

# Standard Code Library

123ZDQ

South China University of Technology

November 19, 2022

# Contents

开始	4
定义	4
取消流同步	4
整数读写	4
数学	4
模乘	4
快速乘	4
龟速乘	5
快速幂	5
光速幂	5
最大公约数（欧几里得算法）	5
binary-gcd	6
乘法逆元	6
exgcd 求逆元	6
线性递推求逆元	6
线性同余方程组的合并（ex 中国剩余定理）	6
大指数运算（欧拉降幂）	7
Miller-Rabin 素性检测	7
Pollard-Rho 分解质因数	8
原根	9
寻找质数原根	9
高次同余方程	9
离散对数（BSGS/exBSGS）	9
模任意数开任意次根/N 次剩余	10
模奇质数开平方根/立方根	11
线性筛	12
欧拉函数 $\phi(x)$	12
整除分块	13
莫比乌斯变换（数论）/狄利克雷卷积	13
莫比乌斯反演（数论）	13
杜教筛	13
Min_25 筛	15
Step.1	15
Step.2	17
组合数的计算	18
基于模数的预处理分治（exlucas）	18
直线下整点数（类欧几里得算法）	19
洛谷模板题	20
高斯整数	20
四则运算与辗转相除	20
$4k + 1$ 型质数的分解（二次剩余）	21
平分分解计数	21
数值积分	21
日期操作	22
用于跳转的常量	22
辅助函数	22
日期和整数的一一对应	22
拉格朗日插值	22
曼哈顿距离与切比雪夫距离	22
几个数论函数的数量级	23
约数个数 $d(n)$ 的数量级	23
素数数量 $\pi(n)$ 的数量级	23
素数前缀和 $\sum_n [n \in prime]n$ 的数量级	23
高斯整数	23

环染色问题	23
置换群	24
Burnside 引理	24
轮换指标	25
多项式快速幂	25
广义二项式	25
Fibonacci 数列	25
Catalan 数	25
错排数	26
盒子放球方案数	26
第二类 Stirling 数	27
分拆数	27
五边形数定理	27
第一类 Stirling 数	27
Stirling 反演 (未验证)	27
<b>二维计算几何</b>	<b>28</b>
点向量基本运算	28
位置关系	28
多边形	28
求多边形面积	28
判断点是否在多边形内	29
凸包	29
凸包直径·平面最远点对	29
平面最近点对	30
圆	30
三点垂心	30
最小覆盖圆	30
<b>图论</b>	<b>31</b>
存图	31
最短路	31
Dijkstra	31
LCA	31
连通性	32
有向图强联通分量	32
二分图匹配	32
匈牙利算法求二分图无权最大匹配	32
KM 算法求二分图带权最大匹配	33
<b>数据结构</b>	<b>34</b>
STL	34
小根堆	34
整数哈希	34
二维树状数组	34
堆式线段树	35
小根堆	36
Treap	36
<b>字符串</b>	<b>38</b>
KMP	38
扩展 KMP(Z 函数)	39
AC 自动机	39
<b>杂项</b>	<b>40</b>
整数哈希	40
GCC 位运算	41
对拍	41

fread/fwrite 实现的读写类 . . . . .	42
Python 最短路 . . . . .	43

## 开始

### 定义

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  typedef long long LL;
5  typedef __int128 LLL;
6  typedef unsigned u32;
7  typedef unsigned long long u64;
8  typedef long double LD;
9  typedef pair<int,int> pii;
10 typedef pair<LL,LL> pll;
11
12 #define pln putchar('\n')
13 #define For(i,a,b) for(int i=(a),(i##i)=(b);i<=(i##i);++i)
14 #define Fodn(i,a,b) for(int i=(a),(i##i)=(b);i>=(i##i);--i)
15
16 const int M=1000000007,INF=0x3f3f3f3f;
17 const long long INFLL=0x3f3f3f3f3f3f3f3fLL;
18 const int N=1000009;
```

### 取消流同步

此时混用 `iostream` 和 `cstdio` 会寄

```
1 ios::sync_with_stdio(0);cin.tie(0);
```

### 整数读写

`read()` 读有符号数极限数据可能溢出未定义行为 `write()` 最长能写 `__int128`

```
1  template <typename T>
2  bool read(T &x) {
3      x = 0; char c = getchar(); int f = 1;
4      while (!isdigit(c) && (c != '-' && (c != EOF))) c = getchar();
5      if (c == EOF) return 0;
6      if (c == '-') f = -1, c = getchar();
7      while (isdigit(c)) { x = x * 10 + (c & 15); c = getchar(); }
8      x *= f; return 1;
9  }
10
11 template <typename T, typename... Args>
12 bool read(T &x, Args &...args) {
13     bool res = 1;
14     res &= read(x);
15     res &= read(args...);
16     return res;
17 }
18
19 template <typename T>
20 void write(T x) {
21     if (x < 0) x = -x, putchar('-');
22     static char sta[55];
23     int top = 0;
24     do sta[top++] = x % 10 + '0', x /= 10; while (x);
25     while (top) putchar(sta[--top]);
26 }
```

## 数学

### 模乘

#### 快速乘

- 可能比 `__int128` 快
- $|a|,|b|<m$ , 且  $m$  在 `longlong` 范围

- 编译环境支持 64 个二进制位精度的 long double (如 x86-64 平台的 GNU G++)

```

1 LL mul(LL a, LL b, LL m) {
2     u64 r = (u64)a * b - (u64)((LD)a / m * b + 0.5L) * m;
3     return r < m ? r : r + m;
4 }

```

### 龟速乘

```

1 template<typename T>
2 T pow_mod(T a,T b,T m){
3     T res=(m!=1);
4     while(b){
5         if(b%2)res=mul_mod(res,a,m);
6         a=mul_mod(a,a,m);
7         b/=2;
8     }
9     return res;
10 }

```

### 快速幂

- longlong 范围用 fpl

```

1 LL fp(LL a, LL b, LL Mod) {
2     LL res = (Mod != 1);
3     for (; b; b >>= 1, a = a * a % Mod)
4         if (b & 1)
5             res = res * a % Mod;
6     return res;
7 }
8
9 LL fpl(LL a, LL b, LL Mod) {
10    LL res = (Mod != 1);
11    for (; b; b >>= 1, a = mul(a, a, Mod))
12        if (b & 1)
13            res = mul(res, a, Mod);
14    return res;
15 }

```

### 光速幂

给定  $(a, m) = 1$ , 要多次询问  $f(b) = a^b \bmod m$

对  $a$  的幂分块, 预处理出  $a^0, a^1, a^2, a^3, \dots, a^t, a^{2t}, a^{3t}, \dots, a^{t^2}$ , 常见的取法是  $t = \lceil \sqrt{\phi(m)} \rceil$ , 或  $t = \lceil \sqrt{m} \rceil$ , 不小于阶即可

有  $f(b) = f(b \bmod \phi(m)) = f(kt) * f(r)$ ,  $O(1)$  回答询问

### 最大公约数 (欧几里得算法)

- 实际上 <algorithm> 有函数 \_\_gcd 在 cpp17 的 numeric 中有函数 std::gcd

```

1 template <typename T>
2 T gcd(T a, T b) {
3     while (b){T t = b;b = a % b;a = t;}
4     return a;
5 }
6
7 template <typename T>
8 T lcm(T a, T b) { return a / gcd(a, b) * b; }
9
10 template<typename T>
11 void exgcd(T a,T b,T&x,T&y,T&d){
12     if (b == 0){ x = 1, y = 0, d = a;}
13     else{
14         exgcd(b, a % b, y, x, d);
15         y -= (a / b) * x;
16     }
17 }

```

## binary-gcd

常数极小, longlong 版本更换对应的 64 位函数

```
1 int bgcd(int a, int b) {
2     int az = __builtin_ctz(a);
3     int bz = __builtin_ctz(b);
4     int z = min(az, bz);
5     b >>= bz;
6     while (a) {
7         a >>= az;
8         int diff = a - b;
9         az = __builtin_ctz(diff);
10        b = min(a, b), a = abs(diff);
11    }
12    return b << z;
13 }
```

## 乘法逆元

利用欧拉定理 (费马小定理) 通过快速幂求逆元代码最短

### exgcd 求逆元

需要 c++11

```
1 int getinv(int a, int p){
2     int m = 0, n = 1, x = 1, y = 0, b = p;
3     while (b){
4         int q = a / b;
5         tie(x, m) = make_tuple(m, x - q * m);
6         tie(y, n) = make_tuple(n, y - q * n);
7         tie(a, b) = make_tuple(b, a - q * b);
8     }
9     (x < 0) && (x += p);
10    return x;
11 }
```

### 线性递推求逆元

模  $p$  时, 设正整数  $i$  和  $p\%i$  均存在逆元, 有

$$i^{-1} = (p - \lfloor p/i \rfloor) * (p \bmod i)^{-1}$$

可以  $O(n)$  求出  $1..n$  模固定质数  $p$  的逆元, 但是空间局部性不佳, 对于  $n = 10^7$  数量级不如用对阶乘求逆的  $O(n + \log P)$  做法

## 线性同余方程组的合并 (ex 中国剩余定理)

用于合并同余方程组

$$x \equiv a_i \pmod{m_i}$$

其中  $a_i \geq 0$ ,  $m_i > 0$ ,  $2 * lcm(m_1, \dots) \in longlong$

前置算法 快速乘和 exgcd

用 `pair<LL,LL>` 存  $\{a_i, m_i\}$

有解返回 True, 并将第二个方程合并进第一个 (得到最小非负解); 否则返回 False

- 注意题目求最小非负解还是正数解!!!

```
1 bool crt(pll &a, pll b){
2     LL g, u, d;
3     exgcd(a.second, b.second, u, d, g);
4     d = b.first - a.first; if (d % g) return 0;
```

```

5     const LL t = b.second / g;
6     a.first += mul(u, d / g, t) * a.second;
7     a.second *= t;
8     a.first %= a.second;
9     (a.first < 0) && (a.first += a.second); //注意题目要求最小非负解还是正数解
10    return 1;
11 }

```

## 大指数运算（欧拉降幂）

$$a^x = \begin{cases} a^{x \bmod \phi(m)} & (a, m) = 1 \\ a^x & (a, m)! = 1 \ x < c \\ a^{(x-c) \bmod c+c} & (a, m)! = 1 \ x \geq c \end{cases}$$

其中  $c = m$  的质因子 + 1，是一个大小为  $O(\log n)$  的常数，为计算方便常用  $\phi(m)$  代替它

只需在快速幂中特判，若  $b^c \geq \phi(m)$ ，则返回  $b^{c \bmod \phi(m) + \phi(m)}$ ，否则返回  $b^{c \bmod \phi(m)}$

```

1 LL exfp(LL a, LL b, LL Mod){
2     /* 注意检查 Mod<=1e9 且 res 初值为 1*/
3     bool f=0; LL res=1;
4     for(; b>=1;){
5         if(a>Mod){f=1; a%=Mod;}
6         if(b&1){
7             res=res*a;
8             if(res>Mod){f=1; res%=Mod;}
9         }
10        a=a*a;
11    }return f?res+Mod:res;
12 }

```

## Miller-Rabin 素性检测

loj143 luoguP4718

$n \leq 10^{18}$

需要 mul fpl

<http://miller-rabin.appspot.com/>

对于  $n \leq 2^{64}$  保证正确的 7 个 base: 2, 325, 9375, 28178, 450775, 9780504, 1795265022

对于  $n \leq 3 * 10^{24}$ ，可以选 a = 2; 3; 5; 7; 11; 13; 17; 19; 23; 29; 31; 37; 41（前 13 个质数）

```

1 bool isPrime(LL n) {
2     if (n <= 3) return n > 1;
3     if (n % 2 == 0 || n % 3 == 0) return 0;
4     LL d = n - 1;
5     while (d % 2 == 0) d /= 2;
6     for (LL a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
7         if (a % n == 0) continue;
8         LL t = d;
9         LL u = fpl(a, t, n);
10        while (t != (n - 1) && u != 1 && u != (n - 1)) {
11            u = mul(u, u, n);
12            t *= 2;
13        }
14        if (t % 2 == 0 && u != (n - 1)) return 0;
15    }
16    return 1;
17 }

```

\_\_int128 版本 51nod 质数检测 V2（陈旧的垃圾代码）

```

1 // a b \in [0,m)
2 template<typename T>
3 T mul_mod(T a, T b, T m){
4     T res=0;
5     while(b){

```



```

6         if(b%2){
7             res+=a;
8             if(res>m)res-=m;
9         }
10        a*=2;
11        if(a>m)a-=m;
12        b/=2;
13    }
14    return res;
15 }
16
17 template<typename T>
18 T pow_mod(T a,T b,T m){
19     T res=(m!=1);
20     while(b){
21         if(b%2)res=mul_mod(res,a,m);
22         a=mul_mod(a,a,m);
23         b/=2;
24     }
25     return res;
26 }
27
28 template<typename T>
29 bool M_R(T a,const T&d,const int&t,const T&n){//注意 a%n==0 时应判定通过测试
30     a%=n;if(a==0)return 1;
31     a=pow_mod(a,d,n);
32     if(a==1||a==(n-1))return 1;
33     for(int i=1;i<t;++i){
34         a=mul_mod(a,a,n);
35         if(a==(n-1))return 1;
36         if(a==1)return 0;
37     }
38     return 0;
39 }
40
41 template<typename T>
42 bool isPrime(T n){
43     if(n==2||n==3)return 1;
44     if(n<2||(n%2==0)|| (n%3==0))return 0;
45     T d=n-1;int t=0;
46     while(d%2==0)d/=2,++t;
47     srand(time(0));
48     for(int i=0;i<8;++i)if(!M_R(rand()%(n-1)+1,d,t,n))return 0;
49     return 1;
50 }

```

## Pollard-Rho 分解质因数

需要 `mul __gcd`

返回  $n$  的一个大于 1 的真因子

不存在就死循环， 寄！ 因此在调用前务必检查  $n$  是大于 1 的合数

时间复杂度猜想为  $O(n^{1/4})$

```

1 mt19937 mt(time(0)); //随机化
2 //Pollard_Rho 返回 n 的一个 >1 的约数 调用前务必判断素性
3 inline LL PR(LL n) {
4     LL x = uniform_int_distribution<LL>(0, n - 1)(mt), s, t,
5     c = uniform_int_distribution<LL>(1, n - 1)(mt); //随机化
6     for (int gol = 1; gol <= 1; s = t, x = 1) {
7         for (int stp = 1; stp <= gol; ++stp) {
8             LL u = mul(t, t, n) + c;
9             t = (u >= n ? u - n : u);
10            x = mul(x, abs(s - t), n);
11            if ((stp & 127) == 0) {
12                LL d = __gcd(x, n);
13                if (d > 1) return d;
14            }
15        }
16    }
17 }

```

```

16         LL d = __gcd(x, n);
17         if (d > 1) return d;
18     }
19 }
20
21 vector<LL> get_prime_divisor(LL n) {
22     vector<LL> res; if (n <= 1) return res;
23     queue<LL> fac; fac.push(n);
24     while (fac.size()) {
25         LL u = fac.front();
26         fac.pop();
27         if (isPrime(u)) res.push_back(u);
28         else {
29             LL d = PR(u);
30             while (d == u) d = PR(u);
31             while (u % d == 0) u /= d;
32             fac.push(d);
33             if (u > 1) fac.push(u);
34         }
35     }
36     sort(res.begin(), res.end());
37     unique(res.begin(), res.end());
38     return res;
39 }

```

## 原根

只有  $2, 4, p^a, 2p^a$  有原根, 其中  $p$  是奇质数

(王元) 质数  $p$  的最小正原根是  $O(p^{0.25+\epsilon})$  的, 因此可以暴力查找质数的原根

设  $g$  为  $m$  的原根, 对任意  $x$  满足  $(x, \phi(m)) = 1$ ,  $g^x$  也是原根, 故原根共有  $\phi(\phi(m))$  个

(原根判定定理) 设  $(g, m) = 1$ , 则  $g$  是  $m$  的原根  $\iff$  对  $\phi(m)$  的任意质约数  $p$  有  $g^{\frac{\phi(m)}{p}} \not\equiv 1 \pmod{m}$

## 寻找质数原根

求任意质数  $p$  的原根

前置算法 `__gcd` 快速乘 快速幂

需要对  $\phi(p)$  分解, 一般要配合 Pollard-rho

时间复杂度猜测为  $O(p^{0.25+\epsilon})$

```

1  int Primitive_Root(LL p){
2      if(p<=3) return p-1;
3
4      vector<LL> g=Pollard_Rho::get_prime_divisor(p-1);
5      for(auto &it:g) it=(p-1)/it;
6
7      for(int q=1;q<p;++q){
8          bool t=1;
9          for(const auto&it:g) if(fpl(q,it,p)==1){t=0;break;}
10         if(t) return q;
11     }
12     return -1;
13 }

```

## 高次同余方程

- 以下  $p$  均为正整数

### 离散对数 (BSGS/exBSGS)

给定  $a, b, p$  求 (最小非负)  $x$  满足

$$a^x \equiv b \pmod{p}$$

其中  $p \leq 10^9$ , 不必是质数

时间复杂度  $O(\sqrt{p})$

- 前置算法 快速幂 map/unorded\_map \_\_gcd() 求逆元
  - 或者自定义哈希表 typedef unordered\_map<int,int,neal> HASH;

视  $p$  范围换用快速乘, 然而由于复杂度, 实则难以对一般的 long long 做 BSGS, 此时多半是想法歪了或者性质没有观察充分

- 若多组询问  $a, p$  为常量且始终满足  $(b, p) = 1$  则应复用哈希表  $h$  以降低常数
- 接口说明传入非负整数  $a, b$  和正整数  $p$ 
  - $\log()$  要求  $(a, p) = 1$  无解返回 -1 只求最小非负整数解
  - $\operatorname{operator}()$  无特殊要求无解返回 -1 可以修改使得求最小正数解, 见注释

```
1 typedef unordered_map<int,int,neal> HASH;
2
3 struct{
4     int log(int a,int b,int p){
5         if((b-1)%p==0)return 0;
6         HASH h;const int t=(int)sqrt(p)+1;
7         for(int j=0;j<t;++j,b=(LL)b*a%p)h[b]=j;
8         int at=(a=fp(a,t,p));
9         for(int i=1;i<=t;++i,a=(LL)a*at%p){
10             auto it=h.find(a);
11             if(it!=h.end())return i*t-(*it).second;
12         }return -1;
13     }
14     int operator()(int a,int b,int p){//(接口)
15         if((b-1)%p==0)return 0;//求正整数解则注释掉
16         if((b-a)%p==0)return 1;
17         a%=p;b%=p;
18         const int _a=a,_b=b,_p=p;
19         int d=__gcd(a,p),ax=1,t=0;
20         while(d>1){
21             if(b%d!=0)break;
22             b/=d;p/=d;ax=(LL)ax*(a/d)%p;++t;
23             d=__gcd(a,p);
24         }
25         for(int i=2,at=(LL)_a*_a%p;i<=t;++i,at=(LL)at*_a%p)if(at==_b)return i;
26         if(d>1)return -1;
27         int res=log(a,(LL)b*getinv(ax,p)%p,p);//res 为-1 说明无解 有解则非负
28         return res<0?-1:res+t;//求正整数解则要修改
29     }
30 }BSGS;
```

### 模任意数开任意次根/ $N$ 次剩余

给定  $a, b, m$ , 求 (最小非负)  $x$  满足

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

对于一般的情形, 暂时不会

#### 特殊情形一

若  $(b, m) = 1$  且  $(a, \phi(m)) = 1$  且易求得  $\phi(m)$

则存在非负整数  $u, v$  满足  $au - \phi(m)v = 1$ , 又  $b^{\phi(m)} \equiv 1 \pmod{m}$

得  $b^{au} \equiv b^{au - \phi(m)v} \equiv b \pmod{m}$

即  $x = b^u$

#### 特殊情形二

若  $(b, m) = 1$ , 且易求得  $m$  的原根  $g$  和  $\phi(m)$ , 则步骤如下:

计算离散对数  $y = \log_g b$  (时间瓶颈所在)

将问题转化为  $x^a = (g^{x'})^a \equiv g^y \pmod{m}$

等价于求解  $x'$  满足线性同余方程  $ax' \equiv y \pmod{\phi(m)}$

## 模奇质数开平方根/立方根

### 二次剩余判定定理

设  $n$  是奇质数  $p$  的一个既约剩余类

$$n^{\frac{p-1}{2}} \pmod p = \begin{cases} 1 & n \\ -1 & n \end{cases}$$

### 求平方根 (Cipolla 算法)

求最小非负整数  $x$  满足

$$x^2 \equiv n \pmod p$$

其中  $p$  为不超过  $10^9$  的质数,  $n$  为非负整数

期望复杂度  $O(2 \log p)$

- 若有两根则另一根为  $p - x$
- 若无解返回 -1
- 对  $10^{18}$  以内的  $p$  需用快速乘见注释
- 接口 `int ans1 = Square_root(p)(n);`

```
1 struct Square_root{
2     typedef int DAT; //操作数类型
3     const DAT P;
4     DAT I2;
5     Square_root(DAT p = 1) : P(p) {}
6     DAT mul(DAT a, DAT b) const{ return (LL)a * b % P;} // DAT=int
7     /*DAT mul(DAT a, DAT b) const{
8         u64 r = (u64)a * b - (u64)((LD)a / P * b + 0.5L) * P;
9         return r < P ? r : r + P;
10    }*/ // DAT=long long
11
12    #define X first
13    #define Y second
14    typedef pair<DAT, DAT> pii;
15    pii mul(pii a, pii b) const { return {
16        (mul(a.X, b.X) + mul(mul(I2, a.Y), b.Y)) % P,
17        (mul(a.X, b.Y) + mul(a.Y, b.X)) % P};}
18
19    template <class T>
20    T pow(T a, DAT b, T x){
21        for (; b; b /= 2, a = mul(a, a)) if (b & 1)x = mul(x, a);
22        return x;
23    }
24
25    DAT operator()(DAT n){
26        if ((n % P) <= 1) return n;
27        if (pow(n, (P - 1) / 2, (DAT)1) == P - 1) return -1;
28        DAT a;
29        do a = rand(); // 随机方法可能寄
30        while (pow(I2 = (mul(a, a) - n + P) % P, (P - 1) / 2, (DAT)1) == 1);
31        DAT x = pow(pii{a, 1}, (P + 1) / 2, {1, 0}).X;
32        return min(x, P - x);
33    }
34    #undef X
35    #undef Y
36 };
```

## 立方根 (三次剩余)

求一个非负整数  $x$  满足

$$x^3 \equiv n \pmod{p}$$

其中  $p$  是质数

复杂度期望  $O(9 \log p)$

参考 <https://blog.csdn.net/skywalkert/article/details/52591343>

- $n = 0$  或  $p \leq 3$  需要特判
- 若  $p \equiv 2 \pmod{3}$ , 则  $x \equiv n^{(2p-1)/3} \pmod{p}$  是唯一解
- 若  $p \equiv 1 \pmod{3}$ , 则  $\epsilon = (\sqrt{-3} - 1)/2$  是三次单位根, 其满足  $\epsilon^3 \equiv 1 \pmod{p}$ 
  - 如果找到一个解  $x$ , 则另外两个解为  $x\epsilon, x\epsilon^2$

接口 `int ans2 = Cube_root(n, p)();` 返回其中一个非负整数解, 无解返回 `-1`

换 `long long` 见注释

```
1 struct Cube_root{
2     typedef int DAT; //操作数类型
3     const DAT n, p; Cube_root(DAT _n=1, DAT _p=1): n(_n%p), p(_p){};
4     struct Z3{DAT x, y, z;};
5     DAT mul(DAT a, DAT b) const {return (LL)a*b%p;} //DAT=int
6     //DAT mul(DAT a, DAT b) const {u64 r=(u64)a*b-(u64)((LD)a/p*b+0.5L)*p; return r<p?r:r+p;} //DAT=long long
7     Z3 mul(Z3 a, Z3 b) const {return (Z3){
8         (mul(a.x, b.x) + mul((mul(a.y, b.z) + mul(a.z, b.y))%p, n))%p,
9         ((mul(a.x, b.y) + mul(a.y, b.x))%p + mul(mul(a.z, b.z), n))%p,
10        ((mul(a.x, b.z) + mul(a.y, b.y))%p + mul(a.z, b.x))%p
11    };}
12     template<class T> T pow(T a, DAT b, T x)
13     {for(; b/=2, a=mul(a, a)); if(b&1) x=mul(x, a); return x;}
14     DAT operator()() {
15         if(n==0 || p<=3) return n;
16         if(p%3==2) return pow(n, (2*p-1)/3, (DAT)1);
17         if(pow(n, (p-1)/3, (DAT)1) != 1) return -1;
18         Z3 r;
19         do r=pow((Z3){rand(), rand(), rand()}, (p-1)/3, (Z3){1, 0, 0});
20         while(r.x==0 || r.y==0 || r.z==0); // 随机寄了就换均匀随机
21         return pow(r.y, p-2, (DAT)1);
22     }
23 };
```

## 线性筛

这也要板子?

```
1 int ntp[N], pri[N], tot;
2 inline void linear_sieve(int n){
3     ntp[1]=1; for(int i=2; i<=n; ++i) ntp[i]=0; tot=0; //初始化 一般可省
4     for(int i=2; i<=n; ++i){
5         if(!ntp[i]) pri[++tot]=i;
6         for(int j=1; j<=tot; ++j){
7             const LL nex=(LL)i*pri[j]; if(nex>n) break; //n<=1e6 则不必开 LL
8             ntp[nex]=1;
9             if(i%pri[j]==0) break;
10        }
11    }
12 };
```

## 欧拉函数 $\phi(x)$

求  $\phi(x)$  的值可以归约为找  $x$  的质因子, 于是可以  $O(\sqrt{x})$  求点值

```
1 template <typename T>
2 inline T phi(T x) {
3     T res = x;
```

```

4     for (T i = 2; i * i <= x; ++i)
5         if ((x % i) == 0) {
6             res = res / i * (i - 1);
7             while ((x % i) == 0) x /= i;
8         }
9     if (x > 1) res = res / x * (x - 1);
10    return res;
11 }

```

或者  $O(n)/O(n\log\log n)/O(n\log n)$  筛出  $\phi(1..n)$

## 整除分块

对于给定的正整数  $n$  和  $i$ , 使得

$$\lfloor \frac{n}{i} \rfloor = \lfloor \frac{n}{j} \rfloor$$

成立的最大正整数 ( $i \leq j \leq n$ )  $j$  的值为

$$j = \lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \rfloor$$

即  $\lfloor \frac{n}{i} \rfloor$  所在块的右端点的标号为  $\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \rfloor$

## 莫比乌斯变换 (数论) / 狄利克雷卷积

P5495

给定  $a[1..n]$ , 求  $b[1..n]$  满足

$$b_k = \sum_{i|k} a_i$$

先筛出所有质数, 视每个质数为一个维度, 做高维前缀和, 本质上与 Sm of Subset DP 相同

时间复杂度  $O(n \log \log n)$

```

1  for (int i = 1; i <= tot; ++i)
2      for (int j = 1; pri[i] * j <= n; ++j)
3          a[pri[i] * j] += a[j];

```

## 莫比乌斯反演 (数论)

- 狄利克雷卷积  $*$  的代数性质
  - 交换律、结合律
  - 对加法 (函数值逐项相加) 分配律
  - 单位元存在  $\epsilon = [n == 1]$
  - 对  $f(1) \neq 0$  的  $f$  存在逆元  $g$
  - 两个积性函数的狄利克雷卷积仍是积性函数
- $\mathbb{1} * \mu = \epsilon$
- $\mathbb{1} * \phi = id$
- 设  $f$  和  $g$  是数论函数 ( $\mathbb{N} \rightarrow \mathbb{R}$ ), 且当  $n > M$  时有  $f(n) = g(n) = 0$ , 则
  - $f(n) = \sum_{n|m} g(m) \iff g(n) = \sum_{n|m} \mu(\frac{m}{n}) f(m)$

## 杜教筛

给定数论函数  $f$  和一个较大的  $n$ , 要计算其前缀和

$$S(n) = \sum_{i=1}^n f(i)$$

考虑构造一个函数  $g$ , 令其与  $f$  作狄利克雷卷积, 有

$$\sum_{i=1}^n (f * g)(i) = \sum_{i=1}^n \sum_{xy=i} f(x)g(y) = \sum_{y=1}^n g(y)S(\lfloor \frac{n}{y} \rfloor)$$

即

$$g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{y=2}^n g(y)S(\lfloor \frac{n}{y} \rfloor)$$

其中,  $f * g$  的前缀和与  $g$  的区间和要求可以快速计算 (不高于计算  $f$  的复杂度即可,  $O(1)$  推式子或者也用杜教筛算都行)

时间复杂度为  $O(n^{3/4})$

如果能线性预处理出  $S(1..n)$  的前  $n^{2/3}$  项, 可以将复杂度平衡至  $O(n^{2/3})$

关于记忆化:

- 对于单组询问, 可以不用 `map/unordered_map`, 在线性预处理的前提下直接索引  $\lfloor \frac{N}{x} \rfloor$  中的  $x$  即可
- 对于多组询问, 用 `unordered_map` 可以利用不同询问的信息, 可能更快

特别注意对 **n** 数论分块时, 最后一次循环之前, 执行 **i=j+1** 之后, 此时  $i = n + 1$ , 如果 **n** 刚好为数据类型上界, 会导致溢出!!!

rqy 大佬的【模板】杜教筛 (Sum) 代码

其中 1300, maxm, maxn 分别是  $N$ ,  $N^{2/3}$ ,  $N^{1/3}$  的最大值

```

1  typedef long long LL;
2  const int maxn = 2147483647;
3  const int maxm = 2000000;
4
5  LL S1[maxm], S2[1300];
6  bool vis[1300];
7  int N;
8
9  LL S(int n) {
10     if (n < maxm) return S1[n];
11     int x = N / n; // 如果存在某个 x 使得 n = floor(N / x),
12                   // 选 x = floor(N / n) 一定可以。
13                   // 且对于 x > N^{1/3} 的情形会直接查线性表
14     if (vis[x]) return S2[x];
15     vis[x] = true;
16     LL &ans = S2[x];
17     ans = (LL)n * (n + 1) / 2; // 对 (f*g)(n) = n 求和
18     // 在加强了数据后, 对于 n=(2^31)-1, 下面这里 i 会溢出成负 2^32, 导致接下来发生除 0 RE, 应该开 long long
19     for (int i = 2, j; i <= n; i = j + 1) {
20         j = n / (n / i);
21         ans -= (j - i + 1) * S(n / i);
22     }
23     return ans;
24 }

```

本人完整代码

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long LL;
5
6  const int N=2000000;
7
8  LL phi[N];
9  int miu[N], pri[N], tot;
10 int n;
11 bool vis[2021];
12 pair<LL, int> S2[2021];
13
14 pair<LL, int> S(int t){
15     if(t<=2000000)return {phi[t], miu[t]};
16     int x=n/t; if(vis[x])return S2[x];

```

```

17     vis[x]=1;
18     pair<LL,int>&res=S2[x];
19     res.first=t*(t+1ll)/2;res.second=1;
20     for(LL i=2,j;i<=t;i=j+1){
21         j=t/(t/i);
22         pair<LL,int> u=S(t/i);
23         res.first-=(j-i+1)*u.first;
24         res.second-=(j-i+1)*u.second;
25     }
26     return res;
27 }
28
29
30 inline void solve(int T){
31     scanf("%d",&n);
32     memset(vis,0,sizeof vis);
33     pair<LL,int> ans=S(n);
34     printf("%lld %d\n",ans.first,ans.second);
35 }
36
37 signed main(){
38     miu[1]=1;phi[1]=1;
39     for(int i=2;i<=20000000;++i){
40         if(phi[i]==0)pri[++tot]=i,phi[i]=i-1,miu[i]=-1;
41         for(int j=1;j<=tot;++j){
42             const LL m=i*pri[j];if(m>20000000)break;
43             phi[m]=phi[i]*(pri[j]-1);
44             if(i%pri[j]){
45                 miu[m]=-miu[i];
46             }else{
47                 phi[m]=phi[i];
48                 break;
49             }
50         }
51         miu[i]+=miu[i-1];
52         phi[i]+=phi[i-1];
53     }
54     int t;scanf("%d",&t);
55     for(int i=1;i<=t;++i)solve(i);
56     return 0;
57 }

```

## Min\_25 筛

给定积性函数  $f(x)$  和较大的  $n$  求

$$S(n) = \sum_{i=1}^n f(i)$$

令第  $i$  小的质数为  $p_i$  (特殊地,  $p_0 = 1$ ), 先线性筛出  $\sqrt{n}$  以内的所有质数

### Step.1

先将原来的积性函数  $f(x)$  拆成若干个完全积性函数之和

对于特定的函数比如素数数量/素数前缀和, 可以用 long long 存, 在 dp 过程中不取模, 算完再取模

对完全积性函数  $f(x)$  求

$$g(n) = \sum_{p \in \text{prime}} f(p)$$

考虑 DP, 设  $g(n, j) = \sum_{i=1}^n [i \leq p_j] f(i)$

则需要的结果是  $g(n, k)$ , 其中  $p_k$  为  $\sqrt{n}$  以内最大质数

从  $j-1$  到  $j$  有转移

$$g(n, j) = g(n, j-1) - f(p_j) (g(\lfloor \frac{n}{p_j} \rfloor, j-1) - g(p_{j-1}, j-1))$$



初始值  $g(n, 0) = \sum_{i=2}^n f(i)$ ，一般而言这是个自然数已知幂的和，可以  $O(1)$  算

时间复杂度  $O(\frac{n^{3/4}}{\log n})$

- 只用到了形如  $\lfloor \frac{n}{x} \rfloor$  的约  $2\sqrt{n}$  处的 DP 值（形如  $p_{j-1}$  的点已经包含在内），对于  $x \leq \sqrt{n}$ ，映射到下标  $x$ ，对于  $x > \sqrt{n}$ ，映射到下标  $n/x$
- 可以滚动掉第二维

```

1  const int N=2e5; // sqrt(n)
2
3  LL n;
4  int p[N],tot;
5  ... // 输入 n, 预处理 sqrt(n) 以内质数 p[1..tot],
6      // 注意一定记得 p[0]=1
7
8  int id1[N],id2[N],m=0; // 存离散化的 n/x
9  LL v[2*N],g[2*N];
10
11 int getid(LL x){ // 找每个点值的下标
12     return x<N?id1[x]:id2[n/x];
13 }
14
15 for(LL i=1,j;i<=n;i=j+1){ // 初始化
16     LL u=n/i; j=n/u;
17     v[++m]=u;
18     if(u<N)id1[u]=m;else id2[n/u]=m;
19     g[m]=...; // g[n/i][0] = sum(f(2..n))
20 }
21
22 for(int j=1;j<=tot;++j){
23     for(int i=1;i<=m && 1ll*p[j]*p[j]<=v[i];++i){
24         g[i]=g[i]-(.../*f(p_j)*/)*( g[getid(v[i]/p[j])] - g[getid(p[j-1])] );
25     }
26 }

```

求  $n$  以内素数数量

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long LL;
4
5  const int N=316333;
6
7  LL n; //1e11
8  int m;
9
10 int id1[N],id2[N],cnt;
11 LL v[2*N+1],g[2*N+1];
12 int getid(LL u){
13     if(u<N)return id1[u];else return id2[n/u];
14 }
15
16 int pri[N],tot; bool ntp[N];
17
18 signed main(){
19     scanf("%lld",&n);m=sqrt(n)+1;
20
21     pri[0]=1;
22     for(int i=2;i<=m;++i){
23         if(!ntp[i])pri[++tot]=i;
24         for(int j=1;j<=tot;++j){
25             const int nex=pri[j]*i;if(nex>m)break;
26             ntp[nex]=1;
27             if(i%pri[j]==0)break;
28         }
29     }
30
31     for(LL i=1,j;i<=n;i=j+1){
32         LL t=n/i;j=n/t;
33         v[++cnt]=t;
34         if(t<N)id1[t]=cnt;else id2[j]=cnt;

```

```

35     g[cnt]=t-1;
36 }
37
38 for(int j=1;j<=tot;++j){
39     const LL pj=pri[j],lim=pj*pj;
40     const LL gpre=g[getid(pri[j-1])];
41     for(int i=1;i<=cnt&&lim<=v[i];++i)
42         g[i]+=gpre-g[getid(v[i]/pj)];
43 }
44
45 printf("%lld\n",g[1]);
46 return 0;
47 }

```

## Step.2

此时已求出积性函数  $f(x)$  在质数处的点值和  $g(n) = \sum_{p \in \text{prime}} f(p)$ ，且需要满足  $f(p^a)$  可以快速计算

### 法一

令  $S(n, j) = \sum_{i > p_j} f(i)$ ，则有

$$S(n, j) = g(n) - g(p_j) + \sum_{j < k, p_k \leq \sqrt{n}} \sum_{e=1}^{p_k^e \leq n} f(p_k^e) (S(\lfloor n/p_k^e \rfloor, k) + [e \neq 1])$$

特殊地，当  $p_j \geq n$  时  $S(n, j) = 0$

考虑  $S(\lfloor n/p_k^e \rfloor, k)$  在  $\lfloor n/p_k^e \rfloor \geq p_k$  时才有值，也可变形为

$$S(n, j) = g(n) - g(p_j) + \sum_{j < k, p_k \leq \sqrt{n}} \sum_{e=1}^{p_k^e * p_k \leq n} (f(p_k^e) S(\lfloor n/p_k^e \rfloor, k) + f(p_k^e * p_k))$$

无需记忆化直接暴力递归，最终答案即为  $S(n, 0) + f(1)$

时间复杂度为  $O(n^{1-\epsilon})$ ，当  $n \leq 10^{13}$  时跑得比  $O(\frac{n^{3/4}}{\log n})$  快

P5325 【模板】Min\_25 筛

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long LL;
4
5  const int N=1000009;
6  const int M=10000000007, i2=(M+1)/2, i6=(M+1)/6;
7
8  LL n;
9  int m;
10
11 int id1[N], id2[N], cnt;
12 LL v[2*N+1];
13 int g1[2*N+1], g2[2*N+1];
14 int getid(LL u){
15     if(u<N) return id1[u]; else return id2[n/u];
16 }
17
18 int pri[N], tot;
19 bool ntp[N];
20
21 LL f(LL pa){ return pa%M*((pa-1+M)%M)%M; }
22
23 int S(LL t, int j){
24     if(pri[j]>=t) return 0;
25     int u=getid(t), v=getid(pri[j]);
26     int res=(LL(g2[u]-g1[u]-g2[v]+g1[v]+M+M)%M);
27     for(int k=j+1; k<=tot&&LL(pri[k])*pri[k]<=t; ++k){

```

```

28         for(LL pa=pri[k];pa*pri[k]<=t;pa*=pri[k]){
29             res=(res+f(pa)*S(t/pa,k)+f(pa*pri[k]))%M;
30         }
31     }
32     return res;
33 }
34
35 signed main(){
36     scanf("%lld",&n);m=sqrt(n)+1;
37
38     pri[0]=1;
39     for(int i=2;i<=m;++i){
40         if(!ntp[i])pri[++tot]=i;
41         for(int j=1;j<=tot;++j){
42             int nex=pri[j]*i;if(nex>m)break;
43             ntp[nex]=1;
44             if(i%pri[j]==0)break;
45         }
46     }
47
48     for(LL i=1,j;i<=n;i=j+1){
49         LL t=n/i;j=n/t;
50         v[++cnt]=t;
51         if(t<N)id1[t]=cnt;else id2[j]=cnt;
52         g1[cnt]=(t+2)%M*((t-1)%M)%M*i2%M;
53         g2[cnt]=(t%M)*((t+1)%M)%M*((2*t+1)%M)%M*i6%M-1;
54         if(g2[cnt]<0)g2[cnt]+=M;
55     }
56
57     for(int j=1;j<=tot;++j){
58         const LL pj=pri[j],lim=pj*pj;
59         const int pre=getid(pri[j-1]);
60         for(int i=1;i<=cnt&&lim<=v[i];++i){
61             const int now=getid(v[i]/pj);
62             g1[i]=(g1[i]-pj*(g1[now]-g1[pre]+M)%M)%M;
63             g2[i]=(g2[i]-pj*pj*(g2[now]-g2[pre]+M)%M)%M;
64         }
65     }
66
67     printf("%d\n", (S(n,0)+1)%M);
68     return 0;
69 }

```

## 法二

$O(\frac{n^{3/4}}{\log n})$ ，代码较长，由于常数大一般跑得不如法一快，暂没学

## 组合数的计算

计算  $C(n, m) \bmod p$

- $m$  非常小， $p$  为质数，此时  $n^m/m!$  项数很小
- 若  $p$  是固定的线性大小的质数， $O(n)$  预处理阶乘逆元
- 若  $n, m$  很大但  $p$  是很小的质数，根据 *Lucas* 定理

$$C_n^m \equiv C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n \bmod p}^{m \bmod p} \pmod{p}$$

- 若  $n, m$  很大但  $p$  是线性大小的任意数，用预处理分治的思想求  $C_n^m \bmod P$ ：

## 基于模数的预处理分治 (exlucas)

luoguP4720

接口 `exlucas(p).C(n, m)`

预处理  $O(p + \sum p_i^{a_i})$

单次询问  $O(\log^2 n)$ ，用光速幂可以做到  $O(\log n)$

- 不要求  $P$  为质数

```

1  struct exlucas{
2      struct dat{
3          const int p,pt,phi,t;
4          vector<int>r;
5          int Pow(int x,int y)const{
6              int res=1;
7              while(y>0){
8                  if(y%2!=0)res=LL(res)*x%pt;
9                  y/=2;x=LL(x)*x%pt;
10             }
11             return res;
12         }
13         dat(int _p,int _pt,int _t):p(_p),pt(_pt),phi(_pt/_p*(p-1)),t(_t),r(pt){
14             r[0]=1;
15             for(int i=1;i<pt;++i)if(i%p==0)r[i]=r[i-1];
16             else r[i]=r[i-1]*LL(i)%pt;
17         }
18         int f(LL n)const{
19             if(n<p)return r[n];
20             return Pow(r.back(),n/pt%phi)*LL(r[n%pt])%pt*f(n/p)%pt;
21         }
22         LL g(LL n)const{
23             LL res=0;
24             while(n>0)res+=n/p,n/=p;
25             return res;
26         }
27         int C(LL n,LL m){
28             LL v=g(n)-g(m)-g(n-m);
29             if(v>=t)return 0;
30             return LL(f(n))*Pow(f(m),phi-1)%pt*Pow(f(n-m),phi-1)%pt*Pow(p,v)%pt;
31         }
32     };
33     vector<dat>mp;
34     vector<LL>u;
35     const int p;
36     exlucas(int _p):p(_p){
37         for(int i=2;i*i<=p;++i)if(_p%i==0){
38             int m=1,cnt=0;
39             while(_p%i==0)_p/=i,m*=i,++cnt;
40             mp.emplace_back(i,m,cnt);
41         }
42         if(_p>1)mp.emplace_back(_p,_p,1);
43         u.resize(mp.size());
44         for(int i=0;i<int(u.size());++i){
45             u[i]=1;
46             for(int j=0;j<int(mp.size());++j)if(i!=j){
47                 u[i]=u[i]*mp[j].pt%p*mp[i].Pow(mp[j].pt,mp[i].phi-1)%p;
48             }
49         }
50     }
51     int C(LL n,LL m){
52         if(n<m||m<0)return 0;
53         int res=0;
54         for(int i=0;i<int(u.size());++i)res=(res+u[i]*mp[i].C(n,m))%p;
55         return res;
56     }
57 };

```

## 直线下整点数（类欧几里得算法）

给定  $n \geq 0, a \geq 0, c > 0, b$ ，求

$$f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$$

不妨设  $0 \leq a, b < c$ （否则直接  $\text{mod } c$  调整）， $m = \lfloor \frac{ai+b}{c} \rfloor \geq 1$ （否则直接 return），

换一维枚举贡献即得  $f(a, b, c, n) = nm - f(c, c - b - 1, a, m - 1)$

相当于  $a$  与  $c$  在辗转相除, 时间复杂度  $O(\log \max(a, c))$

```
LL f(LL a, LL b, LL c, LL n) {
    assert(c > 0 && n >= 0 && a >= 0);
    LL u = a / c; a -= u * c;
    LL v = b / c; b -= v * c;
    if (b < 0) {b += c; --v;} assert(b >= 0);
    LL res = u * (n * (n + 1) / 2) + (n + 1) * v;
    if (a == 0) return res + b / c * (n + 1);
    LL m = (a * n + b) / c; if (m == 0) return res;
    return n * m - f(c, c - b - 1, a, m - 1) + res;
}
```

## 洛谷模板题

$f = \sum [(ai+b)/c]$   $g = \sum i[(ai+b)/c]$   $h = \sum [(ai+b)/c]^2$   $i=0..n$   $a, b, n \in \mathbb{N}$   $c \in \mathbb{N}^*$

```
1 struct dat{LL f, g, h};
2 const LL i2 = 499122177, i3 = 332748118, M = 998244353; //预处理出模 M 意义下 2 和 3 的逆元
3 dat f(LL a, LL b, LL c, LL n){
4     LL ac = a / c, bc = b / c;
5     LL n2 = (n * (n + 1) % M) * i2 % M, n3 = n2 * (2ll * n + 1) % M * i3 % M;
6     dat res = {
7         (n2 * ac % M + (n + 1) * bc % M) % M,
8         (ac * n3 % M + bc * n2 % M) % M,
9         (ac * ac % M * n3 % M +
10          bc * bc % M * (n + 1) % M + ac * bc % M * n2 % M * 2ll) % M};
11     a %= c; b %= c; if (a == 0) return res;
12     LL m = (a * n + b) / c;
13     dat p = f(c, c - b - 1, a, m - 1);
14     LL fc = (n * m % M - p.f + M) % M, gc = (n2 * m % M - i2 * (p.f + p.h) % M + M) % M;
15     return {(res.f + fc) % M, (res.g + gc) % M,
16             (res.h + 2ll * (bc * fc % M + ac * gc % M) % M +
17              n * m % M * m % M - 2ll * p.g - p.f + 3ll * M) % M};}
```

loj 模板题不会

## 高斯整数

### 四则运算与辗转相除

```
1 struct G {
2     long long r, i;
3     G(long long re = 0, long long im = 0) : r(re), i(im) {}
4     long long norm() const { return r * r + i * i; }
5
6     G operator+(const G& b) const { return G(r + b.r, i + b.i); }
7     G operator-(const G& b) const { return G(r - b.r, i - b.i); }
8     G operator*(const G& b) const {
9         return G(r * b.r - i * b.i, i * b.r + r * b.i);
10    }
11    G operator/(const G& b) const {
12        long long l = b.norm();
13        return G(round(1.0 * (r * b.r + i * b.i) / l),
14                 round(1.0 * (i * b.r - r * b.i) / l));
15    }
16    G operator%(const G& b) const { return (*this) - (*this) / b * b; }
17
18    void get() { scanf("%lld%lld", &r, &i); }
19    void write() const { printf("%lld %lld ", r, i); }
20 };
21
22 G gcd(G a, G b) { return (b.norm() == 0) ? a : gcd(b, a % b); }
```

### $4k+1$ 型质数的分解 (二次剩余)

给定质数  $p$  满足  $p \equiv 1 \pmod{4}$ , 解出一对共轭复数  $a \pm bi$  满足  $(a+bi)(a-bi) = a^2 + b^2 = p$

先求正整数  $x$  满足  $x \equiv -1 \pmod{p}$ , 由于  $-1$  是  $4k+1$  型质数的二次剩余, 一定有解

即  $p|(x^2+1)$ , 由于  $x^2+1 = (x+i)(x-i)$ , 计算  $d = \gcd(p, x+i)$ , 则  $d$  即为  $a \pm bi$  其中之一

### 平分分解计数

设  $n$  为正整数, 记  $4 * f(n)$  为二维平面的圆  $x^2 + y^2 = n$  上的整点数, 则  $f(n)$  是积性函数:

$$f(p^a) = \begin{cases} a+1 & p = 4k+1 \\ [a] & p = 4k+3 \\ 1 & p = 2 \end{cases}$$

或者对任意正整数  $d$  定义完全积性函数  $\chi(d)$ :

$$\chi(d) = \begin{cases} 1 & d = 4k+1 \\ -1 & d = 4k+3 \\ 0 & d = 2k \end{cases}$$

易证

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

特别地, 对于  $x^2 + y^2 = n^2$  上的整点数  $4 * g(n)$  有

$$g(p^a) = \begin{cases} 2a+1 & p = 4k+1 \\ 1 & else \end{cases}$$

$\sum f(1..n)$  与  $\sum g(1..n)$  都可以用 min25 筛计算

### 数值积分

- 自适应辛普森算法
- 复杂度  $O(\epsilon)$
- luogu p4542

```

1  struct IG{
2      typedef double Func(double);
3      const Func*f;IG(const Func&g):f(&g){}
4      double simpson(double l,double r)const{
5          double mid=(l+r)/2;
6          return (r-l)*(f(l)+4*f(mid)+f(r))/6;
7      }
8      double find(double l,double r,const double&EPS,double res)const{
9          double mid=(l+r)/2;
10         double fl = simpson(l, mid), fr = simpson(mid, r);
11         if (abs(fl + fr - res) <= 15 * EPS)return fl + fr + (fl + fr - res) / 15;
12         return find(l, mid, EPS / 2, fl) + find(mid, r, EPS / 2, fr);
13     }
14     double operator()(double l,double r,double EPS=1e-8)const{return find(l,r,EPS,simpson(l,r));}
15 };
16
17 struct Integration{
18     typedef double Func(double);
19     const Func*f;Integration(const Func&g):f(&g){}
20     typedef pair<double,double> pdd;
21     pdd add(pdd a,pdd b)const{
22         double mid=(a.first+b.first)/2;
23         return (pdd){mid,f(mid)};

```

```

24     }
25     #define simpson(p1,p2,p3) (((p3).first-(p1).first)*(p1).second+4*(p2).second+(p3).second)/6)
26     double find(pdd p1,pdd p3,pdd p5,double EPS,double res,int dep) const{
27         pdd p2=add(p1,p3),p4=add(p3,p5);
28         double fl=simpson(p1,p2,p3),fr=simpson(p3,p4,p5),d=(fl+fr-res)/15;
29         if(abs(d)<=EPS&&dep<0) return fl+fr+d;
30         return find(p1,p2,p3,EPS/2,fl,dep-1)+find(p3,p4,p5,EPS/2,fr,dep-1);
31     }
32     double operator()(double l,double r,double EPS=1e-6/* 精度 */,int dep=12/* 最小递归深度 */) const{
33         pdd p1(l,f(l)),p3(r,f(r)),p2=add(p1,p3);
34         return find(p1,p2,p3,EPS,simpson(p1,p2,p3),dep);
35     }
36     #undef simpson
37 };

```

## 日期操作

### 用于跳转的常量

```

1  const LL year_1[2]={365, 366};
2  const LL year_400=1460097;
3  const LL m_day[13]={(LL)0x3f3f3f, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

```

### 辅助函数

```

1  inline bool isLeap(LL t){return (t % 400 == 0) || ((t % 4 == 0) && (t % 100 != 0));}
2  inline bool pick(LL a, LL b){return ((isLeap(a) && b <= 2) || (isLeap(a + 1) && b > 2));}
3  inline LL dayThisMonth(LL y, LL m){return m_day[m] + isLeap(y) * (m == 2);}

```

### 日期和整数的一一对应

- LL 可以改成 int

```

1  struct MY_DATE{
2      LL year, month, day;
3      MY_DATE(LL y = 2021, LL m = 1, LL d = 1) : year(y), month(m), day(d){};
4      LL p(MY_DATE op = {0, 0, 0}){//日期转换为整数
5          LL y = year - op.year, m = month - op.month, d = day - op.day;
6          if (m <= 2){ y--; m += 12;}
7          return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307;
8      }
9      MY_DATE run(LL k){//当前日期过 k 天
10         k += p();
11         LL x = k + 1789995, n = 4 * x / 146097, i, j, d;
12         x -= (146097 * n + 3) / 4;
13         i = (4000 * (x + 1)) / 1461001;
14         x -= 1461 * i / 4 - 31;
15         j = 80 * x / 2447;
16         d = x - 2447 * j / 80;
17         x = j / 11;
18         return MY_DATE(100 * (n - 49) + i + x, j + 2 - 12 * x, d);
19     }
20 };

```

## 拉格朗日插值

构造一个过给定  $n$  个点  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  的多项式

$$f(x) = \sum_{i=1}^n (y_i * \prod_{j \neq i} \frac{x - x_j}{x_i - x_j})$$

## 曼哈顿距离与切比雪夫距离

设  $A(x_1, y_1), B(x_2, y_2)$

曼哈顿距离  $d(A, B) = |x_1 - x_2| + |y_1 - y_2| = \max(|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|)$

这正是  $(x_1 + y_1, x_1 - y_1)$  与  $(x_2 + y_2, x_2 - y_2)$  的切比雪夫距离

因此将每个点  $(x, y)$  转化为  $(x + y, x - y)$ ，原坐标系的曼哈顿距离就是新坐标系下的切比雪夫距离

同理，将每个点  $(x, y)$  转化为  $(\frac{x+y}{2}, \frac{x-y}{2})$ ，原坐标系的切比雪夫距离就是新坐标系下的曼哈顿距离

## 几个数论函数的数量级

### 约数个数 $d(n)$ 的数量级

A066150 Maximal number of divisors of any n-digit number

4, 12, 32, 64, 128, 240, 448, 768, 1344, 2304, 4032, 6720, 10752, 17280, 26880, 41472, 64512, 103680, 161280, 245760, 368640, 552960, 860160, 1290240, 1966080, 2764800, 4128768, 6193152, 8957952, 13271040, 19660800, 28311552, 41287680, 59719680, 88473600, 127401984, 181665792, 264241152, 382205952, 530841600

### 素数数量 $\pi(n)$ 的数量级

A006880 Number of primes  $< 10^n$

0, 4, 25, 168, 1229, 9592, 78498, 664579, 5761455, 50847534, 455052511, 4118054813, 37607912018, 346065536839, 3204941750802, 29844570422669, 279238341033925, 2623557157654233, 24739954287740860, 234057667276344607, 2220819602560918840, 21127269486018731928, 201467286689315906290

### 素数前缀和 $\sum_n [n \in prime]n$ 的数量级

A046731  $a(n) = \text{sum of primes} < 10^n$

0, 17, 1060, 76127, 5736396, 454396537, 37550402023, 3203324994356, 279209790387276, 24739512092254535, 2220822432581729238, 201467077743744681014, 18435588552550705911377, 1699246443377779418889494, 157589260710736940541561021, 146923985169080063982257

## 高斯整数

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long LL;
4
5  struct G{
6      LL r,i;
7      G(LL re=0,LL im=0):r(re),i(im){}
8      LL norm()const{return r*r+i*i;}
9      G operator+(const G&b)const{return G(r+b.r,i+b.i);}
10     G operator-(const G&b)const{return G(r-b.r,i-b.i);}
11     G operator*(const G&b)const{return G(r*b.r-i*b.i,i*b.r+r*b.i);}
12     G operator/(const G&b)const{LL l=b.norm();return G(round(1.0*(r*b.r+i*b.i)/l),round(1.0*(i*b.r-r*b.i)/l));}
13     G operator%(const G&b)const{return (*this)-(*this)/b*b;}
14     //void get(){read(r,i);}void write()const{printf("%lld %lld ",r,i);}
15 };G gcd(G a,G b){return (b.norm()==0)?a:gcd(b,a%b);}
```

## 环染色问题

- poj2154 Burnside 引理

```
1  #include<cstdio>
2  using namespace std;
3  typedef long long LL;
4  #define pln putchar('\n')
5  const int N=100009;
6
7  int n,p,ans;
8
9  LL fp(LL a,LL b,LL Mod){
10     LL res=(Mod!=1);
11     for(;b>=1,a=a*a%Mod;if(b&1)res=res*a%Mod;
12     return res;
13 }
14
15 int ntp[N],pri[N],tot;
16
```



```

17  int f[33],t[33],d;
18
19  void dfs(int x,int phi,int u){
20      if(x>d){
21          ans=(ans+phi*fp(n,n/u-1,p))%p;
22          return;
23      }
24      dfs(x+1,phi,u);
25      u*=f[x];
26      phi*=(f[x]-1);
27      dfs(x+1,phi,u);
28      for(int i=2;i<=t[x];++i){
29          u*=f[x];
30          phi*=f[x];
31          dfs(x+1,phi,u);
32      }
33  }
34
35  void solve(int T){
36      scanf("%d%d",&n,&p);d=0;
37      int x=n;
38      for(int i=1;i<=tot;++i){
39          if(pri[i]*pri[i]>x)break;
40          if(x%pri[i])continue;
41          int k=0;while(x%pri[i]==0)x/=pri[i],++k;
42          f[++d]=pri[i];
43          t[d]=k;
44      }
45      if(x>1){
46          f[++d]=x;
47          t[d]=1;
48      }
49      ans=0;
50      dfs(1,1,1);
51      printf("%d\n",ans);
52  }
53
54  signed main(){
55      for(int i=2;i<=40000;++i){
56          if(!ntp[i])pri[++tot]=i;
57          for(int j=1;j<=tot;++j){
58              int nex=pri[j]*i;
59              if(nex>40000)break;
60              ntp[nex]=1;
61              if(i%pri[j]==0)break;
62          }
63      }
64      int t;scanf("%d",&t);
65      for(int i=1;i<=t;++i)solve(i);
66      return 0;
67  }

```

## 置换群

### Burnside 引理

设有限置换群  $G$  作用在有限集  $X$  上, 则  $X$  上的  $G$  — 数量为

$$\frac{1}{|G|} \sum_{g \in G} |\{x | g(x) = x\}|$$

若将  $x \in X$  理解为对  $1..n$  每个元素映射到某种颜色的方案,  $g$  的不动点数量可以理解为:

对于  $g$  生成的图 (可分解为若干不相交简单环的并), 满足同一个环上所有点颜色相同, 的染色方案数

### 可旋转涂色问题

给一个  $n$  元环涂色,  $m$  种颜色, 旋转后得到的方案算同一种, 求不同的方案数

$$\frac{1}{n} \sum_{i=0}^{n-1} m^{gcd(n,i)} = \frac{1}{n} \sum_{d|n} \phi(d) m^{n/d}$$

## 轮换指标

$x_i^j$  表示置换中  $i$  元环有  $j$  个

- 正  $n$  边形的旋转群

$$\frac{1}{n} \sum_{d|n} \phi(d) x_d^{n/d}$$

- 正  $n$  边形的二面体群

$$\frac{1}{2n} \sum_{d|n} \phi(d) x_d^{n/d} + \begin{cases} \frac{1}{2} x_1 x_2^{\frac{n-1}{2}} & n \text{ is odd} \\ \frac{1}{4} (x_2^{\frac{n}{2}} + x_1^2 x_2^{\frac{n-2}{2}}) & n \text{ is even} \end{cases}$$

- 正方体的顶点置换群

$$\frac{1}{24} (x_1^8 + 8x_1^2 x_3^2 + 9x_2^4 + 6x_4^2)$$

- 正方体的边置换群

$$\frac{1}{24} (x_1^{12} + 8x_3^4 + 6x_1^2 x_2^5 + 3x_2^6 + 6x_4^3)$$

- 正方体的面置换群

$$\frac{1}{24} (x_1^6 + 8x_3^2 + 6x_2^3 + 3x_1^2 x_2^2 + 6x_1^2 x_4)$$

## 多项式快速幂

给定  $F(x) \pmod{x^n}$ , 求  $F^k(x) \pmod{x^n}$

$k, n$  不大, 系数集不成域,  $O(n \log n \log k)$  分治 NTT

$k$  很大, 且系数模奇质数  $p$  ( $p > n$ ):

$F[0] = 1$ , 根据  $f(x^p) \equiv f^p(x) \pmod{p}$ , 则  $F^p(x) \equiv F[0] = 1 \pmod{x^p} = 1 \pmod{x^n}$ , 直接令  $k \% p = p$  后计算  $\exp(k \ln F(x))$

$F[0]! = 1$ , 提出最低次非 0 系数  $ux^t$  后化为上一种情形  $F^k(x) = u^k x^{tk} G^k(x)$ , 其中  $G[0] = 1$ , 注意  $u$  的幂模  $p-1$ ,  $G$  的幂模  $p$

## 广义二项式

对于任意实数  $\alpha$  和非负整数  $k$ :

$$C(\alpha, k) = \frac{\alpha^{\underline{k}}}{k!}$$

$$C(-\alpha, k) = (-1)^k C(\alpha + k - 1, k)$$

有牛顿二项式:

$$(1+x)^\alpha = \sum_{k \geq 0} C(\alpha, k) x^k$$

$$\frac{1}{(1-x)^\alpha} = \sum_{k \geq 0} C(\alpha + k - 1, k) x^k$$

## Fibonacci 数列

$$F_1 = 1, F_2 = 1, F_{n+2} = F_{n+1} + F_n$$

$$F_n = \frac{\sqrt{5}}{5} \left( \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right)$$

## Catalan 数

$$H_0 = 1, H_1 = 1$$

$$H_n = \sum_{i=0}^{n-1} H_i H_{n-i-1} \quad n \geq 2$$

$$H(x) = \sum_{n \geq 0} \frac{C(2n, n)}{n+1} x^n$$

## 错排数

容斥原理  $n! \sum_{k=0}^n \frac{(-1)^k}{k!}$

## 盒子放球方案数

luoguP5824 记得检查  $n \leq m$  之类的合法性条件

给定正整数  $n, m$ , 问  $n$  个球全部放入  $m$  个盒子的方案数, 不同的限制条件如下

### 划分数 (与第二类 Stirling 数相关)

- 球之间互不相同, 盒子之间互不相同。
  - $- m^n$
- 球之间互不相同, 盒子之间互不相同, 每个盒子至多装一个球。
  - $- m^{\underline{n}}$
- 球之间互不相同, 盒子之间互不相同, 每个盒子至少装一个球。
  - $- m! S_2(n, m)$
- 球之间互不相同, 盒子全部相同。
  - $- \sum_{i=1}^m S_2(n, i)$
- 球之间互不相同, 盒子全部相同, 每个盒子至多装一个球。
  - $- [n \leq m]$
- 球之间互不相同, 盒子全部相同, 每个盒子至少装一个球。
  - $- S_2(n, m)$

### 一类不定方程解数

- 球全部相同, 盒子之间互不相同。
  - $- C(n + m - 1, m - 1)$
- 球全部相同, 盒子之间互不相同, 每个盒子至多装一个球。
  - $- C(m, n)$
- 球全部相同, 盒子之间互不相同, 每个盒子至少装一个球。
  - $- C(n - 1, m - 1)$

### 盒子有限的整数分拆

- 球全部相同, 盒子全部相同。
  - $- [x^n] (\prod_{i=1}^m \frac{1}{1-x^i})$
- 球全部相同, 盒子全部相同, 每个盒子至多装一个球。
  - $- [n \leq m]$
- (整数的  $m$  分拆) 球全部相同, 盒子全部相同, 每个盒子至少装一个球。
  - OGF:  $n < m ? 0 : [x^{n-m}] (\prod_{i=1}^m \frac{1}{1-x^i})$ 
    - 记答案为  $p(n, m)$ , 递推式  $p(n, m) = p(n - 1, m - 1) + p(n - m, m)$

## 第二类 Stirling 数

$n$  个带标号球全部放入  $m$  个无标号盒子, 且所有盒子非空的方案数

- $S_2(n, 0) = [n = 0]$
- $S_2(n, m) = S_2(n-1, m-1) + mS_2(n-1, m)$
- $S_2(n, m) = \sum_{i=0}^m \frac{(-1)^i}{i!} \frac{(m-i)^n}{(m-i)!}$  可以卷积算一行
- 考虑  $n$  个带标号球全部放入  $m$  个带标号盒子共有  $m^n$  种方案, 有和式  $m^n = \sum_{k=0}^m S_2(n, k) m^{\underline{k}}$
- 列 EGF  $\sum_{n \geq 0} S_2(n, m) \frac{x^n}{n!} = \frac{1}{m!} (e^x - 1)^m$

## 分拆数

$n$  个无标号球全部放入一些 (个数无限制) 无标号盒子, 要求每个盒子非空

记方案数为  $P(n)$

- $P(n) = \sum_{m=1}^n p(n, m)$
- OGF:  $\sum_{n \geq 0} P(n) x^n = \prod_{n \geq 1} \frac{1}{1-x^n}$

## 五边形数定理

广义五边形数  $g_n = \frac{1}{2}n(3n-1)$

对于  $n = 0, 1, -1, 2, -2, 3, -3, \dots$

$g_n$  的前几项为  $0, 1, 2, 5, 7, 12, 15, 22, 26, \dots$

欧拉函数的 OGF

$$\phi(x) = \prod_{i=1}^{\infty} (1 - x^i) = 1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} = 1 + \sum_{i=1}^{\infty} (-1)^i (x^{i(3i-1)/2} + x^{-i(-3i-1)/2})$$

其中  $i(3i-1)/2$  与  $-i(-3i-1)/2$  恰好是相邻的广义五边形数

由  $P(x)\phi(x) = 1$  得

$$P(1) = 1$$

$$P(n) = P(n-1) + P(n-2) - P(n-5) - P(n-7) + \dots$$

有时可以用来  $O(n\sqrt{n})$  预处理分拆数, 比约  $O(50n \log n)$  的龟速 MTT 省事 (HDU6042)

## 第一类 Stirling 数

求有  $k$  个轮换的  $n$  元置换的方案数, 记为  $S_1(n, k)$

- $S_1(n, 0) = [n = 0]$
- $S_1(n, k) = S_1(n-1, k-1) + (n-1)S_1(n-1, k)$
- 行 OGF (无符号)  $x^{\overline{n}} = \sum_{k=0}^n S_1(n, k) x^k$
- 行 OGF (符号)  $x^{\underline{n}} = \sum_{k=0}^n S_1(n, k) x^k$
- 列 EGF (有符号)  $\sum_{n \geq 0} S_1(n, k) \frac{x^n}{n!} = \frac{1}{k!} \ln^k(1+x)$
- 列 EGF (无符号)  $\sum_{n \geq 0} S_1(n, k) \frac{x^n}{n!} = \frac{1}{k!} \ln^k(1/(1-x))$

## Stirling 反演 (未验证)

$$\sum_{k \geq 0} S_2(n, k) S_1(k, m) = [n = m]$$

$$f_n = \sum_{i=0}^n S_2(n, i) g_i \iff g_n = \sum_{i=0}^n S_1(n, i) f^i$$

## 二维计算几何

- Point 直接支持整型和浮点型
- 部分函数可以对整型改写
- 多边形 (凸包) 按逆时针存在下标 1..n

### 点向量基本运算

```
1  template <typename T>
2  struct Point {
3      T x, y;
4      Point() {}
5      Point(T u, T v) : x(u), y(v) {}
6      Point operator+(const Point &a) const { return Point(x + a.x, y + a.y); }
7      Point operator-(const Point &a) const { return Point(x - a.x, y - a.y); }
8      Point operator*(const T &a) const { return Point(x * a, y * a); }
9      T operator*(const Point &a) const { return x * a.x + y * a.y; }
10     T operator%(const Point &a) const { return x * a.y - y * a.x; }
11     double len() const { return hypot(x, y); }
12     double operator^(const Point &a) const { return (a - (*this)).len(); }
13     double angle() const { return atan2(y, x); }
14     bool id() const { return y < 0 || (y == 0 && x < 0); }
15     bool operator<(const Point &a) const { return id() == a.id() ? (*this) % a > 0 : id() < a.id(); }
16 };
17 typedef Point<double> point;
18
19 #define sqr(x) ((x) * (x))
20 const point O(0, 0);
21 const double PI(acos(-1.0)), EPS(1e-8);
22 inline bool dcmp(const double &x, const double &y) { return fabs(x - y) < EPS; }
23 inline int sgn(const double &x) { return fabs(x) < EPS ? 0 : ((x < 0) ? -1 : 1); }
24 inline double mul(point p1, point p2, point p0) { return (p1 - p0) % (p2 - p0); }
```

### 位置关系

```
1  inline bool in_same_seg(point p, point a, point b) {
2      if (fabs(mul(p, a, b)) < EPS) {
3          if (a.x > b.x) swap(a, b);
4          return (a.x <= p.x && p.x <= b.x && ((a.y <= p.y && p.y <= b.y) || (a.y >= p.y && p.y >= b.y)));
5      } else return 0;
6  }
7
8  inline bool is_right(point st, point ed, point a) {
9      return ((ed - st) % (a - st)) < 0;
10 }
11
12 inline point intersection(point s1, point t1, point s2, point t2) {
13     return s1 + (t1 - s1) * (((s1 - s2) % (t2 - s2)) / ((t2 - s2) % (t1 - s1)));
14 }
15
16 inline bool parallel(point a, point b, point c, point d) {
17     return dcmp((b - a) % (d - c), 0);
18 }
19
20 inline double point2line(point p, point s, point t) {
21     return fabs(mul(p, s, t) / (t - s).len());
22 }
23
24 inline double point2seg(point p, point s, point t) {
25     return sgn((t - s) * (p - s)) * sgn((s - t) * (p - t)) > 0 ? point2line(p, s, t) : min((p ^ s), (p ^ t));
26 }
```

### 多边形

#### 求多边形面积

```
1  inline double area(int n, point s[]) {
2      double res = 0;
3      s[n + 1] = s[1];
```

```

4     for (int i = 1; i <= n; ++i)
5         res += s[i] % s[i + 1];
6     return fabs(res / 2);
7 }

```

### 判断点是否在多边形内

- 特判边上的点
- 使用了  $a[1] \dots a[n+1]$  的数组

```

1 inline bool in_the_area(point p, int n, point area[]) {
2     bool ans = 0; double x;
3     area[n + 1] = area[1];
4     for (int i = 1; i <= n; ++i) {
5         point p1 = area[i], p2 = area[i + 1];
6         if (in_same_seg(p, p1, p2)) return 1; //特判边上的点
7         if (p1.y == p2.y) continue;
8         if (p.y < min(p1.y, p2.y)) continue;
9         if (p.y >= max(p1.y, p2.y)) continue;
10        ans ^= (((p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x) > p.x);
11    }
12    return ans;
13 }

```

### 凸包

- *Andrew* 算法
- $O(n \log n)$
- 可以应对凸包退化成直线/单点的情况但后续旋转卡壳时应注意特判
- 注意是否应该统计凸包边上的点

```

1 inline bool pcmp1(const point &a, const point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }
2
3 inline int Andrew(int n, point p[], point ans[]) { //ans[] 逆时针存凸包
4     sort(p + 1, p + 1 + n, pcmp1);
5     int m = 0;
6     for (int i = 1; i <= n; ++i) {
7         while (m > 1 && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
8         ans[++m] = p[i];
9     }
10    int k = m;
11    for (int i = n - 1; i >= 1; --i) {
12        while (m > k && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
13        ans[++m] = p[i];
14    }
15    return m - (n > 1); //返回凸包有多少个点
16 }

```

### 凸包直径·平面最远点对

- 旋转卡壳算法
- $O(n)$
- 凸包的边上只能有端点，否则不满足严格单峰
- 凸包不能退化成直线，调用前务必检查  $n \geq 3$
- 使用了  $a[1] \dots a[n+1]$  的数组

```

1 inline double Rotating_Caliper(int n, point a[]) {
2     a[n + 1] = a[1];
3     double ans = 0;
4     int j = 2;
5     for (int i = 1; i <= n; ++i) {
6         while (fabs(mul(a[i], a[i + 1], a[j])) < fabs(mul(a[i], a[i + 1], a[j + 1]))) j = (j % n + 1);
7         ans = max(ans, max((a[j] ^ a[i]), (a[j] ^ a[i + 1])));
8     }
9     return ans;
10 }

```

## 平面最近点对

- 分治 + 归并
- $O(n \log n)$

```
1 namespace find_the_closest_pair_of_points {
2     const int N = 200010; //maxn
3     inline bool cmp1(const point &a, const point &b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
4     inline bool operator>(const point &a, const point &b) { return a.y > b.y || (a.y == b.y && a.x > b.x); }
5
6     point a[N], b[N];
7     double ans;
8     inline void upd(const point &i, const point &j) { ans = min(ans, i ^ j); }
9
10    void find(int l, int r) {
11        if (l == r) return;
12        if (l + 1 == r) {
13            if (a[l] > a[r]) swap(a[l], a[r]);
14            upd(a[l], a[r]); return;
15        }
16        int mid = (l + r) >> 1;
17        double mx = (a[mid + 1].x + a[mid].x) / 2;
18        find(l, mid); find(mid + 1, r);
19        int i = l, j = mid + 1;
20        for (int k = l; k <= r; ++k) b[k] = a[(j > r) || (i <= mid && a[j] > a[i]) ? (i++) : (j++)];
21        for (int k = l; k <= r; ++k) a[k] = b[k];
22        int tot = 0;
23        for (int k = l; k <= r; ++k) if (fabs(a[k].x - mx) <= ans) {
24            for (int j = tot; j >= 1 && (a[k].y - b[j].y <= ans); --j) upd(a[k], b[j]);
25            b[++tot] = a[k];
26        }
27    }
28
29    //接口
30    inline double solve(int n, point ipt[]){
31        ans = 0x3f3f3f3f3f3f3f3f; //max distance
32        for (int i = 1; i <= n; ++i) a[i] = ipt[i];
33        sort(a + 1, a + 1 + n, cmp1);
34        find(1, n);
35        return ans;
36    }
37 }
```

## 圆

### 三点垂心

```
1 inline point geto(point p1, point p2, point p3) {
2     double a = p2.x - p1.x;
3     double b = p2.y - p1.y;
4     double c = p3.x - p2.x;
5     double d = p3.y - p2.y;
6     double e = sqr(p2.x) + sqr(p2.y) - sqr(p1.x) - sqr(p1.y);
7     double f = sqr(p3.x) + sqr(p3.y) - sqr(p2.x) - sqr(p2.y);
8     return {(f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2};
9 }
```

### 最小覆盖圆

- 随机增量  $O(n)$

```
1 inline void min_circlefill(point &o, double &r, int n, point a[]) {
2     mt19937 myrand(20011224); shuffle(a + 1, a + 1 + n, myrand); //越随机越难 hack
3     o = a[1];
4     r = 0;
5     for (int i = 1; i <= n; ++i) if ((a[i] ^ o) > r + EPS) {
6         o = a[i];
7         r = 0;
8         for (int j = 1; j < i; ++j) if ((o ^ a[j]) > r + EPS) {
9             o = (a[i] + a[j]) * 0.5;
10            r = (a[i] ^ a[j]) * 0.5;
11        }
12    }
```

```

11         for (int k = 1; k < j; ++k) if ((o ^ a[k]) > r + EPS) {
12             o = geto(a[i], a[j], a[k]);
13             r = (o ^ a[i]);
14         }
15     }
16 }
17 }

```

## 图论

### 存图

- 前向星
- 注意边数开够

```

1 int Head[N], Ver[N*2], Next[N*2], Ew[N*2], Gtot=1;
2 inline void graphinit(int n) {Gtot=1; for(int i=1; i<=n; ++i) Head[i]=0;}
3 inline void edge(int u, int v, int w=1) {Ver[++Gtot]=v; Next[Gtot]=Head[u]; Ew[Gtot]=w; Head[u]=Gtot;}
4 #define go(i,st,to) for (int i=Head[st], to=Ver[i]; i; i=Next[i], to=Ver[i])

```

### 最短路

#### Dijkstra

- 非负权图

```

1 namespace DIJK{//适用非负权图 满足当前 dist 最小的点一定不会被松弛
2     typedef pair<long long,int> pii;
3     long long dist[N];//存最短路长度
4     bool vis[N];//记录每个点是否被从队列中取出 每个点只需第一次取出时扩展
5     priority_queue<pii,vector<pii>,greater<pii> >pq;//维护当前 dist[] 最小值及对应下标 小根堆
6
7     inline void dijk(int s,int n){//s 是源点 n 是点数
8         while(pq.size())pq.pop();for(int i=1;i<=n;++i)dist[i]=INFL,vis[i]=0;//所有变量初始化
9         dist[s]=0;pq.push(make_pair(0,s));
10        while(pq.size()){
11            int now=pq.top().second;pq.pop();
12            if(vis[now])continue;vis[now]=1;
13            go(i,now,to){
14                const long long relx(dist[now]+Ew[i]);
15                if(dist[to]>relx){dist[to]=relx;pq.push(make_pair(dist[to],to));}//松弛
16            }
17        }
18    }
19 }

```

### LCA

- 倍增求 lca
- 数组开够

```

1 namespace LCA_Log{
2     int fa[N][22],dep[N];
3     int t,now;
4     void dfs(int x){
5         dep[x]=dep[fa[x][0]]+1;
6         go(i,x,to){
7             if(dep[to])continue;
8             fa[to][0]=x;for(int j=1;j<=t;++j)fa[to][j]=fa[fa[to][j-1]][j-1];
9             dfs(to);
10        }
11    }
12
13    //初始化接口
14    inline void lcainit(int n,int rt){//记得初始化全部变量
15        now=1;t=0;while(now<n)++t,now<=1;
16        for(int i=1;i<=n;++i)dep[i]=0,fa[i][0]=0;
17        for(int i=1;i<=t;++i)fa[rt][i]=0;

```



```

18     dfs(rt);
19 }
20
21 //求 lca 接口
22 inline int lca(int u,int v){
23     if(dep[u]>dep[v])swap(u,v);
24     for(int i=t;~i;--i)if(dep[fa[v][i]]>=dep[u])v=fa[v][i];
25     if(u==v)return u;
26     for(int i=t;~i;--i)if(fa[u][i]!=fa[v][i])u=fa[u][i],v=fa[v][i];
27     return fa[u][0];
28 }
29 }

```

## 连通性

### 有向图强联通分量

- tarjan  $O(n)$

```

1 namespace SCC{
2     int dfn[N],clk,low[N];
3     bool ins[N];int sta[N],tot; //栈 存正在构建的强连通块
4     vector<int>scc[N];int c[N],cnt;//cnt 为强联通块数 scc[i] 存放每个块内点 c[i] 为原图每个结点属于的块
5     void dfs(int x){
6         dfn[x]=low[x]=(++clk);//low[] 在这里初始化
7         ins[x]=1;sta[++tot]=x;
8         go(i,x,to){
9             if(!dfn[to]){dfs(to);low[x]=min(low[x],low[to]);} //走树边
10            else if(ins[to])low[x]=min(low[x],dfn[to]); //走返祖边
11        }
12        if(dfn[x]==low[x]){ //该结点为块的代表元
13            ++cnt;int u;
14            do{u=sta[tot--];ins[u]=0;c[u]=cnt;scc[cnt].push_back(u);}while(x!=u);
15        }
16    }
17    inline void tarjan(int n){ //n 是点数
18        for(int i=1;i<=cnt;++i)scc[i].clear();//清除上次的 scc 防止被卡 MLE
19        for(int i=1;i<=n;++i)dfn[i]=ins[i]=0;tot=clk=cnt=0; //全部变量初始化
20        for(int i=1;i<=n;++i)if(!dfn[i])dfs(i);
21        for(int i=1;i<=n;++i)c[i]+=n; //此行 (可以省略) 便于原图上加点建新图 加新点前要初始化 Head[]=0
22    }
23 }

```

## 二分图匹配

### 匈牙利算法求二分图无权最大匹配

- 复杂度  $O(nm)$

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 namespace Hungary{
4     const int N=1000009;
5     //图中应储存左部到右部的边 左点编号 1..n 右点编号 1..m
6     int Head[N],Ver[N*2],Next[N*2],Gtot=1; //注意边数开够
7     inline void graphinit(int n){Gtot=1;for(int i=1;i<=n;++i)Head[i]=0;}
8     inline void edge(int u,int v){Ver[++Gtot]=v,Next[Gtot]=Head[u],Head[u]=Gtot;}
9     #define go(i,st,to) for(int i=Head[st],to=Ver[i];i;i=Next[i],to=Ver[i]) //st 是左点, to 是 st 能到达的右点
10
11     int match[N],vis[N]; //右点的匹配点和访问标记
12
13     bool dfs(int x){ //x 是左点
14         go(i,x,to){if(!vis[to]){
15             vis[to]=1;
16             if(!match[to]||dfs(match[to])){
17                 match[to]=x;return 1;
18             }
19         }
20         return 0;
21     }

```

```

22
23 inline int ask(int n,int m){//左点数和右点数 返回最大匹配数
24     for(int i=1;i<=m;++i)match[i]=0;int res=0;
25     for(int i=1;i<=n;++i){
26         for(int j=1;j<=m;++j)vis[j]=0;
27         res+=dfs(i);
28     }
29     return res;
30 }
31 }

```

## KM 算法求二分图带权最大匹配

- 要求该图存在完美匹配该算法将最大化完美匹配的权值和
- 复杂度  $O(n^3)$
- naive 的写法复杂度为  $O((n^2)m)$  在完全图上会退化至  $O(n^4)$  luogu P6577

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  namespace KM{
4      const int N=509;
5      const long long INFLL=0x3f3f3f3f3f3f3f3fll;//注意 INF 应足够大 至少远大于 n*n*Max{W}
6
7      int n;//左右点数均为 n 标号均为 1..n
8      int w[N][N]; //邻接矩阵存边权
9      bool r[N][N]; //记录 i j 之间是否有边连接 完全图/非 0 权图则不必要
10
11     inline void init(){//记得重写初始化
12         int m;scanf("%d",&n,&m);
13         for(int i=1;i<=m;++i){
14             int u,v,wt;scanf("%d%d",&u,&v,&wt);
15             w[u][v]=wt;r[u][v]=1;
16         }
17     }
18
19     long long la[N],lb[N],upd[N]; //左右顶标 每个右点对应的最小 delta 要开 longlong
20     int last[N]; //每个右点对应的回溯右点
21     bool va[N],vb[N];
22     int match[N]; //每个右点对应的左匹配点
23     bool dfs(int x,int fa){
24         va[x]=1;
25         for(int y=1;y<=n;++y){if(r[x][y]&&!vb[y]){
26             const long long dt=la[x]+lb[y]-w[x][y];
27             if(dt==0){//相等子图
28                 vb[y]=1;last[y]=fa;
29                 if(!match[y]||dfs(match[y],y)){
30                     match[y]=x;
31                     return 1;
32                 }
33             }else if(upd[y]>dt){//下次 dfs 直接从最小 delta 处开始
34                 upd[y]=dt;
35                 last[y]=fa;//用 last 回溯该右点的上一个右点以更新增广路
36             }
37         }
38         return 0;
39     }
40
41     inline void KM(){
42         for(int i=1;i<=n;++i){//初始化顶标
43             la[i]=-INFLL;lb[i]=0;
44             for(int j=1;j<=n;++j){if(r[i][j])la[i]=max(la[i],(long long)w[i][j]);}
45         }
46         for(int i=1;i<=n;++i){//尝试给每一个左点匹配上右点 匹配失败则扩展相等子图重试至成功
47             memset(va,0,sizeof va); //注意复杂度
48             memset(vb,0,sizeof vb);
49             memset(last,0,sizeof last);
50             memset(upd,0,sizeof upd);
51             int st=0;match[0]=i;//给起点 i 连一个虚右点 标号为 0
52             //不断尝试将右点 st 从已有的匹配中解放 以获得增广路

```

```

53     while(match[st]){//当 st 到达非匹配右点直接退出
54         long long delta=INFL;
55         if(dfs(match[st],st))break;//st 的左点匹配到了新的右点则退出
56         for(int j=1;j<=n;++j)
57             if(!vb[j]&&upd[j]<delta){//下次从最小的 delta 处开始 DFS
58                 delta=upd[j];
59                 st=j;
60             }
61         for(int j=1;j<=n;++j){//将交错树上的左顶标加 delta 右顶标减 delta 使更多的边转为相等边
62             if(va[j])la[j]-=delta;
63             if(vb[j])lb[j]+=delta;
64             else upd[j]-=delta;//小问题：这里在干啥 每次修改顶标后重新计算 upd 是否可行
65         }
66         vb[st]=1;
67     }
68     while(st){//更新增广路
69         match[st]=match[last[st]];
70         st=last[st];
71     }
72 }
73
74 long long ans=0;
75 for(int i=1;i<=n;++i)ans+=w[match[i]][i];
76 printf("%lld\n",ans);
77 for(int i=1;i<=n;++i)printf("%d%c",match[i]," \n"[i==n]);
78 }
79 //signed main(){init();KM();return 0;}
80 }

```

## 数据结构

### STL

#### 小跟堆

```
1 priority_queue<vector<int>, int, greater<int> > q;
```

#### 整数哈希

```

1  #include<bits/stdc++.h>
2  #include <ext/pb_ds/assoc_container.hpp>
3  #include <ext/pb_ds/hash_policy.hpp>
4  using namespace std;
5  typedef unsigned long long u64;
6
7  struct Neal {
8      static u64 A(u64 x) {
9          x += 0x9e3779b97f4a7c15;
10         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
11         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
12         return x ^ (x >> 31);
13     }
14     u64 operator()(u64 x) const{
15         static const u64 C = chrono::steady_clock::now().time_since_epoch().count();
16         return A(x + C);
17     }
18 };
19
20 unordered_map<int, int, Neal> HASH1;
21
22 __gnu_pbds::gp_hash_table<int, int, Neal> HASH2;

```

### 二维树状数组

- LOJ135 区间修改 + 区间查询单次操作复杂度  $\log^2$  空间复杂度  $N^2$
- 对原数组差分  $S(n,m) = \sum \sum A_{ij}$  用树状数组维护差分数组  $A_{ij}$  转化为单点修改
- 区间查询  $SS(n,m) = \sum \sum S(i,j)$  推式子转化为单点查询问题

- $SS(n,m) = \sum \sum \sum A_{ij} = (n+1) * (m+1) * \sum A_{ij} - (n+1) * \sum j A_{ij} - (m+1) * \sum i A_{ij} + \sum ij A_{ij}$

```

1 //省略了文件头
2 const int N=2051;
3 int n,m;
4 struct dat{
5     LL c,cx,cy,cxy;
6     void operator+=(const dat&a){c+=a.c;cx+=a.cx;cy+=a.cy;cxy+=a.cxy;}
7 }c[N][N];
8
9 inline void add(int a,int b,int k){
10     const dat d={k,k*a,k*b,k*a*b};
11     for(int i=a;i<=n;i+=(i&(-i)))
12         for(int j=b;j<=m;j+=(j&(-j)))c[i][j]+=d;
13 }
14
15 inline LL f(int a,int b){
16     dat r={0,0,0,0};
17     for(int i=a;i>0;i-=i&(-i))
18         for(int j=b;j>0;j-=j&(-j))r+=c[i][j];
19     return (a+1)*(b+1)*r.c-(b+1)*r.cx-(a+1)*r.cy+r.cxy;
20 }
21
22 signed main(){
23     read(n,m);
24     int op;while(read(op))if(op==1){
25         int a,b,c,d,k;read(a,b,c,d,k);
26         add(a,b,k);add(c+1,b,-k);add(a,d+1,-k);add(c+1,d+1,k);
27     }else{
28         int a,b,c,d;read(a,b,c,d);
29         printf("%lld\n",f(c,d)-f(a-1,d)-f(c,b-1)+f(a-1,b-1));
30     }
31     return 0;
32 }

```

## 堆式线段树

- 区间求和区间修改
- 空间估算
- 所有数组务必初始化

```

1 struct SegmentTree_Heap{
2     #define TreeLen (N<<2) //N
3     #define lc(x) ((x)<<1)
4     #define rc(x) ((x)<<1|1)
5     #define sum(x) (tr[x].sum) //
6     #define t(x) (t[x]) //
7
8     struct dat{
9         LL sum;
10         /* 满足结合律的基本运算 用于合并区间信息 */
11         dat operator+(const dat&brother){
12             dat result;
13             result.sum=sum+brother.sum;
14             return result;
15         }
16     }tr[TreeLen];
17     LL t[TreeLen]; //lazy tag
18
19     /* 单区间修改 */
20     inline void change(const int&x,const int&l,const int&r,const LL&d){
21         tr[x].sum=tr[x].sum+d*(r-l+1);
22         t[x]=t[x]+d;
23     }
24
25     inline void pushup(int x){tr[x]=tr[lc(x)]+tr[rc(x)];}
26
27     inline void pushdown(int x,const int&l,const int&r,const int&mid){
28         if(t(x)){ // 区间修改注意细节
29             change(lc(x),l,mid,t(x));
30             change(rc(x),mid+1,r,t(x));

```

```

31         t(x)=0;
32     }
33 }
34
35 void build(int x,int l,int r){
36     t(x)=0;    // 记得初始化
37     if(l==r){
38         sum(x)=0;
39         return;
40     }
41     int mid=(l+r)>>1;
42     build(lc(x),l,mid);
43     build(rc(x),mid+1,r);
44     pushup(x);
45 }
46
47 void add(int x,int l,int r,const int&L,const int&R,const LL&d){
48     if(L<=l&&r<=R){
49         change(x,l,r,d);
50         return;
51     }
52     int mid=(l+r)>>1;pushdown(x,l,r,mid);
53     if(L<=mid)add(lc(x),l,mid,L,R,d);
54     if(R>mid)add(rc(x),mid+1,r,L,R,d);
55     pushup(x);
56 }
57
58 LL ask(int x,int l,int r,const int&L,const int&R){
59     if(L<=l&&r<=R)return sum(x);
60     int mid=(l+r)>>1;pushdown(x,l,r,mid);
61     LL res=0;
62     if(L<=mid)res=(res+ask(lc(x),l,mid,L,R));
63     if(mid<R)res=(res+ask(rc(x),mid+1,r,L,R));
64     return res;
65 }
66 };

```

## 小根堆

```

1 namespace MyPQ{
2     typedef int pqdat;    ////
3     pqdat q[N];
4     int tot;
5     void up(int x){
6         while(x>1)
7             if(q[x]<q[x/2]){
8                 swap(q[x],q[x/2]);
9                 x/=2;
10            }else return;
11    }
12    void down(int x){
13        int ls=x*2;
14        while(ls<=tot){
15            if(ls<tot&&q[ls+1]<q[ls])++ls;
16            if(q[ls]<q[x]){
17                swap(q[x],q[ls]);x=ls;ls=x*2;
18            }else return;
19        }
20    }
21    void push(pqdat x){q[++tot]=x;up(tot);}
22    pqdat top(){return q[1];}
23    void pop(){if(!tot)return;q[1]=q[tot--];down(1);}
24    void pop(int k){if(!tot)return;q[k]=q[tot--];up(k);down(k);}
25 }

```

## Treap

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int N=2000007,INF=(1ll<<30)+7;

```

```

4
5 //Treap 维护升序多重集
6 //支持操作: 数 <-> 排名 查询某数前驱后继
7 //操作数 x 可以不在集合中
8 //x 的排名: 集合中 <x 的数的个数 +1
9 //排名 x 的数: 集合中排名 <=x 的数中的最大数
10 //x 的前驱 比 x 小的最大数
11
12 struct treap{//所有点值不同 用副本数实现多重集
13     int l,r;
14     int v,w;//v 是数据 w 是维护堆的随机值
15     int num,sz;//num 是该点副本数 sz 是该子树副本总数
16 }tr[N];int tot,rt;//tr[0] 始终全 0 使用范围 tr[1..n]
17 #define lc(x) tr[x].l
18 #define rc(x) tr[x].r
19 #define sz(x) tr[x].sz
20 #define num(x) tr[x].num
21 #define val(x) tr[x].v
22 #define wt(x) tr[x].w
23
24 inline int New(int x){
25     val(++tot)=x; wt(tot)=rand();
26     num(tot)=sz(tot)=1; return tot;
27 }
28
29 inline void upd(int p){sz(p)=sz(lc(p))+sz(rc(p))+num(p);}
30
31 inline void build(){//初始化 INF 和-INF 两个点
32     srand(time(0));
33     rt=1;tot=2;
34     rc(1)=2;val(1)=-INF;wt(1)=rand();num(1)=1;sz(1)=2;
35     val(2)=INF;wt(2)=rand();num(2)=1;sz(2)=1;
36 }
37
38 //调用时记得减一 askrk(rt,x)-1
39 int askrk(int p,int x){//当前子树中查询 x 的排名
40     if(p==0)return 1;//说明某子树所有数均比 x 大
41     if(x==val(p))return sz(lc(p))+1;
42     return x<val(p)?askrk(lc(p),x):askrk(rc(p),x)+sz(lc(p))+num(p);
43 }
44
45 //调用时记得加一 kth(rt,++rank)
46 int kth(int p,int rk){//当前子树中查询排名 rk 的数
47     if(p==0)return INF;//说明集合大小 <rk
48     if(sz(lc(p))>=rk)return kth(lc(p),rk);
49     rk-=sz(lc(p))+num(p);
50     return (rk>0)?kth(rc(p),rk):val(p);
51 }
52
53 inline void zig(int &p){//与左子节点交换位置
54     int q=lc(p);lc(p)=rc(q);rc(q)=p;
55     upd(p);p=q;upd(p);
56 }
57
58 inline void zag(int &p){//与右子节点交换位置
59     int q=rc(p);rc(p)=lc(q);lc(q)=p;
60     upd(p);p=q;upd(p);
61 }
62
63 //insert(rt,x)
64 void insert(int &p,int x){//当前子树中插入 x
65     if(p==0){p=New(x);return;}//x 首次插入
66     if(x==val(p)){++num(p);++sz(p);return;}
67     if(x<val(p)){
68         insert(lc(p),x);
69         if(wt(p)<wt(lc(p)))zig(p);//维护大根堆
70     }else{
71         insert(rc(p),x);
72         if(wt(p)<wt(rc(p)))zag(p);//维护大根堆
73     }
74     upd(p);

```

```

75 }
76
77 //erase(rt,x)
78 void erase(int &p,int x){//当前子树中删除一个 x
79     if(p==0)return;//已经无需删除
80     if(val(p)==x){//如果找到了 x 的位置
81         if(num(p)>1){//无需删点
82             --num(p);--sz(p);return;//如果有多个 x 维护副本数即可
83         }
84         if(lc(p)||rc(p)){//该点不是叶子节点 则不断向下调整至叶子节点
85             if(rc(p)==0||wt(lc(p))>wt(rc(p)))zig(p),erase(rc(p),x);//由于 rand() 的值域 & 大根堆的实现 故省略左子树为空的判断
86             else zag(p),erase(lc(p),x);
87             upd(p);
88         }else p=0;//是叶子节点则直接删除
89         return;
90     }
91     x<val(p)?erase(lc(p),x):erase(rc(p),x);upd(p);
92 }
93
94 int askpre(int x){
95     int id=1;//-INF 若没有前驱则返回-INF
96     //尝试自顶向下寻找 x 则 x 的前驱有两种情况
97     //1) 未找到 x 或 x 没有左子树 则前驱在搜索路径上
98     //2) 前驱是 x 的左子树中最大值 即 x 的左子树一直向右走
99     int p=rt;
100     while(p){
101         if(x==val(p)){//找到 x
102             if(lc(p)){p=lc(p);while(rc(p))p=rc(p);id=p;}
103             break;
104         }
105         if(val(p)<x&&val(p)>val(id))id=p;//每经过一个点尝试更新前驱
106         p=(val(p)>x?lc(p):rc(p));//找 x
107     }
108     return val(id);
109 }
110
111 int asknxt(int x){
112     int id=2;//INF
113     int p=rt;
114     while(p){
115         if(x==val(p)){
116             if(rc(p)){p=rc(p);while(lc(p))p=lc(p);id=p;}
117             break;
118         }
119         if(val(p)>x&&val(p)<val(id))id=p;
120         p=(val(p)>x?lc(p):rc(p));
121     }
122     return val(id);
123 }

```

## 字符串

### KMP

```

1 //luogu P3375
2 #include <bits/stdc++.h>
3 using namespace std;
4 const int N = 1000009;
5
6 char s1[N], s2[N];
7 int fail[N], n, m;
8
9 signed main() {
10     scanf("%s%s", s1 + 1, s2 + 1);
11     n = strlen(s1 + 1);
12     m = strlen(s2 + 1);
13
14     //fail[1] = 0;
15     for (int i = 2, j = 0; i <= m; ++i) {
16         while (j != 0 && s2[j + 1] != s2[i]) j = fail[j];

```

```

17         if (s2[j + 1] == s2[i]) ++j;
18         fail[i] = j;
19     }
20
21     int p = 0;
22     for (int i = 1; i <= n; ++i) {
23         while (p != 0 && s2[p + 1] != s1[i]) p = fail[p];
24         if (s2[p + 1] == s1[i]) ++p;
25
26         if (p == m) {
27             printf("%d\n", i - p + 1);
28             p = fail[p];
29         }
30     }
31
32     for (int i = 1; i <= m; ++i) printf("%d ", fail[i]);
33     return 0;
34 }

```

## 扩展 KMP(Z 函数)

```

1 //luogu 5410
2 #include <bits/stdc++.h>
3 using namespace std;
4 const int N = 20000009;
5
6 char t[N], p[N];
7 int n, m, z[N], mc[N];
8
9 signed main() {
10     scanf("%s%s", t + 1, p + 1);
11     n = strlen(t + 1);
12     m = strlen(p + 1);
13
14     z[1] = m;
15     for (int i = 2, l = 0, r = 0; i <= m; ++i) {
16         z[i] = ((r < i) ? 0 : min(r - i + 1, z[i - l + 1]));
17         while (z[i] <= m - i && p[i + z[i]] == p[z[i] + 1]) ++z[i];
18         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
19     }
20
21     for (int i = 1, l = 0, r = 0; i <= n; ++i) {
22         mc[i] = ((r < i) ? 0 : min(r - i + 1, z[i - l + 1]));
23         while (mc[i] <= n - i && mc[i] < m && t[i + mc[i]] == p[mc[i] + 1])
24             ++mc[i];
25         if (i + mc[i] - 1 > r) l = i, r = i + mc[i] - 1;
26     }
27
28     unsigned long long u = 0, v = 0;
29     for (int i = 1; i <= m; ++i) u ^= (i * (z[i] + 1ull));
30     for (int i = 1; i <= n; ++i) v ^= (i * (mc[i] + 1ull));
31     printf("%llu\n%llu\n", u, v);
32     return 0;
33 }

```

## AC 自动机

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int N=1000009;
4
5 //洛谷 P3808 AC 自动机 (简单版)
6 int n;
7 char s[N];
8 struct AC{
9     struct dat{
10         int tp[26],fail,ed;
11         void init(){memset(tp,0,sizeof tp);fail=ed=0;}
12     }tr[N];int tot;
13     void init(){for(int i=0;i<=tot;++i)tr[i].init();tot=0;}

```



```

14
15 void insert(char*s){//C 风格字符串 下标从 1 开始
16     int p=0;
17     for(int i=1;s[i]!=0;++i){//C 风格字符串 下标从 1 开始
18         int&tp=tr[p].tp[s[i]-'a'];
19         if(tp==0)tp=(++tot);
20         p=tp;
21     }
22     ++tr[p].ed;
23 }
24
25 void build(){//先 insert 所有模式串 然后 build
26     queue<int>q;
27     for(int i=0;i<26;++i)if(tr[0].tp[i])q.push(tr[0].tp[i]);
28     while(q.size()){
29         dat&now=tr[q.front()];q.pop();
30         for(int i=0;i<26;++i){
31             int&tp=now.tp[i];
32             if(tp)tr[tp].fail=tr[now.fail].tp[i],q.push(tp);
33             else tp=tr[now.fail].tp[i];//路径压缩
34         }
35     }
36 }
37
38 int ask(char*s){//C 风格字符串 下标从 1 开始
39     int p=0,res=0;
40     for(int i=1;s[i]!=0;++i){//C 风格字符串 下标从 1 开始
41         p=tr[p].tp[s[i]-'a'];
42         for(int j=p;tr[j].ed!=-1;j=tr[j].fail)res+=tr[j].ed,tr[j].ed=-1;
43     }
44     return res;
45 }
46 }ac;
47
48 signed main(){
49     scanf("%d",&n);
50     for(int i=1;i<=n;++i){
51         scanf("%s",s+1);
52         ac.insert(s);
53     }
54     ac.build();
55     scanf("%s",s+1);
56     printf("%d\n",ac.ask(s));
57     return 0;
58 }

```

## 杂项

### 整数哈希

```

1  #include<bits/stdc++.h>
2  #include<ext/pb_ds/assoc_container.hpp>
3  #include<ext/pb_ds/hash_policy.hpp>
4  using namespace std;
5  using namespace __gnu_pbds;
6
7  //https://codeforces.com/blog/entry/62393
8
9  struct custom_hash {
10     static uint64_t splitmix64(uint64_t x) {
11         // http://xorshift.di.unimi.it/splitmix64.c
12         x += 0x9e3779b97f4a7c15;
13         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
14         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
15         return x ^ (x >> 31);
16     }
17
18     size_t operator()(uint64_t x) const {
19         static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
20         return splitmix64(x + FIXED_RANDOM);

```

```

21     }
22 };
23
24 unordered_map<long long, int, custom_hash> safe_map;
25 gp_hash_table<long long, int, custom_hash> safe_hash_table;

```

## GCC 位运算

需要 gcc 编译器

- `int __builtin_ffs (unsigned int x)` 返回 `x` 的最后一位 1 的是从后向前第几位，比如 7368 (1110011001000) 返回 4。
- `int __builtin_clz (unsigned int x)` 返回前导的 0 的个数。
- `int __builtin_ctz (unsigned int x)` 返回后面的 0 的个数，和 `__builtin_clz` 相对。
- `int __builtin_popcount (unsigned int x)` 返回二进制表示中 1 的个数。
- `int __builtin_parity (unsigned int x)` 返回 `x` 的奇偶校验位，也就是 `x` 的 1 的个数模 2 的结果。

此外，这些函数都有相应的 `unsigned long` 和 `unsigned long long` 版本，只需要在函数名后面加上 `l` 或 `ll` 就可以了，比如 `int __builtin_clzll`。

来源 <https://blog.csdn.net/yuer158462008/article/details/46383635>

## 对拍

for windows

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  //输入文件  input.in
5  //正确的程序名  ac.cpp  ac.exe  ac.out
6  //待验证的程序名  wa.cpp  wa.exe  wa.out
7
8  void gen(){//生成 INPUT 内的数据
9      FILE *op=fopen("input.in","w");
10     //mt19937 RAND(time(nullptr));
11     //uniform_int_distribution<int>(1,100)(RAND);//范围内均匀随机整数
12     //fprintf(op,"%d\n",233);
13     fclose(op);
14 }
15
16 bool check(){//比较 ac.out 和 wa.out 的内容 一致则返回 0 否则返回非 0
17     return system("fc wa.out ac.out");
18 }
19
20 int main(){
21     system("g++ wa.cpp -o wa.exe");
22     system("g++ ac.cpp -o ac.exe");
23     while(1){
24         gen(); int st,ed;
25
26         st=clock();
27         system("ac.exe < input.in > ac.out");
28         ed=clock();
29         printf("ac cost %lld ms\n",(ed-st)*1000ll/CLOCKS_PER_SEC);
30
31         st=clock();
32         system("wa.exe < input.in > wa.out");
33         ed=clock();
34         printf("wa cost %lld ms\n",(ed-st)*1000ll/CLOCKS_PER_SEC);
35
36         if(check()){
37             puts("Wrong Answer");
38             system("pause");
39             return 0;
40         }
41     }

```

```

42     return 0;
43 }

```

## fread/fwrite 实现的读写类

没用

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct MY_IO {
5      #define MAXSIZE (1 << 20) //缓冲区大小 不小于 1 << 14
6      inline bool isdigit(const char &x) {return x >= '0' && x <= '9';} //字符集 看情况改
7      inline bool blank(const char &c) { return c == ' ' || c == '\n' || c == '\r' || c == '\t'; }
8
9      char buf[MAXSIZE + 1], *p1, *p2, pbuf[MAXSIZE + 1], *pp;
10     MY_IO() : p1(buf), p2(buf), pp(pbuf) {}
11     ~MY_IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }
12
13     char gc() {
14         if (p1 == p2) p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
15         return p1 == p2 ? EOF : *p1++;
16     }
17
18     void pc(const char &c) {
19         if (pp - pbuf == MAXSIZE) fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
20         *pp++ = c;
21     }
22
23     template <typename T>
24     bool read(T &x) {
25         x = 0; char c = gc(); int f = 1;
26         while (!isdigit(c) && (c != '-') && (c != EOF)) c = gc();
27         if (c == EOF) return 0;
28         if (c == '-') f = -1, c = gc();
29         while (isdigit(c)) { x = x * 10 + (c & 15); c = gc(); }
30         x *= f; return 1;
31     }
32
33     template <typename T, typename... Args>
34     bool read(T &x, Args &...args) {
35         bool res = 1;
36         res &= read(x);
37         res &= read(args...);
38         return res;
39     }
40
41     int gets(char *s) {
42         char c = gc();
43         while (blank(c) && c != EOF) c = gc();
44         if (c == EOF) return 0;
45         int len = 0;
46         while (!blank(c) && c != EOF) *s++ = c, c = gc(), ++len;
47         *s = 0; return len;
48     }
49
50     void getc(char &c) { for (c = gc(); blank(c) && c != EOF; c = gc()); }
51
52     template <typename T>
53     void write(T x) {
54         if (x < 0) x = -x, putchar('-');
55         static char sta[55];
56         int top = 0;
57         do sta[top++] = x % 10 + '0', x /= 10; while (x);
58         while (top) putchar(sta[--top]);
59     }
60
61     template <typename T>
62     void write(T x, const char &Lastchar) {write(x); pc(Lastchar);}
63
64     void puts(char *s) {while ((*s) != 0)pc(*s++);}

```

```

65
66     int getline(char *s){
67         char c = gc(); int len = 0;
68         while (c != '\n' && c != EOF)*s++ = c, c = gc(), ++len;
69         *s = 0; return len;
70     }
71
72     void putline(char *s){ while ((*s) != 0) pc(*s++); pc('\n');}
73 } IO;
74
75 #define read IO.read    //读一个/多个整数
76 #define write IO.write  //写一个整数
77 #define gc IO.gc        //getchar()
78 #define pc IO.pc        //putchar()
79 #define gets IO.gets     //读一个指定字符集的 C 风格字符串 到 s[0..len-1] 返回 len
80 #define getc IO.getc     //读一个指定字符集的字符
81 #define puts IO.puts     //写一个 C 风格字符串
82 #define getl IO.getline  //读一行
83 #define putl IO.putline  //写一个 C 风格字符串并换行

```

## Python 最短路

```

1  EPS=1e-6
2  M=1000000007
3  INF=0x3f3f3f3f
4  INFL=0x3f3f3f3f3f3f3f3f
5  N=100010
6  def pause():input("\npress the enter key to exit")
7  import math
8  import sys
9
10 # 二元组
11 class dat:
12     def __init__(self, f=0, s=0):
13         self.first=f
14         self.second=s
15     def __lt__(self, a):
16         return self.second<a.second if self.first==a.first else self.first<a.first
17
18 # 小根堆
19 class Priority_Queue: # 数据类型必须支持 < 比较
20     def __init__(self):
21         self.q=[0]
22         self.tot=0
23     def up(self, x):
24         while(x>1):
25             if(self.q[x]<self.q[x>>1]):
26                 self.q[x], self.q[x>>1]=self.q[x>>1], self.q[x]
27                 x>>=1
28             else:return
29     def down(self, x):
30         s=x<<1
31         while(s<=self.tot):
32             if(s<self.tot and (self.q[s|1]<self.q[s])):s|=1
33             if(self.q[s]<self.q[x]):
34                 self.q[x], self.q[s]=self.q[s], self.q[x]
35                 x=s
36             else:return
37         s<<=1
38     def clear(self):
39         q=[0]
40         self.tot=0
41     def size(self):return self.tot
42     def push(self, thenew):
43         self.q.append(thenew)
44         self.tot+=1
45         self.up(self.tot)
46     def top(self):return self.q[1]
47     def pop(self, k=1):
48         if(self.tot==0):return
49         self.q[self.tot], self.q[k]=self.q[k], self.q[self.tot]

```

```

50         self.tot-=1
51         self.q.pop()
52         self.up(k)
53         self.down(k)
54 # 小根堆
55
56 # 前向星存图
57 class qxx:
58     def __init__(self):
59         self.nex=0
60         self.to=0
61         self.v=0
62 e=[qxx() for i in range(N<<1)]
63 h=[0 for i in range(N)]
64 etot=0
65 def add(u,v,w):
66     global e,h,etot
67     etot+=1
68     e[etot].nex=h[u]
69     h[u]=etot
70     e[etot].to=v
71     e[etot].w=w
72 def alledge(u): # 访问某点的所有出边
73     i=h[u]
74     while i:
75         yield i
76         i=e[i].nex
77 # 前向星存图
78
79 dis=[INF for i in range(N)]
80 q=Priority_Queue()
81 def dij(s):
82     dis[s]=1
83     q.push(dat(1,s))
84     while(q.size()):
85         j=q.top()
86         q.pop()
87         if(dis[j.second]<j.first)and(dis[j.second]!=INF):continue
88         for i in alledge(j.second):
89             k=e[i].to
90             w=e[i].w
91             if(dis[k]>dis[j.second]*w)or(dis[k]==INF):
92                 dis[k]=dis[j.second]*w
93                 q.push(dat(dis[k],k))
94
95
96 def main():
97     n,m=map(int,input().split())
98     for line in sys.stdin:
99         if(m==0):break
100         u,v,w=map(int,line.split())
101         add(u,v,w)
102         m-=1
103     dij(1)
104     print(dis[n]%9987)
105     return 0
106
107
108 main()
109 #pause()

```