[i]=i-1;

```

```

33     hou[i]=(n/i-1); //n 1e9 不用模
        inv[i] = 1./i;
35 }
    for(int p=2;p<=M;++p){
37         if(pre[p]==pre[p-1]) continue;
        primes[tot++] = p;
39         const int q=p*p, pnt=pre[p-1];
        const int mid=M/p;
41         const int End=min(M, n/q);
        for(int i=1;i<=mid;++i) hou[i]-=(hou[i*p]-pnt);
43         const int m=n/p;
        for(int i=mid+1;i<=End;++i){
45             int j = m*inv[i] + 1e-6;
            hou[i]-=(pre[j]-pnt);
47         }
        for(int i=M/p; i>=p ; --i)
49             for(int j=p-1; j>=0;--j){
                pre[i*p+j]-=(pre[i]-pnt);
51             }
        }
53     primes[tot++] = M+1;
    assert(n>1);
55     return dfs(n,0,1);
}

57 void init()
{
59     ni[1]=1;
    Fac[0]=1;
61     Fac_inv[0]=1;
    for(int i=1;i<40;++i){
63         if(i!=1) ni[i]=((mod-mod/i)*(ni[mod%i]))%mod;
        Fac[i]=Fac[i-1]*i%mod;
65         Fac_inv[i]=Fac_inv[i-1]*ni[i]%mod;
    }
67 }

int main()
69 {
    init();
71     int k;
    while(scanf("%d%d",&n,&k)!=EOF){
73         if(n==1 || k>=n){cout<<0<<endl; continue;}
        int ans = 0;
75         for(int i=0;i<k;++i){

```



```

    nowk = k-i;
77     ll tp = Fac[k]*Fac_inv[i]%mod*Fac_inv[k-i]%mod;
    if(!(i&1)) ans = add(ans, tp*solve(n)%mod);
79     else ans = dec(ans, tp*solve(n)%mod);
    }
81     cout<<ans<<endl;
    }
83     return 0;
}

```

code05/Neko-and-function.cpp

例 5.22 题意转化后：函数 f 是积性函数，且 $f(p^k) = (p \% 4 == 1) ? 3k + 1 : 1$ 。求 $\sum_{i=1}^n f(i)$ ，其中 $n \leq 10^9$ 。(2019 牛客暑期多校训练营第七场 K.Function)

解 可以看到根据 $p \% 4$ 的值的不同，计算方法不同，这时候怎么办呢？对于大于 2 的素数 p ， p 模 4 为 1 或 3。如果我们能分别预处理它们的前缀，那么问题就得到了解决。首先我们引入一个函数 $\chi(n)$ 如下：

$$\chi(n) = \begin{cases} 1 & n \% 4 == 1 \\ -1 & n \% 4 == 3 \\ 0 & n \% 4 == 0 \text{ or } 2 \end{cases} \quad n > 0$$

$\chi(n)$ 是一个积性函数，所以我们在 *MLLMO* 的时候可以乘上这个函数，这样我们就可以预处理出 $1 \sim n$ 中 $\%4 = 1$ 的素数减去 $\%4 = 3$ 的素数的个数（那些前缀们）。由于我们可以方便地预处理出 $1 \sim n$ 中 $\%4 = 1$ 的素数加上 $\%4 = 3$ 的素数的个数（那些前缀们，也就是所有素数，就像最常做的那样），所以就能推出 $1 \sim n$ 中模 4 余 1 的素数个数了。

```

//f(1)=1
2 //f(p)= (p%4==1)? 4 : 1
  //f(p^e) = (p%4==1)? 3e+1 : 1
4 #include<bits/stdc++.h>
  using namespace std;
6 typedef long long ll;
  int n,M;
8 vector<int> pre[2],hou[2],primes;
  //0      1~n 素数个数
10 //1      1~n中  %4=1素数个数 减去  %4=3素数个数
  ll dfs(ll res, int last, ll f){
12     ll t=(res > M ? hou[0][n/res] : pre[0][res])-pre[0][primes[last]-1];
    ll ans= t*f;
14     for(int i=last;i<(int) primes.size();++i){
        int p = primes[i];

```

```

16     if(p*p > res) break;
    for(ll q=p, nres=res, nf=f*((p&3)==1)?4:1);q*p<=res;q*=p){
18         ans += dfs (nres/=p,i+1,nf);
        nf += f*((p&3)==1)?3:0;
20         ans += nf;
    }
22 }
    return ans;
24 }
    int help1[4]={0,1,1,0};
26 int help2[4]={0,1,0,-1};
    ll solve(int n){
28     M=sqrt(n);
    for(int i=0;i<2;++i){
30         pre[i].clear();pre[i].resize(M+1);
        hou[i].clear();hou[i].resize(M+1);
32     }
    primes.clear();primes.reserve(M+1);
34     for(int i=1;i<=M;++i){
        pre[0][i]=i-1;
36         hou[0][i]=n/i-1;
        pre[1][i]=help1[i&3]-1;//kafang function
38         hou[1][i]=help1[(n/i)&3]-1;
    }
40     for(int p=2;p<=M;++p){
        if(pre[0][p]==pre[0][p-1]) continue;
42         primes.push_back(p);
        const int q=p*p,m=n/p, pnt0 = pre[0][p-1], pnt1 = pre[1][p-1];
44         const int mid=M/p;
        const int End=min(M,n/q);
46         for(int i=1;i<=mid;++i){
            hou[0][i]-=hou[0][i*p]-pnt0;
48             hou[1][i]-=(hou[1][i*p]-pnt1)*help2[p&3];
        }
50         for(int i=mid+1;i<=End;++i){
            hou[0][i]-=pre[0][m/i]-pnt0;
52             hou[1][i]-=(pre[1][m/i]-pnt1)*help2[p&3];
        }
54         for(int i=M;i>=q;--i){
            pre[0][i]-=pre[0][i/p]-pnt0;
56             pre[1][i]-=(pre[1][i/p]-pnt1)*help2[p&3];
        }
58     }


```

```

primes.push_back(M+1);
60 //由0 和 1 可推出 1~n中模4余1的素数个数 覆盖入1中
for (int i = 2; i <= M; ++i) {
62     assert((pre[0][i] + pre[1][i] - 1)%2==0);
        assert((hou[0][i] + hou[1][i] - 1)%2==0);
64     pre[1][i] = (pre[0][i] + pre[1][i] - 1)/2;
        hou[1][i] = (hou[0][i] + hou[1][i] - 1)/2;
66 }
    hou[1][1] = (hou[0][1] + hou[1][1] - 1)/2;
68 //对于本题 计算的贡献为3*pre[1] + pre[0]
    //覆盖到0中
70 for(int i = 1; i <= M; ++i){
        pre[0][i] += 3*pre[1][i];
72     hou[0][i] += 3*hou[1][i];
    }
74 return n>1 ? 1+dfs(n,0,1) : 1;
}
76 int main()
{
78     int t; cin>>t; while(t--){
        {cin>>n; cout<<solve(n)<<endl;}
80     return 0;
}

```

code05/nowcode-Function.cpp

 **注意** 考虑有没有其他方法，比如直接从 *MLLMO* 的转移式考虑。

第5章 习题

1. HDU6588 2019ICPC 网络赛 经典 gcd 求和
2. HDU6707 2019CCPC 网络赛 杜教筛

第 6 章 不定方程

内容提要

□ 佩尔方程

□ 其它不定方程

6.1 佩尔方程

Pell 方程是指具有形式 $x^2 - Dy^2 = 1$ 的方程, 其中 D 是一个固定的正整数且不是完全平方数。

假设可求得 *Pell* 方程的一个解 x_1, y_1 , 则可以用下面的方法来产生一个新的解。将已知解因式分解为

$$1 = x_1^2 - Dy_1^2 = (x_1 + y_1\sqrt{D})(x_1 - y_1\sqrt{D})$$

两边同时平方得到一个解:

$$1 = 1^2 = (x_1 + y_1\sqrt{D})^2(x_1 - y_1\sqrt{D})^2 = (x_1^2 + y_1^2D)^2 - (2x_1y_1)^2D$$

也就是说, $(x_1^2 + y_1^2D, 2x_1y_1)$ 是一个新解。取 3 次幂、4 次幂, 等等, 可以找到所需的任意多个解。

定理 6.1. Pell 方程定理

设 D 是一个正整数, 且不是完全平方数, 则佩尔方程

$$x^2 - Dy^2 = 1$$

总有正整数解。如果 (x_1, y_1) 是使 x_1 最小的解, 则每个解 (x_k, y_k) 可通过取幂得到:

$$x_k + y_k\sqrt{D} = (x_1 + y_1\sqrt{D})^k, \quad k = 1, 2, 3, \dots$$

矩阵形式求解:

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 & Dy_1 \\ y_1 & x_1 \end{pmatrix}^{n-1} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$



注意 注意, 一些方程的最小解会很大。关于何时最小解大, 何时小, 还没有已知的明确的模式。

6.2 其它不定方程

6.2.1 OEIS A011772

$f(n)$ = 最小的正整数 m 使得 $\frac{m*(m+1)}{2}$ 是 n 的倍数。 $f(1) \sim f(10)$ 分别为 1, 3, 2, 7, 4, 3, 6, 15, 8, 4。
现在给定 n , $1 \leq n \leq 10^{12}$, 求 $f(n)$ 。

题目链接: [鱼跃龙门](#)

解 由于 $m * (m + 1)$ 一定整除 2, 于是问题就是找最小的 m , 使得 $m * (m + 1)$ 是 $2 * n$ 的倍数, 那我们假设是 k 倍, 那就是解二元二次方程 $m * (m + 1) = 2nk$ 中 m 的最小正整数解。乍一看很难做, 这里需要用到 m 和 $m + 1$ 一定互质的性质。我们将 $2n$ 质因数分解, 然后分成 A, B 两块 (暴力枚举), 即 $2n = A * B, \gcd(A, B) = 1$ 。然后设 $m = Ax, m + 1 = By$, 那么有 $By - Ax = 1, y = \frac{Ax+1}{B}$ 。我们的目的是让 $k (= xy)$ 尽量小, 由于 y 随 x 单增, 所以只要 x 最小即可。那么只要解 $AX \equiv -1 \pmod{B}$ 的最小正整数解即可。

时间复杂度为 $O(\frac{\sqrt{2n}}{\ln(\sqrt{2n})} + 2^{12} * \log(n))$ 。

```

1 #include<bits/stdc++.h>
  using namespace std;
3 #define ll __int128
  const int maxn=100;
5 const int maxm =2e6+10;
  ll a[maxn]; //质因子
7 int b[maxn]; //质因子的指数
  int tot; //1~tot
9 bool valid[maxn];
  int prime[maxn];
11 void getprime(int n,int &tot,int ans[])
  {
13     tot=0;
    memset(valid,true,sizeof(valid));
15     for(int i=2;i<=n;++i){
        if(valid[i]){
17         tot++;
        ans[tot]=i;
19     }
    //下面的主角是小于等于i的每个质数
21     for(int j=1;(j<=tot) && (i*ans[j]<=n);++j){
        valid[i*ans[j]]=false;
23         if(i%ans[j]==0) break; //如果整除就break;
    }
25 }
  }
27 void factor(ll n)
  {
29     ll now=n;
    tot=0;
31     for(int i=1;1LL*prime[i]*prime[i]<=now;++i) if(now%prime[i]==0){
        a[++tot]=prime[i]; b[tot]=0;
33         while(now%prime[i]==0){++b[tot]; now/=prime[i];}
    }
  }

```

```

    }
35     if(now>1){
        a[++tot]=now;b[tot]=1;
37     }
    }
39 ll e_gcd(ll a,ll b,ll &x,ll &y)
    {
41     if(b==0){x=1;y=0;return a;}
        ll ans=e_gcd(b,a%b,x,y);
43     ll temp=x; x=y; y=temp-a/b*y;
        return ans;
45 }
    ll newinv(ll a,ll mod)
47 {
        ll ans,tmp;
49     e_gcd(a,mod,ans,tmp);
        //gcd(a,mod) = 1
51     ll c = mod-1;
        ans = (c*ans%mod+mod)%mod;
53     return ans;
    }
55 int main()
    {
57     int Count;
        getprime(2e6,Count, prime);
59     int t;
        cin>>t;
61     while(t-->0)
        {
63         long long n;
            cin>>n;
65         if(n==1){cout<<1<<endl;continue;}
            factor(2*n); //n>1
67         assert(tot<15);
            ll ans=1e18;
69         for(int i=0;i<(1<<tot);++i){
                int tp = i;
71                 int id = 0;
                    ll A=1,B;
73                 while(tp){
                        ++id;
75                 if(tp&1){
                        A=A*(ll)pow((long long)(a[id]),(long long)(b[id]));

```

```
77         }
78         tp>>=1;
79     }
80     B = 2*n/A;
81     ll ni = newinv(A,B);
82     ll ans1 = A*ni;
83     if(ans1 == 0) ans1 = 1e18;
84     ans = min(ans1,ans);
85 }
86 cout<<(long long)ans<<endl;
87 }
88 return 0;
89 }
```

code06/OEIS-A011772.cpp

第 6 章 习题

1. POJ1320 Street Numbers 佩尔方程
2. HDU6222 Heron and His Triangle 佩尔方程

第 7 章 其他

本章内容提要

- 勾股数组
- 圆上整点与高斯素数
- 模线性方程循环节
- 分数转小数
- DFS Similar
- FFT, NTT

7.1 勾股数组

定义 7.1. 本原勾股数组

本原勾股数组是一个三元组 (a, b, c) ，其中 a, b, c 没有公因数，且满足：

$$a^2 + b^2 = c^2$$

定理 7.1. 勾股数组定理

每个本原勾股数组 (a, b, c) (a 为奇数， b 为偶数) 都可从如下公式得出：

$$a = st, \quad b = \frac{s^2 - t^2}{2}, \quad c = \frac{s^2 + t^2}{2}$$

其中 $s > t \geq 1$ 是任意没有公因数的奇数。

7.2 平方数之和、圆上整点

这一节要解决的问题是给定一个数 x ，判断它能否分解为两个整数的平方之和，以及具体来说如何分解。从几何意义上考虑就是以原点为圆心， \sqrt{x} 为半径的圆，是否经过坐标点 (横纵坐标都是整数)，以及具体是哪些点。

先从 x 为素数考虑，这个时候规律相对简单。

7.2.1 将素数分解为平方之和

定理 7.2. 定理

设 p 是素数，则 p 是两平方数之和的充要条件是 $p \equiv 1 \pmod{4}$ 或 $p = 2$ 。

证明 这里有两个断言：

- p 是两平方数之和；
- $p \equiv 1 \pmod{4}$ 或者 $p = 2$ 。

要证明充要性有两个方面，即用一个断言作为条件去证明另外一个，下面分别证明。

1. 若 p 是两平方数之和, 则 $-a^2 \equiv b^2 \pmod{p}$, 接着对两边取勒让德符号: (回顾第四章的内容)

$$\left(\frac{-1}{p}\right)\left(\frac{a}{p}\right)^2 = \left(\frac{b}{p}\right)^2$$

$$\left(\frac{-1}{p}\right) = 1$$

于是由二次互反律知, p 模 4 余 1 或者 $p = 2$ 。□

2. 当 $p \equiv 1 \pmod{4}$ 或 $p = 2$ 时, 要证明 p 一定可以表示成两平方数之和。也就是说只要我们能想到一种方法总能找到如何分解, 就完成了证明。下面介绍一种方法, 称之为**费马降阶法** (Fermat Descent Procedure)。我们后面的代码就是基于此方法。

我们从 $A^2 + B^2 = Mp$ 开始, 如果这里 $M = 1$, 则证明完毕。所以我们考虑 $M \geq 2$ 。费马的想法是, 我们用现有的 A, B, M , 要是能构造出 $a^2 + b^2 = mp$ 且 $m \leq M - 1$, 则迭代下去, $m = 1$ 时就完成了构造。

在说具体的构造方法之前, 先看一个恒等式, 其正确性是显然的, 在下面的构造过程中, 这个恒等式起到关键作用。

$$(u^2 + v^2)(A^2 + B^2) = (uA + vB)^2 + (vA - uB)^2$$

费马降阶法的过程如表 7.1 所示:

可以发现, 每迭代一次, p 的系数至少减半, 即迭代次数为 $O(\log p)$ 次。为了说明上述过程的正确性, 我们还要证明 5 个断言的正确性。

- (a). 一定可以找出数 A, B , 使得 $A^2 + B^2 = Mp$, 且 $M < p$ 。

证明 取同余式 $x^2 \equiv -1 \pmod{p}$ 的一个解, 由二次互反律知, 当 $p \equiv 1 \pmod{4}$ 时, 必定有解 x 。所以取 $A = x, B = 1$ 具有性质 $p \mid A^2 + B^2$, 而且 $M = \frac{A^2 + B^2}{p} \leq \frac{(p-1)^2 + 1^2}{p} < p$ 。□

在降阶程序的第二步, 我们选取 u, v 使其满足

$$u \equiv A \pmod{M}, \quad v \equiv B \pmod{M}, \quad -\frac{1}{2}M \leq u, v \leq \frac{1}{2}M$$

于是有

$$u^2 + v^2 \equiv A^2 + B^2 \equiv 0 \pmod{M}$$

设 $u^2 + v^2 = rM$, 其余四个断言如下:

- (b). $r \geq 1$
 (c). $r < M$
 (d). $uA + vB$ 能被 M 整除
 (e). $vA - uB$ 能被 M 整除

这四个断言比较容易证明, 这里不再给出。□

至此, 对定理 7.2 的证明结束。□

上面求解过程中关键的一步是求解 $x^2 \equiv -1 \pmod{p}$, 可以直接使用二次剩余的模板, 也可以使用随机算法。即随机一个 $a[1, p-1]$, 求解 $b \equiv a^{(p-1)/4} \pmod{p}$, 由欧拉准则可知 $b^2 \equiv \left(\frac{a}{p}\right) \pmod{p}$, 即若选取的 a 不是 p 的二次剩余 (有一半的概率), 则求得的 b 即为解 x 。由二次剩余性质知有两个解 x_1, x_2 , 且 $x_1 + x_2 = p$ 。

表 7.1: 费马降阶法

举例 (p=881)	符号表示 $p \text{ any prime} \equiv 1(mod 4)$
有 $387^2 + 1^2 = 170 * 881, 170 < 881$	有 $A^2 + B^2 = Mp, M < p$
求得数 47, 1 使得 $47 \equiv 387 (mod 170)$ $1 \equiv 1 (mod 170)$ 其中 $-\frac{170}{2} \leq 47, 1 \leq \frac{170}{2}$	求得数 u, v 使得 $u \equiv A (mod M)$ $v \equiv B (mod M)$ 其中 $-\frac{M}{2} \leq u, v \leq \frac{M}{2}$
于是有 $47^2 + 1^2 \equiv 387^2 + 1^2 \equiv 0 (mod 170)$	于是有 $u^2 + v^2 \equiv A^2 + B^2 \equiv 0 (mod M)$
所以可写 $47^2 + 1^2 = 170 * 13$ $387^2 + 1^2 = 170 * 881$	所以可写 $u^2 + v^2 = Mr(1 \leq r < M)$ $A^2 + B^2 = Mp$
相乘可得 $(47^2 + 1^2)(387^2 + 1^2) = 170^2 * 13 * 881$	相乘可得 $(u^2 + v^2)(A^2 + B^2) = M^2 * r * p$
利用恒等式 $(u^2 + v^2)(A^2 + B^2) = (uA + vB)^2 + (vA - uB)^2$	
$(47 * 387 + 1 * 1)^2 + (1 * 387 - 47 * 1)^2$ $= 170^2 * 13 * 881$ 所以 $18190^2 + 340^2 = 170^2 * 13 * 881$	$(uA + vB)^2 + (vA - uB)^2 = M^2 * r * p$
两边同除以 170^2 得 $(\frac{18190}{170})^2 + (\frac{340}{170})^2 = 13 * 881$ $107^2 + 2^2 = 13 * 881$	两边同除以 M^2 得 $(\frac{uA+vB}{M})^2 + (\frac{vA-uB}{M})^2 = rp$ (肯定可以整除)
由此得到能表示成两平方数之和的 p 的更小倍数	
重复上述过程, 直到 p 本身能表成两平方数之和 ($r = 1$)	

代码如下：

```
1 //求解  $x^2 \equiv -1 \pmod{p}$ 
  //  $O(\log p)$ 
3 ll ran(ll n){
    assert(n%4==1);
5    srand(time(0));
    for(;;){
7        ll a=rand()*rand()%(n-1)+1;
        ll b=fast_exp(a,(n-1)/4,n);
9        if(b*b%n==n-1) return b<=(n-1)/2 ? b:n-b;
    }
11 }
```

code07/random-algo-modsqr.cpp

现在我们可以将素数分解为平方数之和了，下面看一下对于一般数该怎么做。

7.2.2 将数分解为平方之和

这一节的内容，不打算给出相关证明，而是给出一些结论和代码。因为我觉得下面这个视频已经讲的不错了，[bilibili: 隐藏在素数规律中的 \$\pi\$](#) 。

定理 7.3. 圆上整点数定理

定义函数 $\chi(n)$ 如下：

$$\chi(n) = \begin{cases} 1 & n \bmod 4 == 1 \\ -1 & n \bmod 4 == 3 \\ 0 & n \bmod 4 == 0 \text{ or } 2 \end{cases} \quad n > 0$$

对于半径为 \sqrt{N} 的圆，圆上整点的数目（即将 N 分成两个平方数之和的方案数）可以这样计算：

将 N 质因数分解：

$$N = p_1^{k_1} * p_2^{k_2} * \dots * p_m^{k_m}$$

则圆上整点数目 $= 4 * \prod_{i=1}^m (\sum_{j=0}^{k_i} \chi(p_i^j))$ 。

上面的这个式子只是为了统一，所以看起来规律不明显。实际上一句话：

如果有模 4 余 3 的素数的指数为奇数，则答案为 0；否则就是所有模 4 余 1 的素数的指数加 1 后乘起来最后再乘 4。



至于具体如何计算分解的方案，推荐大家看上面那个视频。大致思路就是对于可分解的素数利用费马降阶法进行分解，然后再在复数域中对不同质数的结果组合计算一下。下面给出代码：

输入一个半径 r ，输出在圆上的所有整点。

时间复杂度： $O(A + B)$ ，其中 A 为质因子分解 r^2 (thus r) 的时间， B 为圆上整点数。

该代码在 $r \leq 10^9$ 时通过测试，更大时注意下会不会爆范围。

圆上整点/高斯素数 2019 上海网络赛 Peekaboo

```

1 typedef pair<ll,ll> pll;
  #define mp make_pair
3 ll fun45(double x){if(x>=0) return x+0.5;return x-0.5;}
  ll mod_mul(ll a,ll b,ll c){__int128 tp=1;return tp*a*b%c;}
5 ll fast_exp(ll a,ll b,ll c){
    ll res=1;a=a%c;assert(b>=0);
7    while(b>0)
    {
9        if(b&1) res=mod_mul(a,res,c);
        b=b>>1;a=mod_mul(a,a,c);
11    }
    return res;
13 }
  ll ran(ll n){//求解  $x^2 \equiv -1 \pmod n$       //要保证  $n\%4=1$ 
15    assert(n%4==1);
    srand(time(0));
17    for(;;){
        ll a=rand()%(n-1)+1;//1~ n-1
19        ll b=fast_exp(a, (n-1)/4 ,n);
        if(mod_mul(b,b,n)==n-1) return b<=(n-1)/2 ? b:n-b;
21    }
  }
23 pll solveprime(ll p)
  {
25    if(p==2) return mp(1,1);
    if(p%4!=1) return mp(-1,-1);
27    ll A,B,u,v,M;
    A=ran(p); B=1;
29    __int128 tmp=1;
    M=(tmp*A*A+tmp*B*B)/p;
31    assert(M<p);
    while(M>1)
33    {
        u=(A%M+M)%M;v=(B%M+M)%M;
35        if(2*u>M) u-=M;//注意可能是负数
        if(2*v>M) v-=M;
37        assert(u<=M/2 && u>=-M/2);
        assert(v<=M/2 && v>=-M/2);
39        ll ta=A;
        A=(tmp*u*A+tmp*v*B)/M;
41        B=(tmp*v*ta-tmp*u*B)/M;

```

```

        M=(tmp*u*u+tmp*v*v)/M;
43     }
        return make_pair(min(abs(A),abs(B)),max(abs(A),abs(B)));
45 }
const int maxn = 100;
47 ll a[maxn]; //质因子
int b[maxn]; //质因子的指数
49 int tot; //1~tot
void factor(ll n)
51 {
    assert(n>=1);
53     int now=n;
    tot=0;
55     for(int i=2;i*i<=now;++i) if(now%i==0){
        a[++tot]=i;
57         b[tot]=0;
        while(now%i==0){
59             ++b[tot];
            now/=i;
61         }
    }
63     if(now>1){a[++tot]=now;b[tot]=1;}
}
65 vector<pll> ans;
vector<pll> bns;
67 vector<int> cns; //每个模4余1的素数的指数
vector<complex<double> > pre; //维护每个模4余1的素数的a-bi前缀乘积
69 vector<complex<double> > p3; //size = 4    %4=3 和 2 的贡献
void dfs(int last, complex<double> now)
71 {
    if(last==cns.size()){
73         //一种方案 可以扩展为4种 即分别乘以p3  -p3  p3i  -p3i
        //cout<<now.real()<<" * "<<now.imag()<<endl;
75         for(int i=0;i<4;++i){
            complex<double> tp = p3[i]*now;
77             ans.push_back(mp(fun45(tp.real()), fun45(tp.imag())));
        }
79         return ;
    }
81     complex<double> z{1,0};
    complex<double> z1{1.0*bns[last].first, 1.0*bns[last].second};
83     complex<double> z2 = pre[last];
    complex<double> z3{1.0*bns[last].first, -1.0*bns[last].second};

```

```

85     for(int i=0;i<=cns[last];++i){
            dfs(last+1, now*z*z2);
87         z = z*z1;
            z2 = z2/z3;
89     }
    }
91 void solve(ll r)
    {
93     ans.clear();bns.clear();cns.clear();
        pre.clear();p3.clear();
95     //对r质因子分解
        factor(r);
97     for(int i=1;i<=tot;++i) b[i]*=2;//r^2质因子分解
        ll tp=1;    //对那些模4余3的质数 平分贡献
99     for(int i=1;i<=tot;++i) if((a[i]&3) == 3) tp=tp*fast_exp(a[i], b[i]>>1 , 1
        e18);
        int dir[4][2]={1,0},{-1,0},{0,1},{0,-1}};
101    complex<double> z;
        for(int i=0;i<4;++i){
103        z.imag(tp*dir[i][0]);
            z.real(tp*dir[i][1]);
105        if(a[1]==2 && tot>=1){//2的贡献加进去 tot>=1是排除n=1的情况
            complex<double> z1{1.0,1.0};
107            z = z*pow(z1,b[1]);
        }
109        p3.push_back(z);
    }
111    for(int i=1;i<=tot;++i){//对那些模4余1的质数求出它们的解 并且有 指数+1 种情况
        if((a[i]&3) == 1){
113            pll tp = solveprime(a[i]);
                bns.push_back(tp);
115            z.real(tp.first);
                z.imag(-tp.second);
117            pre.push_back(pow(z,b[i]));
                cns.push_back(b[i]);
119        }
    }
121    z.imag(0);z.real(1);
        dfs(0, z);
123    //check ans
        ll res =1;
125    for(int i=1;i<=tot;++i)
        {

```

```

127     if((a[i]&3) == 1) res*=(b[i]+1);
        else if((a[i]&3) == 3 && (b[i]&1)==1){
129         res = 0;
            break;
131     }
        }
133     res*=4;
        assert(res == ans.size()); //由于这里求得是r^2的情况 所以一定存在
135     //当然任意的N都可以求
    }
137 int main()
    {
139     ll r;
        cin>>r;
141     solve(r);
        if(ans.size()==0){
143         cout<<"no solution"<<endl;
            return 0;
145     }
        cout<<ans.size()<<endl;
147     for(auto k:ans){
        assert(k.first*k.first + k.second*k.second == r*r);
149         cout<<k.first<<" "<<k.second<<endl;
    }
151     return 0;
    }

```

code07/Lattice-points.cpp

7.3 循环节问题

7.3.1 二阶常系数齐次线性递推循环节

所谓二阶常系数齐次线性递推式就是类似斐波那契递推式那样的数列。这样的数列在模意义下会存在循环节。下面阐述下如何求其循环节：

先将模数分解，在模素数幂意义下分别求循环节，最后取最小公倍数即可。而对于模素数幂有结论 $G(p_i^{a_i}) = p_i^{a_i-1} G(p_i)$ 。 $G(m)$ 表示在模数为 m 时的循环节大小。

所以问题转为模素数如何处理。即给定 $a, b, f(1), f(2)$ ，且满足 $f(n) = a * f(n-1) + b * f(n-2)$ 。求 $f(n) \bmod p$ 的循环节。

写成矩阵形式，如下：

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} a & b \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix}$$

变形一下：

$$\begin{bmatrix} f(n+2) \\ f(n+1) \end{bmatrix} = \begin{bmatrix} a & b \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f(2) \\ f(1) \end{bmatrix}$$

那么现在的问题就转化为求最小的 n ，使得

$$\begin{bmatrix} a & b \\ 1 & 0 \end{bmatrix}^n \pmod{p} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

如何快一点求 n 呢？这里直接给出结论：设 $c = a^2 + 4b$ ，有两种情况：

1. 若 c 是模 p 的二次剩余，则 n 是 $p-1$ 的因子；
2. 若 c 不是模 p 的二次剩余，则 n 是 $(p+1)(p-1)$ 的因子

所以只要枚举因子并判断即可。时间复杂度 $O(T \cdot 2^3 \cdot \log(p))$ ，其中 T 为 $p-1$ 或 $(p+1)(p-1)$ 的因子数目。

7.3.2 分数小数的循环节

问题 给定一个分数 $\frac{p}{q}$ ，将其转化为小数。

解 考虑一个分数，有以下三种情况：

- 整数
- 有限小数
- 无限循环小数

先来考虑下无限循环小数，我们先将 p 和 q 公共的因子除去，然后利用循环这个性质，可知存在 i, j 满足， $p \cdot 10^i \equiv p \cdot 10^j \pmod{q}$, where $j > i \geq 0$ 。这里 i 代表循环出现前有多少位数， $j-i$ 表示循环节长度。化简后得 $p \cdot 10^i \cdot (10^{j-i} - 1) \equiv 0 \pmod{q}$ 。也就是说， $q \mid p \cdot 10^i \cdot (10^{j-i} - 1)$ 。由于 $10^{j-i} - 1$ 中一定不存在因子 2 和 5，所以要想是 q 的倍数，2 和 5 的贡献均来自 10^i 。于是我们记录 q 中 2 的次幂数为 num2 ，5 的次幂数为 num5 ，则 $i = \max(\text{num2}, \text{num5})$ 。记 q 除去所有 2 和 5 因子后，为 m ，则 $10^{j-i} \equiv 1 \pmod{m}$ 。然后我们解同余方程 $10^x \equiv 1 \pmod{m}$, where $x > 0$ ， j 就等于 $x+i$ 。至于这个方程， x 一定是 $\varphi(m)$ 的因子，枚举因子判定即可。

如果 j 无解，表示该小数为有限小数， i 也就表示了小数点后共有几位；如果 j 有解，含义如上。

luogu P1530 分数化小数

时间复杂度： $\sqrt{q} \cdot \log(q)$ （确定 i, j ）

输出格式：按照下面规则，如果结果长度大于 76，每行输出 76 个字符。

- $2/2 = 1.0$
- $3/8 = 0.375$
- $45/56 = 0.803(571428)$

```
typedef long long ll;
2 ll gcd(ll a, ll b){if(b==0) return a; return gcd(b,a%b);}
ll fast_exp(ll a,ll b,ll c){
```



```

4    ll res=1;a=a%c;
    while(b){
6        if(b&1) res=(res*a)%c;
        b>>=1; a=(a*a)%c;
8    }
    return res;
10 }
ll phi(ll x)
12 {
    if(x==1) return 1;
14    ll res=x;
    for(int i=2;i*i<=x;++i){
16        if(x%i==0){
            res-=res/i;
18            do{
                x/=i;
20            }while(x%i==0);
        }
22    }
    if(x>1) res-=res/x;
24    return res;
}
26 //返回 循环节前面有多少位 以及 最小循环节长度
    //ans的size为1表示有限小数
28 //ans的size为2表示无限循环小数
vector<ll> solve(ll p, ll q)
30 {
    p = p%q;
32    ll d = gcd(p,q);
    p/=d;
34    q/=d;
    vector<ll> ans;
36    ll m=q;    //看q里有多少个2和5
    int num2=0, num5=0;
38    while(!(m&1)){
        m>>=1; num2++;
40    }
    while(m%5==0){
42        m/=5; num5++;
    }
44    ll i = max(num2,num5),j=-1;
    //10^{j-i} \equiv 1 (mod m)
46    //so solve 10^x = 1 (mod m) and j=x+i

```

```
    ll varphi = phi(m);
48 vector<ll> fac1; //从小到大
    vector<ll> fac2; //从大到小
50 for(ll x=1; x*x<=varphi; ++x){
        if(varphi%x==0){
52             fac1.push_back(x);
             fac2.push_back(varphi/x);
54         }
    }
56 reverse(fac2.begin(), fac2.end());
    fac1.insert(fac1.end(), fac2.begin(), fac2.end());
58 for(int k=0; k<fac1.size(); ++k){
        if(fast_exp(10, fac1[k], m)==1){
60             j = i + fac1[k];
             break;
62         }
    }
64 ans.push_back(i); //循环节前面有多少位
    if(j==-1) return ans; //ans的size为1表示有限小数
66 ans.push_back(j-i); //最小循环节长度
    return ans; //ans的size为2表示无限循环小数
68 }

int main()
70 {
    stringstream ss;
72     streambuf* buffer = cout.rdbuf(); //oldbuffer, STDOUT的缓冲区
    cout.rdbuf(ss.rdbuf());

74
    ll p, q;
76     cin >> p >> q;
    ll zheng = p/q;
78     p = p%q;
    vector<ll> ans = solve(p, q);
80     cout << zheng; //整数部分

82     if(p==0){
        cout << ".0"; //视题目情况 这一题是整除时输出".0"
84         assert(ans[0]==0); //整除时一定是0
    }
86     else cout << ".";

88     int num1=ans[0]; //循环节前面有多少位
    while(num1){
```

```

90     cout<<p*10/q;
      p = p*10%q;
92     num1--;
    }
94     if(ans.size()>1){
        cout<<"(";
96         int num2 = ans[1];
        while(num2){
98             cout<<p*10/q;
            p = p*10%q;
100            num2--;
        }
102        cout<<")";
    }
104    string out(ss.str());
    cout.rdbuf(buffer); //重新载入STDOUT的缓冲区
106
    for(int i=0;i<out.length();++i){
108        if(i%76==0 && i>0){
            cout<<endl;
110        }
        cout<<out[i];
112    }
    cout<<endl;
114    return 0;
}
116 /*
    1/3 = 0.(3)
118 22/5 = 4.4
    1/7 = 0.(142857)
120 2/2 = 1.0
    3/8 = 0.375
122 45/56 = 0.803(571428)
    */

```

code07/fraction2decimal.cpp



注意 由于上面这个题目要求输出时每行只要 76 个字符，我在写代码时一开始没有注意，因此直接使用的 cout。所以代码中我使用了下面的方法：

```

1 #include<bits/stdc++.h>
  using namespace std;
3 int main()
  {

```

```

5   stringstream ss;
   streambuf* buffer = cout.rdbuf(); //oldbuffer, STDOUT的缓冲区
7   cout.rdbuf(ss.rdbuf()); //redirect

9   cout<<"number theory"<<endl;
   cout<<"ok";

11

   string out(ss.str());
13   cout.rdbuf(buffer); //重新载入STDOUT的缓冲区

15   cout<<out<<endl;
   return 0;
17 }

```

code07/redirect-cout.cpp

先将 cout 重定向到 stringstream，然后从其中取出 string，最后将 cout 还原到 stdout。

7.4 DFS Similar

dfs similar 是指一类可以“暴力”搜索的问题，这类问题往往有很好的性质使得暴力的时间不会太长。

7.4.1 Counting Sequences I(18 上海网络赛 D)

问题 我们定义一个由正整数组成的序列 a_1, a_2, \dots, a_n 是好的当：

- $n \geq 2$
- $a_1 + \dots + a_n = a_1 * \dots * a_n$

请输出有多少种这样的序列， $2 \leq n \leq 3000$ 。

解 考虑枚举序列中非 1 的数 (非降序地)，边枚举边统计不同长度序列的答案。直觉复杂度较低。可以证明一下，枚举的数的大小不超过 3000。

```

1  const int mod = 1e9+7;
   ll ans[3010];
3  ll help[3010];
   ll helpni[3010];
5  stack<int> sta;
   int up = 3000;
7  int n=3000;
   ll fast_exp(ll a,ll b,ll c){
9     ll res=1;a=a%c;
     while(b)
11    {

```

```
        if(b&1) res=(res*a)%c;
13        b>>=1;a=(a*a)%c;
    }
15    return res;
}
17 void solve(int len)
{
19    int len2 = sta.size();
    assert(len2>1);
21    int len1 = len-len2;
    ll res = help[len]*helpni[len1]%mod;
23    stack<int> sta_tp = sta;
    vector<int> vec;
25    while(!sta_tp.empty()){
        vec.push_back(sta_tp.top());
27        sta_tp.pop();
    }
29    vec.push_back(-1);
    int num=1;
31    for(int i=1;i<vec.size();++i){
        if(vec[i]==vec[i-1]) num++;
33        else{
            res = res*helpni[num]%mod;
35            num=1;
        }
37    }
    ans[len] = (ans[len] + res)%mod;
39 }

41 void dfs(int len, ll sum, ll jie)
{
43    ll tp = jie-sum+len; //tp为序列长度
    if(tp<=n){
45        if(tp>=3){
            //1的数量: jie-sum >1的数量: len
47            //ans[tp]++;
            //取当前的sta 然后计算
49            solve(tp);
        }
51        else ;
    }
53    else return;
    int low;
```

```

55     if(sta.empty()) low = 2;
        else low = sta.top();
57     for(int i=low;i<=up;++i){
            sta.push(i);
59         dfs(len+1, sum+i,jie*i);
            sta.pop();
61     }
    }
63 void init()
    {
65         help[1] = 1;helpni[1] = 1;
        for(int i=2;i<=3000;++i){
67             help[i] = help[i-1]*i%mod;
            helpni[i] = fast_exp(help[i], mod-2,mod);
69         }
    }
71 int main()
    {
73         init();
        dfs(0,0,1);
75         int t;
        cin>>t;
77         while(t-->0)
        {
79             int m;
            cin>>m;
81             if(m==2) cout<<1<<endl;
            else cout<<ans[m]<<endl;
83         }
    }

```

code07/counting-sequences.cpp

7.4.2 反欧拉函数 (洛谷 P4780)

问题 求最小的正整数 x ，使得 $\phi(x) = n$ 。输出 x ，如果 $x \geq 2^{31}$ 或者不存在则输出 -1 。

解 枚举 x 的可能质因子 p ，则必须有 $n\%(p-1) = 0$ ，同时再枚举该质因子的指数，此时要求每有几个 p （指数）， n 就要能整除 p 几次。

时间复杂度有点迷，但感觉再大点还是可以做的。

```

const int maxn=1e5+10;
2 bool valid[maxn];
int ans[maxn]; //素数

```

```

4 int tot;
  ll all;
6 void get_prime(int n)
{
8     tot=0;
    memset(valid,true,sizeof(valid));
10    valid[1]=false;
    for(int i=2;i<=n;++i){
12        if(valid[i]) ans[++tot]=i;
        for(int j=1;(j<=tot) && (i*ans[j]<=n) ;++j){
14            valid[i*ans[j]]=false;
            if(i%ans[j]==0) break;
16        }
    }
18 }

bool is_prime(ll x){
20     for(int i=2;i<=floor(sqrt(x));++i)
        if(x%i==0) return false;
22     return true;
}

24 ll up;
void dfs(int last,ll num,ll phi){//对num进行分解 last是最后一次选到第几个
26     if(num==1){
        all=min(all,phi);
28         return ;
    }

30     if(num>up && is_prime(num+1)){//如果当前因子num比\sqrt{n}大
        all=min(all,phi*(num+1));
32         return ;
    }

34     for(int i=last+1;i<=tot && ans[i]<=num;i++)
        if(num%(ans[i]-1)==0){//如果num能整除当前素数-1
36             ll num_ = num/(ans[i]-1);//除去
            ll phi_ = phi*ans[i];//搞上
38             dfs(i,num_,phi_);
            while(num_%ans[i]==0){//同时phi可以有更多的该素数
40                 num_/=ans[i];
                phi_*=ans[i];
42                 dfs(i,num_,phi_);
            }
44         }
    }
46 int main()

```

```

{
48     get_prime(1e5); //1e5很稳
        ll n;
50     cin>>n;
        up=floor(sqrt(n)); //注意至少要加1
52     all=1LL<<31;
        dfs(0,n,1);
54     if(all<(1LL<<31)) cout<<all<<endl; //本题的特殊要求
        else cout<<-1<<endl;
56     return 0;
}

```

code07/Anti-Euler.cpp

7.4.3 因子个数最多的数 (51nod1060)

问题 把一个数的约数个数定义为该数的复杂程度，给出一个 n ，求 $1 \sim n$ 中复杂程度最高的那个数。如果有多个数复杂度相等，输出最小的。 $1 \leq n \leq 10^{18}$ 。

即给定 N ，求 $[1, N]$ 之间最大的反素数 (即拥有因子数目最多的数)。

解 性质：

- 一个反素数的质因子们必然是从 2 开始连续的质数。
- $p = 2^{t_1} * 3^{t_2} * 5^{t_3} * 7^{t_4} \dots$ 必然 $t_1 \geq t_2 \geq t_3 \geq \dots$

暴力 dfs，时间复杂度大概几个 \log ? (不会分析.jpg)

```

1 //51nod 1060
    typedef unsigned long long ll;
3 int prime[60];
    bool is_prime(ll x)
5 {
        for(int i=2;i*i<=x;++i){
7             if(x%i==0) return false;
        }
9     return true;
    }
11 int tot;
    void init()
13 {
        tot=0;
15     for(int i=2;i<2000&&(tot<20);++i){
            if(is_prime(i)){
17                 prime[++tot]=i;
            }
19     }

```



```

    }
21 ll ans,res;
    ll up;
23 void dfs(int last,ll cur,ll num,int pre)//last 是最后一个访问的素数下标  cur是当
    前数 num是当前数的约数个数 pre是上一次的指数
    {
25     if(last>=tot) return ;
        if(num>ans){
27         res=cur;//当前数
            ans=num;//约数个数
29     }
        else if(num==ans){
31         res=min(res,cur);
        }
33     for(int i=1;i<=61 && i<=pre;++i){//枚举指数
        if(cur<=up/prime[last+1]){
35             //cout<<cur<<endl;
            cur*=prime[last+1];
37             dfs(last+1,cur,num*(i+1),i);
        }
39         else break;
    }
41 }
int main()
43 {
    init();
45     //cout<<tot<<endl;
    //for(int i=1;i<=tot;++i) cout<<prime[i]<<" ";
47     int t;
    cin>>t;
49     while(t-->0)
    {
51         ans=0;
        cin>>up;
53         dfs(0,1,1,61);
        cout<<res<<" "<<ans<<endl;//该数和因子数目 若因子数目相同则输出最小的数
55     }
    return 0;
57 }

```

code07/mostfactors.cpp

7.5 FFT 与 NTT

7.5.1 FFT

FFT 在算法竞赛中的主要应用之一是加速多项式乘法的计算。


7.5.1.1 多项式

- 多项式的系数表示：

$$A(x) = \sum_{i=0}^{n-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-1} x^{n-1}$$

- 多项式的点值表示：

对于上面的 $n-1$ 次多项式，将一组 (n 个) 互不相同的 x 带入进去得到对应的 y (也就是 n 个点)。这 n 个点可以唯一确定一个多项式。

 **注意** 由多项式的点值表示转系数表示，需要进行多项式插值，朴素的插值算法时间复杂度为 $O(n^2)$ 。


7.5.1.2 多项式乘法

有两个多项式， $A(x) = \sum_{i=0}^{n-1} a_i x^i$ 和 $B(x) = \sum_{i=0}^{n-1} b_i x^i$ (假设两个多项式次数相同，若不同可在后面补零)

则

$$C(x) = A(x) * B(x) = \sum_{k=0}^{2n-2} \left(\sum_{i+j=k} a_i b_j \right) x^k$$

两个 $n-1$ 次多项式相乘，得到的是一个 $2n-2$ 次多项式，时间复杂度为 $O(n^2)$ 。

 **注意** 另外，也可以用点值表达，即选择 $2n-1$ 个互不相同的 x_i 带入 $A(x)$ 和 $B(x)$ 相乘，得到 $2n-1$ 个值，这 $2n-1$ 个点值就唯一确定了这个多项式，时间复杂度 $O(n)$ (注意只是得到点值表达式)。

7.5.1.3 复数

设 a 、 b 为实数， $i^2 = -1$ ，形如 $a + bi$ 的数叫做复数，其中 i 被称为虚数单位。复数域是已知最大的域。

7.5.1.4 复平面

在复平面中， x 轴代表实数、 y 轴 (除原点外的所有点) 代表虚数。每一个复数 $a + bi$ 对应复平面上一个从 $(0, 0)$ 指向 (a, b) 的向量。

该向量的长度 $\sqrt{a^2 + b^2}$ 叫做模长。

从 x 轴正半轴到该向量的转角的有向 (以逆时针为正方向) 角叫做幅角。

复数相加遵循平行四边形定则。

复数相乘时，模长相乘，幅角相加。

7.5.1.5 单位根

下文中，如不特殊指明，均取 n 为 2 的正整数次幂。

在复平面上，以原点为圆心，1 为半径作圆，所得的圆叫做**单位圆**。以原点为起点，单位圆的 n 等分点为终点，作 n 个向量。设所得的**幅角为正且最小**的向量对应的复数为 ω_n ，称为 n 次单位根。

由复数乘法的定义（模长相乘，幅角相加）可知，其余的 $n-1$ 个向量对应的复数分别为 $\omega_n^2, \omega_n^3, \dots, \omega_n^n$ ，其中 $\omega_n^n = \omega_n^0 = 1, \omega_n^1 = \omega_n$ 。

单位根的幅角为圆周角的 $\frac{1}{n}$ ，这为我们提供了一个计算单位根及其幂的公式：

$$\omega_n^k = \cos(k \frac{2\pi}{n}) + i \sin(k \frac{2\pi}{n})$$

7.5.1.6 单位根的性质

- $\omega_{2n}^{2k} = \omega_n^k$

证明 显然

- $\omega_n^{k+\frac{n}{2}} = -\omega_n^k$

证明 因为 $\omega_n^{\frac{n}{2}} = -1$ （带入公式即可验证）

7.5.1.7 离散傅里叶变换 (DFT)

对于一个 n 个系数， $n-1$ 次的多项式，考虑多项式 $A(x)$ 的表示。

将 n 次单位根的 0 到 $n-1$ 次幂（共 n 个）带入多项式的系数表示，所得点值向量为：

$$\{A(\omega_n^0), A(\omega_n^1), A(\omega_n^2), \dots, A(\omega_n^{n-1})\}$$



注意 变换后的这 n 个点，可以唯一确定这个多项式。

称这个结果 $(A(\omega_n^0), A(\omega_n^1), A(\omega_n^2), \dots, A(\omega_n^{n-1}))$ 为其系数向量 $(a_0, a_1, a_2, \dots, a_{n-1})$ 的**离散傅里叶变换**。

按照朴素方法依次带进来求解原系数的离散傅里叶变换，时间复杂度仍为 $O(n^2)$ 。

而这个过程可以分治进行，因而可以优化，即 **FFT**，这是算法竞赛的重点。（但此时还不知道这 n 个点求多项式乘法有什么用，继续看）

定义 7.2. 从信号的角度看

如果从信号的角度看，这个过程就是**时域到频域的转换**，即选择一段时间取样 N 个点，即 0 时刻，1 时刻， \dots ， $N-1$ 时刻。转换的思路是这样的：我去“枚举”一些频率，对于一个固定的频率 f ，去对时域图像（假设为 $x(t)$ ）做“缠绕操作”，而缠绕的时间取样就是上面的 N 个时刻，即 $X(f) = \sum_{n=0}^{N-1} x(n)w_N^{fn}$ (n 是每个时间点)。简单来说，缠绕操作就是对一个无明显频率规律的时域图像，看它在指定（枚举的）

频率上是否有这个频率，衡量的指标就是 X 。（至于缠绕操作，就是一种方便的工具，使得 $x(n)$ 分布在特定的角度上）。所以我们要枚举频率，一般也取 $0 \sim N-1$ 。我们对 $X(f) = \sum_{n=0}^{N-1} x(n)w_N^{fn}$ 简单变个形：

$$X(f) = \sum_{n=0}^{N-1} x(n)(w_N^f)^n$$

再看下朴素的多项式

$$A(x) = \sum_{i=0}^{n-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-1} x^{n-1}$$

即 f 分别取 $0, 1, 2, \dots, n-1$ ，就是多项式 $A(x)$ 带入 $x = w_n^0, w_n^1, \dots, w_n^{n-1}$ 。

而多项式的系数 a_0, a_1, \dots, a_{n-1} 就对应 $x(0), x(1), \dots, x(n-1)$ (即信号在时域离散时间点上的 n 个强度值)。



7.5.1.8 快速傅里叶变换 (FFT)

考虑将多项式按照系数下标的奇偶分为两部分

$$A(x) = (a_0 + a_2 x^2 + a_4 x^4 + \dots + a_{n-2} x^{n-2}) + (a_1 x + a_3 x^3 + a_5 x^5 + \dots + a_{n-1} x^{n-1})$$

令

$$A_1(x) = a_0 + a_2 x + a_4 x^2 + \dots + a_{n-2} x^{\frac{n}{2}-1}$$

$$A_2(x) = a_1 + a_3 x + a_5 x^2 + \dots + a_{n-1} x^{\frac{n}{2}-1}$$

则有

$$A(x) = A_1(x^2) + x A_2(x^2)$$

假设 $k < \frac{n}{2}$ ，现在要求 $A(\omega_n^k)$ ，则根据单位根的性质 1，有：

$$A(\omega_n^k) = A_1(\omega_{\frac{n}{2}}^k) + \omega_n^k A_2(\omega_{\frac{n}{2}}^k)$$

由于前面 $k < \frac{n}{2}$ ，对于 $A(\omega_n^{k+\frac{n}{2}})$ 的部分，使用单位根的性质 1 和性质 2，有：(这个指数范围的变化是精华)

$$A(\omega_n^{k+\frac{n}{2}}) = A(-\omega_n^k) = A_1(\omega_{\frac{n}{2}}^k) - \omega_n^k A_2(\omega_{\frac{n}{2}}^k)$$

这样，当 k 取遍 $[0, \frac{n}{2}-1]$ 时， k 和 $k+\frac{n}{2}$ 取遍了 $[0, n-1]$ ，即全部所求。

那么，如果已知 $A_1(x)$ 和 $A_2(x)$ 在 $\omega_{\frac{n}{2}}^0, \omega_{\frac{n}{2}}^1, \dots, \omega_{\frac{n}{2}}^{\frac{n}{2}-1}$ 的值，就可以在 $O(n)$ 时间内求得 $A(x)$ 在 $\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$ 处的取值。而关于 $A_1(x), A_2(x)$ 的问题正好是原问题规模缩小一半的子问题，分治的边界为一个常数项 a_0 ，即 $A_\phi(x)$ 的系数只有一项。

则该分治算法的时间复杂度为

$$T(n) = 2 * T(n/2) + O(n) = O(n \log n)$$

上述过程称为 *Cooley-Tukey 算法* (JW Cooley 和 John Tukey)。

现在我们可以 $O(n \log n)$ 时间内求得 n 个点对啦！

即 $(\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1})$ 和 $(A(\omega_n^0), A(\omega_n^1), A(\omega_n^2), \dots, A(\omega_n^{n-1}))$ 。



然而还是不知道和多项式乘法有什么关系.....

事实上, 这 n 对取值, 将作为“原料”, 帮助我们反过来去求多项式的系数。

和一般的插值不同 ($O(n^2)$), 这些特殊的点对可以在 $O(n \log n)$ 内求出多项式的系数。

即下面的傅里叶逆变换, 仿佛离最终目标进了一大步.....

7.5.1.9 傅里叶逆变换

将点值表示的多项式转化为系数表示, 同样可以使用快速傅里叶变换, 这个过程叫做傅里叶逆变换。

设 $(y_0, y_1, y_2, \dots, y_{n-1})$ 为 $(a_0, a_1, a_2, \dots, a_{n-1})$ 的离散傅里叶变换。考虑另一个向量 $(C_0, C_1, C_2, \dots, C_{n-1})$, 满足

$$C_k = \sum_{i=0}^{n-1} y_i (\omega_n^{-k})^i, k \in [0, n-1]$$

即多项式 $B(x) = y_0 + y_1x + y_2x^2 + \dots + y_{n-1}x^{n-1}$ 在 $\omega_n^0, \omega_n^{-1}, \omega_n^{-2}, \dots, \omega_n^{-(n-1)}$ 处的点值表示。其中 $B(x)$ 以原系数 a_i 的离散傅里叶变换作为新的系数。

将 C_k 展开:

$$\begin{aligned} C_k &= \sum_{i=0}^{n-1} y_i (\omega_n^{-k})^i \\ &= \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j (\omega_n^i)^j \right) (\omega_n^{-k})^i \\ &= \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j (\omega_n^j)^i \right) (\omega_n^{-k})^i \\ &= \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j (\omega_n^j)^i (\omega_n^{-k})^i \right) \\ &= \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j (\omega_n^{j-k})^i \right) \\ &= \sum_{j=0}^{n-1} a_j \left(\sum_{i=0}^{n-1} (\omega_n^{j-k})^i \right) \end{aligned}$$

观察内层求和, 为了解决这个求和式, 先考虑一个式子先。令首项为 1, 公比为 ω_n^k 的等比数列前 n 项和为 $S(\omega_n^k)$:

$$S(\omega_n^k) = 1 + \omega_n^k + (\omega_n^k)^2 + \dots + (\omega_n^k)^{n-1}$$

- 当 $k \neq 0$ 时, 两边同时乘上 ω_n^k , 得

$$\omega_n^k S(\omega_n^k) = \omega_n^k + (\omega_n^k)^2 + (\omega_n^k)^3 + \dots + (\omega_n^k)^n$$

上面两式相减, 整理后得

$$\omega_n^k S(\omega_n^k) - S(\omega_n^k) = (\omega_n^k)^n - 1$$

$$S(\omega_n^k) = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1}$$

这个式子分子为 0，分母一定不为 0，因此

$$S(\omega_n^k) = 0$$

- 当 $k = 0$ 时， $S(\omega_n^k) = n$

继续考虑 C_k ，

$$\begin{aligned} C_k &= \sum_{j=0}^{n-1} a_j \left(\sum_{i=0}^{n-1} (\omega_n^{j-k})^i \right) \\ &= \sum_{j=0}^{n-1} a_j S(\omega_n^{j-k}) \end{aligned}$$

对每一个 k ，枚举 j ，只有当 $j = k$ 时， $S(\omega_n^{j-k}) = n$ ，否则 $S(\omega_n^{j-k}) = 0$ ，即

$$C_i = na_i$$

$$a_i = \frac{1}{n} C_i$$

也就是说，使用原系数的离散傅里叶变换结果作为新的系数，单位根的倒数代替单位根代入，做一次快速傅里叶变换的过程，再将结果每个数除以 n ，即为傅里叶逆变换的结果，即原系数 a_i 。

这样就可以先做两次正变换再做一次逆变换求得系数啦。

简单来说，求解多项式乘法的大致思路就是：

1. 确定结果多项式 $C(x)$ 的次数，决定要取的点对的数量 N (N 为 2 的次幂)；
2. $O(n \log n)$ 求 $A(x)$ 和 $B(x)$ 在这 N 个点的值，然后依次相乘得到结果多项式在这 N 个点的值；
3. 做傅里叶逆变换求得系数值，即为结果多项式 $C(x)$ 的系数。

7.5.1.10 小结与实现细节

DFT （离散傅里叶变换）是一种对 n 个元素的数组的变换，直接代入依次求解的方法是 $O(n^2)$ 的。但是用分治的方法是 $O(n \log n)$ ，即 FFT （快速傅里叶变换）。

由于 DFT 满足 cyclic convolution（循环卷积）的性质，即定义 $h = a(*)b$ 为 $h_r = \sum_{x+y=r} a_x b_y$ （多项式乘法， h 为结果多项式），则有 $DFT(a(*)b) = DFT(a) \cdot DFT(b)$ ，右边为向量点乘。

所以所求 $a(*)b = DFT^{-1}(DFT(a) \cdot DFT(b))$ ，即只要对 a, b 向量分别进行 DFT ，所得的两个向量点乘之后再逆变换就可以了。



注意

- 由于 FFT 本身算法的要求， n 是 2 的次幂，注意补 0；
- DFT 是定义在复数域上的，有与整数之间变换的要求；
- 不要忘记最后除 n ；

- C++ 的 *STL* 在头文件 'complex' 中提供一个复数的模板实现 'std::complex<T>'，其中 T 为实数类型，一般取 'double'，在对精度要求较高的时候可以使用 'long double' 或 '__float128'（不常用）；
- 单位根的倒数等于其共轭复数，使用 'std::conj()' 取得 *IDFT* 所需的单位根的倒数；
- 对时间要求苛刻时，可以自己实现复数类以提高速度。

7.5.1.11 FFT 的递归实现

直接按照上述思路实现即可，使用 C++ 已经封装的 Complex。[uoj-34 多项式乘法](#)

	结果	用时	内存	语言	文件大小
	100	2501ms	22016kb	C++	1.5kb

图 7.1: fft-recursion

```

1 #include<bits/stdc++.h>
  using namespace std;
3 typedef long long ll;
  const double pi=acos(-1.0);
5 const int maxn=1e5;
  typedef complex<double> xu;
7 xu a[maxn<<2],b[maxn<<2];
  int inver;
9 void FFT(xu *s ,int n){
    if(n==1) return ;//n=1时值就是系数 因为只有w_1^0这一点
11 xu a1[n>>1],a2[n>>1];
    for(int i=0;i<n;i+=2){
13     a1[i>>1]=s[i];
     a2[i>>1]=s[i+1];
15   }
    FFT(a1,n>>1); FFT(a2,n>>1);
17 xu w=xu(cos(2*pi/n),inver*sin(2*pi/n));
    xu wn=xu(1,0);
19 for(int i=0;i<(n>>1);++i,wn=wn*w){
    s[i]=a1[i]+wn*a2[i];
21     s[i+(n>>1)]=a1[i]-wn*a2[i];
    }
23 }
  int main()
25 {
    #ifndef ONLINE_JUDGE
27     freopen("in.txt","r",stdin);
    #endif

```

```

29   ios::sync_with_stdio(false);
    int n,m;
31   cin>>n>>m;
    int tp;
33   n++;
    m++;
35   for(int i=0;i<n;++i){
        cin>>tp;
37       a[i]=xu(tp,0);
    }
39   for(int i=0;i<m;++i){
        cin>>tp;
41       b[i]=xu(tp,0);
    }
43   int N=1; while(N<n+m-1) N<<=1;
    inver=1;
45   FFT(a,N); FFT(b,N);
    for(int i=0;i<N;++i) a[i]=a[i]*b[i];
47   //现在a中的值是上面的傅里叶变化的结果啦 wn0,wn1,wn2,...,wn n-1
    //下面作为逆变化的系数
49   inver=-1;
    FFT(a,N);
51   for(int i=0;i<n+m-1;++i) cout<<ll(a[i].real()/N+0.5)<<' ';
    return 0;
53 }

```

code07/fft-recursion.cpp

7.5.1.12 FFT 的迭代实现

递归实现的 *FFT* 效率不高，因为有栈的调用和参数的传递，实际中一般采用迭代实现。

二进制位翻转

对分治规律的总结：这为迭代实现提供了理论基础。

蝴蝶操作

考虑合并两个子问题的过程，假设 $A_1(w_n^k)$ 和 $A_2(w_n^k)$ 分别存放在 $a[k]$ 和 $a[\frac{n}{2} + k]$ 中， $A(w_n^k)$ 和 $A(w_n^{k+\frac{n}{2}})$ 将要存放在 $b[k]$ 和 $b[\frac{n}{2} + k]$ 中，合并的操作可以表示为：

$$\begin{aligned}
 b[k] &\leftarrow a[k] + w_n^k * a[\frac{n}{2} + k] \\
 b[k + \frac{n}{2}] &\leftarrow a[k] - w_n^k * a[\frac{n}{2} + k]
 \end{aligned}$$

考虑加入一个临时变量 t ，使得这个过程可以在原地完成，而不需要数组 b ，

$$t \leftarrow w_n^k * a[\frac{n}{2} + k]$$


```

1 //以8项为例
2 000 001 010 011 100 101 110 111
3 0   1   2   3   4   5   6   7 //初始要求
4 0   2   4   6 / 1   3   5   7 //分治后的两个多项式的系数 对应于原多项式
5 0   4 / 2   6 / 1   5 / 3   7
6 0 / 4 / 2 / 6 / 1 / 5 / 3 / 7
7 000 100 010 110 001 101 011 111 //发现正好是翻转

```

图 7.2: 分治的规律

$$a[k + \frac{n}{2}] \leftarrow a[k] - t$$

$$a[k] \leftarrow a[k] + t$$

由于 k 和 $k + \frac{n}{2}$ 是对应的, 所以不同的 k 之间不会相互影响。

这一过程被称为蝴蝶操作。

	结果	用时	内存	语言	文件大小	
	100	976ms	26336kb	C++	2.0kb	

图 7.3: fft-iteration

```

1 #include<bits/stdc++.h>
  using namespace std;
3 typedef long long ll;
  const double pi=acos(-1.0);
5 const int maxn=1e5;
  typedef complex<double> xu;
7 xu omega[maxn<<2],omegaInverse[maxn<<2];//辅助
  void init(const int n){
9     for(int i=0;i<n;++i){
        omega[i]=xu(cos(2*pi/n*i),sin(2*pi/n*i));
11        omegaInverse[i]=conj(omega[i]);
    }
13 }
  void trans(xu *a,const int n,const xu *omega){//a是系数 n是2的次幂 omega是要带
    入的点
15     int k=0;
        while((1<<k)<n) k++;
17     //看N是2的多少次幂
        for(int i=0;i<n;++i){
19         int t=0;
            for(int j=0;j<k;++j) if(i&(1<<j)) t|=(1<<(k-j-1));
21         if(i<t) swap(a[i],a[t]);//原地翻转

```

```

    }
23   for(int l=2;l<=n;l*=2){//l=1不用求 就是系数
        int m=l/2;
25   for(xu *p=a;p!=a+n;p+=l){//l为一段要求 由两段l/2合并得到
        //当前处理[p,p+l)
27   for(int i=0;i<m;++i){
            //蝴蝶操作 omega_{l}^{i}=omega_{N}^{N/l*i}
29   xu t=omega[n/l*i]*p[i+m];
            p[i+m]=p[i]-t;
31   p[i]+=t;
        }
33   }
    }
35 }

void dft(xu *a,const int n){
37   trans(a,n,omega);
}

39 void idft(xu *a,const int n){
    trans(a,n,omegaInverse);
41   for(int i=0;i<n;++i) a[i]/=n;
}

43 xu a[maxn<<2], b[maxn<<2];
int main()
45 {
    #ifndef ONLINE_JUDGE
47   freopen("in.txt","r",stdin);
    #endif
49   ios::sync_with_stdio(false);
    int n,m;
51   cin>>n>>m;
    n++;
53   m++;
    int tp;
55   for(int i=0;i<n;++i){
        cin>>tp;
57   a[i]=xu(tp,0);
    }
59   for(int i=0;i<m;++i){
        cin>>tp;
61   b[i]=xu(tp,0);
    }
63   int N=1;
    while(N<m+n-1) N<<=1;

```

```

65     init(N);
        dft(a,N);
67     dft(b,N);
        for(int i=0;i<N;++i) a[i]*=b[i];
69     idft(a,N);
        for(int i=0;i<n+m-1;++i) cout<<(ll)(a[i].real()+0.5)<<" ";
71     return 0;
    }

```

code07/fft-iteration.cpp

7.5.2 NTT

由于 FFT 涉及到复数运算, 难免有精度问题, 在计算一些高精度乘法的时候就有可能由于精度出现错误, 这便让我们考虑是否有在模意义下的方法, 这就是快速数论变换 (*Fast Number – Theoretic Transform, NTT*)。

首先来看 FFT 中能归并地变换用到了单位根 ω_n 的什么性质:

1. $\omega_n^n = 1$;
2. $\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$ 是互不相同的, 这样代入计算出来的点值才可以用来还原出多项式的系数;
3. $\omega_n^{k+\frac{n}{2}} = -\omega_n^k, \omega_{2n}^{2k} = \omega_n^k$, 这使得问题规模得以减半;
4. $\sum_{i=0}^{n-1} (\omega_n^{j-k})^i = \begin{cases} 0, & j \neq k \\ n, & j = k \end{cases}$, 这使得可以用同样的优化方法加速逆变换。

下面我们要在模意义下寻找满足这些性质的数!

7.5.2.1 原根

由第三章中知一个素数 p 的原根 g 满足 $g^0, g^1, \dots, g^{p-2} \pmod{p}$ 均不相同。对于一个素数 p , 将其写成 $p = k * 2^m + 1$ 的形式, 记 $2^m = n$, 有 $g^{kn} \equiv 1 \pmod{p}$ 。我们记 $g_n = g^k$, 则有 $g_n^n \equiv 1 \pmod{p}$ 。由于 g 是 p 的原根, 则 $g_n^0, g_n^1, \dots, g_n^{n-1}$ 互不相同。

到这里我们发现, 在模 p 意义下, p 的原根 g 的 k 次方 g_n , 具有和单位根 ω_n 一样的性质! 上面已经说明了其具有性质 1 和性质 2。下面说明性质 3。

由于 $g_n^n \equiv 1 \pmod{p}$, 所以 $g_n^{\frac{n}{2}} \equiv 1 \text{ or } -1 \pmod{p}$ 。又由于 g 是原根, 则 $g_n^{\frac{n}{2}} \equiv -1 \pmod{p}$ 。所以 $g_n^{k+\frac{n}{2}} \equiv -g_n^k$ 得证。至于 $g_{2n}^{2k} \equiv g_n^k$, 只需证 $g_n^2 \equiv g_{\frac{n}{2}}$ 。这是比较显然的, 因为 $k = \frac{p-1}{2^m}$ 。

对于性质 4, 证明方法类似, 大家可以自行尝试。

因此在 NTT 中, 我们可以用原根 g 的 k 次方 g_n 代替 FFT 中的单位根 ω_n 。

7.5.2.2 NTT 的实现细节

在 FFT 中, 需要预处理 N 项单位根的次幂及其共轭, 对应地, NTT 中也需要。即预处理 $g_n^0, g_n^1, \dots, g_n^{n-1}$, 以及对应的模 p 意义下的逆元。

NTT 的代码相比 FFT 的代码变动很少。

注意

1. 最后逆变换完成后除 n 时, 变为乘上 n 关于 p 的逆元即可;
2. $p = k * 2^m + 1$ 中, 最大能取到的 2^m 为能处理的结果多项式项数极限。如对于素数 $998244353 = 119 * 2^{23} + 1$, 最多能处理的 $N = 2^{23}$ 。当 $N < 2^{23}$ 时, 要将多余的部分乘到 k 上; (常见的素数及其原根见附录表)
3. 由于实数域上的运算比复数域运算要快, 所以 NTT 的运行时间理论上会比 FFT 少; (但由于有取模操作在, 所以上述说明是玄学.....)
4. 如果一个题目不用取模, 使用 NTT 时要注意原本的答案是否会大于等于所选取的模数 p 。

	结果	用时	内存	语言	文件大小
	100	715ms	5356kb	C++	2.4kb

图 7.4: NTT-iteration

```

#include<bits/stdc++.h>
2 using namespace std;
  typedef long long ll;
4 const int maxn=1e6+10;
  const int mod = 998244353;
6 const int groot = 3; //119 * 2^23 + 1 = mod
  int gk = 119;
8 const int limit = 1<<23;
  int omega[maxn<<2], omegaInverse[maxn<<2]; //辅助
10 ll fast_exp(ll a,ll b,ll c){
    ll res=1;
12   a=a%c;
    assert(b>=0);
14   while(b>0)
    {
16       if(b&1) res=(a*res)%c;
        b=b>>1;
18       a=(a*a)%c;
    }
20   return res;
  }
22 inline int add(const int x, const int v){
    return x+v>=mod?x+v-mod: x+v;
24 }
  inline int dec(const int x, const int v){
26   return x-v<0?x-v+mod: x-v;

```

```

}
28 void init(const int n){
    omega[0] = 1;
30    omegaInverse[0] = 1;
    assert(limit>=n);
32    gk = gk* (limit / n); // 很关键 取到恰好用的2^m
    int g_n = fast_exp(groot, gk, mod);
34    int g_n_ni = fast_exp(g_n, mod-2, mod);
    for(int i=1;i<n;++i) {
36        omega[i] = 1LL*omega[i-1]*g_n%mod;
        omegaInverse[i] = 1LL*omegaInverse[i-1]*g_n_ni%mod;
38    }
}

40 void trans(int *a,const int n,const int *omega){ //a是系数 n是2的次幂 omega是要
    带入的点
    int k=0;
42    while((1<<k)<n) k++; //看N是2的多少次幂
    for(int i=0;i<n;++i){
44        int t=0;
        for(int j=0;j<k;++j) if(i&(1<<j)) t|=(1<<(k-j-1));
46        if(i<t) swap(a[i],a[t]); //原地翻转
    }
48    for(int l=2;l<=n;l*=2){ //l=1不用求 就是系数
        int m=l/2;
50        for(int *p=a;p!=a+n;p+=l){ //l为一段要求 由两段l/2合并得到
            //当前处理[p,p+l)
52            for(int i=0;i<m;++i){
                //蝴蝶操作 omega_{l}^{i}=omega_{N}^{N/l*i}
54                int t=1LL*omega[n/l*i]*p[i+m]%mod;
                p[i+m]=dec(p[i], t);
56                p[i]=add(p[i], t);
            }
58        }
    }
60 }

void dft(int *a,const int n){
62    trans(a,n,omega);
}

64 void idft(int *a,const int n){
    trans(a,n,omegaInverse);
66    int n_ni = fast_exp(n, mod-2, mod);
    for(int i=0;i<n;++i) a[i] = 1LL*a[i]*n_ni%mod;
68 }

```

```

int a[maxn<<2], b[maxn<<2];
70 int main()
{
72 #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
74 #endif
    ios::sync_with_stdio(false);
76     int n,m;
    cin>>n>>m;
78     n++;m++;
    for(int i=0;i<n;++i) cin>>a[i];
80     for(int i=0;i<m;++i) cin>>b[i];
    int N=1;
82     while(N<m+n-1) N<<=1;
    init(N);
84     dft(a,N);
    dft(b,N);
86     for(int i=0;i<N;++i) a[i]=1LL*a[i]*b[i]%mod;
    idft(a,N);
88     for(int i=0;i<n+m-1;++i) cout<<a[i]<<" ";
    return 0;
90 }

```

code07/NTT.cpp

7.5.2.3 模数任意的解决方案

NTT 要求模数写成 $p = k * 2^m + 1$ 的形式, 但有些题目, 要求模数写不成这样的形式, 甚至是一个合数。这样的话, 一般选取 s 个常用的素数 p_1, p_2, \dots, p_s , 要求满足:

$$\prod_{i=1}^s p_i > n(m-1)^2$$

其中 n 是结果多项式的项数, m 是要求的模数。然后分别在 $\text{mod } p_i$ 下求解系数, 最后使用中国剩余定理将结果合并 (一般会使用高精度)。

第 7 章 习题

1. 本原勾股数组 poj 1305
2. 本原勾股数组 fzu 1669
3. 圆上整点/高斯素数 2019 上海网络赛
4. 斐波那契数列循环节 51nod-1195
5. dfs similar 2019icpc 亚洲区域赛-南昌站 B 题

6. FFT BZOJ 3992 / SDOI2015 序列统计
7. FFT BZOJ 3771 Triple
8. BZOJ 3527 FFT
9. 模板: 任意模数 NTT



参考文献

-
- [1] DELEGLISE M, RIVAT J. Computing $\phi(x)$: The meissel, lehmer, lagarias, miller, odlyzko method[J]. Mathematics of Computation, 1996, 65(213):235-245.
 - [2] DUSART P. Estimates of some functions over primes without r.h.[Z]. [S.l.: s.n.], 2010.
 - [3] AXLER C. On the sum of the first n prime numbers[J]. Mathematics, 2014.

附录 常用的表

A.1 梅森素数表

Mersenne exponents: primes p such that $2^p - 1$ is prime. Then $2^p - 1$ is called a Mersenne prime. 见表A.1。

表 A.1: 梅森素数指数

n	a(n)	24	19937
1	2	25	21701
2	3	26	23209
3	5	27	44497
4	7	28	86243
5	13	29	110503
6	17	30	132049
7	19	31	216091
8	31	32	756839
9	61	33	859433
10	89	34	1257787
11	107	35	1398269
12	127	36	2976221
13	521	37	3021377
14	607	38	6972593
15	1279	39	13466917
16	2203	40	20996011
17	2281	41	24036583
18	3217	42	25964951
19	4253	43	30402457
20	4423	44	32582657
21	9689	45	37156667
22	9941	46	42643801
23	11213	47	43112609

A.2 卡米歇尔数

见表A.2。

A.3 常见的素数及其原根

见表A.3。

表 A.2: 卡米歇尔数前面一点

a	a(n)
1	561
2	1105
3	1729
4	2465
5	2821
6	6601
7	8911
8	10585
9	15841
10	29341
11	41041
12	46657
13	52633
14	62745
15	63973
16	75361
17	101101
18	115921
19	126217
20	162401
21	172081
22	188461
23	252601
24	278545
25	294409

表 A.3: 常见的素数及其原根

$p = k * 2^m + 1$	k	m	$groot$
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

A.4 一些公式

A.4.1 切比雪夫多项式

$$\cos nx = \begin{cases} \sum_{k=0}^{\frac{n}{2}} (-1)^{\frac{n-2k}{2}} \frac{n \cdot (n+2k-2)!!}{(2k)!(n-2k)!!} \cos^{2k} x, & n \text{ is even} \\ \sum_{k=1}^{\frac{n+1}{2}} (-1)^{\frac{n+1-2k}{2}} \frac{n \cdot (n+2k-3)!}{(2k-1)!(n+1-2k)!!} \cos^{2k-1} x, & n \text{ is odd} \end{cases}$$

此页为草稿

