第八章 数学提高

8.1 数论

8.1.1 Miller-Rabin 定理

$Miller-rabin$ 算法是一个用来快速判断一个正整数是否为素数的算法，它利用了费马小定理和二次探测。

1. 费马小定理：如果 p 是质数且 $a \perp p$，那么 $a^{p-1} \equiv 1 \ (mod \ p)$。

   也就是对于所有小于 p 的正整数 a 来说都应该符合 $a^{p-1} \equiv 1 \ (mod \ p)$。那么根据逆否命题，对于一个 p，我们只要举出一个 $a(a < p)$ 不符合这个恒等式，则可判定 p 不是素数。$Miller-rabin$ 算法就是多次用不同的 a 来尝试 p 是否为素数。

2. 二次探测定理：如果 p 是一个素数，那么对于正整数 $x(x < p)$，若 $x^2 \equiv 1 \ (mod \ p)$，则 $x = 1 \ or \ x = p - 1$。逆否命题：如果对于正整数 $x(x < p)$，若 $x^2 \equiv k \ (mod \ p), k \neq 1$，则 p 不是素数（二次探测定理的正确性可以通过求解二次同余方程验证）

根据二次探测定理，我们要判断 $a^{p-1} \ mod \ p$ 是否和 1 同余时，可以这样计算，设 $p - 1 = k * 2^t$，从 a^k 开始，不断将其平方直到得到 a^{p-1}，一旦发现某次平方并对 p 取模后等于 1，那么说明符合了二次探测定理的逆否命题使用条件，立即检查 x 是否等于 1 或 $p - 1$，如果都不是则可直接判定 p 为合数。

```
inline ll mul(ll a, ll b, ll p) {    //wa 了尝试慢速乘
    if (p <= 1000000000) return a * b % p;  //1e9
    else if (p <= 1000000000000LL) return (((a * (b >> 20) % p) << 20) + (a * (b & ((1 << 20) - 1)))) % p;  //1e12
    else {
        ll d = (ll) floor(a * (long double) b / p + 0.5);
        ll ret = (a * b - d * p) % p;
        if (ret < 0) ret += p;
        return ret;
    }
}

//inline ll mul(ll x, ll n, ll p) {
//    ll ans = 0;
//    x %= p;
//    while (n) {
```

```
//        if (n & 1) ans = (ans + x) % p;
//        x = (x + x) % p;
//        n >>= 1;
//    }
//    return ans;
//}

inline ll qkp(ll x, ll n, ll p) {
    ll ans = 1;
    x %= p;
    while (n) {
        if (n & 1) ans = mul(ans, x, p);
        x = mul(x, x, p);
        n >>= 1;
    }
    return ans;
}

bool miller_rabin(ll n, int t = 20) {
    if (n == 2) return true;
    if (n < 2 || !(n & 1)) return false;
    ll m = n - 1;
    int k = 0;
    while ((m & 1) == 0) {
        m >>= 1;
        k++;
    }
    for (int i = 0; i < t; i++) {
        ll a = rand() % (n - 1) + 1;
        ll x = qkp(a, m, n), y = 0;
        for (int j = 0; j < k; j++) {
            y = mul(x, x, n);
            if (y == 1 && x != 1 && x != n - 1) return false;
            x = y;
        }
        if (y != 1) return false;
    }
    return true;
}
```

如果需要高精度的素数判断，Java 的 BigInterger 类封装了米勒罗宾素数判定算法。

```
import java.math.BigInteger;
import java.util.*;

public class Main{
```

```
    public static void main(String[] args){
        BigInteger x;
        Scanner in = new Scanner(System.in);
        x = in.nextBigInteger();
        if(x.isProbablePrime(50))            //相当于该数是素数的概率超过 99%
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

8.1.2 Pollard-Rho 算法

1. 算法背景

1975 年，John M. Pollard 提出了第二种因数分解的方法，Pollard Rho 快速因数分解。该算法时间复杂度为$O(n^{0.25})$。算法推导过程是由暴力算法逐步优化得到。该算法是快速找到一个大整数 n 的非平凡因子（不是 1 和它本身的因子）。

2. 算法原理

朴素算法：通过 $O(\sqrt{n})$ 复杂度的枚举，可以得到 n 的一个因数。

```
int getNTFactor() {
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0)
            return i;
    return -1;
}
```

尝试的改进：

考虑一个问题，在 $[1,n]$ 里随机选取 i 个数（$i=1$ 时就是它自己），使它们之间有两个数的差值为 k（若只有一个数只有 k 满足）。当 $i=1$ 时成功的概率是 $\frac{1}{n}$，当 时成功的概率是 $\frac{1}{\frac{n}{2}}$，随着 i 的增大，这个概率也会增大最后趋向于 1。

先用米勒罗宾算法判断素数。然后从 $[2,n-1]$ 范围内随机素数。在最坏的情况下，n 是一个质数的平方，设 $n=p^2$，当 $d=p,2p,3p,\dots,(p-1)*p$ 时，都有 $gcd(d,n)>1$，最坏时间复杂度为 $O(\sqrt{n}\log n)$。

虽然增加了时间复杂度，但是为后续算法的优化提供了思路。

```
int getNTFactor() {
    if (MillerRabin(n)) return n;
    int x, d = 2;
    while (d > 1) {
        x = rand() % (n - 3) + 2;  // 生成 2 和 n-1 之间的随机数
```

```
6.          d = gcd(n, x);       // 求 gcd
7.      }
8.      return d;
9. }
```

3. 生日悖论

问题：若不考虑出生年份，则一个房间中至少有多少人，才能使其中两个人生日相同的概率达到$\frac{1}{2}$？

解：假设一年有 n 天，房间中有 k 人，用整数 $1,2,3,\dots,k$ 对这些人进行编号。假定每个人的生日都为均匀分布且相互独立。那么 k 个人生日互不相同的概率为 $P=1\times\frac{n-1}{n}\times\cdots\times\frac{n-k+1}{n}$

至少有两个人生日互不相同的概率为 $1-P$，根据不等式 $1+x\le e^x$，得到：$e^{-\frac{k*(k-1)}{2n}}\le\frac{1}{2}$

代入数据，得到 $k=23$。

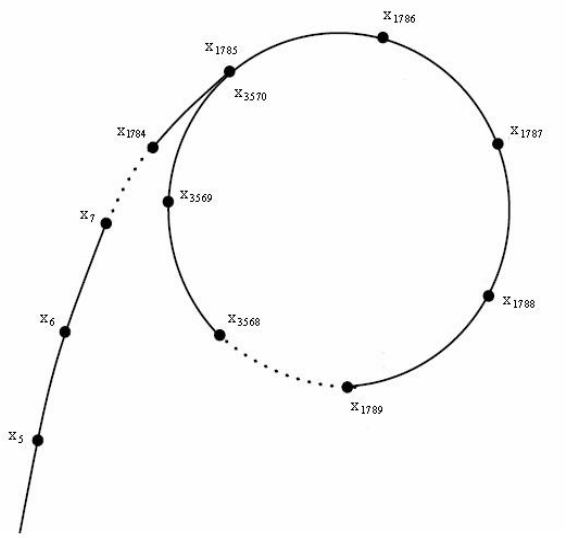
生日悖论启示我们，如果不断在某个范围内生成随机整数，很快便会生成到重复的数，期望大约在根号级别。

在上一个问题中，应用到原题上，即是对于最坏情形 $n=p^2$，如果不断在 $[1,n-1]$ 间生成随机数，那么期望在生成大约 $\sqrt{p}=n^{\frac{1}{4}}$ 个数后，可以出现两个在模 p 下相同的数（因为 $[1,n-1]$ 间的随机数模 p 大致是 $[0,p-1]$ 间的随机数）。那么这两个数的**差**的绝对值 d，就一定满足 $d\equiv 0(\text{mod } p)$，于是也满足 $gcd(d,n)>1$。尽管期望生成次数不多，但是要得到一个非平凡因子是需要在这些数中两两计算比较的，于是最坏的时间复杂度又会退化到 $\sqrt{n}\log n$。

4. 伪随机数序列

使用伪随机函数 $f(x)=(x^2+c)\bmod n$ 来生成一个伪随机序列 $\{x_i\}=x_1,x_2,x_3,\cdots$，其中 $x_2=f(x_1),x_3=f(x_2),\dots,x_n=f(x_{n-1})$

其实这是显然的，因为每个数都是由前一个数决定的，可以生成的数又是有限的，那么一定会进入**循环**。但是循环很可能是**混循环**，生成的序列常常形成下图所示的 ρ 形，这也是为什么 Pollard 把这个算法命名为 rho 算法。



使用伪随机数序列有一个性质。设 m 是 n 的一个非平凡因子，序列中两个数 x_i, x_j，若 $|i-j| \equiv \varphi \pmod{m}$，那么有 $\left|f(x_i)-f\left(x_j\right)\right| = \left|x_i^2 - x_j^2\right| = |x_i + y_i| - |x_i - y_i| \equiv \varphi \pmod{m}$

由此可得，**环上任意两个距离为 d 的数在该问题的意义下是等价的**。

5. 龟兔赛跑算法

Flyod 判环算法又称龟兔赛跑判环算法。假设已经随机生成了伪随机函数的常量 c，设置两个变量 $t = f(0), r = f\left(f(0)\right)$，每次判断是否有 $gcd(|t-r|, N) > 1$，如果没有，就令 $t = f(t), r = f\left(f(r)\right)$。因为 r 每次向前走两步，t 每次向前走一步，如果没有找到答案，最终会与 t 在环上相遇，这时算法结束退出。伪随机函数会重新随机一个 c 重新生成伪随机数，直到找到一个非平凡因子退出。

在 Floyd 判环的过程中，每次移动都相当于在检查一个新的距离 d，这样就不需要进行两两比较了。

```
1. ll f(ll x, ll c) {
2.     return (x * x % n + c) % n
3. }
4.
5. ll PollardRho(ll n) {
6.     if (n == 4) return 2;  //特判 4
7.     if (MillerRabin(n)) return n;  //特判质数
8.     while (1) {
9.         ll c = rand() % (n - 1) + 1;
10.        while (t != r) {
11.            ll d = gcd(abs(t - r), N);
12.            if (d > 1) return d;
13.            t = f(t);
14.            r = f(f(r));
15.        }
16.    }
```

```
17. }
```

6. 倍增优化算法

频繁的使用欧几里得算法会使得最后的算法复杂度带一个 log，可以通过乘法积累来减少使用欧几里得算法的频率。

若 $\gcd(a,n)>1$，则有 $\gcd(ac,n)>1$；并且有 $\gcd(ac \bmod n,n)>1$。

可以每过一段时间对差值的乘积进行欧几里得运算。例如，可以每隔 2^k 个数求一次公因数，这样只需要求期望 $\log\left(n^{\frac{1}{4}}\right)$ 次公因数。但这样在常数上有个缺点，就是到后面的间隔很大，可能已达成目标却迟迟无法退出。所以我们也可以每隔固定的距离 C 就求一次公因数，而且结合起来且取 $C=2^7-1$ 是目前比较好的写法。

```
1. ll f(ll x, ll c) {
2.     return (x * x % n + c) % n
3. }
4.
5. ll Pollard_rho(ll n) {
6.     if (n == 4) return 2;  //特判 4
7.     if (MillerRabin(n)) return n;  //特判质数
8.     ll s = 0, t = 0, c = rand() % (n - 1) + 1, v = 1, ed = 1;
9.     while (1) {
10.        for (int i = 1; i <= ed; i++) {
11.            t = f(t);
12.            v = v * abs(t - s) % n;
13.            if (i % 127 == 0) {
14.                ll d = gcd(v, n);
15.                if (d > 1) return d;
16.            }
17.        }
18.        ll d = gcd(v, n);
19.        if (d > 1) return d;
20.        s = t, v = 1, ed <<= 1;
21.    }
22. }
```

Multiply（计蒜客 42544）

给出一个大小为 n 的序列 a，定义 $Z=\prod_{i=1}^{n} a_i!$；给出 X,Y，若 $b=Z*X^i$，求出最大的 i 使得 b 是 $Y!$ 的因数。

【输入格式】

第一行输入一个正整数 $T(1\le T\le 8)$，代表有 T 个测试用例。

对于每个测试用例，包含两行。

第一行包含三个整数$N,X,Y(1\le N\le 10^5,2\le X,Y\le 10^{18})$，用空格隔开。

第二行包含 N 个正整数 $a_1,a_2,\ldots,a_N(a_i\le 10^{18},a_1+a_2+\cdots+a_N<Y)$。

【输出格式】

对于每个测试用例输出一行一个整数代表结果。

【分析】

将X质因数分解得 $p_1, p_2, p_3 \cdots p_n$。对于某质因子p_i，设Z中有a个，X中有b个，$Y!$中有c个。则该质因子p_i最多对应$(c-a)/b$个，再求出所有质因子中最小的那个i，即为答案

$n!$的质因数分解在数论基础已经介绍，此处可以直接用。因为数据范围较大，那么使用 Pollard-Rho 质因数分解。

```
#include <iostream>
#include <map>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 10;

ll x, y, a[maxn];

struct PollardRho {
  const static int maxm = 1e6 + 16;
  ll prime[maxm], p[maxm], fac[maxm], sz, cnt;
  inline ll mul(ll a, ll b, ll mod) {
      if (mod <= 1000000000) return a * b % mod;
      return (a * b - (ll) ((long double) a / mod * b + 1e-8) * mod + mod) % mod;
  }
  void init(int maxn) {
      int tot = 0;
      sz = maxn - 1;
      for (int i = 1; i <= sz; ++i) p[i] = i;
      for (int i = 2; i <= sz; ++i) {
          if (p[i] == i) prime[tot++] = i;
          for (int j = 0; j < tot && 1ll * i * prime[j] <= sz; ++j) {
              p[i * prime[j]] = prime[j];
              if (i % prime[j] == 0) break;
          }
      }
  }
  ll powl(ll a, ll x, ll mod) {
      ll res = 1LL;
      while (x) {
          if (x & 1) res = mul(res, a, mod);
          a = mul(a, a, mod);
          x >>= 1;
      }
      return res;
```

```
36.  }
37.  bool check(ll a, ll n) {                          //二次探测原理检验 n
38.      ll t = 0, u = n - 1;
39.      while (!(u & 1)) t++, u >>= 1;
40.      ll x = powl(a, u, n), xx = 0;
41.      while (t--) {
42.          xx = mul(x, x, n);
43.          if (xx == 1 && x != 1 && x != n - 1) return false;
44.          x = xx;
45.      }
46.      return xx == 1;
47.  }
48.  bool miller(ll n, int k) {
49.      if (n == 2) return true;
50.      if (n < 2 || !(n & 1)) return false;
51.      if (n <= sz) return p[n] == n;
52.      for (int i = 0; i <= k; ++i) {                //测试 k 次
53.          if (!check(rand() % (n - 1) + 1, n)) return false;
54.      }
55.      return true;
56.  }
57.  inline ll gcd(ll a, ll b) {
58.      return b == 0 ? a : gcd(b, a % b);
59.  }
60.  inline ll Abs(ll x) {
61.      return x < 0 ? -x : x;
62.  }
63.  ll Pollard_rho(ll n) {                    //基于路径倍增的 Pollard_Rho 算法
64.      ll s = 0, t = 0, c = rand() % (n - 1) + 1, v = 1, ed = 1;
65.      while (1) {
66.          for (int i = 1; i <= ed; ++i) {
67.              t = (mul(t, t, n) + c) % n;
68.              v = mul(v, Abs(t - s), n);
69.              if (i % 127 == 0) {
70.                  ll d = gcd(v, n);
71.                  if (d > 1) return d;
72.              }
73.          }
74.          ll d = gcd(v, n);
75.          if (d > 1) return d;
76.          s = t;
77.          v = 1;
78.          ed <<= 1;
79.      }
```

```
80.   }
81.   void getfactor(ll n) {                          //得到所有的质因子(可能有重复的)
82.       if (n <= sz) {
83.           while (n != 1) fac[cnt++] = p[n], n /= p[n];
84.           return;
85.       }
86.       if (miller(n, 6)) fac[cnt++] = n;
87.       else {
88.           ll d = n;
89.           while (d >= n) d = Pollard_rho(n);
90.           getfactor(d);
91.           getfactor(n / d);
92.       }
93.   }
94.   ll cal(ll n, ll x) {
95.       ll num = 0;
96.       while (n) {
97.           num += n / x;
98.           n = n / x;
99.       }
100.      return num;
101.  }
102.  ll solve(int n, ll x, ll y) {
103.      map<ll, ll> mp;
104.      ll ans = 4e18;
105.      cnt = 0;
106.      getfactor(x);
107.      for (int i = 0; i < cnt; ++i) mp[fac[i]]++;
108.      map<ll, ll>::iterator it = mp.begin();
109.      while (it != mp.end()) {
110.          ll num = 0;
111.          for (int i = 1; i <= n; ++i) {
112.              num += cal(a[i], it->first);
113.          }
114.          ans = min(ans, (cal(y, it->first) - num) / it->second);
115.          it++;
116.      }
117.      return ans;
118.  }
119. } Q;
120.
121. int main() {
122.     Q.init(100000);
123.     int T, n;
```

```
124.     scanf("%d", &T);
125.     while (T--) {
126.         scanf("%d %lld %lld", &n, &x, &y);
127.         for (int i = 1; i <= n; ++i)
128.             scanf("%lld", a + i);
129.         printf("%lld\n", Q.solve(n, x, y));
130.     }
131.     return 0;
132. }
```

8.1.3 中国剩余定理及扩展

1. 问题引入

现在有一个数 x，它除以 3 余 2，除以 5 余 3，除以 7 余 2，求 x

2. 前置知识：

1) 关于取模的两个重要性质：
   - 若 $a\%b = c$，那么有 $(a + kb)\%b = c$
   - 若 $a\%b = c$，那么 $(a * k)\%b = k * c$
2) 逆元的定义：

   若 $ax \equiv 1(mod\ \ p)$，且 a 与 p 互质，那么我们就能定义 x 为 a 模 p 意义下的逆元，记为 a^{-1} 或 $inv(a)$

3. 推导思路：

设 n_1 满足 $n_1\%3 = 2$，n_2 满足 $n_2\%5 = 3$，n_3 满足 $n_3\%7 = 2$，问题转化为如何使得 $n_1 + n_2$ 仍满足模 3 余 2，进而 $n_1 + n_2 + n_3$ 仍满足模 3 余 2（模另外两个数时同理）可得出以下推导：

1) 若 $n_1 + n_2 + n_3$ 的和满足模 3 余 2，那么 $n_2 + n_3$ 必须是 3 的倍数
2) 若 $n_1 + n_2 + n_3$ 的和满足模 5 余 3，那么 $n_1 + n_3$ 必须是 5 的倍数
3) 若 $n_1 + n_2 + n_3$ 的和满足模 7 余 2，那么 $n_1 + n_2$ 必须是 7 的倍数

进一步得到：

1) n_1 模 3 余 2 且是 5,7 的公倍数
2) n_2 模 5 余 3 且是 3,7 的公倍数
3) n_3 模 7 余 2 且是 3,5 的公倍数

4. 具体步骤

从 5,7 的公倍数找到模 3 余 2 的数 n_1，这个 n_1 并不是直接找，而是先找到一个模 3 余 1 的数，根据上述性质二乘以 2 即可满足模 3 余 2。那么实际上就是求这个公倍数模 3 下的逆元，再用逆元乘以余数

同理找 n_2, n_3。最后将所有的结果求和得到了问题的一个解。

如何找最小整数解？只需最大程度减去 3,5,7 的最小公倍数，因为它们两两互质，也就是对 $3*5*7$ 的结果取模即可

5. 中国剩余定理

设正整数 $m_1, m_2, \dots, m_k$ 两两互质，则同余方程组：

$$\begin{cases} x \equiv a_1 (mod\ \ m_1) \\ x \equiv a_2 (mod\ \ m_2) \\ \quad\quad\quad\quad \dots\dots \\ x \equiv a_k (mod\ \ m_k) \end{cases}$$

有整数解，并且在模 $M = m_1 * m_2 * \dots * m_k$ 下的解是唯一的。解为：

$x \equiv (a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} + \dots + a_k M_k M_k^{-1})\ mod\ \ M$。

其中 $M_i = \frac{M}{m_i}, M_i^{-1} = inv(M_i)\ mod\ \ M$。

```
ll a[maxn], m[maxn];

void exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return;
    }
    exgcd(b, a % b, y, x);
    y -= (a / b) * x;
}

ll crt(int n) {
    ll M = 1, ans = 0, res, x, y;
    for (int i = 1; i <= n; i++) M *= m[i];
    for (int i = 1; i <= n; i++) {
        res = M / m[i];
        exgcd(res, m[i], x, y);
        ans = (ans + res * x * a[i] % M) % M;
    }
    if (ans < 0) ans += M;
    return ans;
}
```

6. 扩展中国剩余定理

假设给出的 m_i 不互素，如何求解？假设求解的是如下方程组：

$$\begin{cases} x \equiv a_1 (mod\ \ m_1) \\ x \equiv a_2 (mod\ \ m_2) \end{cases}$$

实际上等价于：

$$\begin{cases} x \equiv a_1 + m_1 x_1 ① \\ x \equiv a_2 + m_2 x_2 ② \end{cases}$$

那么得到 $a_1 + m_1x_1 = a_2 + m_2x_2 \Rightarrow m_1x_1 + m_2(-x_2) = a_2 - a_1$

如果上述二元一次不定方程无解那么整个方程式无解，否则就可以继续下去：

令 $g = gcd(m1, m2)$，那么我们可以通过扩欧求出 $m_1x_1 + m_2(-x_2) = g$ 的一组最小整数解 α, β，那么不难得出上述不定方程的解为 $x_1 = \alpha * \frac{(a_2-a_1)}{g}, x_2 = \beta * \frac{(a_2-a_1)}{g}$，

为了方便我们使用 x_1 的最小正整数解来表示答案，即：$x_1 = \left(\alpha\%\left(\frac{m_2}{g}\right) + \frac{m_2}{g}\right)\%\left(\frac{m_2}{g}\right)$，再代入 ① 方程即可求出方程组 $\begin{cases} x \equiv a_1 (mod\ \ m_1) \\ x \equiv a_2 (mod\ \ m_2) \end{cases}$ 的一个最小整数解

对于该方程组求出到了当前的解 x，但是后面还有很多项方程，怎么继续求下面的方程组？

引入一个**定理**：

若有特解 x'，那么 $\begin{cases} x \equiv a_1 (mod\ \ m_1) \\ x \equiv a_2 (mod\ \ m_2) \end{cases}$ 的通解是：$x' + k * lcm(m1, m2)$，即 $x \equiv x' (mod\ \ lcm(m_1, m_2))$

定理的证明略去，现在已经可以得出总体的解题步骤了：

1) 得到所有的同余方程组
2) 取前两个方程，若无解则直接返回；否则求出最小整数解，按上述定理得到一个新的方程组
3) 重复执行上述步骤二，直到只剩下一个方程组，这个方程就是所有方程组的解系

```
ll a[maxn], m[maxn];

ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll gcd = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return gcd;
}

ll mul(ll x, ll n, ll p) {
    ll ans = 0;
    while (n) {
        if (n & 1) ans = (ans + x) % p;
        x = (x + x) % p;
        n >>= 1;
    }
    return ans;
}
```

```
22.
23. ll excrt(int n) {
24.     ll a1 = a[1], m1 = m[1], a2, m2, x, y;
25.     for (int i = 2; i <= n; i++) {
26.         a2 = a[i], m2 = m[i];
27.         ll g = exgcd(m1, m2, x, y);
28.         if((a2 - a1) % g != 0) return -1;
29.         x = x * (a2 - a1) / g, m2 /= g;
30.         x = (x % m2 + m2) % m2;
31.         a1 = a1 + m1 * x, m1 = m1 * m2;
32.     }
33.     return a1;
34. }
```

扩展中国剩余定理（洛谷 P4777）

给定 n 组非负整数 a_i, b_i，求解关于 x 的方程组的最小非负整数解。

$$\begin{cases} x \equiv b_1 (mod\ a_1) \\ x \equiv b_2 (mod\ a_2) \\ \quad\quad ... \\ x_n \equiv b_n (mod\ a_n) \end{cases}$$

【输入格式】

输入第一行包含整数 n。

接下来 n 行，每行两个非负整数 a_i, b_i。

【输出格式】

输出一行，为满足条件的最小非负整数 x。

【分析】

扩展欧几里得的模板题，注意中间计算会溢出，使用 __int128 即可。

```
1. #include <iostream>
2.
3. using namespace std;
4. const int Mod = 1e9 + 7;
5. const int maxn = 2e5 + 10;
6.
7. typedef __int128 ll;
8.
9. ll a[maxn], m[maxn];
10.
11. inline __int128 read() {
12.     __int128 x = 0, f = 1;
13.     char ch = getchar();
14.     while (ch < '0' || ch > '9') {
15.         if (ch == '-') f = -1;
16.         ch = getchar();
```

```
17.     }
18.     while (ch >= '0' && ch <= '9') {
19.         x = x * 10 + ch - '0';
20.         ch = getchar();
21.     }
22.     return x * f;
23. }
24.
25. inline void write(__int128 x) {
26.     if (x < 0) putchar('-'), x = -x;
27.     if (x > 9) write(x / 10);
28.     putchar(x % 10 + '0');
29. }
30.
31. ll exgcd(ll a, ll b, ll &x, ll &y) {
32.     if (!b) {
33.         x = 1, y = 0;
34.         return a;
35.     }
36.     ll gcd = exgcd(b, a % b, y, x);
37.     y -= (a / b) * x;
38.     return gcd;
39. }
40.
41. ll mul(ll x, ll n, ll p) {
42.     ll ans = 0;
43.     while (n) {
44.         if (n & 1) ans = (ans + x) % p;
45.         x = (x + x) % p;
46.         n >>= 1;
47.     }
48.     return ans;
49. }
50.
51. ll excrt(int n) {
52.     ll a1 = a[1], m1 = m[1], a2, m2, x, y;
53.     for (int i = 2; i <= n; i++) {
54.         a2 = a[i], m2 = m[i];
55.         ll g = exgcd(m1, m2, x, y);
56.         x = x * (a2 - a1) / g, m2 /= g;
57.         x = (x % m2 + m2) % m2;
58.         a1 = a1 + m1 * x, m1 = m1 * m2;
59.     }
60.     return a1;
```

```
61. }
62.
63. int main() {
64.     int n;
65.     cin >> n;
66.     for (int i = 1; i <= n; i++) {
67.         m[i] = read();
68.         a[i] = read();
69.     }
70.     write(excrt(n));
71.     return 0;
72. }
```

8.1.4 Lucas 定理及扩展

在求解组合数$\binom{n}{m} \bmod p$时，当m,n比较大甚至达到了$long\ \ long$数据范围，但是模数是一个不大的数时，需要使用卢卡斯定理。

1. 取模数 p 为质数

A,B是非负整数，p是质数。A,B写成p进制：$A=a[n]a[n-1]\dots a[0]$，$B=b[n]b[n-1]\dots b[0]$。

定理：组合数C_A^B与$\left(C_{a[n]}^{b[n]} * C_{a[n-1]}^{b[n-1]} \dots * C_{a[0]}^{b[0]}\right)\ mod\ \ p$同余

即：$Lucas(n,m,p)=C(n\%p,m\%p)*Lucas\left(\frac{n}{p},\frac{m}{p},p\right)$

求解$C(n\%p,m\%p)$时采用组合数公式暴力计算即可，时间复杂度$O(log_p n * p)$

```
1. ll qkp(ll x, ll n, ll p) {
2.     ll ans = 1;
3.     x %= p;
4.     while (n) {
5.         if (n & 1) ans = ans * x % p;
6.         x = x * x % p;
7.         n >>= 1;
8.     }
9.     return ans;
10. }
11.
12. ll inv(ll x, ll p) {  //求逆元
13.     return qkp(x, p - 2, p);
14. }
15.
16. ll cal(ll n, ll m, ll p) {
17.     if (m > n) return 0;
```

```
    ll u = 1, d = 1;
    for (int i = n - m + 1; i <= n; i++) u = u * i % p;
    for (int i = 1; i <= m; i++) d = d * i % p;
    return u * inv(d, p) % p;
}

ll lucas(ll n, ll m, ll p) {
    if (!m) return 1;
    return cal(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
}
```

2. 取模数 p 为合数

求$C_n^m \ mod \ p$，其中p为合数。

1) **原理**

将p质因数分解为$p = p_1^{a_1} * p_2^{a_2} \dots p_k^{a_k}$，问题就变成了求同余方程组：

$$\begin{cases} x \equiv C_n^m (mod \ p_1^{a_1}) \\ x \equiv C_n^m (mod \ p_2^{a_2}) \\ \dots\dots \\ x \equiv C_n^m (mod \ p_n^{a_n}) \end{cases}$$

因为$p_1, p_2, \dots p_n$互质，最后使用中国剩余定理合并答案，此时问题变成了求$C_n^m \ mod \ p^k$。

2) **思路**

根据组合数的定义，上述问题转化为：

$$\frac{n!}{m!\,(n-m)!} \ mod \ p^k$$

因为求逆元的充要条件为$gcd(a, p) = 1$，因此不能对$m!, (n-m)!$求逆元，考虑不能求逆元时就是$x!$中含有质因数p，考虑除掉这个质因子，则问题变为：

$$\frac{\frac{n!}{p^x}}{\frac{m!}{p^y}\frac{(n-m)!}{p^z}} p^{x-y-z} \ mod \ p^k$$

其中p^x代表$n!$中质因子p的个数。

设 $f(n) = \frac{n!}{p^x}$，那么问题又转化为：

$$\frac{f(n)}{f(m)f(n-m)} p^{x-y-z} \ mod \ p^k$$

对$n!$进行变形，将其中p的倍数和非倍数分开看，可以得到 $n! = (p * 2p * 3p \dots tp)(1 * \dots)$，又因为$1-n$中有$\left\lfloor \frac{n!}{p} \right\rfloor$个$p$的倍数，那么又能得到$p^{\left\lfloor \frac{n!}{p} \right\rfloor}(1 * 2 \dots * t)(1 * \dots)$，即：

$$p^{\left\lfloor \frac{n!}{p} \right\rfloor} \left(\left\lfloor \frac{n!}{p} \right\rfloor\right)! \left(\prod_{i=1, i\%p!=0}^{n} i \right) \ ①$$

其中$\prod_{i=1,i\%p!=0}^{n} i$这个式子是有循环节的，例如$15!\ mod\ 7$，那么后面的式子为：

$(1*2*3*4*5*6)*(8*9*10*11*12*13)*15$

$=(1*2*3*4*5*6)*(8\%7*9\%7*10\%7*10\%7*11\%7*12\%7*13\%7)*15$

$=(1*2*3*4*5*6)^2*15$

因此①式又可以化为：

$$p^{\left\lfloor\frac{n!}{p}\right\rfloor}\left(\left\lfloor\frac{n!}{p}\right\rfloor\right)!\left(\prod_{i=1,i\%p!=0}^{p^k} i\right)^{\left\lfloor\frac{n}{p^k}\right\rfloor}\left(\prod_{i=\left\lfloor\frac{n}{p^k}\right\rfloor p^k,i\%p!=0}^{n} i\right)$$

因为$p^{\left\lfloor\frac{n!}{p}\right\rfloor}$是要提取出来化为$p^{x-y-z}$，但是$\left(\left\lfloor\frac{n!}{p}\right\rfloor\right)!$还可能包含$p$，那么需要这样定义$f(n)$：

$$f(n)=f\left(\left\lfloor\frac{n}{p}\right\rfloor\right)\left(\prod_{i=1,i\%p!=0}^{p^k} i\right)^{\left\lfloor\frac{n}{p^k}\right\rfloor}\left(\prod_{i=\left\lfloor\frac{n}{p^k}\right\rfloor p^k,i\%p!=0}^{n} i\right)$$

递归边界为 $f(0)=1$。

最后还需要求p^{x-y-z}，设$g(n)=\left\lfloor\frac{n}{p}\right\rfloor+g\left(\left\lfloor\frac{n}{p}\right\rfloor\right)$，递归边界为若 $n<p$，则 $g(n)=0$。

这样分别求出$g(n),g(m),g(n-m)$，则$p^{x-y-z}=p^{g(n)-g(m)-g(n-m)}$

综上，总的公式为：

$$\frac{f(n)}{f(m)f(n-m)}p^{g(n)-g(m)-g(n-m)}\ mod\ p^k$$

单次计算的时间复杂度为 $O(p\log p)$。

```
1. void exgcd(ll a, ll b, ll &x, ll &y) {
2.     if (!b) {
3.         x = 1, y = 0;
4.         return;
5.     }
6.     exgcd(b, a % b, y, x);
7.     y -= a / b * x;
8. }
9.
10. ll inv(ll a, ll p) {
11.     ll x, y;
12.     exgcd(a, p, x, y);
13.     return (x % p + p) % p;
14. }
15.
16. ll mul(ll x, ll n, ll p) {
17.     ll ans = 0;
18.     x %= p, n %= p;
19.     while (n) {
```

```
20.         if (n & 1) ans = (ans + x) % p;
21.         x = (x + x) % p;
22.         n >>= 1;
23.     }
24.     return ans;
25. }
26.
27. ll qkp(ll x, ll n, ll p) {
28.     ll ans = 1;
29.     x %= p;
30.     while (n) {
31.         if (n & 1) ans = ans * x % p;
32.         x = x * x % p;
33.         n >>= 1;
34.     }
35.     return ans;
36. }
37.
38. ll f(ll n, ll p, ll mod) {
39.     if (!n) return 1;
40.     ll tmp = 1, res = 1;
41.     for (ll i = 1; i <= mod; i++)
42.         if (i % p) {
43.             tmp = tmp * i % mod;
44.         }
45.     tmp = qkp(tmp, n / mod, mod);
46.     for (ll i = mod * (n / mod); i <= n; i++)
47.         if (i % p) {
48.             res = res * (i % mod) % mod;
49.         }
50.     return f(n / p, p, mod) * tmp % mod * res % mod;
51. }
52.
53. ll g(ll n, ll p) {
54.     if (n < p) return 0;
55.     return g(n / p, p) + n / p;
56. }
57.
58. ll cal(ll n, ll m, ll p, ll mod) {
59.     ll fn = f(n, p, mod), fm = inv(f(m, p, mod), mod), fnm = inv(f(n - m, p, mod),
mod);
60.     ll res = qkp(p, g(n, p) - g(m, p) - g(n - m, p), mod);
61.     return fn * fm % mod * fnm % mod * res % mod;
62. }
```

```
63.
64. ll A[maxn], B[maxn];
65.
66. ll exlucas(ll n, ll m, ll p) {
67.     ll res = p, cnt = 0;
68.     for (ll i = 2; i * i <= p && res > 1; i++)
69.         if (res % i == 0) {
70.             ll pk = 1;
71.             while (res % i == 0) pk *= i, res /= i;
72.             A[++cnt] = pk, B[cnt] = cal(n, m, i, pk);
73.         }
74.     if (res != 1) A[++cnt] = res, B[cnt] = cal(n, m, res, res);
75.     ll ans = 0;
76.     for (ll i = 1; i <= cnt; i++) {
77.         ll M = p / A[i], M1 = inv(M, A[i]);
78.         ans = (ans + B[i] * M % p * M1 % p) % p;
79.     }
80.     return ans;
81. }
```

Mysterious For（HDU 4373）

给定 m 个嵌套的 for 循环，如下所示：

```
1. for (int i = 0; i < n; i++)
2.         for (int j = i; j < n; j++)
3.             for (int k = 0; k < n; k++)
4.                 ...
```

可以发现有两种不同的循环类型，一种是从 0 开始枚举，另一种是从上一项开始枚举。题目给定 m 个嵌套循环中最多有 k 个从 0 开始的循环（其中第一个循环保证一定从 0 开始）。求总的循环次数对 364875103 取余的结果。

【输入格式】

输入第一行包含正整数 $T(T \leq 50)$，代表测试用例的数量。

对于每个测试用例，包含两行。

第一行是三个正整数 $n, m, k(1 \leq n \leq 10^6, 1 \leq m \leq 10^5, 1 \leq k \leq 15)$。

第二行包含 k 个正整数，代表从 0 开始枚举循环的编号。

【输出格式】

对于每一个测试用例，输出一行，"Case #T: ans"，其中 T 代表从 1 开始的案例编号，ans 是答案对 364875103 取模的结果。

【分析】

首先不难发现每次从第一种循环开始时，若后面是第二种循环那么就一直计算直到下一次循环为第一次循环。那么问题就变成了所有循环分为若干块，每一块都从第一种循环

开始，计算所有块的结果累乘即为答案

对于单纯的第一种循环显然就是直接乘n；对于第一种循环后面跟了若干个第二种循环，需要找规律：

如果只有一层第一种循环：

$$ans = n = C_n^1$$

如果后面跟了一个第二种循环：

$$ans = \sum_{i=1}^{n} = \frac{n(n+1)}{2} = C_{n+1}^2$$

如果后面跟了两个第二种循环：

$$ans = \sum_{i=1}^{n} + \sum_{i=1}^{n-1} + \sum_{i=1}^{n-2} + \ldots + \sum_{i=1}^{1}$$

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

$$= \frac{1+2+\ldots+n}{2} + \frac{1^2+2^2+\ldots+n^2}{2}$$

$$= \frac{n(n+1)}{4} + \frac{n(n+1)(2n-1)}{12}$$

$$= \frac{n(n+1)(n+2)}{6}$$

$$= C_{n+2}^3$$

至此已经大致得出规律了：若一块中有m层，那么答案就是C_{n+m-1}^m

但是这个时候直接写会发现样例是调不对的，为什么？容易被人忽视的点是取模数不是素数，而是一个合数：$364875103 = 97 \times 3761599$

那么考虑中国剩余定理CRT，就是解一个两项的同余方程组，因为只有两个素数，那么不需要套板子，只需要求出一些东西计算。

```
#include <iostream>

using namespace std;
typedef long long ll;
const int Mod1 = 97;
const int Mod2 = 3761599;
const int Mod = Mod1 * Mod2;
const int maxn = 1e5 + 10;

int a[20];
ll fac1[Mod1 + 10], fac2[Mod2 + 10];
ll inv1, inv2;

ll qkp(ll x, ll n, ll p) {
```

```
15.     ll ans = 1;
16.     x %= p;
17.     while (n) {
18.         if (n & 1) ans = ans * x % p;
19.         x = x * x % p;
20.         n >>= 1;
21.     }
22.     return ans;
23. }
24.
25. ll cal(ll n, ll m, ll p, ll *fac) {
26.     if (n < m) return 0;
27.     return fac[n] * qkp(fac[m] * fac[n - m], p - 2, p) % p;
28. }
29.
30. ll lucas(ll n, ll m, ll p, ll *fac) {
31.     if (!m) return 1;
32.     return cal(n % p, m % p, p, fac) * lucas(n / p, m / p, p, fac);
33. }
34.
35. void init() {
36.     fac1[0] = fac2[0] = 1;
37.     for (int i = 1; i < Mod1; i++)
38.         fac1[i] = fac1[i - 1] * i % Mod1;
39.     for (int i = 1; i < Mod2; i++)
40.         fac2[i] = fac2[i - 1] * i % Mod2;
41.     inv1 = Mod2 * qkp(Mod2, Mod1 - 2, Mod1);
42.     inv2 = Mod1 * qkp(Mod1, Mod2 - 2, Mod2);
43. }
44.
45. int main() {
46.     int t, n, m, k, kase = 0;
47.     init();
48.     scanf("%d", &t);
49.     while (t--) {
50.         scanf("%d%d%d", &n, &m, &k);
51.         for (int i = 0, x; i < k; i++)
52.             scanf("%d", &a[i]);
53.         a[k] = m;
54.         ll ans = 1, m1, m2, m3;
55.         for (int i = 1; i <= k; i++) {
56.             int q = a[i] - a[i - 1];
57.             m1 = lucas(n + q - 1, q, Mod1, fac1);
58.             m2 = lucas(n + q - 1, q, Mod2, fac2);
```

```
            m3 = (m1 * inv1 + m2 * inv2) % Mod;
            ans = ans * m3 % Mod;
        }
        printf("Case #%d: %lld\n", ++kase, ans);
    }
    return 0;
}
```

DP?（HDU 3944）

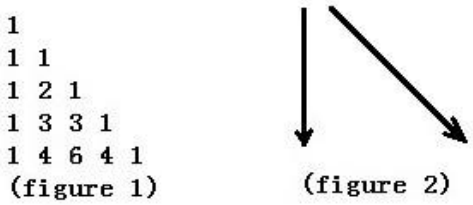


图 1 显示了杨辉三角。我们把行从上到下编号为 0,1,2,…，把列从左到右编号为 0,1,2,… 使用 $C(n,k)$ 代表第 n 行，第 k 列的数量。阳辉三角有如下规律。

$$C(n,0) = C(n,n) = 1\ (n \geq 0)$$

$$C(n,k) = C(n-1,k-1) + C(n-1,k)\ (0 < k < n)$$

编写一个程序，计算从顶部开始到第 n 行第 k 列结束的路线上所通过的数字的最小和，每一步都可以像图 2 那样直接向下或向右斜着走。

由于答案可能非常大，你只需要输出答案对 p 取模的结果，其中 p 是素数。

【输入格式】

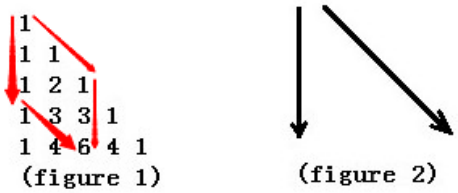
输入将包括一系列多达 100000 个数据集。每个数据有一行包含三个整数$n,k(0 \leq k \leq n < 10^9),p(p < 10^4)$。输入以文件末尾结束。

【输出格式】

对于每一个测试案例，你应该首先输出 "Case #C："，其中 C 表示案例编号，从 1 开始，然后输出最小和对 p 取模的结果。

【分析】

首先不难发现只有两种走法是最划算的：



对于第一种先直走后斜着的走法，显然是如下的一个公式：

$$C(n,k) + C(n-1,k-1) + \ldots + C(n-k+1,1) + C(n-k,0) + n - k$$

根据公式$C(n,k)=C(n-1,k)+C(n-1,k-1)$

我们将最后一项$C(n-k,0)$换成$C(n-k+1,0)$，显然可以和倒数第二项合并得到$C(n-k+2,1)$，那么又可以和倒数第三项合并…，最后得到一个计算公式$C(n+1,k)+n-k$

对于第二种先斜着走后直走的走法，显然是如下的公式：

$$C(n,k)+C(n-1,k)+\ldots+C(k,k)+k$$

同理最后得到的是一个如下的公式：$C(n+1,k+1)+k$

观察不难得知当 $k\le\frac{n}{2}$ 时，选择第一种方案更优；否则选择第二种方案更优

当我们直接计算时，发现实际的复杂度可能最坏为$T*(log_2 n)^2$，因此就超时了。观察可知卢卡斯定理每次计算的组合数的 n,m 都小于 p，那么可以预处理保存 p 的范围内的阶乘对 p 取模的结果，此时的时间复杂度为$O(T*logn)$。

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
typedef long long ll;
const int Mod = 1e9 + 7;
const int maxn = 1e4 + 10;

vector<int> prime;
bool is_prime[maxn];
unordered_map<int, int> mp;
ll fac[maxn][1300];
int num;

void getPrime() {  //欧拉筛求素数
    memset(is_prime, 1, sizeof is_prime);
    is_prime[0] = is_prime[1] = 0;
    for (int i = 2; i < maxn; i++) {
        if (is_prime[i]) prime.push_back(i);
        for (int j = 0; j < prime.size() && 1LL * i * prime[j] < maxn; j++) {
            is_prime[i * prime[j]] = 0;
            if (i % prime[j] == 0) break;
        }
    }
    //cout<<prime.size()<<endl;
    mp.clear(), num = 0;
    for (auto i : prime) mp[i] = ++num;
}

ll qkp(ll x, ll n, ll p) {
```

```
31.     ll ans = 1;
32.     while (n) {
33.         if (n & 1) ans = ans * x % p;
34.         x = x * x % p;
35.         n >>= 1;
36.     }
37.     return ans;
38. }
39.
40. ll inv(ll x, ll p) {  //求逆元
41.     return qkp(x, p - 2, p);
42. }
43.
44. ll cal(ll n, ll m, ll p) {
45.     if (m > n) return 0;
46.     return fac[n][mp[p]] * inv(fac[m][mp[p]], p) % p * inv(fac[n - m][mp[p]], p) %
p;
47. }
48.
49. ll lucas(ll n, ll m, ll p) {
50.     if (!m) return 1;
51.     return cal(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
52. }
53.
54. void init() {
55.     for (auto p : prime) {
56.         int j = mp[p];
57.         fac[0][j] = 1;
58.         for (int i = 1; i < maxn; i++)
59.             fac[i][j] = fac[i - 1][j] * i % p;
60.     }
61. }
62.
63. int main() {
64.     int kase = 0;
65.     ll n, m, p;
66.     getPrime();
67.     init();
68.     while (scanf("%lld%lld%lld", &n, &m, &p) != EOF) {
69.         if (m <= n / 2) printf("Case #%d: %lld\n", ++kase, (lucas(n + 1, m, p) + (n
- m + p) % p) % p);
70.         else printf("Case #%d: %lld\n", ++kase, (lucas(n + 1, m + 1, p) + m) % p);
71.     }
72.     return 0;
```

```
73. }
```

8.1.5 数论函数

8.1.5.1 狄利克雷卷积

1. Dirichlet 卷积

定义两个数论函数 f,g 的狄利克雷卷积 $h=f*g$ 为：

$$h(n)=(f*g)(n)=\sum_{d|n}f(d)g\left(\frac{n}{d}\right)$$

也可以表示为 $h(n)=(f*g)(n)=\sum_{d_1d_2=n}f(d_1)g(d_2)$

2. 性质

交换律：$(f*g)=(g*f)$；

结合律：$(f*g)*h=f*(g*h)$；

分配律：$f*(g+h)=f*g+f*h$；

$f*\varepsilon=f$，其中 ε 为狄利克雷卷积的单位元（任何数卷 ε 都为它本身）；

若 f 积性则 f^{-1} 积性；

若 f,g 均为积性，那么 $h=f*g$ 也为积性。

3. 常见卷积

1) $\varepsilon=\mu*1$

证明：直接按定义展开。

2) $d=1*1$

证明：$d(n)=(1*1)(n)=\sum_{d|n}1$。

3) $\sigma=id*1$

证明：按定义展开即可。

4) $\varphi=\mu*id$

证明：$\varphi(n)=\sum_{d|n}\mu(d)\frac{n}{d}$，在莫比乌斯反演那里有证明。

5) $id_k*1=\sigma_k$

证明：由 $(f*1)(n)=\sum_{d|n}f(n)1\left(\frac{n}{d}\right)=\sum_{d|n}f(n)$，得 $(id_k*1)(n)=\sum_{d|n}i\,d_k(d)=\sum_{d|n}d^k$。

6) $\varphi*1=id$

证明：由 $(\varphi*1)(n)=\sum_{d|n}\varphi(d)$，当 $d=p^x$ 时，$\sum_{d|n}\varphi(d)=\varphi(1)+\sum_{i=1}^{x}(p^i-p^{i-1})=p^x=d$。

对于任意正整数 n，设 $n=\prod p_i^{k_i}$，因为 $\varphi*1$ 是积性函数，那么有 $(\varphi*1)(n)=$

$(\varphi * 1)(\prod p_i^{k_i}) = \prod(\varphi * 1)(p_i^{k_i}) = n$。

8.1.5.2 莫比乌斯函数

1. 莫比乌斯函数

对于 $n > 0$，设数 n 的唯一分解式为 $P_1^{\rho_1} P_2^{\rho_2} \dots P_n^{\rho_n}$

$$\mu(n) = \begin{cases} 1 & n = 1 \\ (-1)^s & \rho_1 = \rho_2 = \dots = \rho_s = 1 \\ 0 & \exists \rho_i \geq 2 \end{cases}$$

通俗地讲，莫比乌斯函数就是：

1) $\mu(1) = 1$
2) 若 n 存在平方因子，$\mu(n) = 0$
3) 若 n 是奇数个不同素数之积，$\mu(n) = -1$
4) 若 n 是偶数个不同素数之积，$\mu(n) = 1$

2. 性质

性质一：莫比乌斯函数是积性函数。若 $gcd(n, m) = 1$，则 $\mu(n * m) = \mu(n) * \mu(m)$

证明：对 n, m 唯一分解然后分类讨论即可

性质二：设 $n \geq 1$，$d|n$（d 是 n 的因子）

$$\sum_{d|n} \mu(d) = \left[\frac{1}{n}\right] = \begin{cases} 1 & n = 1 \\ 0 & n \geq 2 \end{cases}$$

证明：$n = 1$ 时显然满足；设 $n \geq 2$，数 n 的唯一分解式为 $P_1^{\rho_1} P_2^{\rho_2} \dots P_s^{\rho_s}$，$d$ 的唯一分解式为 $P_1^{\alpha_1} P_2^{\alpha_2} \dots P_s^{\alpha_s}$。根据莫比乌斯函数的定义和积性函数的性质可以得出：

$$\sum_{d|n} \mu(d) = \sum_{\alpha_1=1}^{0} \sum_{\alpha_2=1}^{0} \dots \sum_{\alpha_s=1}^{0} \mu(P_1^{\alpha_1} P_2^{\alpha_2} \dots P_s^{\alpha_s})$$

$$\sum_{1}^{0} \mu(P_1^{\alpha_1}) \sum_{1}^{0} \mu(P_2^{\alpha_2}) \dots \sum_{1}^{0} \mu(P_s^{\alpha_s})$$

对于某一项 $\sum_1^0 \mu\left(P_t^{\alpha_t}\right) = 1 + (-1) = 0$，那么所有项乘起来仍为 0，原式得证。

性质三：$\sum_{d|n} \frac{\mu(d)}{d} = \frac{\varphi(n)}{n}$

证明参见莫比乌斯反演关于 $\varphi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$ 的证明。

3. 线性筛选求莫比乌斯函数

若 $i\%prime[j]! = 0$，代表 i 中不含该质因子，也就是说乘上该质因子后恰好会多出一个系数为 1 质因子，因此 $\mu(i * prime[j]) = -\mu(i)$；

否则代表含有该质因子且系数会大于等于 2，那么 $\mu(i * prime[j]) = 0$。

```
bool isprime[maxn];
int mu[maxn], prime[maxn];

void initMu() {
    mu[1] = 1;
    int cnt = 0;
    for (int i = 2; i < maxn; i++) {
        if (!isprime[i]) prime[cnt++] = i, mu[i] = -1;
        for (int j = 0; j < cnt && i * prime[j] < maxn; j++) {
            isprime[i * prime[j]] = 1;
            if (i % prime[j]) mu[i * prime[j]] = -mu[i];
            else {
                mu[i * prime[j]] = 0;
                break;
            }
        }
    }
}
```

8.1.5.3 莫比乌斯反演

1. 莫比乌斯变换

对于 $n \in Z_+$，若有 $f(n) = \sum_{d|n} g(d)$，则称 $f(n)$ 是 $g(n)$ 的莫比乌斯变换。此时 $g(n)$ 称为 $f(n)$ 的逆变换。

2. 莫比乌斯反演

若 $f(n)$ 是 $g(n)$ 的莫比乌斯变换，那么称 $g(n)$ 是 $f(n)$ 的莫比乌斯逆变换，或者称 $g(n)$ 是 $f(n)$ 的莫比乌斯反演，即 $g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right)$

实际上二者互为充要条件，若 $g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right)$，当且仅当 $f(n) = \sum_{d|n} g(d)$

莫比乌斯反演还有另外一种等价描述形式，即：

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

必要性证明：

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

然后根据 d, d' 的关系，固定 d 求出 d' 的范围，可以得出上式的自变量取值集合为 $\{(d, d'), d|n \ \&\& \ d'|\frac{n}{d}\}$，对于 $d'|\frac{n}{d}$ 实际上就是 $dd'|n$，根据求和循环可交换的性质，取值的

集合等价于 $\{(d',d),d'|n \ \&\& \ d|\frac{n}{d'}\}$

那么可以得到：

$$\sum_{d|n}\sum_{d'|\frac{n}{d}}\mu(d)g(d')=\sum_{d'|n}\sum_{d|\frac{n}{d'}}\mu(d)g(d')=\sum_{d'|n}g(d')\sum_{d|\frac{n}{d'}}\mu(d)$$

又根据莫比乌斯函数的性质二：$\sum_{d|n}\mu(d)=\left[\frac{1}{n}\right]$，则 $\sum_{d|\frac{n}{d'}}\mu(d)=\left[\frac{d'}{n}\right]$，进而：

$\sum_{d'|n}g(d')\sum_{d|\frac{n}{d'}}\mu(d)=\sum_{d'|n}g(d')\left[\frac{d'}{n}\right]$，又因为 d' 是 n 的因子，那么对 d' 分类 $d'=n, d'<n$ 讨论之后，该式可以化为 $g(n)$

故原式得证。充分性类似上述必要性证明。

3. 性质

1) 若 $g(n)$ 积性，那么 $f(n)$ 积性；若 $g(n)$ 完全积性，那么 $f(n)$ 完全积性。实际上 $f(n)$ 为积性的充要条件为 $g(n)$ 为积性

当 $n\geq 2$ 时，有该式成立：

$$f(n)=\sum_{\alpha_1=0}^{\beta_1}\sum_{\alpha_2=0}^{\beta_2}\dots\sum_{\alpha_s=0}^{\beta_s}g\left(P_1^{\alpha_1}P_2^{\alpha_2}\dots P_s^{\alpha_s}\right)$$

2) 设 $(m,n)=1$，代表 $gcd(m,n)=1$。那么对于每个 $d|mn$ 成立的充要条件是存在唯一的一对正整数 d_1d_2 满足 $d=d_1d_2$，$d_1^k|m, d_2^k|n$

3) 设 $f(n)$ 为积性函数，p 为 n 分解质因数后可以得到的素数，那么有：

$$\sum_{d|n}\mu(d)f(d)=\prod_{p|n}\left(1-f(p)\right)$$
$$\sum_{d|n}\mu^2(d)f(n)=\prod_{p|n}\left(1+f(p)\right)$$

4) 设 $h(n)=\sum_{d|n}f(d)g\left(\frac{n}{d}\right)$，也写作 $h=f*g$，称 h 为 f 和 g 的狄利克雷卷积，一种重要的卷积形式

4. 常见莫比乌斯变换

1)

$$\sum_{d|n}\mu(d)=\left[\frac{1}{n}\right]=\begin{cases}1 & n=1\\ 0 & n\geq 2\end{cases}$$

该式在莫比乌斯函数章节已经证明。该式等价于 $[gcd(x)=1]$，在题目中经常用到。

2)

$$\varphi(n)=\sum_{d|n}\mu(d)\frac{n}{d}$$

证明：设数 n 的唯一分解式为 $P_1^{\rho_1}P_2^{\rho_2}\dots P_s^{\rho_s}$，$d$ 的唯一分解式为 $P_1^{\alpha_1}P_2^{\alpha_2}\dots P_s^{\alpha_s}$。根据上述的定义和性质可以得出：

$$\sum_{d|n}\mu(d)\frac{n}{d}=n\sum_{\alpha_1=1}^{0}\sum_{\alpha_2=1}^{0}\dots\sum_{\alpha_s=1}^{0}\frac{\mu(P_1^{\alpha_1}P_2^{\alpha_2}\dots P_s^{\alpha_s})}{d}$$

$$=n\sum_{\alpha_1=1}^{0}\sum_{\alpha_2=1}^{0}\dots\sum_{\alpha_s=1}^{0}\frac{\mu(P_1^{\alpha_1}P_2^{\alpha_2}\dots P_s^{\alpha_s})}{P_1^{\alpha_1}P_2^{\alpha_2}\dots P_s^{\alpha_s}}$$

$$=\sum_{1}^{0}\frac{\mu(P_1^{\alpha_1})}{P_1^{\alpha_1}}\sum_{1}^{0}\frac{\mu(P_2^{\alpha_2})}{P_2^{\alpha_2}}\dots\sum_{1}^{0}\frac{\mu(P_s^{\alpha_s})}{P_s^{\alpha_s}}$$

对于某一项 $\sum_1^0\frac{\mu(P_t^{\alpha_t})}{P_t^{\alpha_t}}=1-\frac{1}{P_1}$，那么所有项乘起来为：$n\left(1-\frac{1}{P_1}\right)\left(1-\frac{1}{P_2}\right)\dots\left(1-\frac{1}{P_s}\right)=\varphi(n)$，得证。

实际上就是令上述性质三中 $f(x)=\frac{n}{x}$，得 $\sum_{d|n}\mu(d)\frac{n}{d}=n\prod_{p|n}\left(1-\frac{1}{p}\right)=\varphi(n)$

推论

根据莫比乌斯变换的性质，令恒等函数 $f\left(\frac{n}{d}\right)=\frac{n}{d}$，对于给定的$n$可以得出恒等函数 $f(n)=n$ 为欧拉函数 $\varphi(n)$ 的莫比乌斯变换；或者说 $\varphi(n)$ 是恒等函数 $f(n)=n$ 的逆变换，即 $n=\sum_{d|n}\varphi(d)$。

5. 莫比乌斯反演例题

[HAOI2011]Problem b（洛谷 P2522）

求$\sum_{i=a}^{b}\sum_{j=c}^{d}[gcd(i,j)=k]$。

【输入格式】

第一行一个正整数 $n(1\le n\le 5\times10^4)$。

接下来 n 行每行五个正整数 $a,b,c,d,k(1\le a,b,c,d,k\le 5\times10^4)$。

【输出格式】

共 n 行，每行一个整数表示满足要求数对 (x,y) 的个数。

【分析】

根据容斥定理：

$$\sum_{i=a}^{b}\sum_{j=c}^{d}f(i,j)=\sum_{i=1}^{b}\sum_{j=1}^{d}f(i,j)-\sum_{i=1}^{a-1}\sum_{j=1}^{d}f(i,j)-\sum_{i=1}^{b}\sum_{j=1}^{c-1}f(i,j)+\sum_{i=1}^{a-1}\sum_{j=1}^{c-1}f(i,j)$$

最后的问题变成了求：

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

因为 $gcd(i,j)=k\Rightarrow gcd\left(\lfloor\frac{i}{k}\rfloor,\lfloor\frac{j}{k}\rfloor\right)=1$，那么在上述式子中只需在两个∑处做一个小小的转化：

$$f(n,m)=\sum_{i=1}^{\lfloor\frac{n}{k}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{k}\rfloor}[(i,j)=1]$$

考虑到$\sum_{i=1}^{n}\sum_{j=1}^{m}[(i,j)=1]$的求解，思路如下：

根据$\sum_{d|n}\mu(d)=\left[\frac{1}{n}\right]$，得：

$$\sum_{i}^{n}\sum_{j}^{m}\sum_{d|(i,j)}\mu(d)$$

显然d的范围是$[1,n]$，然后从我们枚举$d\in[1,n],d|(i,j)$得到：

$$f(n,m)=\sum_{d=1}^{n}\mu(d)\sum_{i=1}^{n}[d|i]\sum_{j=1}^{m}[d|j]$$

因为i,j都是d的倍数，实际上问题变成了对于每个d，在$[1,n]$和$[1,m]$出现了多少次，这里可以联系下整除分块，最后得到：

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

联系上面，最后可以得到：

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

```cpp
#include <iostream>
#include <vector>
using namespace std;
typedef long long ll;
const int Mod = 1e9 + 7;
const int maxn = 5e4 + 10;

int mu[maxn], k;
vector<int> prime;
bitset<maxn> vis;

void init() {
    mu[1] = 1;
    for (int i = 2; i < maxn; i++) {
        if (!vis[i]) {
            prime.push_back(i);
            mu[i] = -1;
        }
        for (int j = 0; j < prime.size() && i * prime[j] < maxn; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j]) {
```

```
                mu[i * prime[j]] = -mu[i];
            } else {
                mu[i * prime[j]] = 0;
                break;
            }
        }
        mu[i] += mu[i - 1];
    }
}

ll cal(int n, int m) {
    int up = min(n, m);
    ll ans = 0;
    for (int l = 1, r; l <= up; l = r + 1) {
        r = min(n / (n / l), m / (m / l));
        if (r > up) r = up;
        ans += 1LL * (mu[r] - mu[l - 1]) * (n / (k * l)) * (m / (k * l));
    }
    return ans;
}


int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    int t, a, b, c, d;
    init();
    cin >> t;
    while (t--) {
        cin >> a >> b >> c >> d >> k;
        cout << cal(b, d) - cal(a - 1, d) - cal(b, c - 1) + cal(a - 1, c - 1) <<
"\n";
    }
    return 0;
}
```

[国家集训队]Crash 的数字表格（洛谷 P1829）

求$\sum_{i=1}^{n}\sum_{j=1}^{m}\ \mathrm{lcm}(i,j)$

【输入格式】

输入包含一行两个整数 $n,m(1\le n,m\le 10^7)$。

【输出格式】

输出一个正整数，表示表格中所有数的和对 20101009 取模的结果。

【分析】

常规解法

下面表述默认$n \le m$。

$$f(n,m)=\sum_{i=1}^{n}\sum_{j=1}^{m} lcm(i,j)=\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{ij}{gcd(i,j)}$$

有一个常用的技巧，首先枚举$gcd(i,j)$：

$$f(n,m)=\sum_{d=1}^{n}\sum_{i}^{n}\sum_{j}^{m}\frac{ij}{d}[gcd(i,j)=d]$$

根据

$$[gcd(i,j)=d]\Leftrightarrow\left[gcd\left(\lfloor\frac{i}{d}\rfloor,\lfloor\frac{j}{d}\rfloor\right)=1\right]$$

从而转化为内层的两个求和同除以d，又因为外面乘上了ij，那么需要再多乘上d^2，得：

$$f(n,m)=\sum_{d=1}^{n}\sum_{i}^{\lfloor\frac{n}{d}\rfloor}\sum_{j}^{\lfloor\frac{m}{d}\rfloor} i*j*d*[gcd(i,j)=1]$$

然后对于$[gcd(i,j)=1]$，依然可以利用莫比乌斯变换替换：

$$f(n,m)=\sum_{d=1}^{n}\sum_{i}^{\lfloor\frac{n}{d}\rfloor}\sum_{j}^{\lfloor\frac{m}{d}\rfloor} i*j*d\sum_{k|(i,j)}\mu(k)$$

枚举k，类似上面那样令$i=x*k,j=y*k$：

$$f(n,m)=\sum_{d=1}^{n} d\sum_{k=1}^{\lfloor\frac{n}{d}\rfloor}\mu(k)*k^2\sum_{x=1}^{\lfloor\frac{n}{dk}\rfloor}x\sum_{y=1}^{\lfloor\frac{m}{dk}\rfloor}y$$

两次数论分块时间复杂度$O(n)$，$\sum_d^n d$，$\sum_i^n i$，$\sum_j^m j$都可以使用等差数列通项公式直接求出。

继续优化

上式还可以继续化简优化：

设$T=dk,f(x)=\sum_{i=1}^{x} i$，那么可以得到：

$$f(n,m)=\sum_{d=1}^{n} d\sum_{k=1}^{\lfloor\frac{n}{d}\rfloor}\mu(k)*k^2 f\left(\lfloor\frac{n}{T}\rfloor\right)f\left(\lfloor\frac{m}{T}\rfloor\right)$$

然后仍使用上面枚举$gcd(i,j)$的技巧，枚举T（因为T是d的倍数，然后T可以取遍$[1,n]$，故这样转化）：

$$f(n,m)=\sum_{T=1}^{n} f\left(\lfloor\frac{n}{T}\rfloor\right)f\left(\lfloor\frac{m}{T}\rfloor\right)\sum_{d|T} d*\mu\left(\frac{T}{d}\right)*\left(\frac{T}{d}\right)^2$$

$$=\sum_{T=1}^{n} f\left(\lfloor\frac{n}{T}\rfloor\right)f\left(\lfloor\frac{m}{T}\rfloor\right)\sum_{d|T} d*\mu(d)*T$$

上面的$\sum_{d|T} d*\mu\left(\frac{T}{d}\right)*\left(\frac{T}{d}\right)^2 \Leftrightarrow \sum_{d|T} d*\mu(d)*T$比较巧妙，原理是：

- 当 d 取遍 T 的所有因数时，因为$d|T$，那么$\mu(\frac{T}{d})=\mu(d)$
- 当 d 取遍 T 的所有因数时，那么$d*(\frac{T}{d})\text{^}2=\frac{T^2}{d}=T*\frac{T}{d}=T*d$

设$F(T)=\sum_{d|T} d*\mu(d)$，考虑$a\perp b$：$F(a)=\sum_{d|a} d*\mu(d), F(b)=\sum_{d|b} d*\mu(d)$

在我们枚举d时，它既可以是a的约数也可以是b的约数，根据唯一分解定理不难发现$F(ab)=F(a)*F(b)$，故$F(n)$是一个积性函数，可以线性筛。可以知道$F(1)=1, F(p)=1-p, p\in primes$；考虑$a\%p==0$的情况，那么$a*p$中分解出的因数含$p^{\alpha+\beta}$项的值一定为0，因为其幂次一定大于1，而大于1之后莫比乌斯函数的值即为0，因此可以得出此时$F(a*p)=F(a), a\%==0$，现在就可以欧拉筛预处理出$F(n)$了。

此时再观察公式

$$\sum_{T=1}^{n} f\left(\lfloor\frac{n}{T}\rfloor\right) f\left(\lfloor\frac{m}{T}\rfloor\right) F(T)*T$$

其中$F(T)*T$也可以作为前缀和预处理。

```
#include <iostream>
#include <vector>

using namespace std;
typedef long long ll;
const int Mod = 20101009;
const int maxn = 1e7 + 10;

ll sum[maxn], F[maxn];
bitset<maxn> vis;
vector<int> prime;

void init() {
    sum[1] = F[1] = 1;
    vis.reset(), prime.clear();
    for (int i = 2; i < maxn; i++) {
        if (!vis[i]) {
            prime.push_back(i);
            F[i] = (1 - i + Mod);
        }
        for (int j = 0; j < prime.size() && i * prime[j] < maxn; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j]) {
                F[i * prime[j]] = (F[i] * F[prime[j]]) % Mod;
            } else {
```

```
                F[i * prime[j]] = F[i];
                break;
            }
        }
        sum[i] = (sum[i - 1] + F[i] * i % Mod) % Mod;
    }
}

ll f(ll x) {
    return x * (x + 1) / 2 % Mod;
}

ll solve(ll n, ll m) {
    ll up = min(n, m), ans = 0;
    for (ll l = 1, r; l <= up; l = r + 1) {
        r = min(n / (n / l), m / (m / l));
        if (r > up) r = up;
        ans += (sum[r] - sum[l - 1] + Mod) % Mod * f(n / l) % Mod * f(m / l) % Mod;
        ans %= Mod;
    }
    return ans;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    int n, m;
    init();
    cin >> n >> m;
    cout << solve(n, m) << "\n";
    return 0;
}
```

[SDOI2015]约数个数和（洛谷 P3327）

求$\sum_{i=1}^{n}\sum_{j=1}^{m}d(ij)$，其中$d(x)$代表$x$的约数个数。

【输入格式】

第一行一个整数$T(1 \le T \le 5 \times 10^4)$，代表测试用例的个数。

接下来的T行，每行两个整数$n,m(1 \le n,m \le 5 \times 10^4)$。

【输出格式】

输出共T行，每行一个整数表示答案。

【分析】

一个重要的公式：

$$d(ij)=\sum_{x|i}\sum_{y|j}[(x,y)=1]$$

证明：考虑一个质数p，在a中为p^{α}，在b中为p^{β}，在ij中对因数个数的贡献为$\alpha+\beta+1$。在等式的右边要满足$gcd(x,y)=1$，就有$gcd\left(p^{\alpha_x},p^{\beta_y}\right)=1$，要么$\alpha_x=0,\beta_y\in[0,\beta]$，有$\beta+1$种；要么$\beta_y=0,\alpha_x\in[0,\alpha]$，有$\alpha+1$种。最后减去二者同为零的情况，即$\alpha+\beta+1$，等于左式，得证。

那么原式变为：

$$f(n,m)=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{x|i}\sum_{y|j}[(x,y)=1]$$

根据

$$\sum_{d|n}\mu(d)=\left[\frac{1}{n}\right]\Leftrightarrow[gcd(x,y)=1]$$

得：

$$f(n,m)=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{x|i}\sum_{y|j}\sum_{d|(x,y)}\mu(d)$$

因为i,j会取遍$[1,n],[1,m]$，而x,y分别取遍其所有的因子时就可以取遍$[1,n],[1,m]$的所有数，联系到整除分块不难得知$[1,n],[1,m]$的某个因子x,y分别有$\lfloor\frac{n}{x}\rfloor,\lfloor\frac{m}{y}\rfloor$个。那么可以简化为：

$$f(n,m)=\sum_{x=1}^{n}\sum_{y=1}^{m}\lfloor\frac{n}{x}\rfloor\lfloor\frac{m}{y}\rfloor\sum_{d|(x,y)}\mu(d)$$

设$x=x'*d,y=y'*d$，然后再次令新的$x=x',y=y'$，最后得到：

$$f(n,m)=\sum_{d=1}^{n}\mu(d)\sum_{x=1}^{\lfloor\frac{n}{d}\rfloor}\lfloor\frac{n}{dx}\rfloor\sum_{y=1}^{\lfloor\frac{m}{d}\rfloor}\lfloor\frac{m}{dy}\rfloor$$

令$g(n)=\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor=\sum_{i=1}^{n}d(i)$，右式是可以线性筛预处理的，而$\sum_{d=1}^{n}\mu(d)$可以求前缀和。那么最终的式子为

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

可以通过整除分块来求。

```
1. #include <iostream>
2. #include <vector>
3. using namespace std;
4. typedef long long ll;
5. const int Mod = 1e9 + 7;
```

```
const int maxn = 5e4 + 10;

vector<int> prime;
bitset<maxn> vis;
int mu[maxn], premu[maxn], a[maxn];
ll d[maxn];

void init() {
    mu[1] = d[1] = 1;
    for (int i = 2; i < maxn; i++) {
        if (!vis[i]) {
            prime.push_back(i);
            mu[i] = -1, a[i] = 1, d[i] = 2;
        }
        for (int j = 0; j < prime.size() && i * prime[j] < maxn; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j]) {
                mu[i * prime[j]] = -mu[i];
                d[i * prime[j]] = d[i] * 2, a[i * prime[j]] = 1;
            } else {
                mu[i * prime[j]] = 0;
                d[i * prime[j]] = d[i] / (a[i] + 1) * (a[i] + 2), a[i * prime[j]] =
a[i] + 1;
                break;
            }
        }
    }
    for (int i = 1; i < maxn; i++) {
        premu[i] = premu[i - 1] + mu[i];
        d[i] += d[i - 1];
    }
}


int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    int t, n, m;
    cin >> t;
    init();
    while (t--) {
        cin >> n >> m;
        int up = min(n, m);
        ll ans = 0;
```

```
        for (int l = 1, r; l <= up; l = r + 1) {
            r = min(n / (n / l), m / (m / l));
            if (r > up) r = up;
            ans += (premu[r] - premu[l - 1]) * d[n / l] * d[m / l];
        }
        cout << ans << "\n";
    }
    return 0;
}
```

8.1.6 拉格朗日定理

1. 定义

拉格朗日定理：设 p 为素数，对于模 p 意义下的整系数多项式$f(x) = a_n x^n + a_{n-1}\, x^{n-1} + \cdots + a_0 (p \nmid a_n)$ 的同余方程 $f(x) \equiv 0 \bmod p$ 在模 p 意义下至多有 n 个不同解。

2. 证明

对 n 使用归纳法：

当 $n = 0$ 时，由于 $p \nmid a\_0$，故 $f(x) \equiv 0\ (mod\ p)$ 无解，定理对 $n = 0$ 的多项式 $f(x)$ 都成立。

若命题对于 $\deg f < n$ 的 f 都成立，由反证法，假设存在一个满足题目条件的 f 在模 p 意义下有着至少 $n + 1$ 个不同的解 $x_0, x_1, \ldots, x_n$。

可设 $f(x) - f(x_0) = (x - x_0)g(x)$，则 $g(x)$ 在模 p 意义下是一个至多 $n - 1$ 次的多项式。现在由 $x_0, x_1, \ldots, x_n$ 都是 $f(x) \equiv 0 (\bmod p)$ 的解，知对 $1 \le i \le n$，都有

$$(x_i - x_0)g(x_i) \equiv f(x_i) - f(x_0) \equiv 0 (\bmod p)$$

而 $x_i \not\equiv x_0\ (mod\ p)$，故 $g(x_i) \equiv 0 (mod\ p)$，从而 $g(x) \equiv 0 (mod\ p)$ 有至少 n 个根，与归纳假设矛盾。

所以，命题对 n 次多项式也成立。证毕。

3. 应用

给出一个关于同余方程的引理：对于任意 a，至多有 n 个不同的 x 满足同余方程 $nx \equiv a (mod\ m)$。

证明：反证法。如果存在不同的解 $x_1, x_2, \ldots, x_{n+1}$ 都满足该方程，那么对于方程 $n(x - x_{n+1}) \equiv 0 (mod\ m)$ 也至少有这 $n + 1$ 个解。设 m 与 n 的最大公约数为 d，$m = m_0 d$，那么上述方程可以简化为 $d(x - x_{n+1}) \equiv 0 (mod\ m_0 d)$。所有的解 x 在模 $m * 0$ 意义下都与 $x * n + 1$ 同余，根据中国剩余定理，在模 $m_0 d$ 意义下的 x 至多有 d 个，而 d 是 n 的约数，不可能大于 n，这与假设矛盾，因此原命题成立。

拉格朗日定理可以用在一个抽象代数中的定理中：

在有限可交换群 G 中，以下两个条件等价：

1) G 是循环群；

2) 对于任意一个元素 a，至多有 n 个不同的元素 x 满足条件 $x^n = a$。

证明：先证循环群推 n 个元素条件。如果 G 是循环群，那么每个元素都可以表示成为生成元 g 的幂的形式。

于是将不同的元素 x 记为 g^y，a 记为 g^z，条件变为 $g^{yn} = g^z$。如果群的阶为 m，则条件等价于 $yn \equiv z(mod\ m)$。于是这部分根据引理得证。

再证 n 个元素条件推循环群。如果 G 中对于任意元素 a，至多 n 个 x 使得 $x^n = a$。取 a 为单位元 e，即 $x^n = e$。

根据元素的阶部分的定理，群 G 中必然存在元素 g，阶 d 是所有元素的倍数。对于这个阶 d，所有的元素 x 都满足 $x^d = e$。那么，根据初始条件，这个 d 至少为群 G 的阶 m。

但是显然 d 不能比 m 大，否则会产生重复现象，即存在不同的整数 i 和 j 使得 $g^i = g^j$。因此只能有 $d = m$，元素 g 的幂次恰好把群 G 的所有元素不重不漏地跑了一遍，所以 G 是循环群，g 是生成元。证毕。

因此可以直接得到结论：对于素数 p，模 p 的缩剩余系 $\mathbb{Z}_p^*$ 对于乘法构成的群是循环群。

8.1.7 原根

8.1.7.1 阶

1. 定义

由欧拉定理可知，对 $a \in \mathbb{Z}$，$m \in \mathbb{N}^*$，若 $gcd(a,m) = 1$，则 $a^{\varphi(m)} \equiv 1(mod\ m)$。因此满足同余式 $a^n \equiv 1(mod\ m)$ 的最小正整数 n 存在，这个 n 称作 a 模 m 的阶，记作 $\delta_m(a)$。

2. 性质

性质 1：$a, a^2, \dots, a^{\delta_m(a)}$ 模 m 两两不同余。

证明：考虑反证，假设存在两个数 $i \neq j$，且 $a^i \equiv a^j(mod\ m)$，则有 $a^{|i-j|} \equiv 1(mod\ p)$。但是显然的有：$0 < |i-j| < \delta_m(a)$，这与阶的最小性矛盾，故原命题成立。证毕。

性质 2：若 $a^n \equiv 1(mod\ m)$，则 $\delta_m(a) \mid n$。

证明：对 n 除以 $\delta_m(a)$ 作带余除法，设 $n = \delta_m(a)q + r, 0 \leq r < \delta_m(a)$。若 $r > 0$，则 $a^r \equiv a^r\left(a^{\delta_m(a)}\right)^q \equiv a^n \equiv 1\ (mod\ m)$。这与 $\delta_m(a)$ 的最小性矛盾。故 $r = 0$，即 $\delta_m(a) \mid n$。证毕。

据此还可以推出：若 $a^p \equiv a^q(mod\ m)$，则有 $p \equiv q(mod\ \delta_m(a))$。

8.1.7.2 原根

1. 定义

设 $m \in \mathbb{N}^*$，$a \in \mathbb{Z}$。若 $gcd(a,m)=1$，且 $\delta_m(a)=\varphi(m)$，则称 a 为模 m 的原根。

g 满足 $\mathrm{ord}_n(g)=|Z_n^{\times}|=\varphi(n)$，对于质数 p，也就是说 $g^i\ (mod\ p), 0 \le i < p$ 结果互不相同。

注意：在抽象代数中，原根就是循环群的生成元。这个概念只在模 m 缩剩余系关于乘法形成的群中有“原根”这个名字，在一般的循环群中都称作“生成元”。并非每个模 m 缩剩余系关于乘法形成的群都是循环群，存在原根就表明它同构于循环群，如果不存在原根就表明不同构。

2. 原根判定定理

原根判定定理：设 $m \ge 3, gcd(a,m)=1$，则 a 是模 m 的原根的充要条件是，对于 $\varphi(m)$ 的每个素因数 p，都有 $a^{\frac{\varphi(m)}{p}} \not\equiv 1(mod\ m)$。

证明： 必要性显然，下面用反证法证明充分性。

当对于 $\varphi(m)$ 的每个素因数 p，都有 $a^{\frac{\varphi(m)}{p}} \not\equiv 1(mod\ m)$ 成立时，我们假设存在一个 a，其不是模 m 的原根。因为 a 不是 m 的原根，则存在一个 $t<\varphi(m)$ 使得 $a^t \equiv 1(mod\ m)$。

由裴蜀定理得，一定存在一组 k,x 满足 $kt = x\varphi(m) + gcd\big(t,\varphi(m)\big)$。又由欧拉定理得 $a^{\varphi(m)} \equiv 1(mod\ m)$，故有：

$$1 \equiv a^{kt} \equiv a^{x\varphi(m)+gcd\left(t,\varphi(m)\right)} \equiv a^{gcd\left(t,\varphi(m)\right)}(mod\ m)$$

由于 $gcd\big(t,\varphi(m)\big) \mid \varphi(m)$ 且 $gcd\big(t,\varphi(m)\big) \le t < \varphi(m)$。故存在 $\varphi(m)$ 的素因数 p 使得 $gcd\big(t,\varphi(m)\big) \mid \frac{\varphi(m)}{p}$。则 $a^{\frac{\varphi(m)}{p}} \equiv a^{\left(t,\varphi(m)\right)} \equiv 1(mod\ m)$，与条件矛盾。故假设不成立，原命题成立，证毕

3. 原根个数

若一个数 m 有原根，则它原根的个数为 $\varphi(\varphi(m))$。

证明：若 m 有原根 g，则：

$$\delta_m(g^k) = \frac{\delta_m(g)}{gcd(\delta_m(g),k)} = \frac{\varphi(m)}{gcd(\varphi(m),k)}$$

所以若 $gcd\big(k,\varphi(m)\big)=1$，则有：$\delta_m(g^k)=\varphi(m)$，即 g^k 也是模 m 的原根。

而满足 $gcd(\varphi(m),k)=1$ 且 $1 \le k \le \varphi(m)$ 的 k 有 $\varphi(\varphi(m))$ 个。所以原根就有 $\varphi(\varphi(m))$ 个。证毕

4. 原根存在定理

一个数 m 存在原根当且仅当 $m=2,4,p^{\alpha},2p^{\alpha}$，其中 p 为奇素数，$\alpha \in \mathbb{N}^*$。模 n 有原根的充要条件：$n = 2,4,p^e,2 \times p^e$。

5. 求原根

若 m 有原根，其最小原根是不多于 $m^{0.25}$ 级别的。此处略去证明。这保证了我们暴力找一个数的最小原根，复杂度是可以接受的。

若 m 为素数。

2017 Revenge（牛客 NC52855）

给出 n个数且每个数的范围是 [1,2016]，从中选出若干个数相乘使得二者的乘积对 2017 取模为 r，求方案数的奇偶性。

【输入格式】

输入包含多个测试用例，以文件结尾标志为输入结束。

对于每个测试用例：

第一行包括两个正整数 $n,r(1 \le n \le 2 \times 10^6, 1 \le r \le 2016)$。

第二行包含 n 个正整数 $a_1, a_2, \ldots, a_n (1 \le a_i \le 2016)$。

题目保证 $\sum n \le 2 \times 10^6$。

【输出格式】

对于每个测试用例，输出一行一个 0/1 整数，分别代表答案为偶数/奇数。

【分析】

根据 DP 的思路，设 $d[0/1][j]$ 为考虑第 i 个数且取模后结果为 $j(0 \le j \le 2016)$ 得到的方案数，显然这样的时间复杂度为 $O(2017n)$，一定超时。

因为模数 2017 是素数，那么素数一定能找到原根，那么我们利用原根可以将乘法转化为加法，然后方案数无需累加了，而是使用异或运算代替。但是时间复杂度并没有变化。

这时引入 bitset 优化 DP，两个数相加对 2017 取模，那么开一个大小恰好为 2016 的 bitset，考虑没有溢出和溢出两种情况分别异或。

时间复杂度 $O(n)$。

```
#include <iostream>
#include <bitset>

using namespace std;
typedef long long ll;
const int Mod = 1e9 + 7;
const int maxn = 2e6 + 10;

int n, r;
int a[maxn], p[maxn];
bitset<2016> d;
bool vis[maxn];

int main() {
    ios_base::sync_with_stdio(0);
```

```
    cin.tie(0), cout.tie(0);
    int res = 1;
    for (int i = 0; i < 2016; i++) {
        p[res] = i;
        res = res * 5 % 2017;
    }
    while (cin >> n >> r) {
        d.reset();
        d.set(p[1]);
        for (int i = 1; i <= n; i++) {
            cin >> a[i];
            d ^= (d << p[a[i]]) ^ (d >> (2016 - p[a[i]]));
        }
        cout << d[p[r]] << endl;
    }
    return 0;
}
```

8.1.8 离散对数

8.1.8.1 BSGS 算法

1. BSGS

BSGS(baby-step giant-step)，即大步小步算法。常用于求解离散对数问题。形式化地说，该算法可以在 $O(\sqrt{p})$ 的时间内求解求解 $a^x \equiv b(mod\ \ p), gcd(a,p) = 1$，其中方程的解满足$0 \le x < p$。

算法流程：令 $x = A\lceil\sqrt{p}\rceil - B$，其中 $0 \le A, B < \lceil\sqrt{p}\rceil$。则有 $a^{A\lceil\sqrt{p}\rceil - B} \equiv b(mod\ \ p)$，稍加变换，则有 $a^{A\lceil\sqrt{p}\rceil} \equiv ba^B(mod\ \ p)$。

因为已知 a, b，所以我们能先算出等式右边 ba^B 的所有取值，枚举 B，使用哈希表存起来，然后逐一计算 $a^{A\lceil\sqrt{p}\rceil}$，枚举 A，寻找是否有与之相等的 ba^B，从而得到所有的 x，时间复杂度为 $O(\sqrt{p})$。

```
unordered_map<ll, ll> mp;

ll qkp(ll x, ll n, ll p) {
    ll ans = 1;
    x %= p;
    while (n) {
        if (n & 1) ans = ans * x % p;
        x = x * x % p;
        n >>= 1;
```

```
    }
    return ans;
}

ll bsgs(ll a, ll b, ll p) {
    mp.clear();
    a %= p, b %= p;
    if (b == 1 || p == 1) return 0;
    int m = sqrt(p) + 1;
    ll cur = b;
    for (int i = 0; i < m; i++) {
        mp[cur] = i;
        cur = cur * a % p;
    }
    ll x = qkp(a, m, p), ans = -1, s = 1;
    for (int i = 1; i <= m; i++) {
        s = s * x % p;
        if (mp.count(s)) {
            ans = 1LL * i * m - mp[s];
            break;
        }
    }
    return ans;
}

```

2. 扩展 BSGS

求解 $a^x \equiv b(mod\ \ p)$，其中a,p不一定互质。

思路：想办法使得 a,p 互质。

设 $d_1 = gcd(a,p)$，如果 $d_1 \nmid b$，则原方程无解，否则我们将方程两边同时除以 d_1，得到：

$$\frac{a}{d_1}a^{x-1} \equiv \frac{b}{d_1}\left(mod\ \ \frac{p}{d_1}\right)$$

如果 a 和 $p' = \frac{p}{d_1}$ 仍然不互质就设 $d_2 = gcd(a,p')$，如果 $d_2 \nmid b$，则原方程无解，否则我们将方程两边同时除以 d_2。

重复以上过程直到判定无解或者最后得到了模数和底数互质，假设 k 步之后结束且有解，设 $D = \prod_{i=1}^{k} d_i$，将得到如下方程：

$$\frac{a^k}{D}a^{x-k} \equiv \frac{b}{D}\left(mod\ \ \frac{p}{D}\right)$$

由于 $a \perp \frac{p}{D}$ 且 $\frac{a}{D} \perp \frac{p}{D}$，那么求出 $\frac{a}{D}$ 的逆元然后乘到方程右边，这样就变成了一个普通 $BSGS$ 问题，求出解 x 后加上 k 就是原问题的解。

注意：有可能原问题的解会小于等于k，那么需要$O(k)$验证$a^i \equiv b(mod\ p)$。

```
unordered_map<ll, ll> mp;

ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll gcd = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return gcd;
}

ll inv(ll a, ll p) {
    ll x, y;
    exgcd(a, p, x, y);
    return (x % p + p) % p;
}

ll qkp(ll x, ll n, ll p) {
    ll ans = 1;
    while (n) {
        if (n & 1) ans = ans * x % p;
        x = x * x % p;
        n >>= 1;
    }
    return ans;
}

ll bsgs(ll a, ll b, ll p) {
    mp.clear();
    a %= p, b %= p;
    int m = sqrt(p) + 1;
    ll cur = b;
    for (int i = 0; i < m; i++) {
        mp[cur] = i;
        cur = cur * a % p;
    }
    ll x = qkp(a, m, p), ans = -1, s = 1;
    for (int i = 1; i <= m; i++) {
```

```
        s = s * x % p;
        if (mp[s]) {
            ans = 1LL * i * m - mp[s];
            break;
        }
    }
    return ans;
}

ll exbsgs(ll a, ll b, ll p) {
    a %= p, b %= p;
    if (b == 1 || p == 1) return 0;
    ll d = gcd(a, p), k = 0, t = 1;
    while (d > 1) {
        if (b % d) return -1;
        b /= d, p /= d, ++k, t = t * (a / d) % p;
        if (b == t) return k;
        d = gcd(a, p);
    }
    ll ans = bsgs(a, b * inv(t, p), p);
    return ans >= 0 ? ans + k : ans;
}

```

随机数生成器（洛谷 P3306）

给定递推序列 $x_{i+1} = ax_i + b(mod\ p)$，以及首项 x_1。求出最小的 i 使得 $x_i = t$。

【输入格式】

首先输入一个整数 $T(1 \le T \le 50)$，代表测试用例的数量。

对于每个测试用例输入一行五个正整数 $p, a, b, x_1, t(0 \le a, b, x_1, t \le p, 2 \le p \le 10^9)$。

【输出格式】

对于每个测试用例，输出一行一个整数代表答案。

【分析】

将上述递推公式化为等比数列，使用待定系数法：

$$x_{i+1} + \frac{b}{a-1} = a(x_i + \frac{b}{(a-1)})\ (mod\ p)$$

那么根据等比数列递推公式 $x_n + \frac{b}{a-1} = a^{n-1}(x_1 + \frac{b}{a-1})\ (mod\ p)$

这里只有a^{n-1}是未知的，移项可得：

$$a^{n-1} \equiv (x_n + b * inv(a-1)) * inv(x_1 + b * inv(a-1))\ (mod\ p)$$

使用 BSGS 算法求解即可。此题需要特别注意特判 $a = 0,1$ 两种情况，这个式子没有意义。

```
#include <iostream>
```

```
#include <cmath>
#include <map>
using namespace std;
typedef long long LL;

map<LL, LL> hash;
LL p, a, b, x1, xn;

LL ksm(LL a, LL k) {
    LL ans = 1;
    a %= p;
    for (; k; k >>= 1, a = a * a % p)
        if (k & 1) ans = ans * a % p;
    return ans;
}

LL bsgs(LL a, LL b, LL p) {
    if (a % p == 0) return -1;
    hash.clear();
    LL m = ceil(sqrt(p));
    LL am = ksm(a, m);
    for (int j = 0; j <= m; j++) {
        hash[b] = j;
        b = b * a % p;
    }
    LL mul = 1;
    for (int i = 1; i <= m; i++) {
        mul = mul * am % p;
        if (hash[mul]) return i * m - hash[mul] + 1;
    }
    return -1;
}
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%lld%lld%lld%lld%lld", &p, &a, &b, &x1, &xn);
        if (x1 == xn) {
            printf("1\n");
            continue;
        }
        if (a == 0) {
            if (xn == b) printf("2\n"); else printf("-1\n");
            continue;
```

```
        }
        if (a == 1 && b == 0) {
            printf("-1\n");
            continue;
        }

        if (a == 1) {
            LL nyb = ksm(b, p - 2);
            LL ans = ((((xn - x1) % p + p) % p) * nyb % p) % p;
            printf("%lld\n", ans + 1);
            continue;
        }

        LL inv = b % p * ksm(a - 1, p - 2) % p;
        LL shou = (xn % p + inv) % p;
        LL hou = (x1 % p + inv) % p;
        hou = ksm(hou, p - 2) % p;
        LL ans = bsgs(a, shou * hou % p, p);
        printf("%lld\n", ans);
    }
}
```

8.1.8.2 Pohlig-Hellman

求$a^x \equiv b(mod\ p), p \leq 10^{18}$的最小正整数解。模板实现略，见代码仓库 8-1.cpp。

8.2 线性代数

8.2.1 线性空间

1. 线性空间

线性空间（向量空间）是线性代数的基本概念与重要研究对象。线性空间是由向量集合 V、域 $\mathbb{P}$、加法运算 $+$ 和标量乘法（数乘）组成的模类代数结构。

设 $(V,+)$ 是一个阿贝尔群，$\mathbb{P}$ 是一个域。定义 $\mathbb{P}$ 中的数与 V 中元素的一种代数运算，称为数乘 $\mathbb{P} \times V \to V$，记为 $p \cdot v$ 或 pv，其中 p 在域 $\mathbb{P}$ 中，v 在阿贝尔群 V 中。要求该数乘运算是封闭的，运算结果始终有意义，也在群 V 中。

且满足以下条件：

数乘对向量加法分配律：对于 $\mathbf{u},\mathbf{v} \in V, a \in \mathbb{P}, a(\mathbf{u}+\mathbf{v}) = a\mathbf{u} + a\mathbf{v}$

数乘对标量加法分配律：$\mathbf{u} \in V, a, b \in \mathbb{P}, (a+b)\mathbf{u} = a\mathbf{u} + b\mathbf{u}$

数乘结合律：对于 $\mathbf{u} \in V, a, b \in \mathbb{P}, a(b\mathbf{u}) = (ab)\mathbf{u}$

标量乘法单位元：$1 \in \mathbb{P}$ 是乘法单位元，则对于 $\mathbf{u} \in V, 1\mathbf{u} = \mathbf{u}$

则称代数系统 $(V, +, \cdot, \mathbb{P})$ 是 V 关于 $+, \cdot$ 构成 $\mathbb{P}$ 上的一个线性空间，$\mathbb{P}$ 为线性空间的基域，V 中元素称为向量，$\mathbb{P}$ 中元素称为标量。当域 $\mathbb{P}$ 为实数域时，称为实线性空间。当域 $\mathbb{P}$ 为复数域时，称为复线性空间。

称加法群中的零元为零向量，记作 $\mathbf{0}$。

原阿贝尔群中向量的加减法，与线性空间新定义的数乘，统称为线性运算。

简单来说，线性空间就是存在向量的数乘和向量的加法两种运算，对这两种运算封闭的一系列向量的集合。如下部分定义将采用简单形式的介绍（不严格满足其数学定义）。

2. 生成子集

给定若干个向量集合 $A = \{a_1, a_2, .., a_n\}$，若向量 b 能由集合 A 经过向量加法和数乘表示出来，称 b 能被 A 表出。A 构成的所有向量构成一个线性空间，A 称为这个线性空间的生成子集。

3. 线性相关/无关

对线性空间 $(V, +, \cdot, \mathbb{P})$：

称 $a_1, a_2, \dots, a_n \in V$ 称为 V 的一个向量组。

对于 $k_1, k_2, \dots, k_n \in \mathbb{P}$ 称 $\sum_{i=1}^{n} k_i a_i$ 为向量组 $a_1, a_2, \dots, a_n$ 的一个线性组合。

若向量 $\beta \in V$ 可以表示为向量组 $a_1, a_2, \dots, a_n$ 的一个线性组合，则称 β 能被向量组 $a_1, a_2, \dots, a_n$ 线性表出。

对于 $k_1, k_2, \dots, k_n \in \mathbb{P}$，若向量组 $k_1, k_2, \dots, k_n \in \mathbb{P}$ 满足 $\sum_{i=1}^{n} k_i a_i = \mathbf{0}$，则称向量组 $a_1, a_2, \dots, a_n$ 线性无关，否则称其线性相关。

规定零向量与任意向量线性相关。

4. 基

线性无关的生成子集称为线性空间的基底，简称为基，也就是线性空间的极大生成子集。

5. 矩阵意义下的线性空间

对于一个 $n * m$ 的矩阵，可以将它看做长度为 m 的向量，那么这 n 个行向量能表出的所有行向量构成了一个线性空间。对这个矩阵进行高斯消元得到行最简矩阵（对角矩阵），显然这个行最简矩阵所有非零行线性无关。因为高斯消元本身就是在做向量加法和数乘，因此不改变行向量表出的线性空间，于是行最简矩阵的所有非零行向量就是该线性空间的一个极大无关子集 (基)。

8.2.2 线性基

1. 异或和

给定一个大小为 n 的集合 A，其中所有元素相异或的结果为异或和，即 $a_1\ xor\ a_2\ xor\ a_3\ \dots\ xor\ a_n$。

2. 张成

集合 S 的任何子集的异或和构成的集合称为 S 的张成，记做 $span(S)$。通俗来讲张成就是 S 中任取若干个数相异或得到的所有可能结果构成的集合。

3. 线性相关与线性无关

对于集合 S，若存在一个元素 S_i 使得： S 删去 S_i 后得到 S'，$S_i \in span(S')$ 或者说 S' 的张成仍然包含 S_i。那么称集合 S 线性相关。通俗理解就是 S_i 能通过 S 中不包含 S_i 的一个子集的异或和得到。

若 S 中不存在上述元素 S_i 则称 S 线性无关。

4. 线性基

给定集合 A, B，若满足：

B 中所有元素都能通过 A 的子集的异或和得到，且 A 的任何真子集都不可能表示 B 中的所有元素。

集合 A 是线性无关的。

那么称 A 是 B 的线性基。

5. 实现一

（1）构造

在插入过程中，每次找到 x 的最高位上的 1，然后把它消去，如果最终全部被消去，则表示要插入的元素已经可以由当前线性基中一些元素的异或和表示出，此时不需要插入，这样保证了线性基是线性无关的。 这样当我们处理完所有数时，显然当前的线性基可以表示插入集合中的**所有数**。

可以将插入过程理解为高斯约旦消元的过程，矩阵的初等变换不影响向量之间的线性无关性。维护一个对角矩阵（除了对角线上有元素其他位置均无元素）。设线性基为 $A = \{a_i\}$，集合中二进制的最高位为 $maxlen$，实际上就是对一个 $maxlen * maxlen$ 的方阵进行高斯约旦消元的过程。对于 x 从高到低的每一位 i：

- 若 $a_i \neq 0$，则 $x = x\ xor\ a_i$ 消去这一位。
- 否则：
  - 枚举 $j \in [0, i)$，若 x 的第 j 位为 1，那么消去这一位。之所以枚举 $[0, i)$ 是因为如果前面高位有 1 却仍没有被插入，证明前面的位置线性基元素已经被确定。这两个步骤可以看做将某一行除了主元位置都消为 0。
  - 枚举 $j \in (i, maxlen]$，若 a_i 的第 j 位为 1，则消去。这个步骤可以看做是对主元所在列的上方消去可能存在的 1。

这样构造线性基之后，因为 0 是线性基的空集，因此要引入一个变量判断线性基是否能表示出 0，如果一个数插入失败，代表当前线性基已经可以表示出它了，那么显然这两

个数异或就能得到 0。此外还用一个集合 all 记录线性基集合中有多少个不为 0 的元素，也可以用 $all.size == n$ 判断是否能表示出 0。

(2) **子集最大异或和**

求出线性基后只可能有唯一的一个元素的第 i 位为 1，那么只需要将所有的线性基求异或和即为答案。

(3) **子集最小异或和**

最小的主元就是子集最小异或和，注意特判 0。

(4) **查询元素是否在线性基表示的值域中**

根据线性基的构造规律，如果能插入则不能被表示出来，否则就能被表示出来。实际上也就是拿 x 每一位 1 和线性基中的每一个元素异或，如果最后得到 0，那么就可以被表示出来，否则不能被表示出来。

(5) **查询线性基能表示出的第 k 小的数**

设我们已经求出了线性基，那么显然 $[0, maxlen]$ 只会有若干位置为 1，用集合 all 从小到大保存起来每位不为 1 对应的结果，然后分两种情况讨论：

- 若 0 不会被构造出来，显然若有 cnt 位为 1，那么构成的数的个数为 $2^{cnt}-1$。构造方法是二进制的思路：从最小的位开始，如果当前的第 i 位在 k 中为 1，那么异或上这一位对应的线性基元素。
- 若 0 可以被构造出来，这样构造出的数的个数为 2^{cnt}。我们如果只用 1 的位是无法异或出 0 的，为了方便先将 $k--$，这样相当于从 0 开始是第 0 小，然后仍然安装上面的方法构造，显然能成功构造出 0。

(6) **求x和线性基能异或的最大值**

贪心思想，从最高位开始，如果异或得到的结果会更大那么就异或，否则不异或。

(7) **模板**

```
struct LB {
    ll a[105];
    bool isZero;
    int maxlen;
    vector<ll> all;

    void init(int len) {
        maxlen = len, isZero = 0;
        memset(a, 0, sizeof a);
    }

    void build(int n, ll *b) {
        for (int i = 1; i <= n; i++) {
            if (!insert(b[i])) isZero = 1;
        }
```

```
16.       all.clear();
17.       for (int i = 0; i <= maxlen; i++) {
18.           if (a[i]) all.push_back(a[i]);
19.       }
20.   }
21.
22.   ll queryMax() {
23.       ll ans = 0;
24.       for (int i = 0; i <= maxlen; i++) {
25.           ans ^= a[i];
26.       }
27.       return ans;
28.   }
29.
30.   ll queryMin() {
31.       if (isZero) return 0;
32.       for (int i = 0; i <= maxlen; i++) {
33.           if (a[i]) return a[i];
34.       }
35.   }
36.
37.   bool ask(ll x) {
38.       for (int i = maxlen; i >= 0; i--) {
39.           if (x & (1LL << i)) x ^= a[i];
40.       }
41.       return x == 0;
42.   }
43.
44.   bool insert(ll x) {
45.       for (int i = maxlen; i >= 0; i--) {
46.           if (!(x & (1LL << i))) continue;
47.           if (a[i])
48.               x ^= a[i];
49.           else {
50.               for (int j = 0; j < i; j++) {
51.                   if (x & (1LL << j)) x ^= a[j];
52.               }
53.               for (int j = i + 1; j <= maxlen; j++) {
54.                   if (a[j] & (1LL << i)) a[j] ^= x;
55.               }
56.               a[i] = x;
57.               return 1;
58.           }
59.       }
```

```
60.         return 0;
61.     }
62.
63.     ll queryKth(ll k) {
64.         if (isZero) k--;
65.         int cnt = all.size();
66.         if (k >= (1LL << cnt)) return -1;
67.         ll ans = 0;
68.         for (int i = 0; i < all.size(); i++) {
69.             if (k & (1LL << i)) ans ^= all[i];
70.         }
71.         return ans;
72.     }
73.
74.     ll max_Xor_X(ll x) {
75.         for (int i = maxlen; i >= 0; i--) {
76.             if ((x ^ a[i]) > x) x ^= a[i];
77.         }
78.         return x;
79.     }
80. };
```

装备购买（洛谷 P3265）

题目的意思就是在n个m维的向量$(a_1, a_2, \ldots, a_m)$，找出一组向量集，使得这组向量集能表示出尽可能多的向量，并且这一组基的价值和最小。

【输入格式】

第一行两个数 $n, m(1 \le n, m \le 500$。接下来 n 行，每行 m 个数，其中第 i 行描述装备 i 的各项属性值 $a_i(0 \le a_i \le 1000)$。接下来一行 n 个数，其中 c_i 表示购买第 i 件装备的花费。

【输出格式】

一行两个数，第一个数表示能够购买的最多装备数量，第二个数表示在购买最多数量的装备的情况下的最小花费。

【分析】

把 n 件装备看作 n 个长度为 m 的向量，根据题意，购买的装备对应的向量应该是线性无关的。要买下最多数量的装备，就是求这 n 个向量表出的线性空间的基。

把 $a_{i,j}(1 \le i \le n, 1 \le j \le m)$看作系数矩阵，则每个装备 z_i 都是一个行向量。用高斯消元求出该矩阵的秩，就得到了能买下的装备的最多数量。

本题还要求花最少的钱。我们只需要在高斯消元的过程中，使用贪心策略，对于每个主元 x_i，在前$i-1$列为 0、第i列不为 0 的行向量中，选择价格最低的一个，消去其他行中第 i列的值。

该贪心策略可用反证法证明。假设花费价钱最少的基底是$z[i_1], z[i_2], \ldots, z[i_p]$，其中不包含价格最低的行向量 $z[k]$。因为基底是极大线性无关子集，所以 $z[k]$ 能被 $z[i_1], z[i_2], \ldots, z[i_p]$ 表出，不妨设 $z[k] = b_1 z[i_1] + b_2 z[i_2] + \cdots + b_p z[i_p]$。

移项变换可得 $z[i_p] = (z[k] - b_1 z[i_1] - b_2 z[i_2] - \cdots - b_{p-1} z[i_{p-1}])/b_p$，即 $z[i_p]$ 能被 $z[k]$ 和 $z[i_1], z[i_2], \ldots, z[i_{p-1}]$ 表出。故 z[k],$z[i_1], z[i_2], \ldots, z[i_{p-1}]$ 能与 $z[i_1], z[i_2], \ldots, z[i_p]$ 表出相同的线性空间，是一个总价格更低的基底，与假设矛盾。

```
#include <iostream>
#include <cmath>
using namespace std;
typedef long double ld;
const double eps = 1e-8;
const int maxn = 2e5 + 10;

int n, m, ans, cnt;
ld b[510][510];

struct node {
  int w;
  ld a[510];

  bool operator<(const node &p) const {
      return w < p.w;
  }
} c[510];

//直接消元也是正确的
//void guass() {
//    for (int i = 1; i <= n; i++) {
//        int temp = i;
//        for (int j = i + 1; j <= n; j++)
//            if (fabs(a[j][i]) >= eps && w[j] < w[temp]) {
//                temp = j;
//            }
//        if (fabs(a[temp][i]) < eps) continue;
//        if (temp != i) {
//            for (int j = 1; j <= m; j++) swap(a[i][j], a[temp][j]);
//            swap(w[i], w[temp]);
//        }
//        ans += w[i], cnt++;
//        for (int j = m; j >= i; j--) a[i][j] /= a[i][i];
//        for (int j = i + 1; j <= n; j++) {
//            for (int k = m; k >= i; k--) {
//                a[j][k] -= a[j][i] * a[i][k];
```

```
38. //            }
39. //        }
40. //        for (int j = i + 1; j <= m; j++) {
41. //            for (int k = n; k >= i; k--) {
42. //                a[k][j] -= a[i][j] * a[k][i];
43. //            }
44. //        }
45. //    }
46. //}
47.
48. bool insert(ld *x) {
49.     for (int i = 1; i <= m; i++) {
50.         if (fabs(x[i]) < eps) continue;
51.         if (fabs(b[i][i]) < eps) {
52.             for (int j = i; j <= m; j++) b[i][j] = x[j];
53.             return 1;
54.         }
55.         ld k = x[i] / b[i][i];
56.         for (int j = i; j <= m; j++) x[j] -= k * b[i][j];
57.     }
58.     return 0;
59. }
60.
61.
62. int main() {
63.     scanf("%d%d", &n, &m);
64.     for (int i = 1; i <= n; i++)
65.         for (int j = 1; j <= m; j++)
66.             scanf("%Lf", &c[i].a[j]);
67.     for (int i = 1; i <= n; i++) scanf("%d", &c[i].w);
68. //    guass();
69.     sort(c + 1, c + 1 + n);
70.     for (int i = 1; i <= n; i++) {
71.         if (insert(c[i].a)) {
72.             cnt++;
73.             ans += c[i].w;
74.         }
75.     }
76.     printf("%d %d\n", cnt, ans);
77.     return 0;
78. }
```

albus 就是要第一个出场（洛谷 P4869）

已知一个长度为n的正整数序列A（下标从 1 开始），令$S = \{x \mid 1 \le x \le n\}$, S的幂集

2^S定义为 S 所有子集构成的集合。定义映射$f: 2^S \to Z, f(\emptyset) = 0, f(T) = XOR\{A_t\}, (t \in T)$。

现在 albus 把2^S中每个集合的f值计算出来，从小到大排成一行，记为序列B（下标从 1 开始）。

给定一个数，那么这个数在序列B中第 1 次出现时的下标是多少呢？

【输入格式】

第一行一个数n, 为序列A的长度。接下来一行n个数，为序列A，用空格隔开。最后一个数Q，为给定的数.

【输出格式】

共一行， 一个整数， 为 Q 在序列 B 中第一次出现时的下标模 10086 的值.

【分析】

看到本题不难想到线性基，线性基实际上就是上述序列的不可重复集，但是如何知道一个数会重复多少次呢？

首先对前几个 n 打表，我们发现得到的异或集合中所有的数出现的次数都是相同的，且都是 2 的幂次，这暗示着有规律。

设线性基的大小为 cnt，也就是说这 n 个数里面只有 cnt 个可以插入，那么剩下的 n-cnt 个数都能用线性基中的子集异或表示。也就是说原本的集合的子集异或和个数为2^n，但是得到的线性基的子集却只有2^{cnt}不同的异或和，显然每个不同的答案需要重复2^{n-cnt}次才能凑成2^n。

然后就是求一个一定出现的数在线性基子集所有异或和中的排名，整体的思路就是得到线性基中不为 0 的所有元素，判断这些位上x的每一位是否为 1，如果为 1 就加上$1 << j$，j 实际上需要随着枚举线性基不为 0 的元素不断增大的，这个结合二进制不难理解。

```
#include <iostream>
#include <vector>

using namespace std;
#define ENDL "\n"
typedef long long ll;
typedef pair<int, int> pii;
const int inf = 0x7fffffff;
const int maxn = 1e5 + 10;

struct LB {
    ll a[105];
    bool isZero;
    int maxlen;
    vector<ll> all;

    void init(int len) {
        maxlen = len, isZero = 0;
```

```
19.         memset(a, 0, sizeof a);
20.     }
21.
22.     void build(int n, ll *b) {
23.         for (int i = 1; i <= n; i++) {
24.             if (!insert(b[i])) isZero = 1;
25.         }
26.         all.clear();
27.         for (int i = 0; i <= maxlen; i++) {
28.             if (a[i]) all.push_back(a[i]);
29.         }
30.     }
31.
32.     ll queryMax() {
33.         ll ans = 0;
34.         for (int i = 0; i <= maxlen; i++) {
35.             ans ^= a[i];
36.         }
37.         return ans;
38.     }
39.
40.     ll queryMin() {
41.         if (isZero) return 0;
42.         for (int i = 0; i <= maxlen; i++) {
43.             if (a[i]) return a[i];
44.         }
45.     }
46.
47.     bool ask(ll x) {
48.         for (int i = maxlen; i >= 0; i--) {
49.             if (x & (1LL << i)) x ^= a[i];
50.         }
51.         return x == 0;
52.     }
53.
54.     bool insert(ll x) {
55.         for (int i = maxlen; i >= 0; i--) {
56.             if (!(x & (1LL << i))) continue;
57.             if (a[i])
58.                 x ^= a[i];
59.             else {
60.                 for (int j = 0; j < i; j++) {
61.                     if (x & (1LL << j)) x ^= a[j];
62.                 }
```

```
                for (int j = i + 1; j <= maxlen; j++) {
                    if (a[j] & (1LL << i)) a[j] ^= x;
                }
                a[i] = x;
                return 1;
            }
        }
        return 0;
    }

    ll queryKth(ll k) {
        if (isZero) k--;
        int cnt = all.size();
        if (k >= (1LL << cnt)) return -1;
        ll ans = 0;
        for (int i = 0; i < all.size(); i++) {
            if (k & (1LL << i)) ans ^= all[i];
        }
        return ans;
    }
} q;

ll qkp(ll x, ll n, ll p) {
    ll ans = 1;
    x %= p;
    while (n) {
        if (n & 1) ans = ans * x % p;
        x = x * x % p;
        n >>= 1;
    }
    return ans;
}

ll b[maxn];
int Mod = 10086;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    int n;
    ll x;
    q.init(30);
    cin >> n;
    for (int i = 1; i <= n; i++) {
```

```
107.         cin >> b[i];
108.     }
109.     q.build(n, b);
110.     int cnt = q.all.size();
111.     cin >> x;
112.     ll ans = 0;
113.     for (int i = 0, j = 0; i <= q.maxlen; i++) {
114.         if (q.a[i]) {
115.             if(x & (1 << i)) ans += 1 << j;
116.             j++;
117.         }
118.     }
119.     cout << (ans % Mod * qkp(2, n - cnt, Mod) % Mod + 1) % Mod << endl;
120.     return 0;
121. }
```

8.3 博弈论

1. 博弈论基础定理：

必胜状态为先手必胜的状态，**必败状态**为先手必败的状态。

1) 定理一：没有后继状态的状态就是**必败状态**；

2) 定理二：一个状态是**必胜状态**当且仅当存在至少一个**必败状态**为它的后继状态；

3) 定理三：一个状态是**必败状态**当且仅当它的所有后继状态均为**必胜状态**。

2. SG 函数

1) **mex**

$mex(x)$表示最小不属于集合x的非负整数，例如$mex\{1,2,3\}=0, mex\{0,1,2,4\}=3$。

2) **定理**

上述三个基本定理和SG函数三大定理分别对应：

- 没有后继状态的SG值为 0；
- SG函数不等于0，则该状态为必胜态；
  后继状态至少有一个必败态即SG值等于 0。
- SG函数等于0，则该状态为必败态；
  后继状态一定都是必胜态即SG值不为 0。

3. **SG 定理**

若局面由若干个子游戏$x_1, x_2, \ldots, x_n$构成，那么$SG(X)=SG(x_1)\oplus SG(x_2)\oplus\ldots\oplus SG(x_n)$

4. 例题

Brave Game（HDU 1846）

两个顶尖聪明的人在玩游戏，有n个石子，每人可以随便拿$[1,m]$个石子，不能拿的人为败者，问谁会胜利。

【分析】

当石子个数$n=0$时是必败态。

当石子个数$n\in[1,m]$时，都可以转移到$n=0$的状态，那么此时是必胜态。

当石子个数$n=m+1$，能转移到的状态为必胜态$[1,m]$，那么此时是必败态。

当$n\in[m+2,2m+1]$，都可以转移到必败态$m+1$，那么此时是必胜态。

当$n=2(m+1)$时，能转移到的状态为$[m+2,2m+1]$，那么此时是必败态。

……

综上所述，当且仅当$n=k*(m+1)$先手必败，否则先手必胜。

```
#include <iostream>

using namespace std;

int main() {
    int N, n, m;
    cin >> N;
    while (N--) {
        cin >> n >> m;
        if (n % (m + 1) == 0) {
            cout << "second" << endl;
        } else cout << "first" << endl;
    }
    return 0;
}
```

取石子游戏（HDU 2516）

两个顶尖聪明的人在玩游戏，有n个石子，每人可以随便拿$[1,m]$个石子，不能拿的人为败者，问谁会胜利。

【分析】

先手必败当且仅当石子个数为斐波那契数。

证明：

当$n=2$时显然成立。

若对于$[1,n-1]$上述结论均成立

当$n>2$且为斐波那契数时，不妨设$n=f_{k+1}=f_k+f_{k-1}$

若先手第一步取的石子数不小于f_{k-1}，由于$2f_{k-1}>f_k$，则后手必然可以将剩下的石子取完。

否则的话，可以得知后手必然可以取到前f_{k-1}颗石子中的最后一颗，且最后一步取的石子数不会大于$\frac{2}{3}f_{k-1}$。又因为$\frac{2}{3}f_{k-1}<\frac{1}{2}f_{k-1}$，所以先手必然不可能一步把剩下的$f_k$颗石子

取完，由此可知此时必然先手必败。

当n不是斐波那契数时，由齐肯多夫定理可知任何正整数可以表示为若干个不连续的斐波那契数之和，设为$f_{a_1}, f_{a_2}, \ldots, f_{a_k}$，且$a_1 > a_2 > \cdots > a_k$。

考虑先手第一步取f_{a_k}颗石子，由$2f_{a_k} < f_{a_{k-1}}$，可知后手无法一次性取完$f_{a_{k-1}}$颗石子，所以先手必然可以取掉接下来$f_{a_{k-1}}$颗石子中的最后一颗，如此类推，可得到先手必胜。

```
#include<iostream>
#include<algorithm>
using namespace std;
typedef long long ll;

int main() {
    ll n;
    scanf("%lld", &n);
    while (n) {
        ll x = 1, y = 1;
        while (y < n) x = x + y, swap(x, y);
        if (y == n) puts("Second win");
        else puts("First win");
        scanf("%lld", &n);
    }
    return 0;
}
```

欧几里得的游戏（洛谷 P1290）

两个人玩数字游戏，给定两个正整数n, m，每个人都可以拿较大的数减去较小的数的正整数倍且得到的数不能小于 0，减过之后n, m变成了减之前最小的数和减法得到的差，如此反复，直到某个人得到0，他就取得胜利，问什么情况下先手必胜。

【分析】

设$n \geq m, n = km + r$：

当$r == 0$显然先手必胜。

当$r! = 0$，此时可以考虑k：

若$k == 1$，那么(n, m)下面只有一个状态(m, r)，那么(n, m)取决于(m, r)。

若$k > 1$，(n, m)可以分为很多种状态如$(n - m, r), \ldots, (m + r, m), (m, r)$，注意到其中肯定能到$(m + r, m)$这种状态，而$(m + r, m)$这种状态下只有$(m, r)$一种，那么$(m + r, m), (m, r)$二者的胜负性肯定相反，因此$(n, m)$下同时有必胜态和必败态，根据定理二得出$(n, m)$肯定为必胜态。

```
#include <iostream>

using namespace std;
typedef long long ll;
```

```
5.
6. int cal(int a, int b) {
7.     if (a < b) swap(a, b);
8.     int res = a % b;
9.     if (res == 0) return 1;
10.    if (a / b > 1) return 1;
11.    return cal(b, res) ^ 1;
12. }
13.
14. int main() {
15.    ios_base::sync_with_stdio(0);
16.    cin.tie(0), cout.tie(0);
17.    int t, a, b;
18.    cin >> t;
19.    while (t--) {
20.        cin >> a >> b;
21.        if (cal(a, b)) cout << "Stan wins" << "\n";
22.        else cout << "Ollie wins" << "\n";
23.    }
24.    return 0;
25. }
```

nim 游戏（洛谷 P2197）

有n堆石子，两个人可以从任意一堆石子中拿任意多个石子且不能不拿，最后拿空的人胜利，问什么情况下先手会胜利。

【分析】

尼姆游戏可以抽象成在n个有向图上的游戏，我们将数字抽象为点，则每个有向图上的任意点x都可以到达$[0,x-1]$的所有点。$SG(0)=0,SG(1)=1,SG(2)=2,\ldots,SG(x)=x$，尼姆游戏整个局面的的$SG$值为所有子局面的$SG$值异或之和。

扩展：对于一个有n堆石子$a_1,a_2,\ldots,a_n$的游戏局面，它是必败态当且仅当$a_1 \oplus a_2 \oplus \ldots \oplus a_n = 0$。

```
1. #include <iostream>
2. using namespace std;
3. const int maxn = 1e4 + 10;
4.
5. int a[maxn];
6.
7. int main() {
8.     int T, n;
9.     scanf("%d", &T);
10.    while (T--) {
11.        scanf("%d", &n);
12.        int ans = 0;
```

```
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            ans ^= a[i];
        }
        if (ans) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

高手过招（洛谷 P2575）

在一个阶梯上，每层有若干个石子，每次可以将某一层中若干个石子移动到他下一层阶梯中去，第 0 层为地面，不能操作者失败。

【分析】

对于从偶数阶梯拿走到奇数阶梯的石子，我们可以对称地从那个奇数阶梯中拿走相等的石子到下一个偶数阶梯中去，相当于抵消掉了从偶数阶梯拿石子的操作。因为每个奇数阶梯下都有一个偶数阶梯，因此可以保证我们的操作总是可以进行。

问题转化为了奇数堆的取石子游戏，显然对奇数堆的石子数目求一个异或和即可。

```
#include <iostream>

using namespace std;

bitset<25> vis;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    int n, m;
    int t;
    cin >> t;
    while (t--) {
        cin >> n;
        int ans = 0;
        for (int i = 1, y; i <= n; i++) {
            vis.reset();
            cin >> m;
            while (m--) {
                cin >> y;
                vis[y] = 1;
            }
            int cnt = 0, num = 0;
            for (int j = 20; j >= 0; j--) {
                if (!vis[j]) {
```

```
                    if (cnt) ans ^= num;
                    cnt ^= 1;
                    num = 0;
                } else num++;
            }
        }
        if (ans) cout << "YES" << "\n";
        else cout << "NO" << "\n";
    }
    return 0;
}
```

威佐夫博弈（洛谷 P2252）

有两堆石子，两个顶尖聪明的人在玩游戏，每次每个人可以从任意一堆石子中取任意多的石子或者从两堆石子中取出同样多的石子，不能取的人输，分析谁会获得胜利。

【分析】

假设两堆石子数量分别为 $x,y,x<y$，那么先手必败当且仅当$(y-x)*\frac{\sqrt{5}+1}{2}=x$

```
#include<iostream>
#include <cmath>
using namespace std;

int n, m;

const double lorry = (sqrt(5.0) + 1.0) / 2.0;

int main() {
    cin >> n >> m;
    if(n < m) swap(n, m);
    int a = n - m;
    if(m == int(lorry * (double)a))
        cout << 0 << endl;
    else
        cout << 1 << endl;
}
```

8.4 随机算法

8.4.1 爬山算法

爬山算法是一种局部择优的方法，采用启发式方法，是对深度优先搜索的一种改进，它利用反馈信息帮助生成解的决策。

直白地讲，就是当目前无法直接到达最优解，但是可以判断两个解哪个更优的时候，根据一些反馈信息生成一个新的可能解。

因此，爬山算法每次在当前找到的最优方案 x 附近寻找一个新方案。如果这个新的解 x' 更优，那么转移到 x'，否则不变。

这种算法对于单峰函数显然可行。但是对于多数需要求解的函数，爬山算法很容易进入一个局部最优解。

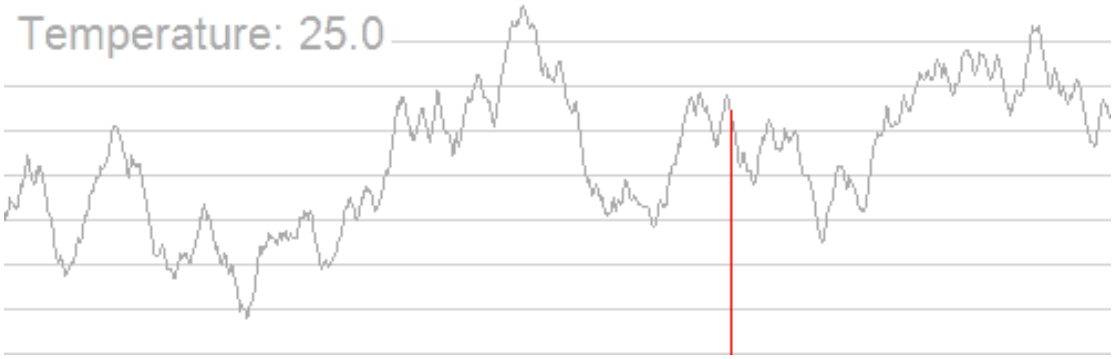
8.4.2 模拟退火算法

模拟退火 (Simulated Annealing[SA]) 的出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。模拟退火算法是一种通用的优化算法，其物理退火过程由加温过程、等温过程、冷却过程这三部分组成。

模拟退火算法是基于 Monte-Carlo 迭代求解策略的一种随机寻优算法，从某一较高初温出发，伴随温度参数的不断下降，结合概率突跳特性在解空间中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。

算法流程：每次随机出一个新解，设当前温度为 t、新解对应的函数值与当前最优解的差为 ΔE。如果这个解更优即 $\Delta E < 0$，则接受它为当前最优解；否则以一定的概率 $e^{\frac{\Delta E}{t}}$ 接受它为当前可能最优解，即随机出的概率小于$e^{\frac{\Delta E}{t}}$。

算法动态执行图如下所示：



算法伪代码如下：

```
1. Status SA() {
2.     t = 初始温度;
3.     delta = 降温参数;
4.     eps = 最终停止的阈值;
5.     ans = 答案, tmp = 可能最优状态;
6.
7.     while (t > eps) {
8.         nxt = 从当前找到的可能最优状态随机找到的一个状态;
9.         del = f(nxt) - f(ans);
```

```
10.         if (del < 0) {
11.             ans = tmp = ans;  //更新最优解和可能最优解
12.         } else if (exp(-del / t) * RAND_MAX > rand()) {
13.             //实际上是 rand()*1.0 / RAND_MAX，这里是减少浮点数计算误差
14.             tmp = nxt;   //以一定概率取得非最优解然后保存尝试
15.         }
16.         t *= delta;
17.     }
18.     return ans;
19. }
20.
```

模拟退火算法如何得到尽可能精确的解：

1. 初始最优解：一般取给出所有初始状态的一个平均值状态，更易得到最优解。
2. 初始温度 T 的设置：大部分题目设置为 100 都可以通过，但是有的范围比较大，一般不超过这个范围的初始温度都可以尝试，自行调整。
3. 温度停止阈值 eps 的设置，一般设置到 10^{-10} 左右即可，如果不行在不超时的情况下可以更小，如10^{-14}。
4. 降温参数 Δ 的设置：一般来说降温参数取 [0.93,0.99] 即可，如果不行可以在不超时的情况下调整区间，如 [0.993,0.999]。
5. 在不超时的前提下多跑几遍模拟退火，即在前面跑出最优解的基础上再去跑模拟退火。
6. 有时函数的峰很多，模拟退火难以跑出最优解。此时可以把整个值域分成几段，每段跑一遍模拟退火，然后再取最优解。

模拟退火算法的时间复杂度和上述的三个影响解的参数 T, eps, Δ 相关，因此得出结论：玄学复杂度。

A Star not a Tree?（POJ 2420）

给出平面内的n个点找出费马点，即找到一个点使得该点到其他所有点的距离之和最小，输出这个最小距离。

【输入格式】

第一行输入包含一个正整数$N \leq 100$，即计算机的数量。接下来是N行；每行给出房间内一台计算机的(x, y)坐标（单位：毫米）。所有坐标都是 0 到 10,000 之间的整数。

【输出格式】

输出包括一个数字，即电缆段的总长度，四舍五入到最近的毫米。

【分析】

本题使用模拟退火算法。

首先任意取一个点(x, y)，然后设置一个最高温，让温度慢慢冷却。对于每个温度，对点(x, y)进行相应幅度的扰动，使其变为点(x', y')。判断新的点是否比原来的点更优。如果

更优则更新，如果更劣则有一定概率更新。

当然本题还有一种解法：

对于费马点问题，当确定x坐标时，距离和函数关于y是一个单峰函数；确定y坐标时，距离和函数关于x是一个单峰函数。

于是我们可以先三分x再三分y，这样确定答案，若浮点误差不好控制，采用三分一百次的写法。

```
#include <cstdio>
#include <cmath>

using namespace std;

const double eps = 1e-14;
const double dinf = 1e300;
const double delta = 0.993;
const double T = 100;

struct Point {
  double x, y;

  Point(double a = 0, double b = 0) : x(a), y(b) {}
} p[105];

int n;
Point s;

int dcmp(double d) {
    if (fabs(d) < eps) return 0;
    return d > 0 ? 1 : -1;
}

double dis(Point a, Point b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

double f(Point cur) {
    double ans = 0;
    for (int i = 0; i < n; i++) {
        ans += dis(cur, p[i]);
    }
    return ans;
}

double SA() {
```

```
38.     double t = T, ans = dinf;
39.     Point tmp = s;
40.     while (t > eps) {
41.         Point nxt(tmp.x + (rand() * 2.0 - RAND_MAX) * t, tmp.y + (rand() * 2.0 -
RAND_MAX) * t);
42.         double res = f(nxt);
43.         double del = res - ans;
44.         if (del < 0) {
45.             ans = res;
46.             s = tmp = nxt;
47.         } else if (exp((ans - res) / t) * RAND_MAX > rand()) {
48.             tmp = nxt;
49.         }
50.         t *= delta;
51.     }
52.     return ans;
53. }
54.
55. void solve() {
56.     scanf("%d", &n);
57.     double sumx = 0, sumy = 0;
58.     for (int i = 0; i < n; i++) {
59.         scanf("%lf%lf", &p[i].x, &p[i].y);
60.         sumx += p[i].x, sumy += p[i].y;
61.     }
62.     s = Point(sumx / n, sumy / n);
63.     printf("%.0lf\n", SA());
64. }
65.
66. int main() {
67.     srand(100000007);
68.     srand(rand()), srand(rand());
69.     solve();
70.     return 0;
71. }
```

Country Meow（Gym 101981D）

给定 N 个三维空间上的点的坐标，求一个半径最小的球，能将所有点覆盖在球内或球上。

【输入格式】

第一行输入包含一个正整数$1 \le N \le 100$。

接下来是N行，每行给出三维坐标系中的一个坐标$(x_i, y_i, z_i)(-10^5 \le x_i, y_i, z_i \le 10^5)$。

【输出格式】

输出包括一个数字，即最小球的半径。

【分析】

本题和寻找费马点很相似，即可以采用模拟退火算法，也可以采用三分套三分，下面代码的模拟退火模板来自红书《ACM 国际大学生程序设计竞赛——算法与实现》（俞勇，清华大学出版社）。

```
#include <cstdio>
#include <math.h>
#include <algorithm>

using namespace std;
const double eps = 1e-14;

struct Tpoint {
  double x, y, z;
};

Tpoint pt[200000], outer[4], res;
double radius;
int n, nouter;

inline double dist(Tpoint p1, Tpoint p2) {
    double dx = p1.x - p2.x, dy = p1.y - p2.y, dz = p1.z - p2.z;
    return (dx * dx + dy * dy + dz * dz);
}

inline double dot(Tpoint p1, Tpoint p2) {
    return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}

void ball() {
    Tpoint q[3];
    double m[3][3], sol[3], L[3], det;
    int i, j;
    res.x = res.y = res.z = radius = 0;
    switch (nouter) {
        case 1 :res = outer[0];
            break;
        case 2 :res.x = (outer[0].x + outer[1].x) / 2;
            res.y = (outer[0].y + outer[1].y) / 2;
            res.z = (outer[0].z + outer[1].z) / 2;
            radius = dist(res, outer[0]);
            break;
        case 3 :
```

```
39.             for (int i = 0; i < 2; i++) {
40.                 q[i].x = outer[i + 1].x - outer[0].x;
41.                 q[i].y = outer[i + 1].y - outer[0].y;
42.                 q[i].z = outer[i + 1].z - outer[0].z;
43.             }
44.             for (int i = 0; i < 2; i++)
45.                 for (int j = 0; j < 2; j++)
46.                     m[i][j] = dot(q[i], q[j]) * 2;
47.             for (int i = 0; i < 2; i++) sol[i] = dot(q[i], q[i]);
48.             if (fabs(det = m[0][0] * m[1][1] - m[0][1] * m[1][0]) < eps)
49.                 return;
50.             L[0] = (sol[0] * m[1][1] - sol[1] * m[0][1]) / det;
51.             L[1] = (sol[1] * m[0][0] - sol[0] * m[1][0]) / det;
52.             res.x = outer[0].x + q[0].x * L[0] + q[1].x * L[1];
53.             res.y = outer[0].y + q[0].y * L[0] + q[1].y * L[1];
54.             res.z = outer[0].z + q[0].z * L[0] + q[1].z * L[1];
55.             radius = dist(res, outer[0]);
56.             break;
57.         case 4 :
58.             for (int i = 0; i < 3; i++) {
59.                 q[i].x = outer[i + 1].x - outer[0].x;
60.                 q[i].y = outer[i + 1].y - outer[0].y;
61.                 q[i].z = outer[i + 1].z - outer[0].z;
62.                 sol[i] = dot(q[i], q[i]);
63.             }
64.             for (int i = 0; i < 3; i++)
65.                 for (int j = 0; j < 3; j++) m[i][j] = dot(q[i], q[j]) * 2;
66.             det = m[0][0] * m[1][1] * m[2][2]
67.                 + m[0][1] * m[1][2] * m[2][0]
68.                 + m[0][2] * m[2][1] * m[1][0]
69.                 - m[0][2] * m[1][1] * m[2][0]
70.                 - m[0][1] * m[1][0] * m[2][2]
71.                 - m[0][0] * m[1][2] * m[2][1];
72.             if (fabs(det) < eps) return;
73.             for (int j = 0; j < 3; j++) {
74.                 for (int i = 0; i < 3; i++) m[i][j] = sol[i];
75.                 L[j] = (m[0][0] * m[1][1] * m[2][2]
76.                     + m[0][1] * m[1][2] * m[2][0]
77.                     + m[0][2] * m[2][1] * m[1][0]
78.                     - m[0][2] * m[1][1] * m[2][0]
79.                     - m[0][1] * m[1][0] * m[2][2]
80.                     - m[0][0] * m[1][2] * m[2][1]
81.                 ) / det;
82.                 for (int i = 0; i < 3; i++) m[i][j] = dot(q[i], q[j]) * 2;
```

```
            }
            res = outer[0];
            for (int i = 0; i < 3; i++) {
                res.x += q[i].x * L[i];
                res.y += q[i].y * L[i];
                res.z += q[i].z * L[i];
            }
            radius = dist(res, outer[0]);
    }
}

void minbool(int n) {
    ball();
    if (nouter < 4)
        for (int i = 0; i < n; i++)
            if (dist(res, pt[i]) - radius > eps) {
                outer[nouter] = pt[i];
                ++nouter;
                minbool(i);
                --nouter;
                if (i > 0) {
                    Tpoint Tt = pt[i];
                    memmove(&pt[1], &pt[0], sizeof(Tpoint) * i);
                    pt[0] = Tt;
                }
            }
}

double smallest_bool() {    //返回最小覆盖球半径
    radius = -1;
    for (int i = 0; i < n; i++) {
        if (dist(res, pt[i]) - radius > eps) {
            nouter = 1;
            outer[0] = pt[i];
            minbool(i);
        }
    }
    return sqrt(radius);
}

void solve() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%lf%lf%lf", &pt[i].x, &pt[i].y, &pt[i].z);
```

```
127.     }
128.     printf("%.8lf\n", smallest_bool());
129. }
130.
131. int main() {
132.     solve();
133.     return 0;
134. }
```

8.4.3 遗传算法

1. 算法起源

遗传算法（Genetic Algorithm，GA）是根据大自然中生物体进化规律而设计提出的。该算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。该算法通过数学的方式,利用计算机仿真运算,将问题的求解过程转换成类似生物进化中的染色体基因的交叉、变异等过程。在求解较为复杂的组合优化问题时,相对一些常规的优化算法,通常能够较快地获得较好的优化结果。

遗传算法的常见应用很多，诸如寻路问题，8 数码问题，囚犯困境，动作控制，找圆心问题，TSP 问题，生产调度问题。遗传算法已被人们广泛地应用于组合优化、机器学习、信号处理、自适应控制和人工生命等领域。

2. 概念介绍

1) 染色体 (Chromosome)：染色体又可称为基因型个体 (individuals)，一定数量的个体组成了群体 (population)，群体中个体的数量叫做群体大小 (population size)。
2) 位串 (Bit String)：其实就是遗传学中的染色体在计算机中的表示。
3) 基因 (Gene)：基因是染色体中的元素，用于表示个体的特征。例如有一个二进制串（即染色体）S=1011，则其中的 1，0，1，1 这 4 个元素分别称为基因。
4) 特征值 (Feature)：在用串表示整数时，基因的特征值与二进制数的权一致；例如在串 S=1011 中，基因位置 3 中的 1，它的基因特征值为 2；基因位置 1 中的 1，它的基因特征值为 8。
5) 适应度 (Fitness)：各个个体对环境的适应程度叫做适应度 (fitness)。为了体现染色体的适应能力，引入了对问题中的每一个染色体都能进行度量的函数，叫适应度函数。这个函数通常会被用来计算个体在群体中的优良等级。
6) 基因型 (Genotype)：或称遗传型，是指基因组定义遗传特征和表现。对应于位串。
7) 表现型 (Phenotype)：生物体的基因型在特定环境下的表现特征。对应于位串解码后的参数。

3. 算法流程

1) 染色体编码，寻找一种对问题潜在解进行“数字化”编码的方案，建立表现型和基因

型的映射关系。

2) 初始化种群。

3) 用适应度函数对每一个个体进行适应度评估。

4) 用选择算子按照某种规定择优选择。

5) 让个体基因变异，保持种群多样性。

6) 然后产生子代。

7) 达到迭代次数或最小误差，算法终止，否则转向步骤 3。

算法流程图如下所示。