ZJUT

MatRush ACM/ICPC Algorithm Reference Document

Zhejiang University of Technology

October, 2011

目录

一：数论和数值算法 6

素数筛法 Prime Sieve 6
质因数分解 6
素数个数 7
最大公约数、最小公倍数 GCD&LCM 8
扩展欧几里德 Extended Euclid 8
模线性同余方程组 8
模的乘法逆元 Modular multiplicative inverse 10
组合数 11
二分快速幂 A^B % C 17
卡特兰数第 N 项 18
Miller Rabin 素性测试+Pollard rho 寻找素因子 19
离散对数 Baby step giant step 20
离散对数 扩展 Baby step giant step 21
质数求原根 23
1..N 中与 N 互质个数 24
高斯消元 25
矩阵 Matrix 27
线性递推 K 倍项和 28
PELL 方程 28
行列式计算 29
FFT 多项式乘法[求卷积] 30
FFT 高精度乘法[万进制] 31
FFT 高精度乘法[十进制] 33
龙贝格积分公式 romberg 35
复合辛普森积分公式 simpson 36
梯形积分法 36
连分数 37
置换群分解 37
N 进制类 38
本原勾股数 39
x^2 + y^2 = r^2 整数解个数 39
Farey 序列构造 40
约瑟夫问题 JOSEPH 42
POLYA 计数 43
循环节 TortoiseHare 算法 44
Sum of power 前 N 个正整数的 K 次幂之和 44
组合函数 47

其他数论常识 50

欧拉函数 50

∑gcd(i,n) 50

与 n 互质的数之和 50

与 n 互质的数的个数 50

约数及素因子个数 51

反素数 51

n!中因子 x 的幂 52

n^n&n!最左位 52

万进制暴力求 N！ 52

n!中 0 的个数 52

a^b 左边 l 位 53

格雷码 53

二进制 53

二：图论算法 **55**

最小生成树 KRUSKAL 55
最小生成树 PRIM 56
次小生成树 56
部分点最小生成树 STEINER TREE 59
最小树形图 60
单源最短路径 SPFA 62
单源最短路径 DIJKSTRA 64
单源最短路径 DIJKSTRA[边表] 64
全源最短路径 FLOYD 65
最小环 65
最短路次短路计数 66
拓扑排序 TOPLOGICAL 67
欧拉路 67
欧拉回路 68
二分图判定 DFS 染色 69
二分图最大匹配 匈牙利 70
二分图最大匹配 HOPCROFT KARP 71
二分图多重匹配 72

带权二分图最佳匹配 KM 73
一般图匹配 76
最大团和最大独立集 78
无向图割边割点 79
有向图弱连通分量 80
有向图强连通分量 KOSARAJU 80
有向图强连通分量 TARJAN 81
有向图最小点基 82
LCA 83
2-SAT 85
最大流 DINIC 88
最大流 SAP 89
最大权闭合子图 90
最小费用最大流 91
无向图全局最小割 STOER-WAGNER 92
有根树的 HASH 93

三：数据结构 96

堆 STL 优先队列 96
分数类 97
多项式类 99
哈希表 100
并查集 101
归并排序 逆序对 101
RMQ 离线 ST 102
二维 RMQ 103
RMQ 点树 线段树 103
树状数组 104
浮点区间并 106
线段树 107
线段树维护区间增量 108
SPLAY 111
虚二叉树 116
划分树 117
左偏树 LEFTISTTREE 119
笛卡尔树 120
动态中位数 121
二维矩形离线求和 122

四：字符串 124

KMP 124
字典树 TRIE 125
扩展 KMP 126
AC 自动机 127
后缀数组 128

字符串 HASH 129
RABIN-KARP[一维] 131
RABIN-KARP[一维可修改] 133
RABIN-KARP[二维] 136
HASH 后缀数组 138
字符串最小最大表示法 138
最长回文子串 138
五：计算几何 139
MATRUSH 几何模板 139
圆面积并 1 160
圆面积并 2 161
圆交 K 次面积 163
圆与多边形面积交 164
凸多边形面积并 166
N 维矩形面积并 169
三维几何 170
三维几何[ZJU] 172
三维凸包 178
三维线段距离[平方分数形式] 181
两球体积交 183
两圆面积交 183
两圆公切线 184
半平面交 184
N 维最近点对 187
多边形费马点 187
面积最大三角形 188
周长最短三角形 189
定长圆覆盖最多点 191
最小圆覆盖[随机增量法] 192
最小球覆盖[随机增量法] 194
最小球覆盖[三分法] 195
点集最小外接矩形 197
扫描线判断某点可见线段数 198
扫描线最近圆对 200
直线切割凸多边形 202
PICK 定理 204
欧拉公式 204
六：动态规划和贪心 204
斜率优化 DP 204
四边形不等式 205
最长上升子序列 LIS 206
最长公共子串 O(NLOGN) 206
多重背包 207

双调路径 208
DigitCount 209
树上最长路径 树形 DP 210
最小区间覆盖 211
七：其他 212
高精度 212
LL 高精度 217
DLX 精确覆盖 218
DLX 重复覆盖 221
表达式计算 223
跳马问题 224
二分查找 227
1x2 矩形完美覆盖 227
矩形切割 228
日期函数 228
带缓冲读入 229
精度问题 230
Java 单关键字排序 230
Java 多关键字排序 231
Java 开根号 232

一：数论和数值算法

素数筛法 PRIME SIEVE

```
#include <iostream>
#include <cmath>
using namespace std;
const int MAXN = 2000001;
int num, prime[MAXN], phi[MAXN];
bool isprime[MAXN + 1];
int sieve(int n) {//求出 1..n 的素数 1..num 素数标记以及欧拉函数
    int p = 0;
    memset(isprime, 1, sizeof(isprime));
    isprime[0] = isprime[1] = 0;
    for (int i = 2; i <= n; i++) {
        if (isprime[i]) {
            prime[p++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; j < p && i * prime[j] <= n; j++) {
            isprime[i * prime[j]] = 0;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            } else {
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            }
        }
    }
    return p;
}

int main() {
    printf("%d\n", num = sieve(MAXN));
}
```

质因数分解

```
//质因数分解 需调用线性筛法
#include <iostream>
#include <cstdio>
#include <cmath>
using namespace std;
const int MAXN = 2000001;
const int MAXP = 150000;
const int MOD = 20100501;
int n, m, num, prime[MAXP], cnt[MAXP], factor[MAXP];

int factorize(int x, int factor[], int cnt[]) {
    int ret = 0;
    for (int i = 0; i < num && prime[i] <= x; i++)
        if (x % prime[i] == 0) {
            factor[ret++] = i;
            while (x % prime[i] == 0) {
                cnt[i]++;
                x /= prime[i];
            }
        }
    return ret;
```

```
}

int main() {
    num = sieve(MAXN);
    while (scanf("%d", &n) != EOF) {
        memset(cnt, 0, sizeof(cnt));
        m = factorize(n, factor, cnt);
        printf("%d = ", n);
        for (int i = 0; i < m; i++) {
            if (i) printf(" * ");
            printf("%d^%d", prime[factor[i]], cnt[factor[i]]);
        }
        printf("\n");
    }
    return 0;
}
```

素数个数

```
/*
2^31 内的素数的个数 需要调用素数筛法
1234567->ans = 95360
2147483647->ans = 105097565
*/
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;
const int MAXP = 50000;
int prime[MAXP], mem[4000][550];
bool isprime[MAXP];
void sieve() {
    memset(isprime, 1, sizeof(isprime));
    int p = 0;
    for (int i = 2; i * i < MAXP; i++)
        if (isprime[i])
            for (int j = i * i; j < MAXP; j += i)
                isprime[j] = 0;
    isprime[0] = isprime[1] = 0;
    for (int i = 2; i < MAXP; i++)
        if (isprime[i])
            prime[p++] = i;
}
int dfs(int n, int lim) {
    if (n < prime[lim])
        return n - 1;
    if (n < 4000 && mem[n][lim] != -1)
        return mem[n][lim];
    int res = 0;
    for (int i = 0; i < lim && prime[i] <= n; i++) {
        unsigned t = (unsigned) n / (unsigned) prime[i];
        res += t;
        if (i > 0 && t >= 2)
            res -= dfs(t, i);
    }
    if (n < 4000)
        mem[n][lim] = res;
    return res;
}
int count(int n) {
    int sqrtn = sqrt((double)n);
    int mlim = upper_bound(prime, prime + 5000, sqrtn) - prime;
    int ans = dfs(n, mlim);
    return n - 1 + mlim - ans;
```

```
}
int main() {
    int n;
    sieve();
    memset(mem, -1, sizeof(mem));
    while (scanf("%d", &n) != EOF)
        printf("ans = %d\n", count(n));
    return 0;
}
```

最大公约数、最小公倍数 GCD&LCM

```
LL gcd(LL a,LL b) {
    return b ? gcd(b,a % b) : a;
}
LL lcm(LL a,LL b) {
    return a / gcd(a,b) * b;
}
```

扩展欧几里德 EXTENDED EUCLID

```
/*
Extended Euclid扩展欧几里得算法
如果gcd(a,b) = d,则存在x,y使ax + by == d(d = gcd(a,b))
如果c % gcd(a,b) != 0 则 ax + by = c没有整数解
exgcd(a,b) = ax + by
方程ax+by=d的第一组解为X0 = x*(d/gcd(a,b)), Y0 = y*(d/gcd(a,b))。
其它的整数解满足:X = X0 + k * (b / gcd(a,b)); Y = Y0 + k * (a / gcd(a, b));
*/
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
int main() {
    LL a, b, x, y;
    while (scanf("%d%d", &a, &b) != EOF)
        printf("%d %d %d\n", exgcd(a, b, x, y), x, y);
}
```

模线性同余方程组

```
/*
中国剩余定理解同余方程组a≡bi(mod mi)返回a的值 m[]互质
线性同余方程ax≡b(mod n)返回最小的x...通解是X=x(mod n/d)
无解返回-1 m[]如果不互质的算法见HDU模板
中国剩余定理
对于每个数n,它可以分解为n=n1*n2*n3*n4*...*nk,且他们之间互质则：
```

```
1,小于 n 的整数 p 都可以唯一地表示成一个 k 元组(a1,a2,a3,...,ak)其中 ai=p%ni;
2,每个 k 元组都对应一个小于 n 的整数
下面给出 2 的证明：
对于 k 元组,我们可以给出 k 个基,第 i 个基是(0,0,...,1,0,0,...0)其中 1 在第 i 个地方
显然地,第 i 个基对应的整数就是( ci=(n/ni)*(模 ni 下 n/ni 的逆元) ).
所以 k 元组(a1,a2,...,ak)对应的整数就是(a1*c1+a2*c2+a3*c3+...ak*ck)mod n
*/
#include <iostream>
using namespace std;
//返回最小的 x...通解是 X=x(mod n/d)
int modularLinerEquation(int a,int b,int n) {
    int x,y,d;
    d = exgcd(a,n,x,y);
    if (b % d != 0)
        return -1; // no solution
    x = b / d * x;
    x = (x % (n/d) + (n/d)) % (n/d);
    return x;
}
//用中国剩余定理解线性同余方程组 m[]要互质
int solModularEquations(int b[],int m[],int z) {
    int M = 1,ret = 0;
    for (int i = 0;i < z;i++)
        M *= m[i];
    for (int i = 0;i < z;i++) {
        int y = modularLinerEquation(M / m[i],1,m[i]);
        ret = (ret + y * b[i] * M / m[i]) % M;
    }
    return ret;
}
int main() {
    int n,b[10],m[10];
    while (scanf("%d",&n) != EOF) {
        for (int i = 0;i < n;i++)
            scanf("%d%d",&b[i],&m[i]);
        printf("%d\n",solModularEquations(b,m,n));
    }
}

//PKU2891 模线性方程组 模板题
//模数可以不是质数
#include <iostream>
#include <cstdio>
using namespace std;
typedef long long LL;
const int MAXN = 10003;
LL aa[MAXN], rr[MAXN];
LL CRT_2(LL a, LL x, LL b, LL y) {
    LL xx, yy, tmp;
    tmp = ex_gcd(a, b, xx, yy);
    LL c = y - x;
    while (c < 0) c += a;
    if (c % tmp != 0) return -1;
    xx *= c / tmp;
    yy *= c / tmp;
    LL t = yy / (a / tmp);
    while (yy - t * (a / tmp) > 0) t++;
    while (yy - (t - 1) * (a / tmp) <= 0) t--;
    return (t * (a / tmp) - yy) * b + y;
}
LL CRT(LL a[], LL r[], int n) {
    LL m = a[0] / gcd(a[0], a[1]) * a[1];
    LL ans = CRT_2(a[0], r[0], a[1], r[1]) % m;
    for (int i = 2; i < n && ans != -1; i++) {
        ans = CRT_2(m, ans, a[i], r[i]);
        m *= a[i] / gcd(m, a[i]);
```

```
        ans %= m;
    }
    return ans;
}
int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; i++) {
            scanf("%d%d",&aa[i],&rr[i]);
        }
        if (n == 1) {
            printf("%lld\n", rr[n]);
        } else {
            printf("%lld\n", CRT(aa, rr, n));
        }
    }
}
```

模的乘法逆元 MODULAR MULTIPLICATIVE INVERSE

```
/*
模的乘法逆元 设m>=1,gcd(a,m) == 1 则必有a*c ≡ 1(mod m) c即a对模m的逆有(c,m) = 1 (a^-1)^-1 ≡ a(mod m)
(a/b) % m = 等于a % (b*m) / b
*/
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
const int MOD = 10007;
typedef long long LL;
LL mul(LL a, LL b, LL c) { // a*b % c
    LL ret = 0, tmp = a % c;
    for (; b; b >>= 1) {
        if (b & 1)
            if ((ret += tmp) >= c)
                ret -= c;
        if ((tmp <<= 1) >= c)
            tmp -= c;
    }
    return ret;
}
LL power(LL a, LL b, LL c) { // a^b % c
    LL res = 1;
    for (; b; b >>= 1) {
        if (b & 1) res = mul(res, a, c);
        a = mul(a, a, c);
    }
    return res;
}
LL exgcd(LL a, LL b, LL &x, LL &y) { //扩展欧几里得定理:解ax+by==1
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
LL mod(LL x, LL n) {
    return (x % n + n) % n;
}
LL inv(LL a, LL n) {
    LL x, y;
    return exgcd(a, n, x, y) == 1 ? mod(x, n) : -1;
```

```
}
//求b % Mod的逆元p p=b^(Mod-2)%Mod 因为b^(Mod-1) % Mod = 1(这里需要Mod为素数)
LL inv2(LL a, LL n) {
    return power(a, n - 2, n);
}

template<class T>
inline T mod(T a, T p) {
    a %= p;
    return (a < 0) ? a + p : a;
}

template<class T>
inline T inv3(T a, T m) {
    a = mod<T>(a, m);
    return (a == 1) ? 1 : mod((1 - m * inv3(m % a, a)) / a, m);
}

const int MAXN = 1111;
int invs[MAXN];
void inverse() {//线性递推逆元
    invs[1] = 1;
    for (int i = 2; i < MAXN; i++) {
        invs[i] = invs[MOD % i] * (MOD - MOD / i) % MOD;
    }
    /*
    a * inv[a] = 1 (% MOD), (MOD % a) * inv[MOD % a] = 1 (% MOD)
    MOD % a = MOD - (MOD / a) * a
    (MOD-(MOD/a)*a)*inv[MOD%a] = 1 (% MOD)
    ((-(MOD/a))*inv[MOD%a])*a = 1 (% MOD)
    inv[a] = (-(MOD/a))*inv[MOD%a]
    */
}
/*LL f[MOD+3];
void init() {
    f[0] = 1;
    for (int i = 1;i < MOD;i++)
        f[i] = i * f[i-1] % MOD;
}
LL C(LL n,LL m,LL MOD){
    if (n < m) return 0;
    LL ans = f[n];
    ans = ans * inv(f[m] * f[n-m] % MOD,MOD) % MOD;
    return ans;
}*/

int main() {
    LL a, b;
    //init();
    while (scanf("%lld%lld", &a, &b) != EOF)
        printf("%lld %lld %lld\n", inv(a, b), inv2(a, b), inv3(a, b));
}
```

组合数

```
LL C(int n, int m) {
    if (2 * m > n) m = n - m; //C(N,M)=C(N,N-M),当N>M/2时,N=M-N优化
    LL ans = 1;
    for (int i = 1; i <= m; i++) ans = ans * (n - i + 1) / i; //C(N,M)/C(N-1,M-1)=(N-M+1)/M递推
    return ans;
}

const int MAXC = 63;
LL c[MAXC + 1][MAXC + 1];
```

```
void init() {
    for (int i = 0; i <= MAXC; i++) {
        for (int j = c[i][0] = 1; j <= i; j++) {
            c[i][j] = c[i - 1][j] + c[i - 1][j - 1];
        }
    }
}
 /*
http://acm.uva.es/board/viewtopic.php?f=22&t=42690&sid=25bd8f7f17abec626f2ee065fec3703b
组合数取模 n 和 m 10^9 取模的数 P 要是质数 需要调用快速幂 逆元 扩展欧几里德
验证->C(43,5) % 7 == 0
*/
LL factmod(LL n, LL p) { //n!里除去 p 的倍数 使之可以求逆元
    LL res = 1;
    while (n > 1) {
        LL cur = 1;
        for (int i = 2; i < p; i++)
            cur = (cur * i) % p;
        res = mul(res, power(cur, n / p, p), p);
        for (int i = 2; i <= n % p; i++)
            res = (res * i) % p;
        n /= p;
    }
    return res % p;
}
int getx(int n, int x) { //计算 n!(或 1..n)中因子 x 的幂
    int ret = 0;
    while (n) {
        ret += n/x;
        n /= x;
    }
    return ret;
}
LL C(int n, int m, int p) {
    //C(n,m) 先检测分子分母 p 的个数
    int num = getx(n, p);//分子中 p 的幂
    int den = getx(m, p) + getx(n - m, p);//分母中 p 的幂
    if (num > den) return 0;
    LL ans = factmod(n, p);
    ans = mul(ans, inv(factmod(m, p), p), p);
    ans = mul(ans, inv(factmod(n - m, p), p), p);
    return ans;
}
/*
Lucas 定理 Lucas(n,m,p)=cm(n%p,m%p) * Lucas(n/p,m/p,p), Lucas(x,0,p)=1;
求 C(n,m) % p (p 是素数)
N = a0 + a1 * p + ... + ak * p^k, m = b0 + b1 * p + ... + bk * p^k
则 C(n,m) = C(a0,b0) * C(a1,b1) * ... * C(ak,bk) % p
注意 t!的逆元 = t! % p 的逆元
*/
//HDU3944 n 和 m 10^9，MOD 是任意 10000 内的质数 个数不多 先预处理再做
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int MAXN = 10001;
int num, prime[MAXN], idx[MAXN];
bool isprime[MAXN + 1];
//先调用筛法 sieve
int exgcd(int a, int b, int &x, int &y) { //扩展欧几里得定理:解 ax+by==1
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
```

```
    return d;
}
int fact[1234][MAXN], inv[1234][MAXN];
void init() {
    num = sieve(MAXN);
    for (int id = 0; id < num; id++) {
        int MOD = prime[id];
        idx[MOD] = id;
        fact[id][0] = 1;
        for (int i = 1; i < MOD; i++) fact[id][i] = fact[id][i - 1] * i % MOD;
        inv[id][1] = 1;
        for (int i = 2; i < MOD; i++) inv[id][i] = inv[id][MOD % i] * (MOD - MOD / i) % MOD;
    }
}
int calc(int n, int m, int MOD) { // n,m < MOD
    return m > n ? 0 : fact[idx[MOD]][n] * inv[idx[MOD]][fact[idx[MOD]][m]] % MOD *
           inv[idx[MOD]][fact[idx[MOD]][n - m]] % MOD;
}
int C(int n, int m, int MOD) {
    int ans = 1;
    while (n) {
        ans = (ans * calc(n % MOD, m % MOD, MOD)) % MOD;
        n /= MOD, m /= MOD;
    }
    return ans;
}

//C(n,m) % MOD n, m<=10^6 MOD 可以不是质数
//分子分母各自分解，幂相减，复杂度 O(n*sqrt(n))
//需要调用筛法 sieve 快速幂 power 和 mul
const int MAXN = 2000001;
const int MAXP = 150000;
const int MOD = 20100501;
int addfact(int n, int cnt[], int d) {
    int i;
    for (i = 0; i < num && prime[i] <= n; i++) {
        int x = n;
        while (x) {
            cnt[i] += d * (x / prime[i]);
            x /= prime[i];
        }
    }
    return i;
}
LL C(int n, int m) {
    int p, cnt[MAXP] = {0};
    p = addfact(n, cnt, 1);//分子 n!
    addfact(m, cnt, -1);//分母 m!
    addfact(n - m, cnt, -1);//分母(n-m)!
    LL ret = 1;
    for (int i = 0; i < p; i++) ret = mul(ret, power(prime[i], cnt[i], MOD), MOD);
    return ret % MOD;
}
int main() {
    num = sieve(MAXN);
    while (scanf("%d%d",&n, &m) != EOF)
        printf("%lld\n",C(n, m));
}

//JLU 内部各种组合函数
//http://www.earthson.net/archives/448
//http://www.earthson.net/archives/577
typedef int typec;
///Lib functions
typec GCD(typec a, typec b) {
    return b ? GCD(b, a % b) : a;
```

```
}
typec extendGCD(typec a, typec b, typec& x, typec& y) {
    if (!b) return x = 1, y = 0, a;
    typec res = extendGCD(b, a % b, x, y), tmp = x;
    x = y, y = tmp - (a / b) * y;
    return res;
}
///for x^k
typec power(typec x, typec k) {
    typec res = 1;
    while (k) {
        if (k&1) res *= x;
        x *= x, k >>= 1;
    }
    return res;
}
///for x^k mod m
typec powerMod(typec x, typec k, typec m) {
    typec res = 1;
    while (x %= m, k) {
        if (k&1) res *= x, res %= m;
        x *= x, k >>= 1;
    }
    return res;
}
/*************************************
Linear congruence theorem
x = a (mod p)
x = b (mod q)
for gcd(p, q) = 1, 0 <= x < pq
*************************************/
typec linearCongruence(typec a, typec b, typec p, typec q) {
    typec x, y;
    y = extendGCD(p, q, x, y);
    while (b < a) b += q / y;
    x *= b - a, x = p * x + a, x %= p * q;
    if (x < 0) x += p * q;
    return x;
}
/*************************************
prime table
O(n)
*************************************/
const int PRIMERANGE = 10000;
int prime[PRIMERANGE + 1];
int getPrime() {
    memset (prime, 0, sizeof (int) * (PRIMERANGE + 1));
    for (int i = 2; i <= PRIMERANGE; i++) {
        if (!prime[i]) prime[++prime[0]] = i;
        for (int j = 1; j <= prime[0] && prime[j] <= PRIMERANGE / i; j++) {
            prime[prime[j]*i] = 1;
            if (i % prime[j] == 0) break;
        }
    }
    return prime[0];
}
/*************************************
get factor of n
O(sqrt(n))
factor[][0] is prime factor
factor[][1] is factor generated by this prime
factor[][2] is factor count

need: Prime Table
*************************************/
///you should init the prime table before
```

```
int factor[80][3], facCnt;
int getFactors(int x) {
    facCnt = 0;
    int tmp = x;
    for (int i = 1; prime[i] <= tmp / prime[i]; i++) {
        factor[facCnt][1] = 1, factor[facCnt][2] = 0;
        if (tmp % prime[i] == 0)
            factor[facCnt][0] = prime[i];
        while (tmp % prime[i] == 0)
            factor[facCnt][2]++, factor[facCnt][1] *= prime[i], tmp /= prime[i];
        if (factor[facCnt][1] > 1) facCnt++;
    }
    if (tmp != 1)
        factor[facCnt][0] = tmp, factor[facCnt][1] = tmp, factor[facCnt++][2] = 1;
    return facCnt;
}


/*****************************************
C(n, k) mod p
O(k) p*p <= typecMAX
*****************************************/
typec combinationModP(typec n, typec k, typec p) {
    if (k > n) return 0;
    if (n - k < k) k = n - k;
    typec a = 1, b = 1, x, y;
    int pcnt = 0;
    for (int i = 1; i <= k; i++) {
        x = n - i + 1, y = i;
        while (x % p == 0) x /= p, pcnt++;
        while (y % p == 0) y /= p, pcnt--;
        x %= p, y %= p, a *= x, b *= y;
        b %= p, a %= p;
    }
    if (pcnt) return 0;
    extendGCD(b, p, x, y);
    if (x < 0) x += p;
    a *= x, a %= p;
    return a;
}
/*****************************************
C(n, k) mod p
Lucas's theorem for combination mod p
O(p * lgn/lgp)
*****************************************/
typec lucas(typec n, typec k, typec p) {
    typec res = 1;
    while (n && k && res) {
        res *= combinationModP(n % p, k % p, p);
        res %= p, n /= p, k /= p;
    }
    return res;
}
/*****************************************
a = n * (n - 1) * ... * (n - k + 1)
b = m * (m - 1) * ... * (m - k + 1)
c = ? from input
if a * c / b is an integer
this function will calculate this value module p^t
and the c input is moduled by p^t, be sure that gcd(c, p^t) = 1
O(len * lgn/lgp) , p^2t < typecMAX
*****************************************/
///the parameter &pcnt caches the factors consists of p
typec productQuotient(typec n, typec m, typec len, typec p, typec pt, typec &c, typec &pcnt) {
    if (!c || n < len) return c = 0;
    typec &a = c, b = 1, x, y;
```

```
    for (int i = 1; i <= len; i++) {
        x = n - i + 1, y = m - i + 1;
        while (x % p == 0) x /= p, pcnt++;
        while (y % p == 0) y /= p, pcnt--;
        x %= pt, y %= pt, a *= x, b *= y;
        a %= pt, b %= pt;
    }
    extendGCD(b, pt, x, y);
    if (x < 0) x += pt;
    a *= x, a %= pt;
    return a;
}

/*****************************************
C(n, k) mod p^t
generalized Lucas's theorem for combination mod p
O(p^t * lgn/lgp), p^2t < typecMAX
*****************************************/
typec generalizedLucas(typec n, typec k, typec p, typec t) {
    if (k > n) return 0;
    if (n - k < k) k = n - k;
    if (t == 1) return lucas(n, k, p);
    typec pt = power(p, t);
    typec c = 1, pcnt = 0, ktable[100], ntable[100], ltable[100];
    int cnt = 0;
    for (; n || k; cnt++) {
        if (k > n) return 0;
        ktable[cnt] = k, ntable[cnt] = n, ltable[cnt] = k % pt;
        n -= k % pt, k -= k % pt, n /= p, k /= p;
    }
    for (--cnt; c && cnt >= 0; cnt--)
        productQuotient(ntable[cnt], ktable[cnt], ltable[cnt], p, pt, c, pcnt);
    if (!c || pcnt >= t) return 0;
    return c * power(p, pcnt) % pt;
}

/*****************************************
C(n, k) mod m
O(min(k, p^t * lgn/lgp)) m * m < typecMAX
   p^t is fractor of m
need:
   prime table
   factor table
   generalizedLucas
   linearCongruence
*****************************************/
///you need to init the prime table
typec combinationModLucas(typec n, typec k, typec m) {
    if (m == 1 || k > n) return 0;
    if (n - k < k) k = n - k;
    getFactors(m);
    typec a, b, p, q;
    for (int i = 0; i < facCnt; i++) {
        if (!i) a = generalizedLucas(n, k, factor[i][0], factor[i][2]), p = factor[i][1];
        else b = generalizedLucas(n, k, factor[i][0], factor[i][2]), q = factor[i][1];
        if (!i) continue;
        a = linearCongruence(a, b, p, q), p *= q;
    }
    return a;
}

int main() {
    int n, p;
    getPrime();
    while (scanf("%d%d", &n, &p) != EOF) {
        typec ans = (combinationModLucas(2 * n, n, p) - combinationModLucas(2 * n, n + 1, p)) % p;
```

```
        while (ans < 0) ans += p;
        printf("%d\n", ans);
        //C(2 * n, n) - C(2 * n, n + 1)
    }
}
```

二分快速幂 A^B % C

```
#include <iostream>
using namespace std;
typedef long long LL;
LL strmod(char *s, LL c) { //s % c
    LL sum = 0;
    int len = strlen(s);
    for (int i = 0; i < len; i++) {
        sum = sum * 10 + s[i] - '0';
        while (sum >= c)
            sum -= c;
    }
    return sum;
}
LL mul(LL a, LL b, LL c) { // a*b % c
    LL ret = 0, tmp = a % c;
    for (; b; b >>= 1) {
        if (b & 1)
            if ((ret += tmp) >= c)
                ret -= c;
        if ((tmp <<= 1) >= c)
            tmp -= c;
    }
    return ret;
}

LL power(LL a, LL b, LL c) { // a^b % c
    LL res = 1;
    for (; b; b >>= 1) {
        if (b & 1) res = mul(res, a, c);
        a = mul(a, a, c);
    }
    return res;
}
/*
1 <= A,B <= 10^10000,1 <= c <= 1000000,求A^B%c
A先对C取模,然后利用结论：当b>=phi(c),(a^b)%c=a^(b%phi(c)+phi(c))%c
b < phi(c)的反例 98^4 % 64 用公式是 0 其实是16 phi(64) = 32
若c是素数则 a^b % c= a^(b%(c-1)) % c
*/
LL phi(int n) {
    LL ret = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            ret -= ret / i;
            while (n % i == 0)
                n /= i;
        }
    }
    if (n != 1)
        ret -= ret / n;
    return ret;
}
char s[1000010];
int main() {
    LL a, b, c;
    while (scanf("%lld%lld%lld", &a, &b, &c) != EOF) {
```

```
        /*while (scanf("%lld%s%lld", &a, s, &c) != EOF) {
            if (strlen(s) < 10) { //注意看 c 的位数范围！
                sscanf(s, "%lld", &b);
                if (b >= phi(c))
                    b = strmod(s, phi(c)) + phi(c);
            }
            else
                b = strmod(s, phi(c)) + phi(c);*/
        printf("%lld\n", power(a, b, c));
    }
}
```

卡特兰数第 N 项

```
//O(n)*log(m)求卡特兰数前 n 项模 MOD 的值(n <= 100000, m = 10^9)
//传入 n, m，返回 sum(catalan[n]) % m;
//做法是把 m 分解质因数，然后与 m 互质的部分可以 O(1)用逆元来求，不互质的部分复杂度 O(logm)暴力计算
//fz 维护前 i 项的不互质的分子， fm 维护前 i 项不互质的分母，每次计算后更新 fz， fm，以及不互质部分的对应幂次数组 cnt2
//catalan[i] = fz * inv(fm, m) * p[1]^cnt2[1] * p[2]^cnt2[2] * ……
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int MAXN = 100010;
const int MAXP = 40000;//sqrt(MOD)
//cnt2 是对应 m 的幂
int idx, factor[30], cnt[30], cnt2[30];
LL dp[MAXN];
//需要调用筛法 sieve 素因子分解 factorize 求逆 inv 快速幂 power
int main() {
    num = sieve(MAXN);
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF && (n || m)) {
        idx = factorize(m, factor, cnt);
        dp[1] = 1;
        memset(cnt2, 0, sizeof(cnt2));
        LL ans = dp[1] % m, fz = 1, fm = 1;
        for (int i = 2; i <= n; i++) {
            //Catalan[n] = (4 * n - 2) / (n + 1) * Catalan[n - 1]
            int now = 4 * i - 2;
            for (int j = 0; j < idx; j++) {
                while (now % prime[factor[j]] == 0) {
                    now /= prime[factor[j]];
                    cnt2[j]++;
                }
            }
            fz = fz * now % m;
            now = i + 1;
            for (int j = 0; j < idx; j++) {
                while (now % prime[factor[j]] == 0) {
                    now /= prime[factor[j]];
                    cnt2[j]--;
                }
            }
            fm = fm * now % m;
            dp[i] = fz % m;
            dp[i] = dp[i] * inv(fm, m) % m;
            for (int j = 0; j < idx; j++) {
                dp[i] = dp[i] * power(prime[factor[j]], cnt2[j], m) % m;
            }
            ans = (ans + dp[i]) % m;
        }
        printf("%lld\n", ans);
```

```
    }
}
```

MILLER RABIN 素性测试+POLLARD RHO 寻找素因子

```
//Miller-Rabin 素数测试 + POLLARD RHO 寻找素因子(<=2^54)
//调用快速幂和 srand((unsigned)time(NULL));
//Carmicheal number: 561,41041,825265,321197185
//如果选用 2, 3, 7, 61 和 24251 作为底数，那么 10^16 内唯一的强伪素数为 46 856 248 255 981。
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cstdio>
using namespace std;
typedef unsigned long long LL;
const int MAXT = 100;
const int MAXN = 30;
LL len, dig, limit;
LL random() {
    LL a = rand() * rand();
    return a * a;
}
bool Miller_Rabin(LL n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (!(n & 1)) return false;
    LL i, j, k, m, a;
    for (m = n - 1, k = 0; !(m & 1); m >>= 1, k++);
    for (i = 0; i < MAXT; i++) {
        a = power(random() % (n - 1) + 1, m, n);
        if (a == 1) continue;
        for (j = 0; j < k; j ++) {
            if (a == n - 1) break;
            a = mul(a, a, n);
        }
        if (j == k) return false;
    }
    return true;
}
LL gcd(LL a, LL b) {
    return b ? gcd(b, a % b) : a;
}
LL f(LL x, LL n) {
    return (mul(x, x, n) + 1) % n;
}
LL Pollard_Rho(LL n) {
    if (n <= 2) return 0;
    if (!(n & 1)) return 2;
    for (LL i = 1; i < MAXT; i++) {
        LL x = random() % n;
        LL xx = f(x, n);
        LL p = gcd((xx + n - x) % n, n);
        while (p == 1) {
            x = f(x, n);
            xx = f(f(xx, n), n);
            p = gcd((xx + n - x) % n, n) % n;
        }
        if (p) return p;
    }
    return 0;
}
LL factor[MAXN], m;
LL Prime(LL a) {
    if (Miller_Rabin(a)) return 0;
```

```
    LL t = Pollard_Rho(a);
    LL p = Prime(t);
    if (p) return p;
    return t;
}
//所有素约数保存于 factor[]中, m 为其个数, 返回分解后的素因子个数
int factorize(LL x, LL factor[]) {
    int m = 0;
    while (x > 1) {
        if (Miller_Rabin(x)) break;
        LL t = Prime(x);
        factor[m++] = t;
        x /= t;
    }
    if (x > 0) factor[m++] = x;
    //所有素约数保存于 factor[]中, m 为其个数
    return m;
}
int main() {
    LL a, t;
    limit = (1LL) << 63;
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%lld", &a);
        int m = factorize(a, factor);
        if (m == 1) printf("Prime\n");
        else {
            int k = 0;
            for (int i = 1; i < m; i++) {
                if (factor[k] > factor[i])
                    k = i;
            }
            printf("%lld\n", factor[k]);
        }
    }
}
```

离散对数 BABY STEP GIANT STEP

```
//baby-step-giant-step 算法
//返回一个[0,m)内的满足 a^x ≡ b(mod m)的最小的 x, a, m 需要互质
//复杂度 O(sqrt(m)logm), 利用哈希表可以优化到 O(sqrt(m))
//PKU3243 PKU2417 FZU1493
//验证 13^621 ≡ 91 (mod 877)
//特别要注意不同题目对于 0, 1 这些边界数据的不同考虑
#include <iostream>
#include <cstdio>
#include <cstring>
#include <map>
using namespace std;
typedef long long LL;
const int INF = 0x3F3F3F3F;

LL baby_step_giant_step(LL a, LL b, LL m) {
    int n = (int)sqrt(m + .0) + 1;
    LL an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * a) % m;
    map<LL, int> vals;
    LL cur = an;
    for (int i = 1; i <= n; ++i) {
        if (!vals.count(cur)) {
            vals[cur] = i;
```

```
        }
        cur = (cur * an) % m;
    }
    cur = b;
    LL x = INF;
    for (int i = 0; i <= n; ++i) {
        if (vals.count(cur)) {
            LL ans = vals[cur] * n - i;
            if (ans > 0 && ans < m) {
                x = min(x, ans);
            }
        }
        cur = (cur * a) % m;
    }
    if (x == INF) x = -1;
    return x;
}
int main() {
    int a, b, m;
    while (scanf("%d%d%d", &a, &b, &m) != EOF) {
        LL x = baby_step_giant_step(a, b, m);
        if (x == -1) {
            puts("no solution");
        } else {
            printf("%lld\n", x);
        }
    }
}
```

离散对数 扩展 BABY STEP GIANT STEP

```
//扩展 baby-step-giant-step 算法
//返回[0,m)内的满足 a^x ≡ b(mod m)的最小的 x，a, b, m 无限制
//复杂度 O(sqrt(m)logm)，利用哈希表可以优化到 O(sqrt(m))
//PKU3243 PKU2417 FZU1493 HDU2815(m 不是质数)
//验证 7987164^5134 ≡ 1691568 (mod 4684532), 13^621 ≡ 91 (mod 877)
//特别要注意不同题目对于 0, 1 这些边界数据的不同考虑 返回大于 0 最小时有点问题
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <map>
using namespace std;
typedef long long LL;
const int INF = 0x3F3F3F3F;

const int MOD = 49997;

template <class TK, class TV, int _MOD = 49997, int _n = 50000>
class HashTable {
public:
    struct H {
        TK key;
        TV val;
        int next;
    } h[_n];
    int list[_MOD], eid;
    int (*hh)(TK);

    HashTable() { //构造函数
        clear();
    }
    void clear() { //清 0
```

```
        memset(list, -1, sizeof(list));
        eid = 0;
    }
    void set(int (*hh)(TK)) { //设置回调函数(哈希函数)
        this->hh = hh;
    }
    int find(TK key) {  //查找关键字是否在哈希表中,没有返回-1
        int ind = hh(key);
        for (int p = list[ind]; p != -1; p = h[p].next) {
            if (h[p].key == key) {
                return p;
            }
        }
        return -1;
    }
    void insert(TK key, TV val) { //插入
        int ind = hh(key);
        h[eid].key = key, h[eid].val = val, h[eid].next = list[ind];
        list[ind] = eid++;
    }
    TV& operator[] (TK key) { //等同于map的[],支持没有key的查询,没有的话创建一个默认的
        int t = find(key);
        if (t == -1) {
            insert(key, TV());
            return h[eid - 1].val;
        } else {
            return h[t].val;
        }
    }
    void print(TK *out, int& N) { //0~N-1
        for (int i = 0; i < eid; i++) {
            out[i] = h[i].key;
        }
        N = eid;
    }
};

inline int h(LL s) {
    return s % MOD;
}

HashTable<LL, int> ht;

inline LL gcd(LL a, LL b) {
    return b ? gcd(b, a % b) : a;
}

inline LL exgcd(LL a, LL b, LL& x, LL& y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
//返回最小的x...通解是X=x0 + k * (n/d)
inline LL modular_linear(LL a, LL b, LL n, LL& d) {
    LL x, y;
    d = exgcd(a, n, x, y);
    if (b % d != 0)
        return -INF;// no solution
    x = b / d * x;
    x = (x % (n / d) + (n / d)) % (n / d);
    return x;
}
```

```
//solve k * a^x ≡ b (mod m), return minimum x
LL baby_step_giant_step(LL k, LL a, LL b, LL m) {
    LL ai = k % m;
    for (int i = 0; i < 50; i++) {
        if (ai == b) {
            return i;
        }
        ai = ai * a % m;
    }
    ht.clear(), ht.set(h);
    int d = 0;
    LL ani = k % m; //notice
    for (LL g; (g = gcd(a, m)) != 1; ) {
        if (b % g) return -1;//无解
        ++d;
        b /= g, m /= g;
        ani = ani * (a / g) % m;
    }
    int n = (int)ceil(sqrt(m + 0.0));
    ai = 1 % m;
    for (int i = 0; i < n; i++) {
        if (ht.find(ai) == -1) {
            ht.insert(ai, i);
        }
        ai = ai * a % m;
    }
    for (int i = 0; i <= n; i++) {
        //k * (a^n)^i * a^j = b (mod m)
        LL g, x = modular_linear(ani, b, m, g);
        int pos = ht.find(x);
        if (pos != -1) {//包含了 x 无解的情况
            return i * n + ht.h[pos].val + d;
        }
        ani = ani * ai % m;
    }
    return -1;
}
int main() {
    int K, P, N;
    while (scanf("%d%d%d", &K, &P, &N) != EOF) {
        if (N >= P)  {
            puts("Orz,I can't find D!");
            continue;
        }
        int Y = baby_step_giant_step(1, K, N, P);
        if (Y == -1) {
            puts("Orz,I can't find D!");
        } else {
            printf("%d\n", Y);
        }
    }
}
```

质数求原根

```
//一个数 g 对于 P 来说是原根，那么 g^i mod P 的结果两两不同,且 1 < g < P, 0 < i < P
//求质数原根 分解质因数复杂度 O(sqrt(phi(p)) 枚举原根复杂度 O(p)
//模 n 有原根的充要条件是 n = 1,2,4,p^m,2p^m，其中 p 是奇质数，m 任意正整数
//若 n 有一个原根，那么它一共有 phi(phi(n))个原根
//如果 p 是质数，那么一共有 phi(p - 1)个原根
//http://e-maxx.ru/algo/primitive_root
//http://zh.wikipedia.org/wiki/%E5%8E%9F%E6%A0%B9
#include <iostream>
```

```cpp
#include <cstdio>
#include <cstring>
#include <vector>
#include <cmath>
using namespace std;
int powmod(int a, int b, int p) {
    int res = 1;
    for (; b; b >>= 1) {
        if (b & 1) {
            res = int(res * 1ll * a % p);
        }
        a = int(a * 1ll * a % p);
    }
    return res;
}
int primitive_root(int p) {
    vector<int> fact;
    int phi = p - 1, n = phi, lim = sqrt(n + 0.0) + 0.5;
    for (int i = 2; i <= lim; ++i) {
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        fact.push_back(n);
    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (size_t i = 0; i < fact.size() && ok; ++i)
            ok &= powmod(res, phi / fact[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}
int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        printf("%d\n", primitive_root(n));
    }
}
```

1..N 中与 N 互质个数

```cpp
//SGU370
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <vector>
using namesisprimece std;
const int MAXN = 1000000;

int isprime[MAXN + 1], prime[MAXN], factor[MAXN + 1], p_cnt;
vector<int> pv[MAXN + 1];

void get_prime() {
    isprime[0] = isprime[1] = 1; factor[1] = 1;
    for (int i = 2; i <= MAXN; ++i) {
        if (!isprime[i]) {
            factor[i] = i;
            prime[p_cnt++] = i;
        }
        for (int j = 0; j < p_cnt && i * prime[j] <= MAXN; ++j) {
```

```
            factor[i * prime[j]] = prime[j];
            isprime[i * prime[j]] = 1;
            if (i % prime[j] == 0)
                break;
        }
    }
    for (int i = 1; i <= MAXN; ++i) {
        int n = i, fa = factor[n];
        while (n != 1) {
            pv[i].push_back(fa);
            while (n % fa == 0) n /= fa;
            fa = factor[n];
        }
    }
}
//计算 1~y 中有几个 gcd(y, i) = 1，容斥原理，复杂度 sqrt(n)+2^(p.size())
int dfs(vector<int>& pvec, int mul, int i, int y) {
    if (i == pvec.size())
        return y / mul;
    return dfs(pvec, mul, i + 1, y) - dfs(pvec, mul * pvec[i], i + 1, y);
}
int main() {
    get_prime();
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF) {
        --n, --m;
        if (n < m) swap(n, m);
        long long ans = 0;
        for (int i = 1; i <= n; ++i)
            ans += dfs(pv[i], 1, 0, m);
        if (n > 0) ++ans;
        if (m > 0) ++ans;
        printf("%I64d\n", ans);
    }
    return 0;
}
```

高斯消元

```
// 高斯消元—方阵
double mat[MAXN][MAXN + 1], answer[MAXN];
bool gauss(int tot) {
    for (int k = 0; k < tot; ++k) {
        int dec = k;
        for (int i = k + 1; i < tot; ++i)
            if (fabs(mat[i][k]) > fabs(mat[dec][k]))
                dec = i;
        if (-1e-6 <= mat[dec][k] && mat[dec][k] <= 1e-6)
            return false;
        if (dec != k) {                          // 交换矩阵的行，解不变！！！
            for (int j = k; j <= tot; ++j)
                swap(mat[k][j], mat[dec][j]);
        }
        for (int i = k + 1; i < tot; ++i) {
            double r = mat[i][k] / mat[k][k];
            mat[i][k] = 0.0;
            for (int j = k + 1; j <= tot; ++j)
                mat[i][j] -= mat[k][j] * r;
        }
    }
    for (int k = tot - 1; k >= 0; --k) {
        answer[k] = mat[k][tot];
        for (int i = k + 1; i < tot; ++i)
            answer[k] -= mat[k][i] * answer[i];
```

```
        answer[k] /= mat[k][k];
    }
    return true;
}

// 高斯消元—解 xor 方程
long long gauss(int n, int m) {
    int k, t;
    for (k = 0, t = 0; k < m && t < n; ++k, ++t) {    // k for column; t for row
        int dec;
        for (dec = t; dec < n && mat[dec][k] == 0; ++dec)
            ;
        if (dec == n) {
            --t;
            continue;
        }
        if (dec != t) {
            for (int j = k; j <= m; ++j)
                swap(mat[t][j], mat[dec][j]);
        }
        for (int i = t + 1; i < n; ++i)
            if (mat[i][k]) {
                for (int j = k; j <= m; ++j)
                    mat[i][j] = mat[i][j] ^ mat[t][j];
            }
    }
    for (int i = t; i < n; ++i)
        if (mat[i][m])
            return 0;
    return 1LL << (m - t);
}

// 高斯消元—一般矩阵
double mat[MAXN][MAXN + 1], answer[MAXN];
// 返回矩阵的秩
int gauss(int n, int m) {
    int k, t;
    for (k = 0, t = 0; k < m && t < n; ++k, ++t) {
        // k for column; t for row
        int dec = t;
        for (int i = t + 1; i < n; ++i)
            if (fabs(mat[i][k]) > fabs(mat[dec][k]))
                dec = i;
        if (fabs(mat[dec][k]) < 1e-8) {
            --t;
            continue;
        }
        if (dec != t) {
            for (int j = k; j <= m; ++j)
                swap(mat[t][j], mat[dec][j]);
        }
        for (int i = t + 1; i < n; ++i) {
            double r = mat[i][k] / mat[t][k];
            mat[i][k] = 0.0;
            for (int j = k + 1; j <= m; ++j)
                mat[i][j] -= mat[t][j] * r;
        }
    }
    for (int i = t; i < n; ++i)
        if (fabs(mat[i][m]) > 1e-8)
            return -1;
    if (t == m)
        for (int k = m - 1; k >= 0; --k) {
            answer[k] = mat[k][m];
            for (int i = k + 1; i < m; ++i)
                answer[k] -= mat[k][i] * answer[i];
```

```
            answer[k] /= mat[k][k];
        }
    return t;
}
```

矩阵 MATRIX

```
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
typedef long long LL;
const int MAXN = 2;//矩阵规格
const int MOD = 100000000;//取模的数
int n, k, vec[MAXN] = {1, 0};
struct Mat {
    LL mat[MAXN][MAXN];//小数据改 int 大数据加 unsigned
} A, ans, E, fib = {1, 1, 1, 0};
Mat operator*(const Mat& a, const Mat& b) {
    Mat c;
    memset(c.mat, 0, sizeof(Mat));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                if (a.mat[i][k] && b.mat[k][j])
                    c.mat[i][j] = (c.mat[i][j] + a.mat[i][k] * b.mat[k][j]) % MOD;
    return c;
}
Mat operator+(const Mat& a, const Mat& b) {
    Mat c;
    memset(c.mat, 0, sizeof(Mat));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            c.mat[i][j] = (a.mat[i][j] + b.mat[i][j]) % MOD;
    return c;
}
Mat operator^(Mat A, int x) { //优先级低加括号
    Mat c = E;
    for (; x; x >>= 1) {
        if (x & 1)
            c = c * A;
        A = A * A;
    }
    return c;
}
Mat sum(const Mat& A, int x) { //A^1+A^2+...+A^x
    if (x == 1)
        return A;
    if (x & 1)
        return (A ^ x) + sum(A, x - 1);
    else
        return sum(A, x / 2) * ((A ^ (x / 2)) + E);
}
int main() {
    n = MAXN;
    for (int i = 0; i < MAXN; i++) E.mat[i][i] = 1;
    A = fib;
    while (scanf("%d", &k) != EOF && k) {
        ans = A ^ k;
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum = (sum + ans.mat[0][i] * vec[i]) % MOD;
        printf("%d\n", sum);
```

```
    }
}
```

线性递推 K 倍项和

```
//CII5112-Sales Prediction
//矩阵，利用向量(a[1],a[2],……,a[R],sum[K],sum[K-1],……,sum[1])递推
//已知 F 前 R 项及递推公式 F[n] = a[1]*F[n-1] + a2*F[n-2] + ... + aR*F[n-R]
//求前 N 项中 K 的倍数项 F[K*0] + F[K*1] + F[K*2] + .... 的和

先调用上面的矩阵
//已知 F 前 R 项及递推公式 F[n] = a[1]*F[n-1] + a2*F[n-2] + ... + aR*F[n-R]
//求前 N 项中 K 的倍数项 F[K*0] + F[K*1] + F[K*2] + .... 的和
LL Sum(LL a[], LL f[], int R, int K, LL N) {
    n = R + K;
    memset(A.mat, 0, sizeof(Mat));
    for (int i = 0; i < R; i++) {
        if (i + 1 < R) {
            A.mat[i][i + 1] = 1;
        } else {
            for (int j = 0; j < R; j++) {
                A.mat[i][j] = a[R - j - 1];
            }
        }
    }
    for (int i = 0; i < K; i++) {
        f[R + i] = 0;
        if (i == 0) {
            A.mat[R + i][0] = 1;
            A.mat[R + i][R + K - 1] = 1;
        } else {
            A.mat[R + i][R + i - 1] = 1;
        }
    }
    ans = A^(N);
    LL sum = 0;
    for (int i = 0; i < n; i++) {
        sum = (sum + (ans.mat[R][i] * f[i]) % MOD) % MOD;
    }
    return sum;
}

int main() {
    n = MAXN;
    for (int i = 0; i < MAXN; i++) E.mat[i][i] = 1;
    int cas;
    scanf("%d", &cas);
    int R, K;
    while (cas--) {
        scanf("%lld%d%d", &k, &R, &K);
        memset(A.mat, 0, sizeof(Mat));
        for (int i = 0; i < R; i++) scanf("%lld", &f[i]);
        for (int i = 0; i < R; i++) scanf("%lld", &a[i]);
        printf("%lld\n", Sum(a, f, R, K, k * K));
    }
}
```

PELL 方程

```
/*
正整数 d 不是完全平方时方程 x^2-dy^2=1 有无穷多组整数解
X*X - n * Y * Y = 1 则有
X(i+1) = X1Xi + n * Y1Yi      Y(i+1) = X1Yi +      Y1Xi、
|x1 D*y1|| x1 |
|y1   x1|| y1 |
枚举一个解构造系数矩阵 X=[3 8;1 3];构造列向量 L=[3;1]第 k 个解=X^k * L
ax^2-by^2=c 先暴力一个 x^2-aby^2=1 的最小解 x0,y0(x0,y0>0,并是 x0 最小的满足条件的解)
继续求 ax^2-by^2=c 的一个最小特解 x1,y1(x1,y1>0),假设要求第 k 个解那么有
| xk | = |x0   by0|^k-1| x1 |
| yk |   |ay0   x0|    | y1 |
n = 2,k = 999888 -> ans = 7181 返回-MAXV 则无解
*/
inline bool issqr(int x) {
    int t = (int)sqrt(double(x));
    return t * t == x;
}
long long pell(int n, int k) { //X^2 + n*Y^2 = 1 的第 k 个解(x)
    if (issqr(n))
        return -MAXV; //pell 方程无解
    int x, y;
    for (y = 1; ; y++) {
        if (issqr(1 + n * y * y)) {
            x = sqrt(double(1 + n * y * y));
            break;
        }
    }
    Mat A, ans;
    A.mat[0][0] = x, A.mat[0][1] = m * y;
    A.mat[1][0] = y, A.mat[1][1] = x;
    ans = A^k;
    return (ans.mat[0][0] + MOD) % MOD;
}
```

行列式计算

```
//计算矩阵行列式的值
long long determinant() {
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         mat[i][j] %= MOD;
   long long ans = 1;
   for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
         while (mat[j][i]) {
            long long t = mat[i][i] / mat[j][i];
            for (int k = i; k < n; k++) {
               mat[i][k] = (mat[i][k] - t * mat[j][k]) % MOD;
               swap(mat[i][k], mat[j][k]);
            }
            ans = -ans;
         }
      }
      if (mat[i][i] == 0) {
         return 0;
      } else {
         ans = ans * mat[i][i] % MOD;
      }
   }
   return (ans + MOD) % MOD;
}
```

```
//FFT 多项式乘法 复杂度 O(NlogN)
//调用 polymul(poly1, len1, poly2, len2, ans, len);
//FFT 主要利用了单位复根 w^n=1 的 w 就是 n 次单位复根的特殊性质
//根是 e^(2PI*k/n)=cos(2*PI*k/n) + i sin(2*PI*k/n),k=0,1,……n-1)
//http://blog.csdn.net/v_july_v/article/details/6684636
//http://numbers.computation.free.fr/Constants/Algorithms/fft.html
//UVa12298
#include <iostream>
#include <complex>
using namespace std;
typedef complex<long double> CP;
typedef long long LL;
const double eps = 1e-8;
const double pi = acos(-1.0);
const int MAXN = 1 << 17; ////大于最大的长度的 2 的幂次的 2 倍，50000->65535*2
const int MAXM = 50001;
void fft(CP a[], CP A[], int n, bool rev) {
    int lg2n = 1;
    while ((1 << lg2n) != n) ++lg2n;
    for (int i = 0; i < n; i++) {
        int x = 0;
        for (int j = 0; j < lg2n; j++) {
            if (i & (1 << j)) x |= 1 << (lg2n - 1 - j);
        }
        A[x] = a[i];
    }
    for (int s = 1; s <= lg2n; ++s) {
        int m = 1 << s;
        long double p = pi * 2 / m * (rev ? -1: 1);
        CP wm(cos(p), sin(p));
        for (int k = 0; k < n; k += m) {
            CP w(1);
            int b = m >> 1;
            for (int j = 0; j < b; j++) {
                CP t = w * A[k + j + b];
                A[k + j + b] = A[k + j] - t;
                A[k + j] += t;
                w *= wm;
            }
        }
    }
    if (rev) for (int i = 0; i < n; i++) A[i] /= n;
};

int find_mi2(int x) {
    int res = 1;
    while (res < x) res <<= 1;
    return res;
}

CP x[MAXN << 1], ya[MAXN << 1], yb[MAXN << 1], yc[MAXN << 1];

void polymul(LL a[], int na, LL b[], int nb, LL c[], int& nc) {
    int n = find_mi2(max(na, nb)) << 1;
    for (int i = 0; i < n; i++) x[i] = i < na ? a[i] : 0;
    fft(x, ya, n, false);
    for (int i = 0; i < n; i++) x[i] = i < nb ? b[i] : 0;
    fft(x, yb, n, false);
    for (int i = 0; i < n; i++) yc[i] = ya[i] * yb[i];
    fft(yc, x, n, true);
    for (int i = 0; i < n; i++) c[i] = (LL)(x[i].real() + 0.5);//注意括号
    for (nc = n; nc > 0 && c[nc - 1] == 0; nc--);
}
```

```
//求卷积 c[k] = sigma(i=k..n-1){a[i] * b[i - k]}
//注意 c 数组要 2MAXN 才够
void convolution(LL a[], LL b[], LL c[], int n) {
    int m;
    LL *rb = new LL[n];
    for (int i = 0; i < n; i++) rb[i] = b[n - 1 - i];
    polymul(a, n, rb, n, c, m);
    for (int i = 0; i < n; i++) c[i] = c[i + n - 1];
    delete rb;
}
//下面是无关函数，应题目要求
long long pol[5][MAXN];
int a, b, c;
bool used[MAXM];
void init() {
    memset(used, 0, sizeof(used));
    for (int i = 2; i < MAXM; i++) {
        if (used[i]) continue;
        if (MAXM / i <= i) continue;
        for (int j = i * i; j < MAXM; j += i) used[j] = true;
    }
}
int to(char c) {
    if (c == 'S') return 0; if (c == 'H') return 1;
    if (c == 'C') return 2; if (c == 'D') return 3;
    return -1;
}
int main() {
    init();
    while (scanf("%d%d%d", &a, &b, &c) == 3 && (a + b + c)) {
        int n = find_mi2(b + 1) * 2;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < n; j++) {
                if (j > b || !used[j]) pol[i][j] = 0;
                else pol[i][j] = 1;
            }
        }
        for (int i = 0; i < c; i++) {
            int x; char ch;
            scanf("%d%c", &x, &ch);
            int s = to(ch);
            pol[s][x] = 0;
        }
        int t = 0;
        for (int i = 1; i < 4; i++) polymul(pol[0], b + 1, pol[i], b + 1, pol[0], t);
        for (int i = a; i <= b; i++) printf("%lld\n", pol[0][i]);
        printf("\n");
    }
}
```

FFT 高精度乘法[万进制]

```
//FFT 高精度乘法[万进制]
//SPOJ-MUL SPOJ-TMUL HDU1402 ZJUT1217
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
#include <algorithm>
#include <complex>
#include <cassert>
using namespace std;
```

```
typedef complex<long double> CP;
const int MAXN = 1 << 14;//大于最大的四位压缩后长度的 2 的幂次，50000->16384
const double pi = acos(-1.0);

void fft(CP a[], CP A[], int n, bool rev) {
    int lg2n = 1;
    while ((1 << lg2n) != n) ++lg2n;
    for (int i = 0; i < n; i++) {
        int x = 0;
        for (int j = 0; j < lg2n; j++) {
            if (i & (1 << j)) {
                x |= 1 << (lg2n - 1 - j);
            }
        }
        A[x] = a[i];
    }
    for (int s = 1; s <= lg2n; ++s) {
        int m = 1 << s;
        long double p = pi * 2 / m * (rev ? -1: 1);
        CP wm(cos(p), sin(p));
        for (int k = 0; k < n; k += m) {
            CP w(1);
            int b = m >> 1;
            for (int j = 0; j < b; j++) {
                CP t = w * A[k + j + b];
                A[k + j + b] = A[k + j] - t;
                A[k + j] += t;
                w *= wm;
            }
        }
    }
    if (rev) for (int i = 0; i < n; i++) A[i] /= n;
};

int find_mi2(int x) {
    int res = 1;
    while (res < x) res <<= 1;
    return res;
}
CP pa[MAXN << 1], pb[MAXN << 1], pt1[MAXN << 1], pt2[MAXN << 1];
//FFT 高精度乘法，传入数 1,len1,数 2,len2，结果乘积在 o 中,长度为 l, n 是对应 FFT 的项数
void mul(CP p1[], int l1, CP p2[], int l2, CP o[], int& l, int n) {
    fft(p1, pt1, n, false);
    fft(p2, pt2, n, false);
    for (int i = 0; i < n; i++) pt1[i] *= pt2[i];
    fft(pt1, o, n, true);
    for (int i = 0; i < n; i++) {
        o[i] = CP((long long)(o[i].real() + 0.5));
    }
    //下面是进位
    for (int i = 0; i < n; i++) {
        o[i + 1].real() += (long long)o[i].real() / 10000;
        o[i].real() = (long long)o[i].real() % 10000;
    }
    for (l = l1 + l2 - 1; (int)o[l].real() == 0 && l; l--);
    l++;
}
char a[MAXN], b[MAXN];
int four[MAXN >> 2];//MAXN / 4 就可以存下数字
void initnum(char s[], int n, CP* p, int len) {
    memset(four, 0, sizeof(four));
    for (int i = 0; i < n; i++) {
        four[(n - i - 1) / 4] = four[(n - i - 1) / 4] * 10 + (s[i] - '0');
    }
    n = (n + 3) / 4;
    for (int i = 0; i < n; i++) {
```

```
        p[i] = CP(four[i]);
    }
    for (int i = n; i < len; i++) {
        p[i] = CP(0);
    }
}

void multiply(char a[], char b[]) {
    int sign = 0;
    if (a[0] == '-') { a++, sign ^= 1; }
    if (b[0] == '-') { b++, sign ^= 1; }
    int la = strlen(a), lb = strlen(b);
    int n = find_mi2(max((la + 3) / 4, (lb + 3) / 4)) * 2; //FFT 乘法需要 2 倍的长度
    initnum(a, la, pa, n);
    initnum(b, lb, pb, n);
    int l;
    mul(pa, (la + 3) / 4, pb, (lb + 3) / 4, pa, l, n);
    if (l - 1 == 0 && (int)pa[l - 1].real() == 0) {//防止负 0
        puts("0");
        return;
    }
    if (sign) putchar('-');
    for (int i = l - 1; i >= 0; i--) {
        if (i == l - 1) {
            printf("%d", (int)pa[i].real());
        } else {
            printf("%04d", (int)pa[i].real());
        }
    }
    puts("");
}

int main() {
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%s%s", a, b);
        multiply(a, b);
    }
}
```

FFT 高精度乘法[十进制]

```
//FFT 高精度乘法[十进制]
//SPOJ-TMUL HDU1402 ZJUT1217
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
#include <algorithm>
#include <complex>
#include <cassert>
using namespace std;
typedef complex<long double> CP;
const int MAXN = 1 << 17;//大于最大的长度的 2 的幂次的 2 倍，50000->65535*2
const double pi = acos(-1.);

void fft(CP a[], CP A[], int n, bool rev) {
    int lg2n = 1;
    while ((1 << lg2n) != n) ++lg2n;
    for (int i = 0; i < n; i++) {
        int x = 0;
        for (int j = 0; j < lg2n; j++) {
```

```
            if (i & (1 << j)) {
                x |= 1 << (lg2n - 1 - j);
            }
        }
        A[x] = a[i];
    }
    for (int s = 1; s <= lg2n; ++s) {
        int m = 1 << s;
        long double p = pi * 2 / m * (rev ? -1: 1);
        CP wm(cos(p), sin(p));
        for (int k = 0; k < n; k += m) {
            CP w(1);
            int b = m >> 1;
            for (int j = 0; j < b; j++) {
                CP t = w * A[k + j + b];
                A[k + j + b] = A[k + j] - t;
                A[k + j] += t;
                w *= wm;
            }
        }
    }
    if (rev) for (int i = 0; i < n; i++) A[i] /= n;
};

int find_mi2(int x) {
    int res = 1;
    while (res < x) res <<= 1;
    return res;
}

CP pa[MAXN], pb[MAXN], pt1[MAXN], pt2[MAXN];

//FFT 高精度乘法，传入数 1,len1,数 2,len2，结果乘积在 o 中,长度为 l, n 是对应 FFT 的项数
void mul(CP p1[], int l1, CP p2[], int l2, CP o[], int& l, int n) {
    fft(p1, pt1, n, false);
    fft(p2, pt2, n, false);
    for (int i = 0; i < n; i++) pt1[i] *= pt2[i];
    fft(pt1, o, n, true);
    for (int i = 0; i < n; i++) {
        o[i] = CP((int)(o[i].real() + 0.5));
    }
    //下面是进位
    for (int i = 0; i < n; i++) {
        o[i + 1].real() += (int)o[i].real() / 10;
        o[i].real() = (int)o[i].real() % 10;
    }
    for (l = l1 + l2 - 1; (int)o[l].real() == 0 && l; ) l--;
    l++;
}

char a[MAXN >> 1], b[MAXN >> 1];

void initnum(char s[], int n, CP* p, int len) {
    for (int i = 0; i < n; i++) {
        p[i] = CP(s[n - i - 1] - '0');
    }
    for (int i = n; i < len; i++) {
        p[i] = CP(0);
    }
}

void multiply(char a[], char b[]) {
    int sign = 0;
    if (a[0] == '-') {
        a++, sign ^= 1;
    }
```

```
    if (b[0] == '-') {
        b++, sign ^= 1;
    }
    int la = strlen(a), lb = strlen(b);
    int n = find_mi2(max(la, lb)) * 2; //FFT 乘法需要 2 倍的长度
    initnum(a, la, pa, n);
    initnum(b, lb, pb, n);
    int l;
    mul(pa, la, pb, lb, pa, l, n);
    if (l - 1 == 0 && (int)pa[l - 1].real() == 0) {//防止负 0
        puts("0");
        return;
    }
    if (sign) putchar('-');
    for (int i = l - 1; i >= 0; i--) printf("%d", (int)pa[i].real());
    puts("");
}

int main() {
    while (scanf("%s%s", a, b) != EOF) {
        multiply(a, b);
    }
}
```

龙贝格积分公式 ROMBERG

```
/*
龙贝格积分(对于∫(a,b) fun(x)*dx)
可以用来算面积也可以用来算长度
*/
#include <iostream>
#include <cstdio>
#include <vector>
#include <cmath>
using namespace std;
template <typename T>
T fun(T x) { //被积函数
    return 3 * x * x;
}
template <typename T>
T romberg(T a, T b, T eps = 1e-5) {
    vector<T> R;
    int k = -1, pow2 = 1;
    double A;
    T r = 0.5 * (b - a) * (fun(a) + fun(b));
    R.push_back(r);
    do {
        A = R[0];
        k += 1;
        r = 0.0;
        for (int i = 0; i < pow2; i++)
            r += fun(a + (b - a) * (i + 0.5) / pow2);
        r *= (b - a) / (2.0 * pow2);
        r += 0.5 * R[k];
        R.push_back(r);
        double pow4 = 4.0;
        for (int m = 0; m <= k; m++, pow4 *= 4.0)
            R[k-m] = (pow4 * R[k + 1 - m] - R[k - m]) / (pow4 - 1);
        pow2 *= 2;
    } while (fabs(R[0] - A) > eps);
    return R[0];
}
int main() {
    printf("%lf\n", romberg(0.0, 5.0, 1e-5));
```

```
}
```

复合辛普森积分公式 SIMPSON

```
/*
复合 Simpson 积分公式
I(f) = ∫(a,b)f(x)dx = ∑k[0,n-1]∫(xk,x(k+1))f(x)dx
     = (h/6)*{f(a) + 4*∑k[0,n-1]f(x((2*k+1)/2))dx + 2*∑k[1,n-1]f(xk)dx + f(b)}
  xk = a+k*h(k = 0,1,...,n)
   h = (b-a) / n;
*/
#include <iostream>
#include <cmath>
#include <cstdio>
#include <algorithm>
using namespace std;
const long double H = 5e-5;
long double r1, r2;
long double f(long double x) { //被积函数
    return x*x;
}
long double simpson(long double a, long double b) {
    long double sum = 0;
    sum += f(a) + f(b);
    int n = int((b - a) / H);
    long double h = (b - a) / n / 2;
    for (int i = 1; i < 2 * n; i++) {
        if (i & 1)
            sum += 4 * f(a + i * h);
        else
            sum += 2 * f(a + i * h);
    }
    return sum * H / 6;
}
int main() {
    int re;
    scanf("%d", &re);
    for (int ri = 1; ri <= re; ++ri) {
        scanf("%Lf%Lf", &r1, &r2);
        if (r1 > r2)
            swap(r1, r2);
        long double ans = simpson(0, r1);
        printf("%.6Lf\n", ans);
    }
}
```

梯形积分法

```
//梯形积分法 适合暴力验证
#include <iostream>
#include <cmath>
using namespace std;
const int LEN = 1 << 18;//精度
double f(double x) { //被积函数
    return 2 * x;
}
double calc(double x, double y) {
    double sum = 0, dx = (y - x) / LEN;
    for (int i = 0; i < LEN; i++)
        sum += f(x + i * dx + dx / 2);
    return sum * dx;
```

```
}
int main() {
    double x, y;
    while (cin >> x >> y)
        printf("%lf\n", calc(x, y));
}
```

连分数

```
//输入一个小数 返回该小数的连分数序列
//连分数序列是 x = a0 + 1 / (a1 + 1 / (a2 + 1 / ...))
#include <iostream>
#include <cmath>
using namespace std;
typedef long long LL;
LL Num[1000];//记录连分数数列
int Fraction(LL r, LL f) { //Continued fractions
    LL m = 1;
    while (m <= f) m *= 10;
    LL s = r * m + f;
    int len = 0;
    while (s % m != 0) {
        LL t = s % m;
        Num[len++] = s / m;
        s = m;
        m = t;
    }
    Num[len++] = s / m;
    //将分母依次存入数列
    return len;
}

int main() {
    LL r, f;
    while (scanf("%lld.%lld", &r, &f) != EOF) {
        int len = Fraction(r, f);
        for (int i = 0; i < len; i++)
            printf("%d:%lld\n", i + 1, Num[i]);
    }
    return 0;
}
```

置换群分解

```
//置换群分解，把 a[]到 b[]的置换分解，结果在 loop 中
#include <iostream>
const int MAXN = 201;
int n, m, a[MAXN], b[MAXN], len[MAXN];
bool vis[MAXN];
vector<pair<int, int> > loop[MAXN];
map<int, int> id;
void dfs(int x) {
    if (vis[id[a[x]]]) return;
    vis[id[a[x]]] = 1;
    loop[m].push_back(make_pair(a[x], b[x]));
    dfs(id[b[x]]);
}

int main() {
    while (scanf("%d", &n) != EOF) {
```

```
        id.clear();
        m = 0;
        memset(vis, 0, sizeof(vis));
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            id[a[i]] = i;
        }
        for (int i = 1; i <= n; i++) scanf("%d", &b[i]);
        for (int i = 1; i <= n; i++)
            if (!vis[id[a[i]]]) {
                loop[m].clear();
                dfs(id[a[i]]);
                len[m] = loop[m].size();
                m++;
            }
        for (int i = 0; i < m; i++) {
            printf("%d %d\n", i, len[i]);
            for (int j = 0; j < loop[i].size(); j++)
                printf("%d->%d\n", loop[i][j].first, loop[i][j].second);
        }
    }
}
```

N 进制类

```
//用于连通性状态压缩 dp
template <int N>
struct Number {
    static int w[30]; //X = b0 * w[0] + b1 * w[1] + ... + b[n] * w[n]
    Number() { init(); }
    static void init() { //对于每个实例必须先初始化
        w[0] = 1; for(int i = 1; i < 30; i++) w[i] = w[i - 1] * N;
    }
    int toInt(int d[], int ind) { //将一个数组计算出一个 N 进制数, ind base0
        int ret = 0;
        for (int i = 0; i < ind; i++) ret += d[i] * w[i];
        return ret;
    }
    void parse(int val, int d[], int &ind) { //将一个 N 进制数解析成一个数组
        ind = 0;
        while (val) {
            d[ind++] = val % N;
            val /= N;
        }
        if (ind == 0) d[ind++] = 0;
    }
    int get(int val, int pos) {  //取出 N 进制数 val 的第 pos 位
        return val % w[pos + 1] / w[pos];
    }
    void set(int &val, int pos, int b) { //将 N 进制数 val 的第 pos 位设置为 b
        val += (b - get(val, pos)) * w[pos];
    }
};
template <int N>
int Number<N>::w[30];
int main() {
    Number<20> base;
    int len, x[100];
    base.parse(1234, x, len);
    for (int i = 0; i < len; i++) {
        printf("%d ", x[i]);
    }
    puts("");
}
```

本原勾股数

```
//生成本原勾股数 a^2+b^2=c^2,其中 a,b,c 互质
//生成本原勾股数 a^2 + b^2 = c^2, 其中 a, b, c 互质
#include <iostream>
using namespace std;
typedef long long LL;
const int LIMIT = 100000;
const int RLIMIT = sqrt(2.0 * LIMIT);
LL gcd(LL a, LL b) {
    return b ? gcd(b, a % b) : a;
}
int main() {
    int ans = 0;
    LL a, b, c;
    //for (LL t = 1; t <= RLIMIT; t += 2) {
    //    for (LL s = t + 2; ; s += 2) {
    //s > t > 1, a = s * t, b = (s^2 - t^2) / 2, c = (s^2 + t^2) / 2
    for (LL s = 3; s <= RLIMIT; s += 2) {
        for (LL t = 1; t < s; t += 2) {
            if (gcd(s, t) != 1) continue;
            a = s * t;
            b = (s * s - t * t) / 2;
            if (a > b) {
                swap(a, b);
            }
            c = (s * s + t * t) / 2;
            if (c > LIMIT) break;
            //ans++;
            //printf("%lld %lld %lld\n", a ,b ,c);
            for (LL x = a, y = b, z = c; z <= LIMIT; x += a, y += b, z += c) {
                //printf("%lld %lld %lld\n", x ,y ,z);
                ans++;
            }
        }
    }
    printf("%d\n", ans);
}
```

X^2 + Y^2 = R^2 整数解个数

```
//x^2 + y^2 = r^2 整数解个数 HAOI2008 圆上整点
#include <iostream>
#include <cstdio>
#include <cmath>
using namespace std;
int ans;
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
void solve(int r) {
    if (r == 1 || r % 4 != 1) return;
    for (int n = 1; n * n * 2 < r; n++) {
        int m = sqrt(r - n * n) + 0.5;
        if (m * m + n * n != r) continue;
        if (gcd(m, n) == 1 && m > n) ans++;
    }
}
int main() {
    int r;
    while (scanf("%d", &r) != EOF) {
        ans = 0;
```

```
        for (int k = 1; k * k <= r; k++) {
            if (r % k) continue;
            solve(r / k);
            solve(k);
        }
        printf("%d\n", ans * 8 + 4);
    }
}
```

FAREY 序列构造

```
/*
Farey 序列构造 PKU3374
输出格式 t 个第 n 行第 k 个分数
Farey 序列
Fn = {a/b | gcd(a,b)=1 && 0<=a,b<=n};
即由小于或等于 n 的整数所组成的不可再约分数的递增序列,并满足分子分母互质.

性质:
1:除了 F1,其余 Farey 序列都有奇数个元素,并且中间值是 1/2.
2:Farey 序列是一个对称序列,头尾之和为 1.
3:若相邻两个元素是 a/b 和 c/d (a/b<c/d),则这两个数的差为 1/bd
4:假如序列中有三个连续元素 x1/y1, x2/y2, x3/y3,则有 x2 = x1+x3; y2 = y1+y3;
  并且有 x1*y2 - x2*y1 = 1.这条性质保证构造出来的分式肯定是不可约分式.
5:Fn 的个数即∑phi[i](1..n);F1000 有 300927 个

F1 = {0/1, 1/1}
F2 = {0/1, 1/2, 1/1}
F3 = {0/1, 1/3, 1/2, 2/3, 1/1}
F4 = {0/1, 1/4, 1/3, 1/2, 2/3, 3/4, 1/1}
F5 = {0/1, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 1/1}
F6 = {0/1, 1/6, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 1/1}
F7 = {0/1, 1/7, 1/6, 1/5, 1/4, 2/7, 1/3, 2/5, 3/7, 1/2, 4/7, 3/5, 2/3, 5/7, 3/4, 4/5, 5/6, 6/7, 1/1}
F8 = {0/1, 1/8, 1/7, 1/6, 1/5, 1/4, 2/7, 1/3, 3/8, 2/5, 3/7, 1/2, 4/7, 3/5, 5/8, 2/3, 5/7, 3/4, 4/5,
5/6, 6/7, 7/8, 1/1}

Farey 序列一种二分的构造法:a/b,c/d (a/b<c/d)是一个 n 级法雷数列中的两个元素,且 b+d<=n, 则可以在 a/b, c/d 中间
插入一个分数 (a+c)/(b+d),比如构造 f(5)
0/1, 1/1
0/1, 1/2, 1/1
0/1, 1/3, 1/2, 2/3, 1/1
0/1, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 1/1
0/1, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 1/1


构造
从第 1、4 两个性质我们可以得出它的构造方法.求 N 阶 Farey 序列:
这个方法适合随机给定 N,求 Farey 序列.
void makefarey(int x1,int y1,int x2,int y2) {
    if (x1 + x2 > N || y1 + y2 > N) return;
    farey[++ind] = {x1+x2,y1+y2};
    makefarey(x1+x2,y1+y2,x2,y2);
}
如果求出连续的 F1,F2,F3,F4…Fn 的话,朴素构造方法更好.
Procedure make farey(n : integer)
farey[1] = {0/1, 1/1}
total = 2
    For i=2 to n
        farey [i] = {0/1}
        For j=2 to total
            If farey [i-1][j].denominator + farey [i][j]. denominator = N then
                farey[i] += { farey [i-1][j] + farey [i][j] }
            End if
```

```
            farey [i] += {farey [i-1][j]}
        End for
    End for
End Procedure.
```

所谓的 Stern-Brocot 树，其实已经在第一个构造算法里面隐含了.
Stern-Brocot 扩展了 Farey 序列，它能构造出小于某个分式的所有分式，当然这些分式是无穷的.所以它能从另一方面证明素数是无穷的.

查找第 K 大元素
pku 3374 Cake Share 有较多查询，得预先构造出 Fn 序列.
如果查询对不可预知 Fn 的时候，我们可以简单的改造算法 1，当 total 达到 k 时输出后，立即跳出.时间复杂度 O(K).K 最大就是 |Fn|.
Fn 的序列大小是可以递推出来的，有一个近似公式，可以让我们大致了解下 Fn 的大小程度.
|Fn| = 0.304*N^2.如 |F5000| ≈ 7600000，实际有 7600459 个.
那可以看出这个算法复杂度是 O(N^2)的.
黑书上的解决方案是二分加上统计，可以如下操作：
规定另一个操作计算出小于给定分式的不可约分式数目，要求<=O(NlogN).
再对 X/N 的 X 进行二分枚举，找出区间 X/N ~ (X+1)/N.
在枚举出改区间的所有不可约分式，最多也就 N 个.
统计再输出答案，总过程时间复杂度是 O(N (logN)^2)的.关键就在于实现 O(NlogN)的操作.
*/

```
//Farey 序列构造 PKU3374
#include <iostream>
using namespace std;
const int MAXN = 4000000;
int ind;
int n, k;
int farey[2][MAXN];
void makefarey(int x1, int y1, int x2, int y2) {
    if (x1 + x2 > n || y1 + y2 > n) return;
    makefarey(x1, y1, x1 + x2, y1 + y2);
    ind++;
    farey[0][ind] = x1 + x2;
    farey[1][ind] = y1 + y2;
    makefarey(x1 + x2, y1 + y2, x2, y2);
}
int main() {
    int t;
    while (scanf("%d %d", &n, &t) != EOF) {
        if (n == 1) {
            while (t--) {
                scanf("%d", &k);
                if (k == 1) puts("0/1");
                else if (k == 2) puts("1/1");
                else puts("No Solution");
            }
            continue;
        }
        ind = 1;
        farey[0][1] = 0, farey[1][1] = 1;
        makefarey(0, 1, 1, 2);
        ind++;
        farey[0][ind] = 1, farey[1][ind] = 2;
        int all = 2 * ind;
        while (t--) {
            scanf("%d", &k);
            if (k >= all) {
                puts("No Solution");
            } else if (k <= ind) {
                printf("%d/%d\n", farey[0][k], farey[1][k]);
            } else if (k > ind) {
                printf("%d/%d\n", farey[1][all-k] - farey[0][all-k], farey[1][all-k]);
            }
        }
    }
}
```

```
//约瑟夫问题 可输出步骤和最后一个人的位置
#include<iostream>
using namespace std;
typedef long long LL;
//next = (k + m-1 - 1) % 人数 + 1
LL Josephus(LL n, int m, LL k) { //分别为：人数 出圈步长 起使报数位置
    LL x;
    if (m == 1) k = (k == 1) ? n : (k + n - 1) % n;
    else {
        for (LL i = 1; i <= n; i++) {
            if (k + m < i) {
                x = (i - k + 1) / m - 1;
                if (i + x < n) {
                    i = i + x;
                    k = (k + m * x);
                } else {
                    k = k + m * (n - i);
                    i = n;
                }
            }
            k = (k + m - 1) % i + 1;
        }
    }
    return k;//返回最后一人的位置
}
const int N = 500005;
struct SegTree {
    int l, r;
    int sum;//sum 表示在[a,b]中还在环中的人数
} tree[3*N];
void build(int a, int b, int p) {
    tree[p].l = a;
    tree[p].r = b;
    tree[p].sum = b - a + 1;
    if (a == b) return;
    int mid = (a + b) >> 1;
    build(a, mid, 2*p);
    build(mid + 1, b, 2*p + 1);
}
int sum(int a, int b, int p) {
    if (tree[p].l > b || tree[p].r < a)
        return 0;
    if (a <= tree[p].l && tree[p].r <= b)
        return tree[p].sum;
    else
        return sum(a, b, 2*p) + sum(a, b, 2*p + 1);
}
int get_kth(int k, int p) { //获得第 k 个人的相对坐标
    tree[p].sum--;
    if (tree[p].l == tree[p].r)
        return tree[p].l;
    if (k <= tree[2 * p].sum)
        return get_kth(k, 2 * p);
    else
        return get_kth(k - tree[2 * p].sum, 2 * p + 1);
}
int main () {
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF) {
        printf("the last one is: %lld\n", Josephus(n, m, 1));
        build(0, n - 1, 1);
        int p = n, now = -1;
        for (int i = 0; i < n; i++) {
```

```
            int s = sum(now + 1, n - 1, 1), t = (m - 1) % p + 1;
            if (t <= s)
                now = get_kth((p - s + t - 1) % p + 1, 1);
            else
                now = get_kth((t - s - 1) % p + 1, 1);
            printf("#%d\n", now + 1);
            if (--p == 0)
                break;
        }
    }
    return 0;
}
```

POLYA 计数

```
#include<iostream>
#include<cmath>
using namespace std;
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
//如果结果过大、double 精度不够就换用 int64 或者 long long，并且自己写对应整数的 pow 比较保险 或高精度
//用 c 种颜色染 n 个珠子，构成不同项链的种数。
double polya1(int c, int n) { //旋转和翻转视为相同
    double t = 0;
    for (int i = 0; i < n; i++)
        t += pow((double)c, gcd(i, n));
    if (n % 2) {
        for (int i = 0; i < n; i++)
            t += pow((double)c, n / 2 + 1);
    } else {
        for (int i = 0; i < n / 2; i++)
            t += pow((double)c, n / 2) + pow((double)c, n / 2 + 1);
    }
    return t / (2 * n);
}
double polya2(int c, int n) { //旋转视为相同，翻转为异
    int bj[10000];
    int k, x, y;
    double t = 0;
    memset(bj, 0, sizeof(bj));
    for (int i = 0; i <= n - 1; i++) {
        for (int j = x = y = 0; j <= n - 1; j++)
            if (!bj[(i + j) % n])
                for (x++, k = (i + j) % n; !bj[k]; k = (i + k) % n)
                    bj[k] = true;
        t += pow(c, x);
    }
    return t / n;
}
double polya3(int c, int n) { //翻转视为相同，旋转为异
    int x = n / 2;
    if (n % 2) x++;
    return (pow(c, n) + pow(c, x)) / 2;
}
int main() {
    int c, n;
    while (scanf("%d%d", &c, &n) && c && n)
        printf("%d\n", (int)polya1(c, n));
    return 0;
}
```

循环节 TORTOISEHARE 算法

```
//TortoiseHare 算法找循环节 时间复杂度 O(n)，空间复杂度 O(1)
//SGU 455
//传入起点位置，保证值 f(x0) 值已经算出
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
int A, B, C;
int next(int x) {
    if (x == 0) return 1;
    return (A * x + x % B) % C;
}
pair<int, int> cycle_finding(int x) {//传入起点位置，返回循环节起点和长度
    int start = x, length = 1;
    //起始令 l = f(x0), r = f(f(x0))
    int l = next(next(x)), r = next(l);
    while (l != r) {
        l = next(l), r = next(next(r));
    }
    l = next(x);
    while (l != r) {
        l = next(l), r = next(r), start++;
    }
    r = next(l);
    while (l != r) {
        r = next(r), length++;
    }
    return make_pair(start, length);
}
int main() {
    while (scanf("%d%d%d", &A, &B, &C) != EOF) {
        int f = 1;
        for (int i = 0; i < 20; i++) {
            printf("%d ", f);
            f = next(f);
        }
        puts("");
        pair<int, int> ret = cycle_finding(0);
        printf("%d %d\n", ret.first, ret.second);
    }
}
```

SUM OF POWER 前 N 个正整数的 K 次幂之和

```
//求前 N 个正整数的 K 次幂之和，也叫伯努利幂之和
//k^2 的算法：http://hi.baidu.com/unber/blog/item/e1e4d911b85ee217203f2e69.html
//sigma(i=1..n)i^k = sigma(i=1..k)(sigma(j=0..i-1)(-1)^j*C(i, j)*(i - j)^(k + 1))*C(n + 1, i + 1);
//1^k+2^k+...+n^k = sigma(P[k][i] * C[n + 1][i + 1]);//i = 1, 2, ...k;
//              = sigma((sigma((-1)^j * C[i][j] * (i - j)^k)) * C[n + 1][i + 1]);//i = 1..k, j = 0..i
- 1;
//其中 C[i][j] 为组合数，C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
//P[i][0] = 0;  P[i][1] = 1;
//P[i][i] = 1 * 2 * ... * i = i !;
//P[i][j] = j * (P[i - 1][j - 1] + P[i - 1][j]); (1 <= j < i )
//ZJUT1706[奇数项加偶数项减]
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
```

```
const int MAXP = 63;
const int MOD = 1000000007;
LL c[MAXP][MAXP];
LL p[MAXP][MAXP];

void init() {
    for (int i = 0; i < MAXP; i++) {
        for (int j = c[i][0] = 1; j <= i; j++) {
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % MOD;
        }
    }
    p[0][1] = 1;
    for (int i = 0; i < MAXP; i++) {
        for (int j = 1; j <= i; j++) {
            p[i][j] = (j * (p[i - 1][j - 1] + p[i - 1][j])) % MOD;
        }
    }
}

LL exgcd(LL a, LL b, LL &x, LL &y) { //扩展欧几里得定理:解 ax+by==1
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
LL mod(LL x, LL n) {
    return (x % n + n) % n;
}
LL inv(LL a,LL n) {
    LL x, y;
    return exgcd(a, n, x, y) == 1 ? mod(x, n) : -1;
}

LL power(LL a, LL b, LL c) { // a^b % c
    LL res = 1;
    for (; b; b >>= 1) {
        if (b & 1) res = (res * a) % c;
        a = (a * a) % c;
    }
    return res;
}

LL powersum(int k, int n) {
    LL ret = 0;
    for (int i = 1; i <= k; i++) {
        LL now = p[k][i];
        /*for (int j = 0; j < i; j++) {
            LL tmp = (j & 1) ? -1 : 1;
            tmp = (tmp * c[i][j]) % MOD;
            tmp = (tmp * power(i - j, k, MOD)) % MOD;
            now = (now + tmp) % MOD;
        }*/
        for (int j = n - i + 1; j <= n + 1; j++) {
            now = (now * j) % MOD;
        }
        for (int j = 2; j <= i + 1; j++) {
            now = (now * inv(j, MOD)) % MOD;
        }
        ret = (ret + now) % MOD;
    }
    return ret;
}
```

```
int main() {
    init();
    int n, k, cas;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%d%d", &n, &k);
        LL ans = (-powersum(k, n) + (powersum(k, n / 2) * power(2, k + 1, MOD)) % MOD) % MOD;
        while (ans < 0) ans += MOD;
        printf("%lld\n", ans);
    }
}

//0 < n <= 10^17, 0 < p <= 10^17, m 为奇素数，且 2 < m < 10^8 mod 为素数，可以利用循环节处理 XMU1231
#include <iostream>
using namespace std;
typedef long long LL;
//调用一下快速幂 power
int main() {
    LL n, p, m;
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%lld%lld%lld", &n, &p ,&m);
        p %= m - 1;
        int len = n - (n / m) * m;
        LL ans = 0, tmp = 0;
        for (int i = 1; i <= len; i++) {
            tmp = (tmp + power(i, p, m)) % m;
        }
        if (n / m) {
            ans = tmp;
            for (int i = len + 1; i <= m; i++) {
                ans = (ans + power(i, p, m)) % m;
            }
        }
        ans = (ans * (n / m)) % m;
        ans = (ans + tmp) % m;
        //while (ans < 0) ans += m;
        printf("%lld\n", ans);
    }
}

//用矩阵递推方法
//二项式定理 n^k - (n - 1)^k = C(k, 0) * n^0 + C(k, 1) * n^1 + C(k, 2) * n^2 + …… + C(k, k - 1) * n^(k
- 1)), 利用上式矩阵递推,奇偶项可用 get(n / 2, k) * 2^(k + 1) - get(n, k)计算
const int MAXC = 65;
LL c[MAXC + 1][MAXC + 1];
void init() {
    for (int i = 0; i <= MAXC; i++) {
        for (int j = c[i][0] = 1; j <= i; j++) {
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % MOD;
        }
    }
}

void getA(int k) {
    memset(A.mat, 0, sizeof(Mat));
    for (int i = 0; i <= k; i++) {
        for (int j = 0; j <= i; j++) {
            A.mat[i][j] = c[i][j];
        }
    }
    A.mat[k + 1][k] = 1;
    A.mat[k + 1][k + 1] = 1;
    for (int i = 0; i <= k; i++) vec[i] = 1;
```

```
    vec[k + 1] = 0;
}

LL sum(int nn, int k) {
    n = k + 2;//矩阵规格
    getA(k);
    ans = A^nn;
    LL sum = 0;
    for (int i = 0; i < n; i++)
        sum = (sum + ans.mat[n - 1][i] * vec[i]) % MOD;
    return sum;
}
```

组合函数

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int MAXN = 252;

//组合数 二项式系数
//C[n, m]是从 n 个不同元素中取出 m 个元素的方案数
//C(n, m) = P(n, m) / m! = n! / ((n - m)! * m!)
//C(n, m) = C(n, n - m) = C(n - 1, m - 1) + C(n - 1, M)
LL C[MAXN + 1][MAXN + 1];
void Combination(int n) {
    for (int i = 0; i <= n; i++) {
        for (int j = C[i][0] = 1; j <= i; j++) {
            C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
        }
    }
}

//阶乘
//Fact[n] = n! = n * (n - 1) * (n - 2) * ... * 1
LL Fact[MAXN + 1];
void Factorial(int n) {
    Fact[0] = 1;
    for (int i = 1; i <= n; i++) {
        Fact[i] = Fact[i - 1] * i;
    }
}

//排列数
//P[n, m]是从 n 个不同元素中取出 m 个元素按照一定的顺序排成一列的方案数
//P(n, m) = C(n, m) * m! = n! / (n - m)!
LL P[MAXN + 1][MAXN + 1];
void Permutation(int n) {
    for (int i = 0; i <= n; i++) {
        for (int j = P[i][0] = 1; j <= i; j++) {
            P[i][j] = P[i - 1][j] + P[i - 1][j - 1];
        }
    }
    Factorial(n);
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= i; j++) {
            P[i][j] *= Fact[j];
        }
    }
}

//错位排列数
```

```
//D[n]表示 n 个相异的元素排成一排 a1, a2, ..., an,且 ai(i = 1, 2, ..., n)不在第 i 位的排列个数
//由容斥原理 D[n]=n! - |A1∪A2∪...∪An| = n! - C(n, 1)(n - 1)! + C(n, 2)(n - 2)! - ... +(-1)^n*C(n, n)*0!
//D[n] = n!(1 - 1 / 1! + 1 / 2! - 1 / 3! + ... + (-1)^n * 1 / n!)
//D[n] = (n - 1) * (D[n - 1] + D[n - 2]) = n * d[n - 1] + (-1)^n, D[1] = 0, D[2] = 1
//近似公式 D[n] = [n! / e] (n odd), f(n)=[n! / e] + 1 (n even);[]为取整函数
//全错位排列数前几项为： 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841
LL D[MAXN + 1];
void Derangement(int n) {
    D[1] = 0, D[2] = 1;
    for (int i = 3; i <= n; i++) {
        D[i] = (i - 1) * (D[i - 1] + D[i - 2]);
    }
}

//第一类斯特灵数
//StirlingS1[n, m]是有正负的，其绝对值是包含 n 个元素的集合分成 m 个环排列的方法数目
//又即有 n 个人分成 m 组，每组内再按特定顺序围圈的分组方法的数目
//S1(n, m) = (n - 1) * S1(n - 1, m) + S1(n - 1, m - 1)
//S1(n, 0) = 0, S1(1, 1) = 1, S1(4, 2) = 11
//..n:...total...m=..1......2......3.....4.....5.....6....7...8...9
//..1:.......1.......1
//..2:.......2.......1......1
//..3:.......6.......2......3......1
//..4:......24.......6.....11......6.....1
//..5:.....120......24.....50.....35....10.....1
//..6:.....720.....120....274....225....85....15.....1
//..7:....5040.....720...1764...1624...735...175....21....1
//..8:...40320....5040..13068..13132..6769..1960...322...28...1
//..9:..362880...40320.109584.118124.67284.22449..4536..546..36...1
LL S1[MAXN + 1][MAXN + 1];
void StirlingS1(int n) {
    S1[0][0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            S1[i][j] = (i - 1) * S1[i - 1][j] + S1[i - 1][j - 1];
        }
    }
}

//第二类斯特林数及贝尔数
//StirlingS2[n, m]给出把 n 个元素的集合分到 m 个 非空子集的分法的数目 也叫集合的分拆
//B[n]是基数为 n 的集合的划分方法的数目
//S2(n, m) = m * S2(n - 1, m) + S2(n - 1, m - 1) (n > 1, m > 1)
//S2(n, 0) = S2(n, k) = 0 (k > n), S2(n, 1) = S2(n, n) = 1
//S2(4, 2) = 7, 前几个贝尔数是：1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597
LL S2[MAXN + 1][MAXN + 1];
LL B[MAXN + 1];
void StirlingS2(int n) {
    S2[0][0] = B[0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            S2[i][j] = j * S2[i - 1][j] + S2[i - 1][j - 1];
            B[i] += S2[i][j];
        }
    }
}

//卡特兰数
//Catalan[n]是 n 个+1 和 n 个-1 构成 2n 项，其部分和满足 a1 + a2 + ... + ak >= 0 的序列个数
//Catalan[n] = 1 / (n + 1) * C(2 * n, n) = C(2 * n, n) - C(2 * n, n + 1)
//Catalan[n] = (4 * n - 2) / (n + 1) * C[n - 1], Catalan[1] = 1
//Catalan[n] = sigma(catalan[i] * catalan[n - 1 - i]) (0 <= i < n)
//n 个节点的二叉树的所有可能形态数 n 个非叶节点的满二叉树(补上叶子)的形态数 n 层的阶梯切割为 n 个矩形的切法数
//凸 n+2 边形进行三角形分割(只连接顶点对形成 n 个三角形) n 个数入栈后的出栈的排列总数 2n 大小集合的不交叉划分的数目
//对于一个 n*n 的网格，每次向右或向上移动一格，从左下角到右上角的所有在副对角线右下方的路径总数
//前几个 Catalan 数是：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900
```

```
//广义卡特兰数
//求由 n 个 1、m 个 0 组成，并且任意前缀中 1 的个数不少于 0 的个数的字符串的个数是 C(n + m, n) - C(n + m, n + 1)
//从(0, 0)点走到(m, n)点的方案是 C(n + m, n) - C(n + m, n + 1) = (n - m + 1) / (n + 1) * C(n + m, m)
LL catalan[MAXN + 1];
void Catalan(int n) {
    catalan[0] = 1;
    for (int i = 1; i <= n; i++) {
        catalan[i] = 0;
        for (int j = 0; j < i; j++) {
            catalan[i] += catalan[j] * catalan[i - 1 - j];
        }
    }
}


//分拆函数 P
//PartP[n][m]表示数 n 分拆最大部分为 m 的分拆个数(n >= m)
//PartP[i][j] = sigma(PartP[i - j][k]);(1 <= k <= j)
//PartP[i - 1][j - 1] = sigma(PartP[(i - 1) - (j - 1)][k]) = sigma(PartP[i - j][k]);(1 <= k <= j -
1)
//联立两式有 Part[i][j] = PartP[i - 1][j - 1] + PartP[i - j][j];
//n 的 m 部分分拆的个数 P[n][m] = n 的最大分部是 m 的分拆个数(由 Ferrers 图共轭分拆可知)
//n 的至多有 m 个分部的分拆的个数 = n 的最大分部至多是 m 的分拆个数
//PartitionP[n]是正整数 n 的不同(排序后)分拆的方案数目
//n 的一个 k 部分拆是把 n 表示成 k 个正整数之和 n = n1 + n2+ ... + nk(n1 <= n2 <= ... <= nk)
//分拆函数 P 前几项是：1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, 135, 176, 231, 297
//PartitionP[n] = sigma(PartP[n][i]);(1 <= i <= n)
LL PartP[MAXN + 1][MAXN + 1];
LL PartitionP[MAXN + 1];
void Partition_P(int n) {
    PartP[0][0] = PartitionP[0] = 1;
    for (int i = 1; i <= n; i++) {
        PartitionP[i] = 0;
        for (int j = 1; j <= i; j++) {
            PartP[i][j] = PartP[i - 1][j - 1] + PartP[i - j][j];
            PartitionP[i] += PartP[i][j];
        }
    }
}

//分拆函数 Q
//PartQ[n][m]表示数 n 分拆为 m 个不同的正整数的方案数目
//PartQ[i][j] = PartP[i - C[j][2]][j];PartP 是分拆函数 P C 是组合数
//PartitionQ[n]表示把整数 n 拆分为不同的正整数的和的方案数目
//PartitionQ[n]也表示把 n 拆分为奇数个正整数的方案数目
//PartitionQ[n]还表示其生成函数 G[x] = Product_{m=1..inf} (1 + x^m)的展开式 x^n 项系数;
//分拆函数 Q 前几项是：1, 1, 1, 2, 2, 3, 4, 5, 6, 8, 10, 12, 15, 18, 22, 27, 32, 38, 46, 54, 64, 76, 89
LL PartQ[MAXN + 1][MAXN + 1];
LL PartitionQ[MAXN + 1];
void Partition_Q(int n) {
    Partition_P(n);
    for (int i = 0; i <= n; i++) {
        PartitionQ[i] = 0;
        for (int j = 0; j <= i; j++) {
            if (i - j * (j - 1) / 2 < 0) break;
            PartQ[i][j] = PartP[i - j * (j - 1) / 2][j];
            PartitionQ[i] += PartQ[i][j];
        }
    }
}

//自然数前 n 项 k 次方和
//1^k+2^k+...+n^k = sigma(P[k][i] * C[n + 1][i + 1]);//i = 1, 2, ...k;
//              = sigma((sigma((-1)^j * C[i][j] * (i - j)^k)) * C[n + 1][i + 1]);//i = 1..k, j = 0..i
- 1;
//其中 C[i][j]为组合数，C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
```

```
//P[i][0] = 0;  P[i][1] = 1;
//P[i][i] = 1 * 2 * ... * i = i !;
//P[i][j] = j * (P[i - 1][j - 1] + P[i - 1][j]); (1 <= j <= i)
LL PS[MAXN + 1][MAXN + 1];
void PowerSum(int n) {
    Combination(n);
    PS[0][1] = 1;
    for (int i = 0; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            PS[i][j] = (j * (PS[i - 1][j - 1] + PS[i - 1][j]));
        }
    }
    //完整的加上逆元、快速幂模块用公式求即可
}

//根号方程(sqrt(a) + sqrt(b))^(2n) + (sqrt(a) - sqrt(b))^(2n)
//递推 Z[n+1] = 2 * (a + b) * Z[n] - (a - b)^2 * z[n - 1], Z0 = 2, Z1 = 2 * (a + b)
int main() {
    Partition_Q(MAXN);
    int cas, n;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%d", &n);
        printf("%lld\n", PartitionQ[n]);
    }
}
```

其他数论常识

欧拉函数

```
int phi(int n) {
    int ret = n;
    for (int i = 2; (LL)i * i <= n; i++)
        if (n % i == 0) {
            ret -= ret / i;
            while (n % i == 0) n /= i;
        }
    if (n != 1) ret -= ret / n;
    return ret;
}
```

∑GCD(I,N)

```
//∑gcd(i,n),(1<=i<=n) = ∑phi(n/i)*i,i 是 n 的因子
LL SigmaGcd(int n) {
    LL sum = 0;
    int p = sqrt(double(n));
    for (int i = 1;i <= p;i++)
        if (n % i == 0) {
            sum += phi(n/i) * i;
            if (i * i != n)
                sum += phi(i) * n / i;
        }
    return sum;
}
```

与 N 互质的数之和

```
//1..n-1 中与 n 互质的数(gcd(n,i) == 1)的和 sum[n] = phi[n] * n / 2;
sum = ((phi(x) * x / 2) % MOD) % MOD;
```

与 N 互质的数的个数

```
int _dfs(vector<int> &p,int d,int x,int y){
    if (d==p.size()) return y/x;
    return _dfs(p,d+1,x,y) + _dfs(p,d+1,x * (-p[d]),y);
}
int cal(int n,int y){ //计算 1~y 中有几个 gcd(n,i)=1,容斥原理,复杂度 sqrt(n)+2^(p.size())
```

```
    vector<int> p;
    for (int i = 2;i * i <= n;i++) {
        if (n % i == 0) {
            p.push_back(i);
            while (n % i == 0) n /= i;
        }
    }
    if (n != 1) p.push back(n);
    return _dfs(p,0,1,y);
}
```

约数及素因子个数

```
//求 1..n 中每个 n 的约数及素因子个数
const int MAXN = 500000;
int c[MAXN+10];
//int 内最大 2095133040 是 1600 个
//long long 内最大 9200527969062830400 是 161280 个
void DivisorNum(int n, int c[]) { //约数个数
    for (int i = 1; i <= n; i++)
       c[i] = 1;
    for (int i = 2; i <= n; i++)
       for (int j = i; j <= n; j += i)
          c[j]++;
}
void FactorNum(int n, int c[]) { //素因子个数
    for (int i = 1; i <= n; i++)
       c[i] = 1;
    for (int i = 2; i <= n; i++) {
       if (c[i] != 1) continue;
       for (long long j = i; j <= n; j *= i) {
          for (int k = j; k <= n; k += j) {
             c[k]++;
          }
       }
    }
}
```

反素数

```
//反素数是 1..n 内的所有数里面因子数最多的
const int antiprime[] = {
    1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, 1680, 2520, 5040, 7560, 10080,
    15120, 20160, 25200, 27720, 45360, 50400, 55440, 83160, 110880, 166320, 221760, 277200,
    332640, 498960, 554400, 665280
};
const int factor[] = {
    1, 2, 3, 4, 6, 8, 9, 10, 12, 16, 18, 20, 24, 30, 32, 36, 40, 48, 60, 64, 72, 80, 84, 90, 96, 100,
108, 120, 128, 144, 160, 168, 180, 192, 200, 216, 224
};
//DP 方法- f[i][j] 表示前 i 个质数，因数个数为 j 的最小数
#include <cstdio>
#include <cstring>
using namespace std;
const long long RANGE = 1LL << 30;
int prime[] = {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
long long f[11][2000], ans[2000];
inline void zy(long long &a, long long b) {
    if (a == -1) a = b;
    else a = min(a, b);
}
inline long long mymin(long long a, long long b) {
    if (a == -1) return b; if (b == -1) return a;
    return min(a, b);
}
int main() {
    memset(f, 0xff, sizeof(f));
    f[0][1] = 1;
    for (int i = 0; i <= 10; ++i)
```

```
        for (int j = 1; j < 2000; ++j)
            if (f[i][j] != -1) {
                long long a = prime[i + 1];
                int k = 1;
                while (f[i][j] * a < RANGE && j * (k + 1) < 2000) {
                    zy(f[i + 1][j * (k + 1)], f[i][j] * a);
                    ++k;
                    a *= prime[i + 1];
                }
            }
    memset(ans, 0xff, sizeof(ans));
    for (int j = 1; j < 2000; ++j)
        for (int i = 0; i <= 10; ++i)
            ans[j] = mymin(ans[j], f[i][j]);
    int n;
    while (scanf("%d", &n) != EOF) {
        int j;
        for (j = 1999; j > 0; --j)
            if (ans[j] != -1 && ans[j] <= n)
                break;
        printf("%lld\n", ans[j]);
    }
    return 0;
}
```

N!中因子 X 的幂

```
//计算 n!(或 1..n)中因子 x 的幂
int getx(int n, int x) {
    return n ? n / x + getx(n / x, x) : 0;
}
```

N^N&N!最左位

```
//n^n&n!最左位
const double PI = acos(-1.0);
const double e = exp(1.0);
//n^n
        double a = n * log10(n);
        printf("%.0lf\n", pow(10.0, a - (long long)a));
        //n!
        double b = (log10(2 * PI) + log10(n)) / 2 + n * (log10(n) - log10(e));
        printf("%.0lf\n", pow(10.0, b - (long long)b));
```

万进制暴力求 N！

```
//返回 n!位数，结果为字符数组 s
int factorial(char s[], int n) {
    int a[10000];
    int c, m = 0;
    int len;
    a[0] = 1;
    for (int i = 1; i <= n; i++) {
        c = 0;
        for (int j = 0; j <= m; j++) {
            a[j] = a[j] * i + c;
            c = a[j] / 10000;
            a[j] = a[j] % 10000;
        }
        if (c > 0) a[++m] = c;
    }
    len = m * 4 + log10(a[m]) + 1;
    sprintf(s, "%d", a[m]);
    for (int i = m - 1; i >= 0; i--)
        sprintf(s + strlen(s), "%4.4d", a[i]);
    return len;//strlen(s);
}
```

N!中 0 的个数

```
const int Factor = 1005;
int zeros[Factor], a[2577] = {1};
void process(){
```

```
    double bitNum = 1.0;
    for (int n = 2, b = 0, e = 0; n <= Factor; n++){
        e = bitNum += log10(n * 1.0);
        while (a[b] == 0)
            b++;
        for (int j = b, and = 0; j < e; j++, and /= 10)
            a[j] = (and += n * a[j]) % 10;
        zeros[n] = count(a + b,a + e,0) + b;
    }
}
```

A^B 左边 L 位

```
/*真正的数字还是靠幂的小数部分 3,9,5->19683*/
int E(int a,int b,int l) {
    double ret = b * 1.0 * log10((double)a);
    ret -= floor(ret);
    ret = pow(10.0,ret) + 1e-9;
    while (--l)
        ret *= 10.0;
    return floor(ret);
}
```

格雷码

```
long long dec2gray(long long x) {
    return x ^ (x >> 1);
}
long long gray2dec(long long x) {
    long long y = x;
    while (x >>= 1) {
        y ^= x;
    }
    return y;
}
```

二进制

```
#include <iostream>
using namespace std;
int countbit(int n) { //Counting out the bits
    return n ? countbit(n & (n-1)) + 1 : 0;
}

inline int count_bits(int x) {
    x = (x & 0x55555555) + ((x & 0xaaaaaaaa) >> 1);
    x = (x & 0x33333333) + ((x & 0xcccccccc) >> 2);
    x = (x & 0x0f0f0f0f) + ((x & 0xf0f0f0f0) >> 4);
    x = (x & 0x00ff00ff) + ((x & 0xff00ff00) >> 8);
    x = (x & 0x0000ffff) + ((x & 0xffff0000) >> 16);
    return x;
}

int reverse(int x) { //Reversing the bits in an integer
    x = ((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1);
    x = ((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2);
    x = ((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4);
    x = ((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8);
    x = ((x & 0xffff0000) >> 16) | ((x & 0x0000ffff) << 16);
    return x;
}
void nCk(int n,int k) { //nCk 二进制方案枚举
    for (int comb = (1 << k) - 1;comb < 1 << n;) {
        printf("%d\n",comb);
        for (int j = 31;j >= 0;j--)
            printf("%d",comb >> j & 1);
        printf("\n");
        int x = comb & -comb, y = comb + x;
        if (!x) break;
        comb = ((comb & ~y) / x >> 1) | y;
    }
}
```

```
inline unsigned int Log2(unsigned int x) {
    unsigned int ret;
    __asm__ __volatile__("bsr %1, %%eax" : "=a"(ret) : "m"(x));
    return ret;
}
int main () {
    int x;
    scanf("%d",&x);
    printf("%d\n",reverse(x));
    int n = 9,k = 7;
    nCk(n,k);
}
```

二：图论算法

最小生成树 KRUSKAL

```
//最小生成树 Kruskal 复杂度 O(ElogE)
//次小生成树枚举删边
#include <iostream>
using namespace std;
const int MAXN = 101;
const int MAXM = MAXN * MAXN;
int n,m,Set[MAXN];
struct edge {
    int x,y,z;
}e[MAXM];
inline bool cmp(const edge& a,const edge& b) {
    return a.z < b.z;
}
void init_set() {
    memset(Set,-1,sizeof(Set));
}
int find_set(int x) {
    while (Set[x] >= 0) {
        //Set[x] = find_set(Set[x]);//路径压缩
        x = Set[x];
    }
    return x;
}
int union_set(int x,int y) {
    int fx = find_set(x);
    int fy = find_set(y);
    if (fx == fy)
        return 0;
    Set[fx] += Set[fy];
    Set[fy] = fx;
    return 1;
}
int kruskal(int n,int m,edge e[]) {
    init_set();
    sort(e,e + m,cmp);
    int j = 1,ans = 0;
    for (int i = 0;i < m && j < n;i++) {
        if (union_set(e[i].x,e[i].y)) {
            j++;
            ans += e[i].z;
        }
    }
    if (j == n) return ans;
    else return 0; //不存在最小生成树
}
int main() {
    while ((scanf("%d%d",&n,&m)!= EOF) && (n || m)) {
        for (int i = 0;i < m;i++)
            scanf("%d%d%d",&e[i].x,&e[i].y,&e[i].z);
```

```
        printf("%d\n",kruskal(n,m,e));
    }
}
```

最小生成树 PRIM

```
#include <iostream>
using namespace std;
const int MAXN = 201;
const int MAXV = 0x3F3F3F3F;
int n,m,t,x,y,z,c,n1,m1,n2,m2;
int dis[MAXN][MAXN],low[MAXN],near[MAXN];
int prim(int n) {
    int sum = 0;
    near[1] = -1;
    for(int i = 2;i <= n;i++) {
        low[i] = dis[1][i];//第 1 个点到其他点的代价
        near[i] = 1;//初始时,所有点的起点都是点 1
    }
    int k = 1;
    for(int i = 2;i <= n;i++) {
        int min = MAXV;//MAXV 一个很大的数
        for(int j = 2;j <= n;j++)
            if(near[j] >= 0 && low[j] < min) {
                min = low[j];//在 v 中找到最小的代价点
                k = j;
            }
        //printf("(%d %d)\n",near[k],k);//near[k]->起点,k->终点
        sum += low[k];
        near[k] = -1;//k 点归入 u 中
        for(int j = 2;j <= n;j++)
            if(near[j] >=0 && dis[k][j] < low[j]) {
                low[j] = dis[k][j];//以 k 点为起点进行新一轮的代价计算
                near[j] = k;
            }
    }
    for (int i = 1;i <= n;i++)
        if (near[i] >= 0)
            return -1;
    return sum;
}
int main() {
    while (scanf("%d%d",&n,&m) != EOF) {
        memset(dis,63,sizeof(dis));
        for (int i = 0;i < m;i++) {
            scanf("%d%d%d",&x,&y,&z);
            if (z < dis[x][y])
                dis[x][y] = dis[y][x] = z;
        }
        printf("%d\n",prim(n));
    }
}
```

次小生成树

```
//BJOI2010 次小生成树 URAL1416
//复杂度 O(NlogN + MlogM + M) 需要特判没有次小生成树的情况
//@http://www.cppblog.com/MatoNo1/archive/2011/08/04/149812.html
//@http://www.cppblog.com/MatoNo1/archive/2011/05/29/147627.aspx
#include <iostream>
#include <cstdio>
```

```cpp
#include <algorithm>
using namespace std;
#define re(i, n) for (int i = 0; i < n; i++)
#define re1(i, n) for (int i = 1; i <= n; i++)
const int MAXN = 100001, MAXM = 300001;
const long long INF = ~0Ull >> 2;
struct edge {
    int a, b, len;
    friend bool operator< (edge e0, edge e1) {
        return e0.len < e1.len;
    }
} E[MAXM];
struct edge0 {
    int a, b, id, pre, next;
} E0[MAXM + MAXM];
int n, m, m0, u[MAXN], pr[MAXN], No[MAXN], s[MAXM], Q[MAXN], fl[MAXN], _s;
long long mst_v = 0, res;
bool inmst[MAXM], vst[MAXN];
void init() {
    scanf("%d%d", &n, &m);
    re(i, m) scanf("%d%d%d", &E[i].a, &E[i].b, &E[i].len);
}
int find(int x) {
    int r = x, r0;
    while (u[r] > 0) r = u[r];
    while (u[x] > 0) {
        r0 = u[x];
        u[x] = r;
        x = r0;
    }
    return r;
}
void uni(int s1, int s2) {
    int tmp = u[s1] + u[s2];
    if (u[s1] > u[s2]) {
        u[s1] = s2;
        u[s2] = tmp;
    } else {
        u[s2] = s1;
        u[s1] = tmp;
    }
}
void init_d() {
    re1(i, n) {
        E0[i].a = i;
        E0[i].pre = E0[i].next = i;
    }
    if (n % 2) m0 = n + 1;
    else m0 = n + 2;
}
void add_edge(int a, int b, int id) {
    E0[m0].a = a, E0[m0].b = b, E0[m0].id = id;
    E0[m0].pre = E0[a].pre, E0[m0].next = a, E0[a].pre = m0, E0[E0[m0].pre].next = m0++;
    E0[m0].a = b, E0[m0].b = a, E0[m0].id = id;
    E0[m0].pre = E0[b].pre, E0[m0].next = b, E0[b].pre = m0, E0[E0[m0].pre].next = m0++;
}
bool prepare() {
    sort(E, E + m);
    re1(i, n) u[i] = -1;
    init_d();
    int s1, s2, z = 0;
    re(i, m) {
        s1 = find(E[i].a);
        s2 = find(E[i].b);
        if (s1 != s2) {
            z++;
```

```
            mst_v += E[i].len;
            add_edge(E[i].a, E[i].b, i);
            inmst[i] = 1;
            uni(s1, s2);
            if (z == n - 1) {
                break;
            }
        }
    }
    return z == n - 1;
}
void bfs() {
    re1(i, n) vst[i] = 0;
    Q[0] = 1;
    vst[1] = 1;
    int i, j;
    for (int front = 0, rear = 0; front <= rear; front++) {
        i = Q[front];
        for (int p = E0[i].next; p != i; p = E0[p].next) {
            j = E0[p].b;
            if (!vst[j]) {
                vst[j] = 1;
                Q[++rear] = j;
                pr[j] = i;
                No[j] = E0[p].id;
            }
        }
    }
}
int LCA(int x, int y) {
    while (x && y) {
        if (fl[x] == _s) return x;
        else fl[x] = _s;
        if (fl[y] == _s) return y;
        else fl[y] = _s;
        x = pr[x];
        y = pr[y];
    }
    if (x) while (1) if (fl[x] == _s) return x;
            else x = pr[x];
    else while (1) if (fl[y] == _s) return y;
            else y = pr[y];
}
void sol0(int a, int b, int l) {
    int p = LCA(a, b), p0, No0;
    while (a != p) {
        No0 = No[a];
        if (l >= E[No0].len && l - E[No0].len < s[No0]) {
            s[No0] = l - E[No0].len;
        }
        p0 = pr[a];
        pr[a] = p;
        a = p0;
    }
    while (b != p) {
        No0 = No[b];
        if (l >= E[No0].len && l - E[No0].len < s[No0]) {
            s[No0] = l - E[No0].len;
        }
        p0 = pr[b];
        pr[b] = p;
        b = p0;
    }
}
void solve() {
    pr[1] = 0;
```

```
    bfs();
    memset(s, 63, sizeof(s));
    re(i, m) if (!inmst[i]) {
        _s = i + 1;
        sol0(E[i].a, E[i].b, E[i].len);
    }
    res = INF;
    re(i, m) if (inmst[i] && s[i] < res) res = s[i];//附加条件 s[i] != 0 即严格次小
    res += mst_v;
}
void pri() {
    if (mst_v == -1 || res >= 0x3F3F3F3F) {
        res = -1;
    }
    printf("%lld\n", res);
}
int main() {
    init();
    if (!prepare()) mst_v = -1;
    else solve();
    pri();
    return 0;
}
```

部分点最小生成树 STEINER TREE

```
//Steiner Tree Problem
//dp[x][1 << m] 代表以 x 为根的树并且包含 state 里的点的最小费用
//HDU3311 复杂度 O(N*4^M) N 小可以 floyd
//给定 6 个基础点，和 1000 个辅助点，以及无向边若干，求一棵代价最小的生成子树，使得 6 个基础点连通。
#include <iostream>
#include <queue>
#include <cstring>
#include <cstdio>
using namespace std;
const int MAXN = 1100;
const int MAXM = 13000;
const int MAXV = 0x3F3F3F3F;
struct edge {
    int v, w, next;
} e[MAXM];
int head[MAXN], E;
queue<int> q;
int n, m, dp[1<<6][MAXN], dis[MAXN][MAXN], p[6];
bool vis[MAXN];
//dp[x][1<<m] 代表以 x 为根的树并且包含 state 里的点的最小费用
void init() {
    memset(head, -1, sizeof(head));
    E = 0;
}
inline void addedge(int u, int v, int w) {
    e[E].v = v; e[E].w = w; e[E].next = head[u]; head[u] = E++;
}
void SPFA(int now) {
    while (!q.empty()) q.pop();
    memset(dis[now], 63, sizeof(dis[now]));
    memset(vis, 0, sizeof(vis));
    q.push(now);
    vis[now] = 1;
    dis[now][now] = 0;
    int u, v;
    while (!q.empty()) {
        u = q.front(), q.pop();
        vis[u] = 0;
```

```
        for (int j = head[u]; j != -1; j = e[j].next) {
            v = e[j].v;
            if (dis[now][v] > dis[now][u] + e[j].w) {
                dis[now][v] = dis[now][u] + e[j].w;
                if (!vis[v]) {
                    q.push(v); vis[v] = 1;
                }
            }
        }
    }
}
int steiner(int n, int m, int p[]) { //0~n 共 n 个点里面选出 p 中的 m 个点
    memset(dp, 63, sizeof(dp));
    for (int i = 0; i < n; i++)
        SPFA(i);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            dp[1<<i][j] = dis[p[i]][j];
    for (int i = 1; i < (1 << m); i++)
        if (((i - 1) & i) != 0) {//二进制不止一个 1
            for (int j = 0; j < n; j++) {
                dp[i][j] = MAXV;
                for (int k = (i - 1) & i; k > 0; k = (k - 1) & i)//枚举子集
                    dp[i][j] = min(dp[i][j], dp[k][j] + dp[i^k][j]);
            }
            for (int j = 0; j < n; j++)
                for (int k = 0; k < n; k++)
                    dp[i][j] = min(dp[i][j], dp[i][k] + dis[k][j]);
        }
    return dp[(1<<m)-1][p[0]];
}
int main() {
    int n, m, l;
    while (scanf("%d%d%d", &n, &m, &l) != EOF) {
        init();
        int x, y, z;
        for (int i = 1, j; i < n + m + 1; i++) {
            scanf("%d", &j);
            addedge(0, i, j);
            addedge(i, 0, j);
        }
        while (l--) {
            scanf("%d%d%d", &x, &y, &z);
            addedge(x, y, z);
            addedge(y, x, z);
        }
        for (int i = 0; i <= n; i++) p[i] = i;
        printf("%d\n", steiner(n + m + 1, n + 1, p));
    }
    return 0;
}
```

最小树形图

```
//最小树形图,复杂度 O(VE),可以打印树的构造 by zjut_DD
//HDU4009 一个有向图,找一个根然后使最小树形图权值最小
#include <iostream>
typedef int type;
#define inf 0x7fffffff
#define maxn 1005
#define maxm 1000005
struct Edge {//入边表
    int iu, iv, ru, rv;//虚(imagine),实(real)  u->v
    type val;
```

```
    int next;
} E[2][maxm];
int l[2][maxn], e[2];
int ip[maxn], rp[maxn];//rp 是实际的点的父节点,用于输出怎么构造
int sign[maxn], ID[maxn];
type In[maxn];
void init(bool f = false) {//默认参数为第一次调用服务
    memset(l[f], -1, sizeof(l[f]));
    e[f] = 0;
}
inline void _ins(bool f, int iu, int iv, int ru, int rv, type val) {
    E[f][ e[f] ].iu = iu; E[f][ e[f] ].iv = iv; E[f][ e[f] ].ru = ru; E[f][ e[f] ].rv = rv;
    E[f][ e[f] ].val = val; E[f][ e[f] ].next = l[f][iv];
    l[f][iv] = e[f]++;
}
inline void insert(int u, int v, type val) {//第一次插入
    if (u == v) return;
    _ins(false, u, v, u, v, val);
}
bool Directed_MST(int root, int N, type &ret) { //0~N-1
    ret = 0;
    bool f = false;//从 f 到!f
    while ( true ) {
        //找最小入边
        for (int i = 0; i < N; i++) {
            if (i == root) continue;
            In[i] = inf;
            int ind;
            for (int p = l[f][i]; p >= 0; p = E[f][p].next) {
                //下面这个判断是基于最小编号考虑的,hdu2121
                if ((E[f][p].val < In[i]) || (
                          (E[f][p].val == In[i]) && (E[f][p].rv < E[f][ind].rv) ) ) {
                    In[i] = E[f][p].val;
                    ind = p;
                }
            }
            if (In[i] < inf) {
                ret += In[i];
                ip[ E[f][ind].iv ] = E[f][ind].iu;
                rp[ E[f][ind].rv ] = E[f][ind].ru;
            } else
                return false; //无解
        }
        //找环,重新标号
        memset(sign, -1, sizeof(sign));
        memset(ID, -1, sizeof(ID));
        int cnt = 0;
        for (int i = 0; i < N; i++) {
            int v = i;
            while (sign[v] == -1 && v != root) { //没搜到过 & 不是根
                sign[v] = i;
                v = ip[v];
            }
            if (sign[v] == i) {
                int u = v;
                do {
                    ID[v] = cnt;
                    v = ip[v];
                } while (v != u);
                cnt++;
            }
        }
        if ( cnt == 0 ) break;
        for (int i = 0; i < N; i++) if (ID[i] == -1) ID[i] = cnt++;
        //转移为下一幅图
        init(!f);
```

```
#define EE (E[f][p])
        for (int i = 0; i < N; i++) {
            for (int p = l[f][i]; p != -1; p = E[f][p].next) {
                if (ID[EE.iu] != ID[EE.iv]) {
                    _ins(!f, ID[EE.iu], ID[EE.iv], EE.ru, EE.rv, EE.val - In[EE.iv]);
                }
            }
        }
        N = cnt;
        root = ID[root];
        f = !f;
    }
    return true;
}//****************模板结束****************

int a[maxn], b[maxn], c[maxn];
inline type dist(int i, int j) {
    return abs(a[i] - a[j]) + abs(b[i] - b[j]) + abs(c[i] - c[j]);
}
int main() {
    int n, x, y, z;
    int T = 1;
    while (scanf("%d%d%d%d", &n, &x, &y, &z) != EOF) {
        if (n == 0 && x == 0 && y == 0 && z == 0) break;
        init();
        for (int i = 1; i <= n; ++i) scanf("%d%d%d", a + i, b + i, c + i);
        for (int i = 1; i <= n; ++i) {
            int k;
            scanf("%d", &k);
            insert(0, i, x * c[i]);
            for (int j = 0; j < k; ++j) {
                int d;
                scanf("%d", &d);
                if (i != d) {
                    if (c[d] <= c[i]) {
                        if (c[d] * x > y * dist(i, d))
                            insert(i, d, y * dist(i, d));
                    } else {

if (c[d] * x > y * dist(i, d) + z)
                            insert(i, d, y * dist(i, d) + z);
                    }
                }
            }
        }
        type ret;
        Directed_MST(0, n + 1, ret);
        printf("%d\n", ret);
    }
    return 0;
}
```

单源最短路径 SPFA

```
/*
SPFA 差分约束是<=号 某些情况下栈更好
a[i] - a[j] <= x; -> addedge(j,i,x);
a[i] - a[j] >= y; -> addedge(i,j,-y);
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
```

```cpp
#include <stack>
using namespace std;
const int MAXN = 500, MAXM = MAXN * MAXN, MAXV = 0x3F3F3F3F;
int n, m, edge;//偶原奇反
bool vis[MAXN];
int dis[MAXN], head[MAXN], cnt[MAXN];
int to[MAXM], next[MAXM], cost[MAXM];
queue<int> q;//int que[100000];//stack<int> q;
inline void addedge(int u, int v, int c) { //添加一条 u->v 权为 c 的边
    to[edge] = v; cost[edge] = c; next[edge] = head[u]; head[u] = edge++;
}
void init() {
    memset(head, -1, sizeof(head));
    edge = 0;//边数清零
    while (!q.empty()) q.pop();
}

int SPFA(int src, int dst) { //,int rev
    memset(vis, 0, sizeof(vis));
    memset(cnt, 0, sizeof(cnt));//入队次数
    memset(dis, 63, sizeof(dis));
    //for (int i = 0;i <= n;i++) dis[i] = MAXV;//防 TLE
    //int h = 0,t = 0;
    dis[src] = 0;
    vis[src] = true;
    ++cnt[src];
    q.push(src);//que[t++] = src;
    while (!q.empty()) {//while (h < t) {
        int u = q.front();//que[h++];
        q.pop();
        vis[u] = false;
        for (int i = head[u]; i != -1; i = next[i]) {
            //if ((i & 1) != rev) continue;//求逆置
            int v = to[i];
            if (dis[u] + cost[i] < dis[v]) {
                dis[v] = dis[u] + cost[i];
                if (!vis[v]) {
                    q.push(v);//que[t++] = v;
                    vis[v] = true;
                    if (++cnt[v] > n)
                        return -MAXV;//入队次数>n,存在负权回路
                }
            }
        }
    }
    if (dis[dst] == MAXV)
        return MAXV;//src 无法到达 dst
    return dis[dst];
}

int main() {
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        init();
        scanf("%d%d", &n, &m);
        for (int i = 0; i < m; i++) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            addedge(x, y, z);
            //addedge(y,x,z);//无向图或有向图求其他所有点到 src 的距离
        }
        printf("%d\n", SPFA(1, n));
    }
}
```

单源最短路径 DIJKSTRA

```
//HDU2544
#include <iostream>
using namespace std;
const int MAXN = 105;
const int INF = 0x3F3F3F3F;
int d[MAXN], dis[MAXN][MAXN];
void dijkstra(int s, int n, int d[]) {
    typedef pair<int, int> type;
    bool done[MAXN];
    memset(done, 0, sizeof(done));
    for (int i = 0; i < n; i++) d[i] = INF;
    d[s] = 0;
    priority_queue<type, vector<type>, greater<type> > q;
    q.push(make_pair(d[s], s));
    while (!q.empty()) {
        int x = q.top().second;
        q.pop();
        if (done[x]) continue;
        done[x] = 1;
        for (int i = 0; i < n; i++) {
            if (d[i] > d[x] + dis[x][i]) {
                d[i] = d[x] + dis[x][i];
                q.push(make_pair(d[i], i));
            }
        }
    }
}
int main() {
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF && (n || m)) {
        memset(dis, 63, sizeof(dis));
        for (int i = 0; i < m; i++) {
            int x, y, z;
            scanf("%d%d", &x, &y);
            x--, y--;
            scanf("%d", &z);
            if (z < dis[x][y]) dis[x][y] = dis[y][x] = z;
        }
        dijkstra(0, n, d);
        printf("%d\n", d[n - 1]);
    }
}
```

单源最短路径 DIJKSTRA[边表]

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
const int MAXN = 50011, MAXM = 2 * 50011;
const int INF = 0x3F3F3F3F;
int d[MAXN], edge, head[MAXN];
int to[MAXM], next[MAXM], cost[MAXM];
void init() {
    memset(head, -1, sizeof(head));
    edge = 0;//边数清零
}
inline void addedge(int u, int v, int c) { //添加一条 u->v 权为 c 的边
```

```
    to[edge] = v; cost[edge] = c; next[edge] = head[u]; head[u] = edge++;
}
void dijkstra(int s, int n, int d[]) {
    typedef pair<int, int> type;
    bool done[MAXN];
    memset(done, 0, sizeof(done));
    for (int i = 0; i < n; i++) d[i] = INF;
    d[s] = 0;
    priority_queue<type, vector<type>, greater<type> > q;
    q.push(make_pair(d[s], s));
    while (!q.empty()) {
        int x = q.top().second;
        q.pop();
        if (done[x]) continue;
        done[x] = 1;
        //for (int i = 0; i < n; i++) {
        for (int i = head[x]; i != -1; i = next[i]) {
            int v = to[i];
            if (d[v] > d[x] + cost[i]) {
                d[v] = d[x] + cost[i];
                q.push(make_pair(d[v], v));
            }
        }
    }
}
int main() {
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        for (int i = 0; i < m; i++) {
            int x, y, z;
            scanf("%d%d", &x, &y);
            x--, y--;
            scanf("%d", &z);
            addedge(x, y, z), addedge(y, x, z);
        }
        dijkstra(0, n, d);
        printf("%d\n", d[n - 1]);
    }
}
```

全源最短路径 FLOYD

```
//记得初始化 dis 数组为-1
void floyd(int dis[][MAXN], int n) {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            if (i == k) continue;
            for (int j = 0; j < n; j++) {
                if (i == j || j == k) continue;
                if (dis[i][k] == -1 || dis[k][j] == -1) continue;
                if (dis[i][j] == -1 || dis[i][k] + dis[k][j] < dis[i][j])
                    dis[i][j] = dis[i][k] + dis[k][j];
            }
        }
    }
}
```

最小环

```
//可以求最短路的时候存中间点,求最小环的时候存起始和末尾点,递归输出方案//
```

```
int dist[MAXN][MAXN], mat[MAXN][MAXN], ans;
int mincycle(int mat[][MAXN], int n) {
    int dist[n][n], ans = MAXV;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            dist[i][j] = mat[i][j];
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= k - 1; i++)
            for (int j = i + 1; j <= k - 1; j++)
                ans = min(ans, dist[i][j] + mat[i][k] + mat[k][j]);//存起始和末尾点
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
    }
    return ans;//MAXV 则不存在
}
```

最短路次短路计数

```
//HDU1688 最短路+次短路计数
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>
#include <vector>
using namespace std;
const int MAXN = 1001;
const int INF = 0x3F3F3F3F;
int dp[MAXN][2], cnt[MAXN][2];// 最大次大
vector<pair<int, int> > G[MAXN];
void dijkstra(int s, int n, int dp[][2]) {
    typedef pair<int, int> type;
    bool done[MAXN << 1];
    for (int i = 0; i < n; i++) {
        done[i] = done[i + MAXN] = 0;
        dp[i][0] = dp[i][1] = INF;
        cnt[i][0] = cnt[i][1] = 0;
    }
    dp[s][0] = 0;
    cnt[s][0] = 1;
    priority_queue<type, vector<type>, greater<type> > q;
    q.push(make_pair(dp[s][0], s));
    while (!q.empty()) {
        int u = q.top().second, f = u / MAXN;
        q.pop();
        if (done[u]) continue;
        done[u] = 1;
        if (f) u -= MAXN;
        for (int k = 0; k < G[u].size(); k++) {
            int v = G[u][k].first, w = G[u][k].second;
            if (dp[v][0] > dp[u][f] + w) {
                dp[v][1] = dp[v][0];
                cnt[v][1] = cnt[v][0];
                q.push(make_pair(dp[v][1], v + MAXN));
                dp[v][0] = dp[u][f] + w;
                cnt[v][0] = cnt[u][f];
                q.push(make_pair(dp[v][0], v));
            } else if (dp[v][0] == dp[u][f] + w) {
                cnt[v][0] += cnt[u][f];
            } else if (dp[v][1] > dp[u][f] + w) {
                dp[v][1] = dp[u][f] + w;
                cnt[v][1] = cnt[u][f];
                q.push(make_pair(dp[v][1], v + MAXN));
```

```
            } else if (dp[v][1] == dp[u][f] + w) {
                cnt[v][1] += cnt[u][f];
            }
        }
    }
}
int main() {
    int cas, n, m;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%d%d", &n, &m);
        for (int i = 0; i < n; ++i) G[i].clear();
        for (int i = 0; i < m; i++) {
            int x, y, z;
            scanf("%d%d", &x, &y);
            x--, y--;
            scanf("%d", &z);
            G[x].push_back(make_pair(y, z));
        }
        int s, t;
        scanf("%d%d", &s, &t);
        s--, t--;
        dijkstra(s, n, dp);
        if (dp[t][1] - 1 == dp[t][0]) {
            cnt[t][0] += cnt[t][1];
        }
        printf("%d\n", cnt[t][0]);
    }
}
```

拓扑排序 TOPLOGICAL

```
//拓扑排序 默认点 1..n 点数 top 答案在 topo 里
bool topsort(int mat[][MAXN], int n) { //拓扑排序
    while (top < n) {
        int now = MAXV;
        for (int i = 1; i <= n; i++)//找到所有入度为零且未被使用(或是未知)的点中最小的点
            if (in[i] == 0 && !v[i])
                now = min(now, i);
        if (now == MAXV) return false;
        v[now] = 1;//标记使用
        topo[top++] = now;
        for (int i = 1;i <= n;i++)
            if (mat[now][i])
                in[i]--;//对所有 now->i,i 的入度减一
    }
    return true;
}
```

欧拉路

```
/*
欧拉路:
如果给定无孤立结点图 G,若存在一条路,经过图中每边一次且仅一次,这条路称为欧拉路;
如果给定无孤立结点图 G,若存在一条回路,经过图中每边一次且仅一次,那么该回路称为欧拉回路.
存在欧拉回路的图,称为欧拉图.
对于无向图 G,具有一条欧拉路,当且仅当 G 是连通的,且有零个或两个奇数度结点.
且有零个奇数度结点,存在欧拉回路;有两个奇数度结点,存在欧拉路.
判断无向图 G 是否连通,可以从任意结点出发,进行深度优先遍历,如果可以遍历到所有点,说明图 G 连通,否则图 G 不连通.
对于有向图 G,具有一条单向欧拉路,当且仅当 G 是连通的,且每个结点入度等于出度,或者,
除两个结点外,每个结点的入度等于出度,而这两个结点满足,一个结点的入度比出度大 1,
```

另一个结点的入度比出度小 1.
判断有向图 G 是否连通,可以某一结点出发,进行深度优先遍历,如果存在一点作为初始结点
可以遍历到所有点,说明,图 G 连通,否则,图 G 不连通.
汉密尔顿路
给定图 G,若存在一条路,经过图中每个结点恰好一次,这条路称作汉密尔顿路.
给定图 G,若存在一条回路,经过图中每个结点恰好一次,这条回路称作汉密尔顿回路.
具有汉密尔顿回路的图,称作汉密尔顿图.
定理若图 G=<V,E>具有汉密尔顿回路,则对于结点集 V 的每个非空子集 S,均有 W(G-S)<=|S|成立.
其中 W(G-S)是 G-S 的连通分支数,|S|表示集合 S 的元素数.
定理设 G 是具有 n 个结点的简单图,如果 G 中每一对结点度数之和大于等于 n-1,则在 G 中存在一条汉密尔顿路.
定理设 G 是具有 n 个结点的简单图,如果 G 中每一对结点度数之和大于等于 n,则在 G 中存在一条汉密尔顿回路.
判断一个图是否存在欧拉回路(每条边只遍历一次，并且回到出发点的路径)，在以下三种情况中有三种不同的算法:
一．无向图
每个顶点的度数都是偶数，则存在欧拉回路.
二．纯有向图(所有边都是单向的)
每个顶点的入度都等于出度，则存在欧拉回路.
三．混合欧拉回路图(有向边和无向边同时存在)
混合图欧拉回路采用网络流解法，看混合图是否能够转化为纯有向图(即第二种情况)．把该图中的无向边随便定向，计算每个点的入度和出度．如果有某个点出入度之差为奇数，那么肯定不存在欧拉回路．因为欧拉回路要求每点的入度 = 出度，意味着"总度数为偶数"．存在奇数点比不能有欧拉回路.
现在每个点入度和出度之差均为偶数，那么将该偶数除以 2 得到 x．也就是说，对于每一个点，只要将 x 条边改变方向(入>出就是变入，出>入就是变出)．就能保证出 = 入．如果每个点都是出 = 入，那么很明显，该图就存在欧拉回路.
现在的问题就变成了：该改变那些边，可以让每个点出 = 入？构造网络流模型．首先，有向边是不能改变方向的，要之无用，删除．一开始不是把无向边定向了吗？定的什么方向，就把网络构建成那个样子，边上的容量上限为 1．添加 s 和 t 点，对于入 > 出的点 u，连接边(u, t)，容量为 x，对于出 > 入的点 v，连接边(s, v)，容量为 x(注意对于不同点 x 不同)．之后，查看是否有满流的分配．有就是能有欧拉回路，没有就是不构成欧拉回路．欧拉回路是哪个？查看流值分配，将所有流量非 0(上限是 1，流值不是 0 就是 1)的边反向，就能得到每点入度 = 出度的欧拉图.
由于是满流，所以每个 入 > 出度点都有 x 条边进来，将这些进来的边反向，OK．入 = 出．对于出 > 入的情况亦然．那么，没和 s, t 连接的点怎么办？和 s 连接的条件是出 > 入，和 t 连接的条件是入 > 出．那么这个既没和 s 也没和 t 连接的点，自然开始就已经满足 入 = 出了．那么在网络流过程中，这些点属于"中间点"．我们知道中间点流量不允许有累积．这样，进去多少就出来多少，反向之后，自然仍保持平衡.
*/

欧拉回路

```
/*
求欧拉回路或欧拉路,邻接阵形式,复杂度 O(n^2)
返回路径长度,path 返回路径(有向图时得到的是反向路径)
传入图的大小 n 和邻接阵 mat,不相邻点边权 0
可以有自环与重边,分为无向图和有向图
判断方法:连通无向图各顶点偶数个则存在欧拉回路
欧拉路径的点的出入满足的条件 in-out==1&&out-in==1 的个数都为 1,其他的 in=out
用以上方法验证记得判连通*****
*/
#include <iostream>
using namespace std;
const int MAXN = 101;
int mat[MAXN][MAXN], ans[MAXN];
void find_path_u(int n, int mat[][MAXN], int now, int& step, int path[]) {
    for (int i = n - 1; i >= 0; i--)
        while (mat[now][i]) {
            mat[now][i]--, mat[i][now]--;
            find_path_u(n, mat, i, step, path);
        }
    path[step++] = now;
}
void find_path_d(int n, int mat[][MAXN], int now, int& step, int path[]) {
    for (int i = n - 1; i >= 0; i--)
        while (mat[now][i]) {
            mat[now][i]--;
            find_path_d(n, mat, i, step, path);
        }
    path[step++] = now;
}
```

```
int euclid_path(int n, int mat[][MAXN], int start, int path[]) {
    int ret = 0;
    find_path_u(n, mat, start, ret, path);//无向
    //find_path_d(n,mat,start,ret,path);//有向
    return ret;
}
```

二分图判定 DFS 染色

```
/*
DFS 染色二分图判定
平面上有 n 个基站,可以发射 1、2 两种频率的信号
相同频率的信号会产生干扰,找一个最小的发射距离使得平面上所有的点都没有干扰
首先想到二分覆盖半径,然后如果两个基站的距离不超过覆盖直径的话,那么他们应该使用不同的频率,也就是应该是一个二分图,如果不是二分图,说明二分的半径偏大
判断二分图只要 DFS 着色就可以了.也可以用并查集来做,如果产生矛盾,说明二分的半径偏大.
*/
struct point {
    int x, y;
} p[MAXN];
int d[MAXN][MAXN];
int edge, head[MAXN], next[MAXM], e[MAXM];
int n, color[MAXN];
bool dfs(int now, int c) {
    color[now] = c;
    bool flag = 1;
    for (int i = head[now]; i != -1; i = next[i]) {
        int v = e[i];
        if (color[v] == 0) {
            flag &= dfs(v, 3 - c);
            if (!flag) return 0;
        } else if (color[v] == c) return 0;
    }
    return flag;
}
bool check(int x) {
    init();
    for (int i = 0; i < n; i++) {
        color[i] = 0;
        for (int j = 0; j < i; j++)
            if (d[i][j] < x)
                addedge(i, j), addedge(j, i);
    }
    bool flag = 1;
    for (int i = 0; i < n; i++)
        if (!color[i]) flag &= dfs(i, 1);
    return flag;
}
int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; i++) {
            scanf("%d%d", &p[i].x, &p[i].y);
            for (int j = 0; j < i; j++)
                d[i][j] = d[j][i] = (p[i].x - p[j].x) * (p[i].x - p[j].x) + (p[i].y - p[j].y) * (p[i].y - p[j].y);
        }
        int l = 0, r = 16e8;
        while (r - l != 1) {
            int mid = (l + r) >> 1;
            if (check(mid)) l = mid;
            else r = mid;
        }
        printf("%.10lf\n", sqrt(l) / 2);
        check(l);
```

```
        for (int i = 0; i < n; i++) {
            if (i) putchar(' ');
            printf("%d", color[i]);
        }
        puts("");
    }
}
```

二分图最大匹配 匈牙利

```
/*
匈牙利算法复杂度 O(VE),优点：实现简洁容易理解,适用于稠密图,DFS 找增广路快
路径覆盖就是在 dag 图中找一些路经,使之覆盖了图中的所有顶点,且任何一个顶点有且只有一条路径与之关联(建图要分点的)
对于公式：最小路径覆盖 = |P| - 最大匹配数；可以这么来理解；
如果匹配数为零,那么 P 中不存在有向边,于是显然有：
最小路径覆盖 = |P| - 最大匹配数 = |P| - 0 = |P| ；即 P 的最小路径覆盖数为 |P| ；
如果在 P' 中增加一条匹配边 pi'->pj',那么在图 P 的路径覆盖中就存在一条由 pi 连接 pj 的边,也就是说 pi 与 pj 在一条路径上,于是路径覆盖数就可以减少一个；
pku 3216 最小路径覆盖
*/
#include <iostream>
using namespace std;
const int MAXN = 1001;
const int MAXM = 1001;
int nx, ny, m, ans;//nx,ny 分别为二分图两边节点的个数,两边的节点分别用 1..nx,1..ny 编号,m 为边数
bool g[MAXN][MAXM];//图 G 邻接矩阵 g[x][y]
bool y[MAXM];//Y 集合中点 i 访问标记
int link[MAXM];//link[y]表示当前与 y 节点相邻的 x 节点

void init() {
    int x, y;
    memset(g, 0, sizeof(g));
    ans = 0;
    scanf("%d%d%d", &nx, &ny, &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &x, &y);
        g[x][y] = true;
    }
}
bool find(int x) { //是否存在 X 集合中节点 x 开始的增广路
    for (int i = 1; i <= ny; i++)
        if (g[x][i] && !y[i]) { //如果节点 i 与 x 相邻并且未访问过
            y[i] = true;
            if (link[i] == -1 || find(link[i])) { //如果找到一个未盖点 i 中或从与 i 相邻的节点出发有增广路
                link[i] = x;
                return true;
            }
        }
    return false;
}
int MaximumMatch() {
    int ret = 0;
    memset(link, -1, sizeof(link));
    for (int i = 1; i <= nx; i++) {
        memset(y, 0, sizeof(y));
        if (find(i))
            ret++;
    }
    return ret;
}
int main() {
    init();
    /*for (int j = 1; j <= ny; j++)
        for (int i = 1; i <= nx; i++)
```

```
            if (g[i][j] && !link[j])
                link[j] = i;//贪心初始解优化*/
    ans = MaximumMatch();
    printf("%d\n", ans);
    return 0;
}
```

二分图最大匹配 HOPCROFT KARP

```
//Hopcroft Karp 算法在增加匹配集合 M 时,每次寻找多条增广路,复杂度 O(V^0.5*E)
//SPOJ MATCHING 点 1..nx
#include <iostream>
using namespace std;
const int MAXN = 50005;
const int MAXM = 150005;
int nx, ny, m, x, y, ans;
int edge, head[MAXN], next[MAXM], e[MAXM];
int cx[MAXN], cy[MAXN];// cx[i]表示 xi 对应的匹配 , cy[i]表示 yi 对应的匹配.
int distx[MAXN], disty[MAXN]; // 层的概念 , 即在 BFS 中的第几层.
int que[MAXN];
void init() {
    memset(cx, -1, sizeof(cx));
    memset(cy, -1, sizeof(cy));
    memset(head, -1, sizeof(head));
    edge = 0;
    ans = 0;
}
inline void addedge(int u, int v) {
    e[edge] = v;
    next[edge] = head[u];
    head[u] = edge++;
}
bool BFS() {
    bool flag = 0;
    memset(distx, 0, sizeof(distx));
    memset(disty, 0, sizeof(disty));
    int l = 0, r = 0, h, t;
    for (int i = 1; i <= nx; i++)
        if (cx[i] == -1)
            que[r++] = i;
    while (l != r) {
        for (h = l, t = r; h != t; h = (h + 1) % MAXN) {
            int u = que[h];
            for (int i = head[u]; i != -1; i = next[i]) {
                int v = e[i];
                if (!disty[v]) {
                    disty[v] = distx[u] + 1;
                    if (cy[v] == -1) {
                        flag = 1;
                    } else {
                        distx[cy[v]] = disty[v] + 1;
                        que[r] = cy[v];
                        r = (r + 1) % MAXN;
                    }
                }
            }
        }
        l = t;
    }
    return flag;
}
bool DFS(int u) {
    for (int i = head[u]; i != -1; i = next[i]) {
        int v = e[i];
```

```
        if (disty[v] == distx[u] + 1) { //说明 v 是 u 的后继结点.
            disty[v] = 0; // v 被用过了,不能再作为其他点的后继结点了.
            if (cy[v] == -1 || DFS(cy[v])) {
                cx[u] = v, cy[v] = u;
                return 1;
            }
        }
    }
    return 0;
}
void Hopcroft_Karp() {
    while (BFS()) {
        for (int i = 1; i <= nx; i++)
            if (cx[i] == -1 && DFS(i)) ans++;
    }
}
int main() {
    while (scanf("%d %d %d", &nx, &ny, &m) != EOF) {
        init();
        for (int i = 0; i < m; i++) {
            scanf("%d%d", &x, &y);
            addedge(x, y);
        }
        Hopcroft_Karp();
        printf("%d\n", ans);
    }
    return 0;
}
```

二分图多重匹配

```
//二分图多重匹配 集合 X 点度 1，集合 Y 点度可以大于 1 求 X 的最大匹配 HDU3605
#include<iostream>
#include<algorithm>
using namespace std;
const int MAXN = 100100, MAXM = 11;
int nx, ny, map[MAXN][MAXM], vis[MAXM], cap[MAXM], sum[MAXM], cnt[MAXM][MAXN];
bool find(int p) {
    for (int i = 0;i < ny;i++) {
        if (!vis[i] && map[p][i]) {
            vis[i] = 1;
            if (sum[i] < cap[i]) {
                cnt[i][sum[i]++] = p;
                return 1;
            }
            for (int j = 0;j < sum[i];j++) {
                if (find(cnt[i][j])) {
                    cnt[i][j] = p;
                    return 1;
                }
            }
        }
    }
    return 0;
}
int main() {
    while (scanf("%d%d", &nx, &ny) != EOF) {
        for(int i = 0; i < nx; i++)
            for(int j = 0; j < ny; j++)
        scanf("%d", &map[i][j]);
        for(int i = 0; i < ny; i++)
            scanf("%d", &cap[i]);
        memset(sum, 0, sizeof(sum));
        int ans = 1;
```

```
        for (int i = 0; i < nx; i++) {
            memset(vis, 0, sizeof(vis));
            if (!find(i)) { //输出答案改成 ans++
                ans = 0;
                break;
            }
        }
        puts(ans ? "YES" : "NO");
    }
}
```

带权二分图最佳匹配 KM

```
/*
KM 算法 复杂度 O(V^3)
左边 N 个点 0..N-1 右边 M 个点 0..M-1 求匹配要保证 N <= M(swap(N,M))否则循环无法终止
mat[i][j]表示 i 和 j 之间有边,
lx[],ly[]是 X,Y 上各点的标号
linky[y] = x 代表 y 被 x 匹配 未匹配点 linky[] = -1
最小权匹配可将权值取相反数或用一个大数去减
*/
//KM 算法 左边 N 个点 0..N-1 右边 M 个点 0..M-1 求匹配要保证 N <= M(swap(N,M))
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int MAXN = 502;
const int MAXV = 0x3F3F3F3F;
int mat[MAXN][MAXN];
bool visitx[MAXN], visity[MAXN];
int lx[MAXN], ly[MAXN], linky[MAXN];
int N, M, slack[MAXN];
int find(int u) {
    visitx[u] = 1;
    for (int i = 0; i < M; i++) {
        if (!visity[i]) {
            int tmp = lx[u] + ly[i] - mat[u][i];
            if (tmp == 0) {
                visity[i] = 1;
                if (linky[i] == -1 || find(linky[i])) {
                    linky[i] = u;
                    return 1;
                }
            } else if (tmp < slack[i])
                slack[i] = tmp;
        }
    }
    return 0;
}
int KM() {
    for (int i = 0; i < N; i++) {
        lx[i] = 0;
        for (int j = 0; j < M; j++)
            if (mat[i][j] > lx[i])
                lx[i] = mat[i][j];
    }
    for (int j = 0; j < M; j++) ly[j] = 0;
    memset(linky, -1, sizeof(linky));
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++)
            slack[j] = MAXV;
        while (1) {
            memset(visitx, 0, sizeof(visitx));
            memset(visity, 0, sizeof(visity));
```

```
            int min = MAXV;
            if (find(i)) break;
            for (int j = 0; j < M; j++)
                if (!visity[j] && slack[j] < min)
                    min = slack[j];
            for (int j = 0; j < N; j++)
                if (visitx[j])
                    lx[j] -= min;
            for (int j = 0; j < M; j++)
                if (visity[j])
                    ly[j] += min;
                else
                    slack[j] -= min;
        }
    }
    int ret = 0, cnt = 0;
    for (int i = 0; i < M; i++) {
        int v = linky[i];
        if (v >= 0 && mat[v][i] != -MAXV) {
            cnt++;
            ret += mat[v][i];
        }
    }
    if (cnt < N) //不存在最佳匹配
        return -1;
    else
        return ret;
}
int main() {
    int E, cnt;
    int cas = 0;
    while (scanf("%d%d%d", &N, &M, &E) != EOF) {
        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                mat[i][j] = -MAXV;
        while (E--) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            if (z < 0) continue;
            mat[x][y] = z;
        }
        printf("Case %d: ", ++cas);
        if (N > M) {
            puts("-1");
            continue;
        }
        int ans = KM();
        printf("%d\n", ans);
    }
    return 0;
}

//另一个 KM
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
LL INF = 1LL << 40;
struct KM {
    int n;
    LL a[20][20], lx[20], ly[20];
    int visx[20];
    int visy[20];
    LL slack[20];
    int match[20];
```

```
    void init(int _n) {
        n = _n;
        memset(a, 0, sizeof(a));
        memset(slack, 0, sizeof(slack));
    }
    int dfs(int u) {
        int i;
        visx[u] = 1;
        for (i = 0; i < n; i++) {
            if (visy[i]) continue;
            if (lx[u] + ly[i] == a[u][i]) {
                visy[i] = 1;
                if (match[i] == -1 || dfs(match[i])) {
                    match[i] = u;
                    return 1;
                }
            } else {
                slack[i] = min(slack[i], lx[u] + ly[i] - a[u][i]);
            }
        }
        return 0;
    }
    LL km() {
        int i, j;
        memset(lx, 0, sizeof(lx));
        memset(ly, 0, sizeof(ly));
        memset(match, -1, sizeof(match));
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                lx[i] = max(lx[i], a[i][j]);
            }
        }
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++)slack[j] = INF;
            while (1) {
                memset(visx, 0, sizeof(visx));
                memset(visy, 0, sizeof(visy));
                if (dfs(i)) break;
                LL d = INF;
                for (j = 0; j < n; j++) {
                    if (!visy[j])d = min(d, slack[j]);
                }
                for (j = 0; j < n; j++) {
                    if (visx[j])lx[j] -= d;
                    if (visy[j])ly[j] += d;
                    else slack[j] -= d;
                }
            }
        }
        LL ans = 0;
        for (i = 0; i < n; i++) {
            ans += lx[i] + ly[i];
        }
        return ans;
    }
} g;
int main() {
    while (scanf("%d", &n) != EOF) {
        g.init(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                scanf("%d", &g.a[i][j]);
            }
        }
        printf("%lld\n", g.km());
    }
```

```
}
```

一般图匹配

```
/*
一般图匹配,带花树算法 初始化需要 n 的大小,adj 矩阵
author: momodi@whuacm URAL1099
*/
#include<iostream>
using namespace std;
#define maxn 1000
int n;//节点个数,用的是 0~n-1,
int match[maxn];//每个点匹配情况
bool adj[maxn][maxn];//邻接阵,连通为 true
void clear() {
    memset(adj, 0, sizeof(adj));
    n = 0;
}
void insert(const int &u, const int &v) {
    adj[u][v] = adj[v][u] = 1;
}
int Q[maxn], pre[maxn], base[maxn];
bool _h[maxn];
bool in_blossom[maxn];
void argument(int u) {
    while (u != -1) {
        int v = pre[u];
        int k = match[v];
        match[u] = v;
        match[v] = u;
        u = k;
    }
}
void change_blossom(int b, int u) {
    while (base[u] != b) {
        int v = match[u];
        in_blossom[base[v]] = in_blossom[base[u]] = true;
        u = pre[v];
        if (base[u] != b) {
            pre[u] = v;
        }
    }
}
int find_base(int u, int v) {
    bool in_path[maxn] = {};
    while (true) {
        in_path[u] = true;
        if (match[u] == -1) {
            break;
        }
        u = base[pre[match[u]]];
    }
    while (!in_path[v]) {
        v = base[pre[match[v]]];
    }
    return v;
}
int contract(int u, int v) {
    memset(in_blossom, 0, sizeof(in_blossom));
    int b = find_base(base[u], base[v]);
    change_blossom(b, u);
    change_blossom(b, v);
    if (base[u] != b) {
```

```
            pre[u] = v;
        }
        if (base[v] != b) {
            pre[v] = u;
        }
        return b;
    }
    int bfs(int p) {
        memset(pre, -1, sizeof(pre));
        memset(_h, 0, sizeof(_h));
        for (int i = 0; i < n; ++i) {
            base[i] = i;
        }
        Q[0] = p;
        _h[p] = 1;
        for (int s = 0, t = 1; s < t; ++s) {
            int u = Q[s];
            for (int v = 0; v < n; ++v) {
                if (adj[u][v] && base[u] != base[v] && v != match[u]) {
                    if (v == p || (match[v] != -1 && pre[match[v]] != -1)) {
                        int b = contract(u, v);
                        for (int i = 0; i < n; ++i) {
                            if (in_blossom[base[i]]) {
                                base[i] = b;
                                if (_h[i] == 0) {
                                    _h[i] = 1;
                                    Q[t++] = i;
                                }
                            }
                        }
                    } else if (pre[v] == -1) {
                        pre[v] = u;
                        if (match[v] == -1) {
                            argument(v);
                            return 1;
                        } else {
                            Q[t++] = match[v];
                            _h[match[v]] = 1;
                        }
                    }
                }
            }
        }
        return 0;
    }
    int max_match() { //节点编号从0~n-1
        memset(match, -1, sizeof(match));//没匹配的就是-1
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            if (match[i] == -1) {
                ans += bfs(i);
            }
        }
        return ans;
    }
    //**************Author: momodi  模板结束**************
    int main() {
        int N, a, b;
        scanf("%d", &N);
        n = N;
        memset(adj, false, sizeof(adj));
        while (scanf("%d%d", &a, &b) != EOF) {
            adj[a-1][b-1] = adj[b-1][a-1] = true;
        }
        int ret = max_match();
        printf("%d\n", ret*2);
```

```
    for (int i = 0; i < N; i++) {
        if ( match[i] != -1) {
            printf("%d %d\n", i + 1, match[i] + 1);
            match[ match[i] ] = -1;
        }
    }
}
```

最大团和最大独立集

```
/*
团：指 G 的一个完全子图，该子图不包含在任何其他的完全当中
最大团 ：指其中包含顶点最多的团
最大独立集：补图的最大团
ZJU1492 HDU1530
*/
#include <iostream>
using namespace std;
const int MAXN = 51;
int mat[MAXN][MAXN];
int dp[MAXN];
bool vis[MAXN];
int n, maxclique, clique[MAXN];//方案(0..maxclique-1)点(0..n-1)
void dfs(int n, int* u, int mat[][MAXN], int size, int& max, int& bb, int* res, int* rr, int* c) {
    int i, j, vn, v[MAXN];
    if (n) {
        if (size + c[u[0]] <= max) return;
        for (i = 0; i < n + size - max && i < n; ++ i) {
            for (j = i + 1, vn = 0; j < n; ++ j)
                if (mat[u[i]][u[j]])
                    v[vn ++] = u[j];
            rr[size] = u[i];
            dfs(vn, v, mat, size + 1, max, bb, res, rr, c);
            if (bb) return;
        }
    } else if (size > max) {
        max = size;
        for (i = 0; i < size; ++ i)
            res[i] = rr[i];
        bb = 1;
    }
}

int max_clique(int n, int mat[][MAXN], int *ret) {
    int max = 0, bb, c[MAXN], i, j;
    int vn, v[MAXN], rr[MAXN];
    for (c[i = n - 1] = 0; i >= 0; -- i) {
        for (vn = 0, j = i + 1; j < n; ++ j)
            if (mat[i][j])
                v[vn++] = j;
        bb = 0;
        rr[0] = i;
        dfs(vn, v, mat, 1, max, bb, ret, rr, c);
        c[i] = max;
    }
    return max;
}
int main() {
    while (scanf("%d", &n) != EOF && n) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                scanf("%d", &mat[i][j]);
        memset(dp, 0, sizeof(dp));
        maxclique = max_clique(n, mat, clique);
```

```
        printf("%d\n", maxclique);
        for (int i = 0; i < maxclique; i++) {
            if (i) putchar(' ');
            printf("%d", clique[i]);
        }
        printf("\n");
    }
}
```

无向图割边割点

```
/*
连通无向图求割点个数割边条数 默认 DFS 根为 1
割点在 cut[]里 割边在 bridge[]里
求两点间割边:从一点开始 DFS 碰到另一点则反回 true 否则返回 false 若 true 则标记割边
边双连通分支 floodfill 对桥分割开的块涂色,color 中存储每个节点所属的分支
若求加几条边能构造双连通图,使图没有割边,则只需统计块缩点后叶子的个数 cnt,ans = (cnt+1)/2  PKU3177 PKU3352
*/
#include <iostream>
using namespace std;
#define out(x) cerr << #x << " = " << (x) << endl
const int MAXN = 110, MAXM = 10010 * 2;
int n, m, x, y, cnt, root = 1;
int edge, head[MAXN], e[MAXM], next[MAXM];
int num, dfn[MAXN], low[MAXN];
bool cut[MAXN], used[MAXM], bridge[MAXM];;
void init() {
    memset(used, 0, sizeof(used));
    memset(head, -1, (n + 1)*sizeof(int));
    memset(dfn, 0, (n + 1)*sizeof(int));
    memset(cut, 0, (n + 1)*sizeof(int));
    memset(bridge, 0, sizeof(bridge));
    edge = cnt = num = 0;
}
inline void addedge(int u, int v) {
    e[edge] = v;
    next[edge] = head[u];
    head[u] = edge++;
}
void dfs(int u) {
    dfn[u] = low[u] = ++num;
    for (int i = head[u]; i != -1; i = next[i]) {
        if (used[i]) continue;
        else {
            used[i&(~1)] = 1;
            used[i|1] = 1;
        }
        int v = e[i];
        if (!dfn[v]) {
            if (u == root) cnt++;//统计 DFS 根的父子边数量
            dfs(v);
            low[u] = min(low[u], low[v]);//uv 是父子边
            if (low[v] >= dfn[u] && u != root)//不为根
                cut[u] = 1;
            if (low[v] > dfn[u]) //形成桥
                bridge[i&(~1)] = 1;
        } else
            low[u] = min(low[u], dfn[v]);//uv 是返祖边
    }
}
int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        for (int i = 0; i < m; i++) {
```

```
            scanf("%d%d", &x, &y);
            addedge(x, y);
            addedge(y, x);
        }
        dfs(root);
        cut[root] = (cnt >= 2);//DFS 根至少有 2 条父子边才为割点
        int ans1 = 0, ans2 = 0;
        for (int i = 1; i <= n; i++)
            if (cut[i])
                ans1++;
        for (int i = 0; i < edge; i++)
            if (bridge[i])
                ans2++;
        printf("%d %d\n", ans1, ans2);
    }
}
```

有向图弱连通分量

```
/*
强连通图：在有向图中，若对于每一对顶点 v1 和 v2，都存在一条从 v1 到 v2 和从 v2 到 v1 的路径，则称此图是强连通图.
弱连通图：将有向图的所有的有向边替换为无向边，所得到的图称为原图的基图.如果一个有向图的基图是连通图，则有向图是弱连通图.
单向连通图:如果有向图中，对于任意节点 v1 和 v2，至少存在从 v1 到 v2 和从 v2 到 v1 的路径中的一条，则原图为单向连通图.
三者之间的关系是，强连通图必然是单向连通的，单向连通图必然是弱连通图.
求弱连通图：首先强连通，然后缩点，对缩点形成的图最多只能有一个入度为 0 的点，
如果有多个入度为 0 的点，则这两个连通分量肯定是不连通的.
缩点后形成的图是一棵树，入度为 0 的点是这颗树的根，这棵树只能是单链，不能有分叉，如果有分叉，则这些分叉之间是不可达的，所以就对这棵树进行 DFS，如果是单链则是 YES.
*/
```

有向图强连通分量 KOSARAJU

```
//有向图强连通分量 Kosaraju 算法 正反两次 DFS 复杂度 O(N+M)
#include <iostream>
#include <sstream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef pair<int, int> PII;
const int MAXN = 10001, MAXM = 50001;
int n, m;
int edge, head[MAXN], e[MAXM], next[MAXM];
int scc, id[MAXN];//scc 为强连通分量数 id 为节点所在 scc 序号
int top, S[MAXN], deg[MAXN];
bool vis[MAXN];
PII E[MAXM];
void init() {
    memset(head, -1, (n + 1)*sizeof(int));
    edge = 0;
}
void addedge(int u, int v) {
    e[edge] = v;
    next[edge] = head[u];
    head[u] = edge++;
}
void dfs1(int u) {
    vis[u] = 1;
    for (int i = head[u]; i != -1; i = next[i])
        if (!vis[e[i]])
```

```
            dfs1(e[i]);
    S[top++] = u;
}
void dfs2(int u) {
    vis[u] = 1;
    id[u] = scc;
    for (int i = head[u]; i != -1; i = next[i])
        if (!vis[e[i]])
            dfs2(e[i]);
}
void kosaraju(int n, int m) { //点1..n,id从1..scc
    init();
    for (int i = 0; i < m; i++)
        addedge(E[i].first, E[i].second);
    memset(vis, 0, (n + 1)*sizeof(bool));
    top = 0;
    for (int i = 1; i <= n; i++)//注意1..n还是0..n-1
        if (!vis[i])
            dfs1(i);
    memset(vis, 0, (n + 1)*sizeof(bool));
    init();
    for (int i = 0; i < m; i++)
        addedge(E[i].second, E[i].first);
    scc = 0;
    for (int i = top - 1; i >= 0; i--)
        if (!vis[S[i]]) {
            scc++;
            dfs2(S[i]);
        }
}
int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        for (int i = 0; i < m; i++)
            scanf("%d%d", &E[i].first, &E[i].second);
        //sort(E,E+m);
        //m = unique(E,E+m) - E;//去重
        kosaraju(n, m);
        printf("%d\n", scc);
        for (int i = 1; i <= n; i++)
            printf("%d %d\n", i, id[i]);
    }
}
```

有向图强连通分量 TARJAN

```
/*
有向图强连通分量 Tarjan算法 复杂度O(N+M)
默认点1..n,SCC:1..scc
*/
#include <iostream>
using namespace std;
typedef pair<int, int> PII;
const int MAXN = 10001, MAXM = 50001;
int n, m;
int edge, head[MAXN], e[MAXM], next[MAXM];//边表
int num, dfn[MAXN], low[MAXN];//tarjan相关
int scc, id[MAXN];//scc为强连通分量数 id为节点所在scc序号1..scc
int top, S[MAXN];//判断横叉边的栈S
bool instack[MAXN];//某点是否入栈
PII E[MAXM];//读入边表
void init() {
    memset(head, -1, sizeof(head));
    edge = 0;
```

```
}
void addedge(int u,int v) {
    e[edge] = v;
    next[edge] = head[u];
    head[u] = edge++;
}
void tarjan(int u) {
    dfn[u] = low[u] = ++num;
    instack[u] = true;
    S[++top] = u;
    int v;
    for (int i = head[u]; i != -1; i = next[i]) {
        v = e[i];
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);//uv 是父子边
        } else if (instack[v]) {//防止 uv 是横叉边
            low[u] = min(low[u], dfn[v]);//uv 是返祖边
        }
    }
    if (low[u] == dfn[u]) { //u 下面形成一个块
        scc++;
        do {
            v = S[top--];
            instack[v] = false;
            id[v] = scc;
        } while (v != u);
    }
}
int SCC() { //点 1..n
    scc = num = top = 0;
    memset(dfn, 0, sizeof(dfn));
    memset(instack, 0, sizeof(instack));
    for (int i = 1; i <= n; i++)
        if (!dfn[i])
            tarjan(i);
    return scc;
}
int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        for (int i = 0; i < m; i++)
            scanf("%d%d", &E[i].first, &E[i].second);
        //sort(E,E+m);
        //m = unique(E,E+m) - E;//去重
        for (int i = 0; i < m; i++)
            addedge(E[i].first, E[i].second);
        printf("%d\n", SCC());
        for (int i = 1; i <= n; i++)
            printf("%d %d\n", i, id[i]);
    }
}
```

有向图最小点基

```
/*
有向图最小点基 就是求出入度为 0 的强连通分量个数 PKU1236
最小权点基只要在每个连通分量内找权值最小的点即可
可求加最少边使任意点可达全图 答案为 max(入度为 0 的点的个数,出度为 0 的点的个数)
*/
int in[MAXN], out[MAXN];
bool link[MAXN][MAXN];
kosaraju(n, m);
memset(in, 0, (n + 1)*sizeof(int));
```

```
memset(out, 0, (n + 1)*sizeof(int));
memset(link, 0, sizeof(link));
for (int i = 0; i < m; i++) {
    int x = id[E[i].first], y = id[E[i].second];
    if (x != y) {
        if (link[x][y]) continue;
        else link[x][y] = 1;
        in[y]++;
        out[x]++;
    }
}
int ans1 = 0, ans2 = 0;
for (int i = 1; i <= scc; i++) {
    if (in[i] == 0) ans1++;
    if (out[i] == 0) ans2++;
}
printf("%d\n", ans1);//最小点基
/*if (scc == 1) puts("0");
else printf("%d\n",max(ans1,ans2));*///加最少边使任意点可达全图
```

LCA

```
/*
LCA 利用 RMQ 区间最值离线 ST 算法,复杂度 O(NlogN)
默认下标范围 0..n-1,求最大值，根默认为 0 先调用 DFS(0,0)再 initRMQ() //ZJUT1504
H[]是点第一次 dfs 到的位置 L[]是这个点的层数 E[]是对应 dfs 到的点
*/
#include <iostream>
#include <cmath>
using namespace std;
const int MAXN = 120001;
int n, m, x, y;
int edge, head[MAXN], next[2*MAXN], to[2*MAXN];
int len, H[MAXN], L[2*MAXN], E[2*MAXN], M[2*MAXN][17];//log2(40000);
void init() {
    memset(head, -1, sizeof(head));
    memset(H, -1, sizeof(H));
    len = edge = 0;
}
inline void addedge(int u, int v) {
    to[edge] = v; next[edge] = head[u]; head[u] = edge++;
}
void dfs(int u, int dep) {
    H[u] = len;
    L[len] = dep;
    E[len++] = u;
    for (int i = head[u]; i != -1; i = next[i]) {
        if (H[to[i]] != -1) continue;
        dfs(to[i], dep + 1);
        L[len] = dep;
        E[len++] = u;
    }
}
/*非递归
int S[MAXN],now[MAXN],top,dep;

void push(int u) {
    S[++top] = u;
    H[S[top]] = len;
    L[len] = dep;
    E[len++] = S[top];
}
void dfs() {
    top = dep = 0;
    int v;
```

```
    memcpy(now,head,(n+1)*sizeof(int));
    push(0);
    while (top) {
        if (now[S[top]] == -1) {
            dep--;
            if (!--top) break;
            L[len] = dep;
            E[len++] = S[top];
        }
        v = to[now[S[top]]];
        now[S[top]] = next[now[S[top]]];
        if (H[v] != -1) continue;
        ++dep;
        push(v);
    }
}
*/
void initRMQ(int n) {
    for (int i = 0; i < n; i++)
        M[i][0] = i;
    for (int j = 1; 1 << j <= n; j++)
        for (int i = 0; i + (1 << j) - 1 < n; i++)
            if (L[M[i][j-1]] < L[M[i+(1<<(j-1))][j-1]])
                M[i][j] = M[i][j-1];
            else
                M[i][j] = M[i+(1<<(j-1))][j-1];
}

int RMQ(int l, int r) {
    int m = (int)floor(log(double(r - l + 1)) / log(2.0));
    if (L[M[l][m]] < L[M[r-(1<<m)+1][m]])
        return M[l][m];
    else
        return M[r-(1<<m)+1][m];
}
int lca(int x, int y) {
    if (H[x] > H[y]) swap(x, y);
    return E[RMQ(H[x], H[y])];
}
int main() {
    int cas = 0;
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        for (int i = 0; i < n - 1; i++) {
            scanf("%d%d", &x, &y);
            addedge(x, y);
            addedge(y, x);
        }
        dfs(0, 0);
        initRMQ(len);
        printf("Case %d:\n", ++cas);
        for (int i = 0; i < m; i++) {
            scanf("%d%d", &x, &y);
            printf("%d\n", lca(x, y));
        }
    }
}
/*
```

[点连通度与边连通度]

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以后，原图变成多个连通块，就称这个点集为割点集合．一个图的点连通度的定义为，最小割点集合中的顶点数．

类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为割边集合．一个图的边连通度的定义为，最小割边集合中的边数．

[双连通图、割点与桥]

如果一个无向连通图的点连通度大于 1，则称该图是点双连通的(point biconnected)，简称双连通或重连通．一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为割点(cut point)，又叫关节点(articulation point)．

如果一个无向连通图的边连通度大于 1，则称该图是边双连通的(edge biconnected)，简称双连通或重连通．一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为桥(bridge)，又叫关节边(articulation edge)．
可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通．
[双连通分支]
在图 G 的所有子图 G′中，如果 G′是双连通的，则称 G′为双连通子图．如果一个双连通子图 G′它不是任何一个双连通子图的真子集，则 G′为极大双连通子图．双连通分支(biconnected component)，或重连通分支，就是图的极大双连通子图．特殊的，点双连通分支又叫做块．
[求割点与桥]
该算法是 R.Tarjan 发明的．对图深度优先搜索，定义 DFS(u)为 u 在搜索树（以下简称为树）中被遍历到的次序号．定义 Low(u)为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点．根据定义，则有：

```
Low(u)=Min{       DFS(u)
                  DFS(v)  (u,v)为后向边(返祖边) 等价于 DFS(v)<DFS(u)且 v 不为 u 的父亲节点
                  Low(v)  (u,v)为树枝边(父子边) }
```

一个顶点 u 是割点，当且仅当满足(1)或(2)
(1) u 为树根，且 u 有多于一个子树．
(2) u 不为树根，且满足存在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)．
一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)．
[求双连通分支]
下面要分开讨论点双连通分支与边双连通分支的求法．
对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出．建立一个栈，存储当前双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中．如果遇到某时满足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)，取出的这些边与其关联的点，组成一个点双连通分支．割点可以属于多个点双连通分支，其余点和每条边只属于且属于一个点双连通分支．
对于边双连通分支，求法更为简单．只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分支．桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属于一个边双连通分支．
[构造双连通图]
一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图．把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1．
统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf．则至少在树上添加(leaf+1)/2 条边，就能使树达到边二连通，所以至少添加的边数就是(leaf+1)/2．具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一定是双连通的．然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2 次，把所有点收缩到了一起．
*/

```
/*
无向图 tarjan 求割点求桥什么的*****************************
1, 求割点测试过 pku 1144 正确
2, 求桥 zjut 上那个两条路测试 AC 的....
dfs 之后判断一条边是不是桥
if( dep[a]>dep[b] ) swap(a,b);
if( low[b] > dep[a] ) continue; //说明 a->b 是桥
*/
```

2-SAT

```
/*
2-SAT 对于两个不相容的点 i,j 构图方式为：i->j'和 j->i'
简要意思就是给定 N 个组（每个组 2 个元素）、M 个互斥关系，从每个组里挑 1 个使得给定的不满足任何互斥关系。
但是解决这类问题的关键还是在于建模，基本建模就是对于两个不相容的点 i、j
构图方式为：i->j'(i 和 j 冲突，选 i 只能选 j')和 j->i'（i 和 j 冲突，选 j 只能选 i'）。
解 2-SAT 方法是，对原图求一次强连通分量，然后看每组中的两个点是否属于同一个强连通分量，如果存在这种情况，那么无解。
*/
#include <iostream>
using namespace std;
typedef pair<int, int> PII;
const int MAXN = 8001 * 2, MAXM = 40001;
int n, m, N;//n 个组 m 个关系 N 个人 N = 2*n
int edge, head[MAXN], e[MAXM], next[MAXM];//边表
int num, dfn[MAXN], low[MAXN];//tarjan 相关
int scc, id[MAXN];//scc 为强连通分量数 id 为节点所在 scc 序号 1..scc
int cnt, ans[MAXN];
vector<int> tree[MAXN];//有向无环图的边连接情况(新图)
vector<int> contain[MAXN];//新图中每个 scc 都包含原图中的哪些点
int top, S[MAXN];//判断横叉边的栈 S
bool instack[MAXN];//某点是否入栈
```

```
void init() {
    memset(head, -1, sizeof(head));
    edge = 0;
}
void addedge(int u, int v) {
    e[edge] = v; next[edge] = head[u]; head[u] = edge++;
}
void tarjan(int u) {
    dfn[u] = low[u] = ++num;
    instack[u] = true;
    S[++top] = u;
    int v;
    for (int i = head[u]; i != -1; i = next[i]) {
        v = e[i];
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]); //uv 是父子边
        } else if (instack[v]) //防止 uv 是横叉边
            low[u] = min(low[u], dfn[v]); //uv 是返祖边
    }
    if (low[u] == dfn[u]) { //u 下面形成一个块
        //scc++;
        do {
            v = S[top--];
            instack[v] = false;
            id[v] = scc;
        } while (v != u);
        scc++;
    }
}
int SCC() { //点 0..2*n-1
    scc = num = top = 0;
    memset(dfn, 0, sizeof(dfn));
    memset(instack, 0, sizeof(instack));
    for (int i = 0; i < N; i++)
        contain[i].clear();
    for (int i = 0; i < N; i++)//记得是 2*n 还是 n
        if (!dfn[i])
            tarjan(i);
    for (int i = 0; i < N; i++)//统计每个强连通分量所包含的点 如果不求可行解，可注释掉。
        contain[id[i]].push_back(i);
    return scc;
}
bool TwoSat() { //2-sat 判定
    SCC();
    for (int i = 0; i < n; i++)
        if (id[2*i] == id[2*i+1])
            return 0;
    return 1;
}
void shrink() { //缩点，得到的是一个有向无环图(原图无重边)
    for (int i = 0; i < scc; i++)
        tree[i].clear();
    for (int u = 0; u < N; u++)
        for (int i = head[u]; i != -1; i = next[i]) {
            int v = e[i];
            if (id[u] != id[v])
                tree[id[v]].push_back(id[u]);
        }
}
void DFS(int u) { //拓扑排序
    dfn[u] = ++num;
    for (int i = 0; i < tree[u].size(); i++) {
        int v = tree[u][i];
        if (!dfn[v])
            DFS(v);
```

```
    }
    ans[cnt++] = u;
}
//函数 ColDfs 是对新图进行着色，新图中着色为 1 的点组成一组可行解
void ColDFS(int u) {
    dfn[u] = 2;
    for (int i = 0; i < tree[u].size(); i++) {
        int v = tree[u][i];
        if (!dfn[v])
            ColDFS(v);
    }
}
void GetAns() {
    shrink();
    num = cnt = 0;
    memset(dfn, 0, sizeof(dfn));
    for (int i = 0; i < scc; i++)
        if (!dfn[i])
            DFS(i);
    memset(dfn, 0, sizeof(dfn));
    for (int i = cnt - 1; i >= 0; i--)//逆拓扑序
        if (!dfn[ans[i]]) {
            int a = contain[ans[i]][0], b;
            //在原图中一属于强连通分量 ans[i]的点 a 点 a 对应的点 b 所属的强连通分量 id[b]一定是 ans[i]矛盾点。
            b = a ^ 1;
            dfn[ans[i]] = 1;//新图中 ans[i]着色为 1 后，它的矛盾点应标记为 2
            if (!dfn[id[b]])
                ColDFS(id[b]);//由于依赖关系，有 id[b]能达的点都是 low[i]的矛盾点
        }
    memset(ans, 0, sizeof(ans));
    for (int i = 0; i < cnt; i++)
        if (dfn[i] == 1)
            for (int j = 0; j < contain[i].size(); j++)
                ans[contain[i][j]] = 1;
    for (int i = 0; i < N; i++)
        if (ans[i] == 1)//(dfn[id[i]] != 1)
            printf("%d\n", i + 1);
}
/*void build() { //对于两个不相容的点 i,j 构图方式为：i->j'和 j->i'
    init();
    for (int i = 0;i < N; i++)
        for (int j = i+1;j < N;j++) {
            if ((i>>1) != (j>>1) && conflict[i][j]) { //注意修改冲突函数
                addedge(i,j^1);
                addedge(j,i^1);
            }
        }
}*/
int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        int x, y;
        for (int i = 0; i < m; i++) {
            scanf("%d%d", &x, &y);
            x--, y--;
            addedge(x, y ^ 1);
            addedge(y, x ^ 1);
        }
        N = 2 * n;
        //build();
        if (TwoSat()) GetAns();
        else puts("NIE");
    }
    return 0;
}
```

```
#include <iostream>
using namespace std;
namespace MF {
    #define MAXN 501
    #define MAXM 10010
    #define wint int
    const wint wEPS = 0;
    const wint wINF = 0x3F3F3F3F;//1001001001001001001LL;
    int n, m, ptr[MAXN], next[MAXM], zu[MAXM];
    wint capa[MAXM], tof;
    int lev[MAXN], see[MAXN], que[MAXN], *qb, *qe;
    void init(int _n) {
        n = _n; m = 0; memset(ptr, ~0, n << 2);
    }
    void ae(int u, int v, wint w0, wint w1 = 0) {
        next[m] = ptr[u]; ptr[u] = m; zu[m] = v; capa[m] = w0; ++m;
        next[m] = ptr[v]; ptr[v] = m; zu[m] = u; capa[m] = w1; ++m;
    }
    wint augment(int src, int ink, wint flo) {
        if (src == ink) return flo;
        wint f;
        for (int &i = see[src]; ~i; i = next[i]) if (capa[i] > wEPS && lev[src] < lev[zu[i]]) {
            if ((f = augment(zu[i], ink, min(flo, capa[i]))) > wEPS) {
                capa[i] -= f; capa[i ^ 1] += f; return f;
            }
        }
        return 0;
    }
    bool dinic(int src, int ink, wint flo = wINF) {
        wint f;
        int i, u, v;
        for (tof = 0; tof + wEPS < flo; ) {
            qb = qe = que;
            memset(lev, ~0, n << 2);
            for (lev[*qe++ = src] = 0, see[src] = ptr[src]; qb != qe; ) {
                for (i = ptr[u = *qb++]; ~i; i = next[i]) if (capa[i] > wEPS && !~lev[v = zu[i]]) {
                    lev[*qe++ = v] = lev[u] + 1; see[v] = ptr[v];
                    if (v == ink) goto au;
                }
            }
            return 0;
        au: for (; (f = augment(src, ink, flo - tof)) > wEPS; tof += f);
        }
        return 1;
    }
}
int n, m, k, num[15];
int v[15][300][2];
bool check(int mid) {
    int S = 0;
    int T = n + m + 1;
    MF::init(T + 1);
    int i;
    for (i = 1; i <= n; i++) {
        MF::ae(S, i, 1);
    }
    for (i = 0; i < m; i++) {
        MF::ae(n + i + 1, T, k);
        for (int j = 0; j < num[i]; j++) {
            if (v[i][j][1] >= mid) {
                MF::ae(v[i][j][0], n + i + 1, 1);
            }
        }
```

```
    }
    MF::dinic(S, T);
    return MF::tof == n;
}
int main() {
    while (scanf("%d%d%d", &n, &m, &k) != EOF) {
        for (int i = 0; i < m; i++) {
            scanf("%d", &num[i]);
            for (int j = 0; j < num[i]; j++) {
                scanf("%d%d", &v[i][j][0], &v[i][j][1]);
            }
        }
        int l = 0, r = 20001;
        while (l + 1 != r) {
            int mid = (l + r) >> 1;
            if (check(mid)) {
                l = mid;
            } else {
                r = mid;
            }
        }
        printf("%d\n", l);
    }
}
```

最大流 SAP

```
//最大流 SAP 复杂度 O(V^2*E) NOI2006 最大获利
//Shortest Augmenting Paths 最短增广路
//SAP + GAP + 当前弧优化
//千万记得修改传点数 aug 与 sap 里的点数 N，共 5 处
//错了试试 long long 或者把 MAXV 改小一点
#include <iostream>
using namespace std;
const int MAXN = 70001;
const int MAXM = 400001;
const int MAXV = 0X3F3F3F3F;
struct edge {
    int v, c;
    edge *next, *pair;
}*head[MAXN], *cur[MAXN], mem[MAXM];
int N, M, E, x, y, z;
int layer[MAXN], cnt[MAXN], flow;
int S, T, ans;//源 S 汇 T 最大流 ans
bool found;
void init() {
    memset(head, 0, sizeof(head));
    E = 0;
}
void addedge(int u, int v, int c1, int c2) { //u->v 加 c1 边,v->u 加 c2 边
    edge *x = &mem[E++], *y = &mem[E++];
    x->v = v; x->c = c1; x->pair = y; x->next = head[u]; head[u] = x;
    y->v = u; y->c = c2; y->pair = x; y->next = head[v]; head[v] = y;
}
void aug(int u, int T, int N) {
    if (u == T) { //找到增广路
        found = 1;
        ans += flow;
        return;
    }
    int tmp = flow;
    for (edge* i = cur[u]; i; i = i->next) {
        int v = i->v;
        if (i->c > 0) {
```

```
            if (layer[u] == layer[v] + 1) { //找到允许弧
                cur[u] = i;//标记当前弧
                flow = min(flow, i->c);
                aug(v, T, N);
                if (layer[S] > N) return;//断层
                if (found) { //找到增广路,减去流量
                    i->c -= flow;
                    i->pair->c += flow;
                    break;
                }
                flow = tmp;
            }
        }
    }
    if (!found) {
        int minl = N;
        for (edge* i = cur[u] = head[u]; i; i = i->next)
            if (i->c > 0)
                minl = min(minl, layer[i->v]);
        if (!(--cnt[layer[u]]))//GAP 优化 断层
            layer[S] = N + 1;
        layer[u] = minl + 1;//更新标号
        ++cnt[layer[u]];//层内数量+1
    }
}
void sap(int S, int T, int N) { //源点 S 汇点 T 点数 N
    ans = 0;
    memset(cnt, 0, sizeof(cnt));
    memset(layer, 0, sizeof(layer));
    cnt[0] = N + 1;
    memcpy(cur, head, sizeof(head));
    while (layer[S] <= N) {
        found = 0;
        flow = MAXV;
        aug(S, T, N);
    }
}
int main() {
    while (scanf("%d%d", &N, &M) != EOF) {
        init();
        S = 0;
        T = N + M + 1;
        for (int i = 1; i <= N; i++) {
            scanf("%d", &x);
            addedge(i, T, x, 0);
        }
        int sum = 0;
        for (int i = 1; i <= M; i++) {
            scanf("%d%d%d", &x, &y, &z);
            addedge(i + N, x, MAXV,0);
            addedge(i + N, y, MAXV,0);
            addedge(S, i + N, z, 0);
            sum += z;
        }
        sap(S, T, N + M + 2);//S 到 T 共 N+M+2 个点
        printf("%d\n", sum - ans);
    }
    return 0;
}
```

最大权闭合子图

```
/*
最大权闭合图：
```

定义：一个有向图的闭合图 G=(V,E)是该有向图的一个点集，且该点集的所有出边都还指向该点集。
即闭合图内的任意点的任意后继也一定在闭合图中。
给每个点 v 分配一个点权（任意实数，可正可负）。最大权闭合图，是一个点权之和最大的闭合图。
闭合图的性质恰好反映了事件之间的必要条件的关系：一个事件发生，它需要的所有前提都要发生。
下面通过构图，我们将最大权闭合图问题转化成最小割问题，即最大流问题求解。
定义 W[I]代表顶点 I 的权值，新增源点 S，汇点 T。
1、若 W[I]>0，则 S 向 I 连一条容量为 W[I]的边。
2、若 W[I]<0，则 I 向 T 连一条容量为-W[I]的边。
3、原图中的边，容量设置为正无穷。
这样，最小割就对应了最大权闭合图，而总盈利-最大流就是权和。
*/

最小费用最大流

```
//最小费用流 O(V*E*f) PKU2135
//错了试试 long long 或者把 MAXV 改小一点
//加边注意单向还是双向
#include <iostream>
using namespace std;
const int MAXN = 1001;
const int MAXM = 50000;
const int MAXV = 0x3F3F3F3F;
struct edge {
    int u, v, c, f, next;
} e[MAXM];
int N, M, S, T, E, head[MAXN];
int p[MAXN], dis[MAXN];
bool vis[MAXN];
queue<int> q;
void init() {
    memset(head, -1, sizeof(head));
    E = 0;
}
void add(int u, int v, int c, int f) { //u->v capacity = c,fee = f
    e[E].u = u; e[E].v = v; e[E].c = c; e[E].f = f; e[E].next = head[u];
    head[u] = E++;
}
void addedge(int u, int v, int c, int f) {
    add(u, v, c, f);
    add(v, u, 0, -f);//反向边流量 0 费用-c
}
bool SPFA(int S, int T, int N) { //寻找费用最小的可增广路
    memset(p, -1, N*sizeof(int));
    memset(vis, 0, N*sizeof(bool));
    memset(dis, 63, N*sizeof(int)); //最大费用-MAXV
    int u, v;
    vis[S] = 1;
    dis[S] = 0;
    q.push(S);
    while (!q.empty()) {
        u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i != -1; i = e[i].next) {
            if (e[i].c > 0) {
                v = e[i].v;
                if (dis[v] > dis[u] + e[i].f) { //最大费用<
                    dis[v] = dis[u] + e[i].f;
                    p[v] = i;
                    if (!vis[v]) {
                        q.push(v);
                        vis[v] = 1;
                    }
                }
```

```
            }
        }
    }
    return dis[T] != MAXV; //最大费用>
}
int mincost(int S, int T, int N) { //源 S 到汇 T 点数 N 的最小费用流
    int cost = 0, flow = 0;
    while (SPFA(S, T, N)) {
        int tflow = MAXV;
        for (int i = p[T]; i != -1; i = p[e[i].u]) //可改进量
            if (e[i].c < tflow)
                tflow = e[i].c;
        for (int i = p[T]; i != -1; i = p[e[i].u]) { //调整
            e[i].c -= tflow;
            e[i^1].c += tflow;
        }
        flow += tflow;
        cost += tflow * dis[T];
    }
    return cost;
    //return flow;//流量
}
int main() {
    while (scanf("%d%d", &N, &M) != EOF) {
        init();
        int x, y, z;
        S = 0;
        T = N + 1;
        for (int i = 0; i < M; i++) {
            scanf("%d%d%d", &x, &y, &z);
            addedge(x, y, 1, z);
            addedge(y, x, 1, z);
        }
        addedge(S, 1, 2, 0);
        addedge(N, T, 2, 0);
        printf("%d\n", mincost(S, T, N + 2));
    }
}
```

无向图全局最小割 STOER-WAGNER

```
/*
无向图全局最小割 Stoer-Wagner 算法 用求 prim 类似方法 O(N^3)
注意判断 n = 0 和 n = 1 的情况 用并查集判下连通性.
PKU2914
*/
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int MAXN = 501;
const int MAXV = 0x3F3F3F3F;
int n, m, v[MAXN], mat[MAXN][MAXN], dis[MAXN];
bool vis[MAXN];
int Stoer_Wagner(int n) {
    int res = MAXV;
    for (int i = 0; i < n; i++)
        v[i] = i;
    while (n > 1) {
        int maxp = 1, prev = 0;
        for (int i = 1; i < n; i++) {//初始化到已圈集合的割大小
            dis[v[i]] = mat[v[0]][v[i]];
            if (dis[v[i]] > dis[v[maxp]])
                maxp = i;
```

```
        }
        memset(vis, 0, sizeof(vis));
        vis[v[0]] = true;
        for (int i = 1; i < n; i++) {
            if (i == n - 1) { //只剩最后一个没加入集合的点，更新最小割
                res = min(res, dis[v[maxp]]);
                for (int j = 0; j < n; j++) { //合并最后一个点以及推出它的集合中的点
                    mat[v[prev]][v[j]] += mat[v[j]][v[maxp]];
                    mat[v[j]][v[prev]] = mat[v[prev]][v[j]];
                }
                v[maxp] = v[--n];//缩点后的图
            }
            vis[v[maxp]] = true;
            prev = maxp;
            maxp = -1;
            for (int j = 1; j < n; j++)
                if (!vis[v[j]]) { //将上次求的 maxp 加入集合，合并与它相邻的边到割集
                    dis[v[j]] += mat[v[prev]][v[j]];
                    if (maxp == -1 || dis[v[maxp]] < dis[v[j]])
                        maxp = j;
                }
        }
    }
    return res;
}
int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        memset(mat, 0, sizeof(mat));
        int x, y, z;
        while (m--) {
            scanf("%d%d%d", &x, &y, &z);
            mat[x][y] += z;
            mat[y][x] += z;
        }
        printf("%d\n", Stoer_Wagner(n));
    }
}
```

有根树的 HASH

```
//有根树的 hash 无根树枚举根
//HDU4013 求不同的子树数目
#include <iostream>
using namespace std;
typedef unsigned long long LL;
#define maxn 17
struct Edge {
    int v, x;
} E[555555];
int e;
struct AdjList {
    int l[maxn];
    void init() {
        memset(l, -1, sizeof(l));
    }
    inline void insert(int u, int v) {
        E[e].v = v;
        E[e].x = l[u];
        l[u] = e++;
    }
} adj, adj2;
LL rnd[maxn][maxn];
bool vst[maxn];
LL hash(int u, int D) {
```

```
    vst[u] = true;
    LL t[maxn];
    int j = 0;
    for (int i = adj2.l[u]; i >= 0; i = E[i].x) {
        int v = E[i].v;
        if ( vst[v] ) continue;
        t[j++] = hash(v, D + 1);
    }
    if ( j == 0 ) return 1;
    sort(t, t + j);
    LL ret = 0;
    for (int k = 0; k < j; k++) ret = ret + rnd[D][k] * t[k];
    return ret;
}
void dfs(int sel, int u) {
    vst[u] = true;
    for (int i = adj.l[u]; i >= 0; i = E[i].x) {
        int v = E[i].v;
        if ( vst[v] || (sel&(1 << v)) == 0 ) continue;
        adj2.insert(u, v);
        adj2.insert(v, u);
        dfs(sel, v);
    }
}
set<LL> st;
int main() {
    srand(time(NULL));
    for (int i = 0; i < maxn; i++) for (int j = 0; j < maxn; j++)
          rnd[i][j] = (LL)rand() * rand() * rand() * rand();
    int cas, Te = 1;
    scanf("%d", &cas);
    while ( cas-- ) {
        int n;
        scanf("%d", &n);
        e = 0;
        adj.init();
        st.clear();
        for (int i = 1; i < n; i++) {
            int u, v;
            scanf("%d %d", &u, &v);
            u--, v--;
            adj.insert(u, v);
            adj.insert(v, u);
        }
        int ans = 0;
        for (int sel = 1; sel < (1 << n); sel++) {
            memset(vst, false, sizeof(vst));
            adj2.init();
            for (int i = 0; i < n; i++) {
                if ( sel&(1 << i) ) {
                    dfs(sel, i);
                    break;
                }
            }
            bool can = true;
            int num = 0;
            for (int i = 0; i < n; i++) {
                if ( sel&(1 << i) ) {
                    num++;
                    if ( vst[i] == false) can = false;
                }
            }
            LL h[maxn];
            int j = 0;
            int cnt = 0;
            for (int i = 0; i < n; i++) {
```

```
            if ( sel&(1 << i) ) {
                memset(vst, false, sizeof(vst));
                h[j] = hash(i, 0);
                if ( st.find(h[j]) != st.end() ) cnt++;
                j++;
            }
        }
        if (cnt != num && can ) {
            ans++;
            for (int i = 0; i < j; i++) st.insert(h[i]);
        }
    }
    printf("Case #%d: %d\n", Te++, ans);
  }
}
```

三：数据结构

堆 STL 优先队列

```
struct data {
    int id, priority, value;
    char name[200];
    friend bool operator <(data a,data b) {
        return a.priority == b.priority ? a.id > b.id : a.priority > b.priority;
    }
};
const int MEM = 60001;
template <class T, class Pred = less<T> >
struct PQueue {
    int _end;
    T _q[MEM];
    Pred _cmp;
    PQueue(Pred cmp = Pred()): _end(0), _cmp(cmp) {}
    void clear() { _end = 0; }
    T& top() { return _q[0]; }
    const T& top() const { return _q[0]; }
    void push(const T& t) {
        _q[_end++] = t;
        push_heap(_q, _q + _end, _cmp);
    }
    void pop() { pop_heap(_q, _q + _end--, _cmp); }
    bool empty() const { return _end == 0; }
    int size() const { return _end; }
};
//priority_queue<data> q;
PQueue<data> q;
priority_queue<data> pq1; //默认最大堆
priority_queue<data, vector<data>, greater<data> > pq2; //最小堆
//priority_queue<int,vector<int>,less_equal<int> > pq2;
/*addition
equal_to      相等
not_equal_to   不相等
less          小于
greater       大于
less_equal     小于等于
greater_equal  大于等于
这些在所有的排序算法中同样适用
*/


//二叉堆 默认最大堆
#include <iostream>
using namespace std;
const int MAXN = 100001;
struct heap {
    int n,size,h[MAXN];
    void init() {
        size = n;
    }
    void up(int x) { //sift up
        while (x != 1 && h[x >> 1] < h[x]) {
```

```
            swap(h[x >> 1],h[x]);
            x >>= 1;
        }
    }
    void down(int x) { //sift down
        int pos;
        while ((pos = x << 1) <= size) {
            if (pos < size && h[pos+1] < h[pos])
                pos++;
            if (h[x] > h[pos])
                swap(h[x],h[pos]);
            else
                return;
            x = pos;
        }
    }
    void push(int x) { //insert a element
        h[++size] = x;
        up(size);
    }
    int pop() { //get and del minimum
        swap(h[1],h[size--]);
        down(1);
        return h[size+1];
    }
    int get() {//get minimum;
        return h[1];
    }
    void build() { //build a heap
        for (int i = size / 2;i > 0;i--)
            down(i);
    }
    void heapsort() { //sort h[]
        for (int i = size;i > 0;i--) {
            swap(h[1],h[size--]);
            down(1);
        }
        size = n;
    }
}hp;

int main() {
    scanf("%d",&hp.n);
    hp.init();
    for (int i = 1;i <= hp.size;i++)
        scanf("%d",&hp.h[i]);
    hp.build();
    hp.heapsort();
    for (int i = 1;i <= hp.size;i++)
        printf("%d ",hp.h[i]);
}
```

分数类

```
//8个8凑数
#include <iostream>
using namespace std;
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
struct Fraction {
    int num, den;//Numerator & Denominator 注意加减要long long
    Fraction(int n = 0, int d = 1) {
        int g = gcd(n, d);
```

```
        n /= g, d /= g;
        if (d < 0) {
            n = -n;
            d = -d;
        }
        num = n, den = d;
    }
};
Fraction operator +(const Fraction& lhs, const Fraction& rhs) {
    return Fraction(lhs.num * rhs.den + lhs.den * rhs.num, lhs.den * rhs.den);
}
Fraction operator -(const Fraction& lhs, const Fraction& rhs) {
    return Fraction(lhs.num * rhs.den - lhs.den * rhs.num, lhs.den * rhs.den);
}
Fraction operator *(const Fraction& lhs, const Fraction& rhs) {
    return Fraction(lhs.num * rhs.num, lhs.den * rhs.den);
}
Fraction operator /(const Fraction& lhs, const Fraction& rhs) {
    return Fraction(lhs.num * rhs.den, lhs.den * rhs.num);
}
bool operator <(const Fraction& lhs, const Fraction& rhs) { //map用
    return lhs.num == rhs.num ? lhs.den < rhs.den : lhs.num < rhs.num;
    //return lhs.num * rhs.den < rhs.num * lhs.den;//要比大小用这个
}
bool cmp(const Fraction& lhs, const Fraction& rhs) { //按大小排序
    return lhs.num * rhs.den < rhs.num * lhs.den;
}
map<Fraction, string> mp[9];
#define GAO(op) \
    t = x->first op y->first;\
    if (mp[i].count(t) == 0) {\
        mp[i][t] = "(" + x->second + #op + y->second + ")";\
    }
int main() {
    Fraction t;
    string s = "";
    int f = 0;
    for (int i = 1; i <= 8; i++) {
        f = f * 10 + 8;
        s += '8';
        mp[i][f] = s;
        for (int j = 1; j < i; j++) {
            int k = i - j;
            map<Fraction, string>::const_iterator x, y;
            for (x = mp[j].begin(); x != mp[j].end(); x++) {
                for (y = mp[k].begin(); y != mp[k].end(); y++) {
                    if (j <= k) {
                        GAO(+)
                        GAO(*)
                    }
                    if (y->first.num != 0) {
                        GAO( / )
                    }
                    GAO(-)
                }
            }
        }
    }
    while (scanf("%d", &f) != EOF) {
        t = Fraction(f);
        if (mp[8].count(t) == 0) {
            puts("Impossible");
        } else {
            s = mp[8][t];
            puts(s.c_str());
        }
```

```
    }
}
```

多项式类

```
//多项式,用于母函数展开
typedef double type;
struct Polynomial {
    map<int, type> mp;
    //*************************
    Polynomial() {
        mp.clear();
    }
    void init(vector< pair<int, type> > &v) {
        mp.clear();
        for (int i = 0; i < v.size(); i++) {
            mp[ v[i].first ] += v[i].second;
        }
    }
    void init(type val) {
        mp.clear();
        mp[0] = val;
    }
    Polynomial operator*(Polynomial &p) {
        map<int, type>::iterator it1, it2;
        Polynomial ret;
        ret.mp.clear();
        for (it1 = mp.begin(); it1 != mp.end(); ++it1) {
            for (it2 = p.mp.begin(); it2 != p.mp.end(); ++it2) {
                int n = it1->first + it2->first;
                ret.mp[ n ] += it1->second * it2->second;
            }
        }
        return ret;
    }
    void print() {
        map<int, type>::iterator it1;
        for (it1 = mp.begin(); it1 != mp.end(); ++it1) {
            cout << it1->second;
            printf("x^%d ", it1->first);
        }
        cout << endl;
    }
};

#include <iostream>
using namespace std;
const int MAXN = 100;
struct poly {
    int n;
    //map<int,double> m; //幂 系数
    double m[MAXN];
    poly() {
        n = MAXN;
        memset(m, 0, sizeof(m));
    }
    void print() {
        for (int i = 0, j = 0; i < n; i++) {
            if (m[i] == 0) continue;
            printf(j++ ? " + " : "");
            printf("%.2lfx^%d", m[i], i);
        }
        puts("");
    }
```

```
};
poly operator *(poly& a, poly& b) {
    poly c;
    for (int i = 0; i < a.n; i++) {
        for (int j = 0; j < b.n; j++) {
            c.m[i + j] += a.m[i] * b.m[j] ;
        }
    }
    return c;
}
poly operator +(poly& a, poly& b) {
    poly c;
    for (int i = 0; i < a.n; i++) {
        c.m[i] = a.m[i] + b.m[i];
    }
    return c;
}
int main() {
    poly a, b, c;
    a.m[0] = 1, a.m[5] = 2, b.m[6] = 3, b.m[7] = 4;
    c = a * b;
    c.print();
}
```

哈希表

```
//映射关系 TK -> TV, 映射范围_mod, 总节点数目_n
//默认没有重复的 key
const int MOD = 1999997;

template <class TK, class TV, int _MOD = 1999997, int _N = 2000000>
class HashTable {
public:
    struct H {
        TK key;
        TV val;
        int next;
    } h[_N];
    int head[_MOD], E;
    int (*hh) (TK);

    HashTable() { //构造函数
        clear();
    }
    void clear() { //清 0
        memset(head, -1, sizeof(head));
        E = 0;
    }
    void set(int (*hh)(TK)) { //设置回调函数(哈希函数)
        this->hh = hh;
    }
    int find(TK key) {  //查找关键字是否在哈希表中,没有返回-1
        int ind = hh(key);
        for (int p = head[ind]; p != -1; p = h[p].next) {
            if ( h[p].key == key ) return p;
        }
        return -1;
    }
    void insert(TK key, TV val) { //插入
        int ind = hh(key);
        h[E].key = key, h[E].val = val, h[E].next = head[ind];
        head[ind] = E++;
    }
```

```
    TV& operator[] (TK key) { //等同于 map 的[],支持没有 key 的查询,没有的话创建一个默认的
        int t = find(key);
        if ( t == -1 ) {
            insert(key, TV());
            return h[E - 1].val;
        } else return h[t].val;
    }
    void print(TK *out, int &N) { //0~N-1
        for (int i = 0; i < E; i++) out[i] = h[i].key;
        N = E;
    }
};

inline int h(LL s) {
    return s % MOD;
}
HashTable<LL, int> ht;
//***************模板结束*********************
int main() {
    ht.clear();
    ht.set(h);
    ht.insert(now.h, now.step);
    if (ht.find(123) == -1) {// || ht.h[ht.find(next.h)].val > next.step) {
    }
}
```

并查集

```
int fa[MAXN], cnt[MAXN];
int find(int x) {
    int p = x;
    while (p != fa[p])
        p = fa[p];
    while (x != fa[x]) {
        int t = fa[x];
        fa[x] = p;
        x = t;
    }
    return p;
}
void Union(int x, int y) {
    int fx = find(x), fy = find(y);
    if (fx == fy) return;
    fa[fy] = fx;
    cnt[fx] += cnt[fy];
    cnt[fy] = 0;
}
```

归并排序 逆序对

```
#include <iostream>
using namespace std;
const int MAXN = 10000;
const int INF = 0x3F3F3F3F;
int a[MAXN], n;
int MergeSort(int a[], int l, int mid, int r) {
    int n1 = mid - l + 1, n2 = r - mid, cnt = 0;
    int L[n1 + 1], R[n2 + 1];
    memcpy(L, a + l, n1 * sizeof(int));//L[i] = a[l + i],i∈(0,n1-1);
    memcpy(R, a + mid + 1, n2 * sizeof(int));//R[i] = a[mid + i + 1],i∈(0,n2-1);
    L[n1] = INF;//顺序对 -INF
```

```
    R[n2] = INF;//顺序对 -INF
    int i = 0, j = 0;
    for (int k = l; k <= r; k++) {
        if (L[i] > R[j]) { //顺序对 L[i] < R[j],可等时加=
            a[k] = R[j++];
            cnt += n1 - i;
        } else
            a[k] = L[i++];
    }
    return cnt;
}
int Merge(int a[], int l, int r) {
    int cnt = 0;
    if (l < r) {
        int mid = (l + r) >> 1;
        cnt += Merge(a, l, mid);
        cnt += Merge(a, mid + 1, r);
        cnt += MergeSort(a, l, mid, r);
    }
    return cnt;
}
#define _cp(a,b) ((a)<(b))//顺序对>,逆序对<,可等时加=
int _tmp[MAXN+1];
long long inv(int n, int *a) {
    int l = n >> 1, r = n - l, i, j;
    long long ret = (r > 1 ? (inv(l, a) + inv(r, a + l)) : 0);
    for (i = j = 0; i <= l; _tmp[i+j] = a[i], i++)
        for (ret += j; j < r && (i == l || !_cp(a[i], a[l+j])); _tmp[i+j] = a[l+j], j++);
    memcpy(a, _tmp, sizeof(int)*n);
    return ret;
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> a[i];
    cout << Merge(a, 0, n - 1) << endl;
    cout << inv(n, a) << endl;
}
```

RMQ 离线 ST

```
/*
RMQ 区间最值离线 ST 算法,复杂度 O(NlogN)
默认下标范围 0..n-1,求最大值
*/
#include <iostream>
#include <cmath>
using namespace std;
const int MAXN = 40001;
int n, a[MAXN], M[MAXN][16];//log2(40000);
void initRMQ(int n) {
    for (int i = 0; i < n; i++)
        M[i][0] = a[i];
    for (int j = 1; 1 << j <= n; j++)
        for (int i = 0; i + (1 << j) - 1 < n; i++)
            M[i][j] = max(M[i][j - 1], M[i + (1 << (j - 1))][j - 1]);
}
int RMQ(int l, int r) {
    int m = (int)floor(log(r - l + 1) / log(2.0));
    return max(M[l][m], M[r - (1 << m) + 1][m]);
}
int main() {
    scanf("%d",&n);
    for (int i = 0; i < n; i++)
```

```
        scanf("%d", &a[i]);
    initRMQ(n);
    int l, r;
    while (scanf("%d%d", &l, &r) != EOF) {
        printf("%d\n",RMQ(l, r));
    }
}
```

二维 RMQ

```
//二维 RMQ
#define maxn 152
#define LOG 8
int _log[maxn+1];
inline int getlog(int m) {
    return _log[m];
}
void RMQ(int d[][maxn], int M, int N, int ret[][maxn][LOG][LOG] ) { //下标从 1 开始,M 是行,N 是列
    for (int m = 1; m <= maxn; m++)     //需要计算很多次时耗时很大!!!需要预处理
        _log[m] = (int)(log(m * 1.0) / log(2.0));
    for (int i = 1; i <= M; i++)
        for (int j = 1; j <= N; j++)
            ret[i][j][0][0] = d[i][j];
    int row = getlog(M), col = getlog(N);
    for (int R = 0; R <= row; R++) {
        for (int C = 0; C <= col; C++) {
            for (int i = 1; i + (1 << R) - 1 <= M; i++) {
                for (int j = 1; j + (1 << C) - 1 <= N; j++) {
                    if ( R == 0 && C == 0 ) continue;
                    if ( R == 0 )
                        ret[i][j][R][C] = max(ret[i][j][R][C-1], ret[i][j + (1 << (C - 1))][R][C-1]);
                    else if ( C == 0 )
                        ret[i][j][R][C] = max(ret[i][j][R-1][C], ret[i + (1 << (R - 1))][j][R-1][C]);
                    else
                        ret[i][j][R][C] = max(max(ret[i][j][R - 1][C - 1], ret[i + (1 << (R - 1))][j][R
- 1][C - 1]),
                                              max(ret[i][j + (1 << (C - 1))][R - 1][C - 1], ret[i + (1 <<
(R - 1))][j + (1 << (C-1))][R - 1][C - 1]));
                }
            }
        }
    }
}
inline int query(int r1, int c1, int r2, int c2, int ret[][maxn][LOG][LOG]) {
    int R = getlog(r2 - r1 + 1);
    int C = getlog(c2 - c1 + 1);
    return max(max( ret[r1][c1][R][C], ret[r2-(1<<R)+1][c1][R][C]),
               max( ret[r1][c2-(1<<C)+1][R][C], ret[r2 - (1 << R) + 1][c2 - (1 << C) + 1][R][C])
              );
}
```

RMQ 点树 线段树

```
//点树 线段树 RMQ,支持单个端点的修改操作 HUST1387
#include <iostream>
using namespace std;
const int MAXN = 100010;
int a[MAXN];
struct node {
    int left, right;//[left,right] //线段树[left,right)
    int max;
```

```
} seg[MAXN*4];

void build(int l, int r, int p) { //l,r 分别为当前节点的左右端点，p 为节点在数组中的编号
    seg[p].left = l;
    seg[p].right = r;
    if (l == r) { //线段树 if (l + 1 == r) {
        seg[p].max = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, 2*p);
    build(mid + 1, r, 2*p + 1);//线段树 build(mid,r,2*p+1);
    seg[p].max = max(seg[2*p].max, seg[2*p+1].max);
}

void modify(int x, int p) { //线段树 void modify(int l,int r,int p) {
    if (seg[p].left == x && seg[p].right == x) {
        seg[p].max = a[x];
        return;
    }
    int mid = (seg[p].left + seg[p].right) >> 1;
    if (x <= mid) //线段树 r <= mid
        modify(x, 2*p);
    else if (x > mid)//线段树 l >= mid
        modify(x, 2*p + 1);
    seg[p].max = max(seg[2*p].max, seg[2*p+1].max);
}

int query(int l, int r, int p) {
    if (seg[p].left == l && seg[p].right == r)
        return seg[p].max;
    if (seg[p].left == seg[p].right) //线段树 seg[p].left + 1 == seg[p].right
        return seg[p].max;
    int mid = (seg[p].left + seg[p].right) >> 1;
    if (r <= mid)
        return query(l, r, 2*p);
    else if (l > mid)//线段树 l>= mid
        return query(l, r, 2*p + 1);
    else
        return max(query(l, mid, 2*p), query(mid + 1, r, 2*p + 1));//线段树 query(mid,r,2*p+1)
}

int main() {
    int n, m, x, y, z;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    build(1, n, 1);//线段树 build(1,n+1,1);!!!!!!!!!!
    for (int i = 0; i < m; i++) {
        scanf("%d%d%d", &z, &x, &y);
        if (z == 1) {
            a[x] = y;
            modify(x, 1);//线段树 modify(x,x+1,1);
        } else
            printf("%d\n", query(x, y, 1));
    }
}
```

树状数组

```
//树状数组
//支持①单点加值，询问整段和 ②整段加值，询问单点值 ③整段加值，询问整段和
#include <iostream>
using namespace std;
```

```
const int MAXN = 100001;
int a[MAXN];
//支持单点加一个值，询问整段和
//下标传入均为 1..n
struct BIT {//更新单点，询问整段
    int n, c[MAXN + 1];
    void init(int _n) {
        n = _n;
        memset(c, 0, sizeof(c));
    }
    void add(int x, int y) {
        while (x && x <= n) {
            c[x] += y;
            x += x & -x;
        }
    }
    int get(int x) {
        int ret = 0;
        while (x > 0) {
            ret += c[x];
            x -= x & -x;
        }
        return ret;
    }
    //统计个数时可以得到第 k 小数 复杂度 O(log(n))
    int find_kth(int k) {
        int now = 0;
        for (int i = 20; i >= 0; i--) {
            now |= (1 << i);
            if (now > n || c[now] >= k) {
                now ^= (1 << i);
            } else {
                k -= c[now];
            }
        }
        return now + 1;
    }
};
//支持整段加一个值，询问单点和
//下标传入均为 1..n
struct BIT2 {//更新单点，询问整段
    int n, c[MAXN + 1];
    void init(int _n) {
        n = _n;
        memset(c, 0, sizeof(c));
    }
    void add(int x, int y) {
        while (x > 0) {
            c[x] += y;
            x -= x & -x;
        }
    }
    int get(int x) {
        int ret = 0;
        while (x && x <= n) {
            ret += c[x];
            x += x & -x;
        }
        return ret;
    }
};
//支持整段加一个值，询问整段和
//下标传入均为 1..n
struct BIT3 {
    BIT tree;
    BIT2 tree2;
```

```
    void init(int n) {
        tree.init(n), tree2.init(n);
    }
    void add(int x, int y, int z) {//[x, y] += z
        if (y) tree.add(y, y * z);
        if (x - 1) tree.add(x - 1, (x - 1) * -z);
        if (y - 1) tree2.add(y - 1, z);
        if (x - 2) tree2.add(x - 2, -z);
    }
    int get(int x) {
        return tree.get(x) + tree2.get(x) * x;
    }
    int query(int x, int y) {
        return get(y) - get(x - 1);
    }
}bit;
int main() {
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF) {
        bit.init(MAXN);
        int x, y, z;
        for (int i = 0; i < m; i++) {
            scanf("%d%d%d", &x, &y, &z);//1 2 3
            bit.add(x, y, z);
        }
        for (int i = 0; i < m; i++) {
            scanf("%d%d", &x, &y);
            cout << bit.query(x, y) << endl;
        }
    }
}
```

浮点区间并

```
//浮点区间(a,b]求并及查询点是否在区间内
#include <iostream>
using namespace std;
const double eps = 1e-10;
const double pi = acos(-1.0);
const double inf = 1e200;
inline int sgn(double x) {
    return x < -eps ? -1 : x > eps ? 1 : 0;
}
// A data structure that maintains a union of intervals (a,b]
// on the real line. Adding new intervals and testing if a point
// is in the union is done in logarithmic time.
struct IntervalUnion {
    typedef map<double, double> Mdd;
    Mdd ints; // If a is in the map, ints[a] is
    // the end of the interval starting at a.
    IntervalUnion() {
        const double Inf = numeric_limits<double>::max();
        ints[-Inf] = -Inf;
    }
    bool contains(double a) {
        Mdd::iterator l = ints.lower_bound(a);//闭区间改upper_bound
        l--;
        return sgn(a - l->second) <= 0;
    }
    // Insert an interval (a,b] where a<b.
    void insert(double a, double b) { //[a,b]则a-eps
        if (a >= b) return;
        Mdd::iterator l = ints.lower_bound(a);//闭区间改upper_bound
        Mdd::iterator u = ints.upper_bound(b);
```

```
        l--, u--;
        double a2 = max(u->second, b);
        bool extend = (sgn(a - l->second) <= 0);
        if (extend) l->second = a2;
        l++, u++;
        ints.erase(l, u);
        if (!extend) ints[a] = a2;
    }
    bool contain(double a, double b) { //是否包含[a,b]
        //a += eps;
        Mdd::iterator l = ints.lower_bound(a);//闭区间改 upper_bound
        Mdd::iterator u = ints.upper_bound(b);
        l--, u--;
        return u == l && sgn(a - l->second) <= 0 && sgn(b - l->second) <= 0;
    }
};
int main() {
    IntervalUnion is;
    is.insert(1, 2);
    cout << is.contains(2.0) << endl;
}
```

线段树

```
/*
线段树+延迟操作+离散化求线段最多覆盖次数 HDU3577
询问是线段出现覆盖次数 插入是该线段覆盖次数+1
*/
#include <iostream>
#include <algorithm>
using namespace std;
const int MAXN = 100010;
int n, m, k, num, x[MAXN], y[MAXN];
bool flag;
struct node {
    int l, r;
    int cover;//该段被覆盖次数
    int inc;//给儿子的增量
    bool flag;//是否要延迟
    inline int mid() {
        return (l + r) >> 1;
    }
} seg[MAXN*10];
int idx[MAXN*2];
int id(int x) {
    return lower_bound(idx, idx + num, x) - idx;
}
void build(int l, int r, int p) {
    seg[p].l = l, seg[p].r = r;
    seg[p].cover = seg[p].flag = seg[p].inc = 0;
    if (l + 1 == r) return;
    int mid = seg[p].mid();
    build(l, mid, 2 * p);
    build(mid, r, 2 * p + 1);
}
void down(int p) {
    if (seg[p].l + 1 == seg[p].r) return;
    seg[2 * p].flag = seg[2 * p + 1].flag = 1;
    seg[2 * p].cover += seg[p].inc;
    seg[2 * p + 1].cover += seg[p].inc;
    seg[2 * p].inc += seg[p].inc;
    seg[2 * p + 1].inc += seg[p].inc;
    seg[p].inc = 0;
    seg[p].flag = 0;
```

```
}
void insert(int l, int r, int p) {
    if (seg[p].l == l && seg[p].r == r) {
        seg[p].flag = 1;
        seg[p].cover++;
        seg[p].inc++;
        down(p);
        return;
    }
    if (seg[p].flag)
        down(p);
    int mid = seg[p].mid();
    if (r <= mid)
        insert(l, r, 2 * p);
    else if (l >= mid)
        insert(l, r, 2 * p + 1);
    else {
        insert(l, mid, 2 * p);
        insert(mid, r, 2 * p + 1);
    }
    seg[p].cover = max(seg[2 * p].cover, seg[2 * p + 1].cover);
}
int query(int l, int r, int p) {
    if (seg[p].l + 1 == seg[p].r)
        return seg[p].cover;
    if (seg[p].flag)
        down(p);
    if (seg[p].l == l && seg[p].r == r)
        return seg[p].cover;
    int mid = seg[p].mid();
    if (r <= mid)
        return query(l, r, 2 * p);
    else if (l >= mid)
        return query(l, r, 2 * p + 1);
    else
        return max(query(l, mid, 2 * p), query(mid, r, 2 * p + 1));
}
int main() {
    int cas;
    scanf("%d", &cas);
    for (int T = 1; T <= cas; T++) {
        scanf("%d%d", &k, &m);
        num = 0;
        for (int i = 1; i <= m; i++) {
            scanf("%d%d", &x[i], &y[i]);
            idx[num++] = x[i];
            idx[num++] = y[i];
        }
        sort(idx, idx + num);
        num = unique(idx, idx + num) - idx;
        build(1, num + 1, 1);
        printf("Case %d:\n", T);
        for (int i = 1; i <= m; i++) {
            if (query(id(x[i]) + 1, id(y[i]) + 1, 1) < k) {
                insert(id(x[i]) + 1, id(y[i]) + 1, 1);
                printf("%d ", i);
            }
        }
        printf("\n\n");
    }
}
```

线段树维护区间增量

```
//线段树维护一段区间增量
//用延迟标记复杂度O(NlogN)
//HDU4037 需要维护c[i], c[i] * i, d[i], d[i] * i, d[i] * i^2
#include <iostream>
using namespace std;
const int MAXN = 100010;
const int MOD = 20110911;
const int INV = 10055456;
int n, m;
int c[MAXN], d[MAXN];
inline int add(int& x, int y) {
    return x = (x + y) % MOD;
}
inline int mul(int x, int y) {
    return x * (long long)y % MOD;
}
int sumi[MAXN], sumii[MAXN];
struct node {
    int l, r;
    int sumci, sumcii, sumdi, sumdii, sumdiii;
    int addc, addd;
    void init(int i) {
        sumci = c[i];
        sumcii = mul(c[i], i);
        sumdi = d[i];
        sumdii = mul(d[i], i);
        sumdiii = mul(mul(d[i], i), i);
    }
    void upc(int c) {
        add(addc, c);
        add(sumci, mul(c, (r - 1) - (l - 1)));
        add(sumcii, mul(c, (sumi[r - 1] - sumi[l - 1] + MOD + MOD) % MOD));
    }
    void upd(int d) {
        add(addd, d);
        add(sumdi, mul(d, (r - 1) - (l - 1)));
        add(sumdii, mul(d, (sumi[r - 1] - sumi[l - 1] + MOD + MOD) % MOD));
        add(sumdiii, mul(d, (sumii[r - 1] - sumii[l - 1] + MOD + MOD) % MOD));
    }
    inline int mid() {
        return (l + r) >> 1;
    }
}seg[MAXN << 2];
void down(int p) {//pushdown
    if (seg[p].l + 1 == seg[p].r) {
        seg[p].addc = seg[p].addd = 0;
        return;
    }
    for (int i = 0; i < 2; i++) {
        node& son = seg[2 * p + i];
        son.upc(seg[p].addc);
        son.upd(seg[p].addd);
    }
    seg[p].addc = seg[p].addd = 0;
}
void up(int p) {//update
    if (seg[p].l + 1 == seg[p].r) return;
    seg[p].sumci = (seg[2 * p].sumci + seg[2 * p + 1].sumci) % MOD;
    seg[p].sumcii = (seg[2 * p].sumcii + seg[2 * p + 1].sumcii) % MOD;
    seg[p].sumdi = (seg[2 * p].sumdi + seg[2 * p + 1].sumdi) % MOD;
    seg[p].sumdii = (seg[2 * p].sumdii + seg[2 * p + 1].sumdii) % MOD;
    seg[p].sumdiii = (seg[2 * p].sumdiii + seg[2 * p + 1].sumdiii) % MOD;
}
void build(int l, int r, int p) {
    seg[p].l = l, seg[p].r = r;
    seg[p].addc = seg[p].addd = 0;
```

```
    if (l + 1 == r) {
        seg[p].init(l);
        return;
    }
    int mid = seg[p].mid();
    build(l, mid, 2 * p);
    build(mid, r, 2 * p + 1);
    up(p);
}
void update(int l, int r, int p, int c, int d) {
    if (seg[p].l == l && seg[p].r == r) {
        seg[p].upc(c);
        seg[p].upd(d);
        return;
    }
    if (seg[p].addc || seg[p].addd) {
        down(p);
    }
    int mid = seg[p].mid();
    if (r <= mid) {
        update(l, r, 2 * p, c, d);
    } else if (l >= mid) {
        update(l, r, 2 * p + 1, c, d);
    } else {
        update(l, mid, 2 * p, c, d);
        update(mid, r, 2 * p + 1, c, d);
    }
    up(p);
}
node query(int l, int r, int p) {
    if (seg[p].l == l && seg[p].r == r) {
        return seg[p];
    }
    if (seg[p].addc || seg[p].addd) {
        down(p);
    }
    int mid = seg[p].mid();
    node t, tt;
    if (r <= mid) {
        t = query(l, r, 2 * p);
    } else if (l >= mid) {
        t = query(l, r, 2 * p + 1);
    } else {
        t = query(l, mid, 2 * p);
        tt = query(mid, r, 2 * p + 1);
        add(t.sumci, tt.sumci);
        add(t.sumcii, tt.sumcii);
        add(t.sumdi, tt.sumdi);
        add(t.sumdii, tt.sumdii);
        add(t.sumdiii, tt.sumdiii);
    }
    up(p);
    return t;
}
int main() {
    for (int i = 1; i < MAXN; i++) {
        sumi[i] = (sumi[i - 1] + i) % MOD;
        sumii[i] = (sumii[i - 1] + (LL)i * i) % MOD;
    }
    int cas, T = 0;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            scanf("%d%d", &c[i], &d[i]);
        }
```

```
        build(1, n + 1, 1);
        char cmd[10];
        int x, y, z, ans;
        node res;
        scanf("%d", &m);
        printf("Case %d:\n", ++T);
        while (m--) {
            scanf(" %s %d %d", cmd, &x, &y);
            if (cmd[0] == 'Q') {
                res = query(x, y + 1, 1);
                ans = (mul(-res.sumci, y + 1) + res.sumcii + MOD + MOD) % MOD;
                add(ans, mul(mul(res.sumdi, (mul(y, y) + y) % MOD), INV));
                add(ans, mul(mul(res.sumdii, -2 * y - 1), INV));
                add(ans, mul(res.sumdiii, INV));
                printf("%d\n", ans);
            } else if (cmd[2] == 's') {
                scanf("%d", &z);
                update(x, y + 1, 1, z, 0);
            } else if (cmd[2] == 'l') {
                scanf("%d", &z);
                update(x, y + 1, 1, 0, z);
            }
        }
    }
}
```

SPLAY

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <cmath>
#include <algorithm>
using namespace std;

const int INF = 0x3F3F3F3F;
const int MAXN = 2000100;
const int MAXMEM = 4 * MAXN;

struct node {
    int val; //val
    int size;
    node *fa, *ch[2]; //左 0 右 1

    node() {}
    void init() {
        val = 0; //val
        size = 1;
        fa = ch[0] = ch[1] = 0;
    }

    node* get_end(int d) {
        node* x = this;
        while (x->ch[d]) {
            x->down();
            x = x->ch[d];
        }
        return x;
    }

    node* get_near(int d) {
        if (ch[d])
            return ch[d]->get_end(!d);
        node* x = this;
```

```
        while (x->fa && x->fa->ch[d] == x)
            x = x->fa;
        return x->fa;
    }

    node* get_prev() {return get_near(0);}
    node* get_succ() {return get_near(1);}

    int get_size() {
        return this ? size : 0;
    }

    void down() {
    }

    void up() {
        size = 1 + ch[0]->get_size() + ch[1]->get_size();
    }
};

struct Splay {
    node *root, mem[MAXMEM];
    node *nil; //默认就是 NULL
    int ind;

    void init() {
        ind = 0;
        root = 0;
    }

    void build(int a, int b) {
        root = make_tree(a, b);
        root->fa = nil;
    }

    node* make_tree(int a, int b) {
        if (a > b) return 0;
        int pos = (a + b) >> 1;
        node *now = mem + pos;
        now->ch[0] = make_tree(a, pos - 1);
        if (now->ch[0])
            now->ch[0]->fa = now;
        now->ch[1] = make_tree(pos + 1, b);
        if (now->ch[1])
            now->ch[1]->fa = now;
        now->up();
        return now;
    }

    void rotate(node* x, int d) { //0 左旋 1 右旋
        node *y = x->fa;
        y->down();x->down(); //y 在 x 上 先 y 后 x
        y->ch[!d] = x->ch[d];
        if (x->ch[d])
            x->ch[d]->fa = y;
        x->fa = y->fa;
        if (y->fa != nil)
            y->fa->ch[y->fa->ch[0] != y] = x;
        else
            root = x;
        x->ch[d] = y;
        y->fa = x;
        y->up();
    }

    void splay(node* x, node* f) { //把 x 转到 f 下
```

```
    x->down();
    while (x->fa != f) {
        if (x->fa->fa == f) //父结点的父亲为 f 执行单旋
            rotate(x, x->fa->ch[0] == x);
        else {
            node *y = x->fa, *z = y->fa;
            if (z->ch[0] == y) {
                if (y->ch[0] == x)
                    rotate(y, 1),rotate(x, 1); //zagzag
                else
                    rotate(x, 0),rotate(x, 1); //zigzag
            }
            else {
                if (y->ch[1] == x)
                    rotate(y, 0),rotate(x, 0); //zigzig
                else
                    rotate(x, 1),rotate(x, 0); //zagzig
            }
        }
    }
    x->up();
}

void select(int k, node* f) { // 将中序遍历的第 k 个结点旋转到结点 f 下面
    int tmp;
    node *x = root;
    while (1) {
        x->down();
        tmp = x->ch[0]->get_size();
        if (k == tmp + 1) break;
        if (k <= tmp) {
            x = x->ch[0];
        } else {
            k -= tmp + 1;
            x = x->ch[1];
        }
    }
    splay(x,f);
}

node* find(int v) { //val
    if (!root) return 0;
    node *x = root;
    while (x) {
        if (x->val == v)
            break;
        x = x->ch[v > x->val];
    }
    if (!x) return 0;
    splay(x, nil);
    return x;
}

node* get_kth(int k) {
    int tmp;
    node *x = root;
    while (1) {
        x->down();
        tmp = x->ch[0]->get_size();
        if (k == tmp + 1) break;
        if (k <= tmp)
            x = x->ch[0];
        else {
            k -= tmp + 1;
            x = x->ch[1];
        }
```

```
    }
    splay(x,nil);
    return root;
}

node* lower_bound(int v) { //寻找第一个 >= v 的地址 超出最大的返回 0
    node *x = root, *y = nil;
    while (x) {
        y = x;
        x = x->ch[v > x->val];
    }
    if (y == nil)
        return 0;
    if (v <= y->val)
        return y;
    else
        return y->get_succ();
}

int query_less(int val) {
    node* x = lower_bound(val);
    if (x == 0)                // 全部都小于
        return root->get_size();
    splay(x, nil);              // 通过 splay 提高效率
    return x->ch[0]->get_size();
}

int query_rank(int val) {
    return query_less(val) + 1;
}

int count() { //先加入 INF 和-INF
    return query_less(INF) - 1;
}

node* insert(int v) { //val
    //node *t = find(v);
    //if (t) return t; //是否可重复加入键值
    node *nd = &mem[ind++];
    nd->init();
    nd->val = v;
    if (!root) {
        root = nd;
        root->fa = nil;
        return root;
    }
    node *x = root, *y = nil;
    while (x) {
        y = x;
        x = x->ch[v > x->val];
    }
    x = nd;
    x->fa = y;
    y->ch[v > y->val] = x;
    splay(x, nil);
    return x;
}

void del(node* x) { //删除结点 x
    splay(x, nil);
    node *y = x->get_prev(), *z = x->get_succ();
    if (y && z) {
        splay(y, nil);
        splay(z, root);
        z->ch[0] = 0;
        z->up();
```

```
            y->up();
        }
        else if (!y && z) {
            splay(z, nil);
            z->ch[0] = 0;
            z->up();
        }
        else if (y && !z) {
            splay(y, nil);
            y->ch[1] = 0;
            y->up();
        }
        else if (!y && !z) {
            root = 0;
        }
    }

    void remove(int av,int bv) {
        //a是 >= av的pos b是 > bv的pos
        int a = query_less(av) + 1,b = query_less(bv + 1) + 1;
        select(a, nil);
        node *aa = root->get_prev();
        select(b, nil);
        node *bb = root;
        splay(aa, nil);
        splay(bb, root);
        bb->ch[0] = 0;
        bb->up();
        aa->up();
    }

    void graph(node *root, int offset) {
        if (root == NULL) {
            for (int i = 0; i < offset; i++)
                putchar('\t');
            puts("~");
        }
        else {
            graph(root->ch[1], offset + 1);
            for (int i = 0; i < offset; i++)
                putchar('\t');
            printf("%d\n", root->val);
            graph(root->ch[0], offset + 1);
        }
    }

};
Splay tree;
int main() {
    int n, min;
    while (scanf("%d%d", &n, &min) != EOF) {
        tree.init();
        char c;
        int k, add = 0, sum = 2;
        getchar();
        tree.insert(-INF);
        tree.insert(INF);
        for (int i = 0; i < n; i++) {
            scanf("%c%d", &c, &k);
            getchar();
            if (c == 'I') {
                if (k >= min) {
                    tree.insert(k - add);
                    sum++;
                }
            } else if (c == 'A') {
```

```
            add += k;
        } else if (c == 'S') {
            add -= k;
            tree.remove(-INF + 1, min - add - 1);
        } else if (c == 'F') {
            if (tree.root->get_size() >= k + 2) {
                tree.select(tree.root->get_size() - k, tree.nil);
                printf("%d\n", tree.root->val + add);
            } else {
                printf("-1\n");
            }
        }
    }
    printf("%d\n", sum - tree.root->get_size());
  }
}
```

虚二叉树

```
//虚二叉树 Sicily1802
#include <iostream>
using namespace std;
const int N = 131072;//2 << 16
const int INF = 0x3F3F3F3F;
struct node {
    int minv, maxv;
    int mindiff;
}seg[N << 1];
void init() {
    for (int i = 1; i < (N << 1); i++) {
        seg[i].minv = INF;
        seg[i].maxv = -INF;
        seg[i].mindiff = INF;
    }
}
inline int get_parent(int p) {
    return (p >> 1);
}
inline int get_left(int p) {
    return (p << 1);
}
inline int get_right(int p) {
    return (p << 1) + 1;
}
inline int get_node(int x) {
    return N + x - 1;
}
void update(int p) {
    int left = get_left(p), right = get_right(p);
    seg[p].minv = min(seg[left].minv, seg[right].minv);
    seg[p].maxv = max(seg[left].maxv, seg[right].maxv);
    seg[p].mindiff = min(seg[left].mindiff, seg[right].mindiff);
    seg[p].mindiff = min(seg[p].mindiff, seg[right].minv - seg[left].maxv);
}
void insert(int x) {
    int p = get_node(x);
    if (seg[p].minv != INF) return;
    seg[p].minv = x;
    seg[p].maxv = x;
    do {
        p = get_parent(p);
        update(p);
    } while (p > 1);
}
```

```
void remove(int x) {
    int p = get_node(x);
    if (seg[p].minv == INF) return;
    seg[p].minv = INF;
    seg[p].maxv = -INF;
    do {
        p = get_parent(p);
        update(p);
    } while (p > 1);
}
int n, m;
int main() {
    int cas;
    scanf("%d", &cas);
    for (int T = 0; T < cas; T++) {
        if (T) puts("");
        scanf("%d", &m);
        char s[20];
        int x;
        init();
        for (int i = 0; i < m; i++) {
            scanf("%s", s);
            if (s[0] == 'g') {
                scanf("%d", &x);
                insert(x);
            } else if (s[0] == 'r') {
                scanf("%d", &x);
                remove(x);
            } else if (s[0] == 'q') {
                if (seg[1].minv == INF || seg[1].maxv == INF || (seg[1].minv == seg[1].maxv)) {
                    printf("-1\n");
                } else {
                    printf("%d\n", seg[1].mindiff);
                }
            }
        }
    }
}
```

划分树

```
/*
划分树
从整体来看是一棵二叉树,最顶上是一个节点,第二层是两个节点...
其次还是一棵划分树,第 k+1 层是对第 k 层的一个划分(二分)
*/
/*
hdu 3473 给你区间[a,b],求 sigma| x-val[i] |的最小值
输入 5 3 6 2 2 4 2 1 4 0 2 输出 6 4
*/
//划分树 支持重复数 寻找第 K 大数
//PKU2761 PKU2104 HDU2665 HDU3473
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int MAXN = 100001;
const int LOGN = 20;
struct node {
    int l, r;
    inline int mid() {
        return (l + r) >> 1;
    }
```

```
}seg[MAXN * 4];
int len[MAXN], sorted[MAXN];
int val[LOGN][MAXN], toleft[LOGN][MAXN];
void build(int l, int r, int d, int p) {
    seg[p].l = l, seg[p].r = r;
    if (seg[p].l == seg[p].r) return;
    int mid = seg[p].mid();
    int lsame = mid - l + 1;//lsame 表示和 val[mid]相等且分到左边的
    for (int i = l; i <= r; i++) {
        if (val[d][i] < sorted[mid]) {
            lsame--;//减去不符合的
        }
    }
    int lpos = l, rpos = mid + 1, same = 0;
    for (int i = l; i <= r; i++) {
        toleft[d][i] = i == l ? 0 : toleft[d][i - 1];//toleft[i]表示[tt[idx].left, i]中分到左边的数个
数
        if (val[d][i] < sorted[mid]) {
            toleft[d][i]++;
            val[d + 1][lpos++] = val[d][i];
        } else if (val[d][i] > sorted[mid]) {
            val[d + 1][rpos++] = val[d][i];
        } else {
            if (same < lsame) {//lsame 个分到左边
                same++;
                toleft[d][i]++;
                val[d + 1][lpos++] = val[d][i];
            } else {
                val[d + 1][rpos++] = val[d][i];
            }
        }
    }
    build(l, mid, d + 1, 2 * p);
    build(mid + 1, r, d + 1, 2 * p + 1);
}
int query(int l, int r, int k, int d, int p) {
    if (l == r) {
        return val[d][l];
    }
    int a, b;//a 表示[l, r]分到左边个数, b 表示[seg[p].l, l - 1]分到左边个数
    b = l == seg[p].l ? 0 : toleft[d][l - 1];
    a = toleft[d][r] - b;
    if (a >= k) {//在左边找
        int newl = seg[p].l + b;
        int newr = seg[p].l + b + a - 1;
        return query(newl, newr, k, d + 1, 2 * p);
    } else {//在右边找
        int mid = seg[p].mid();
        int x = l - seg[p].l - b;//x 表示[seg[p].l, l - 1]分到右边个数
        int y = r - l + 1 - a;//y 表示[l, r]分到右边个数
        int newl = mid + x + 1;
        int newr = mid + x + y;
        return query(newl, newr, k - a, d + 1, 2 * p + 1);
    }
}
int main() {
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF) {
        for (int i = 1; i <= n; i++) {
            scanf("%d", &val[0][i]);
            sorted[i] = val[0][i];
        }
        sort(sorted + 1, sorted + n + 1);
        build(1, n, 0, 1);
        while (m--) {
            int l, r, k;
```

```
            scanf("%d%d%d", &l, &r, &k);
            printf("%d\n", query(l, r, k, 0, 1));
        }
    }
}
```

左偏树 LEFTISTTREE

```
/*
HDU1512 HDU1434
1 左偏树(Leftist Tree)是一种可并堆(Mergeable Heap) ，它除了支持优先队列的三个基本操作(插入，删除，取最小节点)，还支持一个很特殊的操作—合并操作。
2 左偏树是一棵堆有序(Heap Ordered)二叉树。
3 左偏树满足左偏性质(Leftist Property)。
[性质 1] 节点的键值小于或等于它的左右子节点的键值。
[性质 2] 节点的左子节点的距离不小于右子节点的距离。
[性质 3] 节点的左子节点右子节点也是一颗左偏树。
http://www.dgp.toronto.edu/people/JamesStewart/378notes/10leftist/
*/
//本题选择其中的两个猴子，如果他们是不同群体的话，就选出各自最大的一个，然后 PK，减半，并合并在一起。
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
const int MAXN = 100001;
struct node {
    int dist;
    int key, left, right, fa;
}nd[MAXN];
int n, m;
int merge(int a, int b) {
    if (!a || !b) return a ? a : b;
    if (nd[a].key < nd[b].key) {//最大堆<
        swap(a, b);
    }
    nd[a].right = merge(nd[a].right, b);
    nd[nd[a].right].fa = a;
    if (nd[nd[a].left].dist < nd[nd[a].right].dist) {
        swap(nd[a].left, nd[a].right);
    }
    nd[a].dist = nd[nd[a].right].dist + 1;
    return a;
}
int pop(int x) {
    int l = nd[x].left, r = nd[x].right;
    nd[l].fa = l, nd[r].fa = r;
    nd[x].left = nd[x].right = nd[x].dist = 0;
    return merge(l, r);
}
int find(int x) {
    return nd[x].fa == x ? x : find(nd[x].fa);
}
int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; i++) {
            scanf("%d", &nd[i].key);
            nd[i].left = nd[i].right = nd[i].dist = 0;
            nd[i].fa = i;
        }
        scanf("%d", &m);
        while (m--) {
            int a, b;
```

```
            scanf("%d%d", &a, &b);
            int ida = find(a), idb = find(b);
            if (ida == idb) {
                printf("-1\n");
            } else {
                nd[ida].key /= 2;
                int u = pop(ida);
                u = merge(u, ida);
                nd[idb].key /= 2;
                int v = pop(idb);
                v = merge(v, idb);
                printf("%d\n", nd[merge(u, v)].key);
            }
        }
    }
}
```

笛卡尔树

```
/*
笛卡尔树(就是一棵 Treap 的 O(n)构造过程)
zju 1985 一列宽为 1 高为 height[i]的木棍,求最大延伸面积,
利用了 T=i,H=height[i],然后父子之间夹住的区间.....类似 splay 的想法...
*/
#include<iostream>
#include<stdio.h>
using namespace std;
#define maxn 110000
struct data {
    int T, H;
    data *ch[2]; //Tree,Heap
    void init(int _T, int _H) {
        T = _T;
        H = _H;
        ch[0] = ch[1] = NULL;
    }
    friend bool operator<(const data &a, const data &b) {
        return a.T < b.T;
    }
} dd[maxn], *root;

//节点 0~n-1
data *st[maxn];
int top;
void makeTree(int n) {
    top = 0;
    for (int i = 0; i < n; i++) {
        data *lson = NULL;
        while ( top > 0 ) {
            if ( dd[i].H < st[top-1]->H ) {
                lson = st[--top];
            } else break;
        }
        if ( top == 0 ) {
            dd[i].ch[0] = lson;
            root = &dd[i];
        } else {
            st[top-1]->ch[1] = &dd[i];
            dd[i].ch[0] = lson;
        }
        st[top++] = &dd[i];
    }
}
long long ans;
```

```
int dfs(data*p) {
    if ( p == NULL ) return 0;
    int num = 1;
    num += dfs(p->ch[0]);
    num += dfs(p->ch[1]);
    ans = max(ans, (long long)num * p->H);
    return num;
}
int main() {
    int n;
    while ( scanf("%d", &n) && n) {
        for (int i = 0; i < n; i++) {
            int H;
            scanf("%d", &H);
            dd[i].init(i, H);
        }
        makeTree(n);
        ans = 0;
        dfs(root);
        printf("%lld\n", ans);
    }
}
```

动态中位数

```
//动态增加查询中位数 复杂度O(mlogn) TJU3515 http://acm.tju.edu.cn/toj/showp3515.html
//弄两个堆，根据大小动态调整
#include <iostream>
using namespace std;
const int MAXN = 100000;
int a[MAXN];
priority_queue<int,vector<int>,greater<int> > big;
priority_queue<int> small;
int main() {
    int cas,n,m;
    scanf("%d",&cas);
    while (cas--) {
        while (!small.empty()) small.pop();
        while (!big.empty()) big.pop();
        scanf("%d",&n);
        for (int i = 0;i < n;i++) scanf("%d",&a[i]);
        sort(a,a + n);
        int mid;
        int pos = n & 1 ? n / 2 : n / 2 - 1;
        for (int i = 0;i < pos;i++) small.push(a[i]);
        mid = a[pos];
        for (int i = pos + 1;i < n;i++) big.push(a[i]);
        scanf("%d",&m);
        char s[4];
        int x;
        while (m--) {
            scanf("%s",s);
            if (s[0] == 'a') {
                scanf("%d",&x);
                if (x < mid) small.push(x);
                else big.push(x);
                if (small.size() > big.size()) {
                    big.push(mid);
                    mid = small.top();
                    small.pop();
                } else if (big.size() > small.size() + 1) {
                    small.push(mid);
                    mid = big.top();
                    big.pop();
```

```
            }
        } else {
            printf("%d\n",mid);
        }
    }
  }
}
```

二维矩形离线求和

```
//ZJUT1714 HDU3890
//http://blog.csdn.net/pouy94/article/details/5967746
//http://blog.csdn.net/jasonzhu8/article/details/5962599
//能支持单点增/减/乘以及一个矩形和的询问
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int MAXN = 50001;
const int MAXQ = 200001;
struct BIT {
    int _n, c[MAXN];
    int t[MAXN], ts;//time stamp avoid memset
    void add(int x, int y) {
        while (x <= _n + 1) {
            if (t[x] != ts) {
                c[x] = 0, t[x] = ts;
            }
            c[x] += y;
            x += x & -x;
        }
    }
    int get(int x) {
        int ret = 0;
        while (x > 0) {
            if (t[x] != ts) {
                c[x] = 0, t[x] = ts;
            }
            ret += c[x];
            x -= x & -x;
        }
        return ret;
    }
};
int idx, qid;//总操作个数 询问个数
struct query {
    int type, id;//type = 0 表示更新, type = 1, 2, 3, 4 表示询问, id 对于询问是编号，对于增加操作是值
    int x, y;
    bool operator <(const query& q) const {
        return x == q.x ? type < q.type : x < q.x;
    }
}q[MAXQ], tmp[MAXQ];
int ans[MAXQ];
const int dv[] = {0, 1, -1, -1, 1};
struct matrix {
    int n, m;
    BIT tree;
    void init(int _n, int _m) {
        n = _n;
        tree._n = _m;
    }
    void work(query q[], int l, int r) {
        if (l >= r) return;
```

```
        int mid = (l + r) >> 1;
        work(q, l, mid), work(q, mid + 1, r);
        int s1 = l, s2 = mid + 1;
        //优先做左边段的更新操作，因为会对右边段的询问操作产生影响
        tree.ts++;//要清空一下树状数组
        for (int i = l; i <= r; i++) {
            if ((s2 > r) || (s1 <= mid && s2 <= r && q[s1] < q[s2])) {
                if (q[s1].type == 0) {
                    tree.add(q[s1].y + 1, q[s1].id);
                }
                tmp[i] = q[s1++];
            } else {
                if (q[s2].type != 0) {
                    ans[q[s2].id] += dv[q[s2].type] * tree.get(q[s2].y + 1);
                }
                tmp[i] = q[s2++];
            }
        }
        memcpy(q + l, tmp + l, (r - l + 1) * sizeof(query));
    }
}M;
int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        idx = qid = 0;
        int t;
        while (1) {
            scanf("%d", &t);
            if (t == 3) {
                break;
            } else if (t == 1) {
                scanf("%d%d%d", &q[idx].x, &q[idx].y, &q[idx].id);
                q[idx++].type = 0;
            } else if (t == 2) {
                int x1, y1, x2, y2;
                scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
                q[idx].x = x1 - 1, q[idx].y = y1 - 1, q[idx].id = qid, q[idx++].type = 1;
                q[idx].x = x1 - 1, q[idx].y = y2    , q[idx].id = qid, q[idx++].type = 2;
                q[idx].x = x2    , q[idx].y = y1 - 1, q[idx].id = qid, q[idx++].type = 3;
                q[idx].x = x2    , q[idx].y = y2    , q[idx].id = qid, q[idx++].type = 4;
                ans[qid++] = 0;
            }
        }
        M.init(n, n);
        M.work(q, 0, idx - 1);
        for (int i = 0; i < qid; i++) {
            printf("%d\n", ans[i]);
        }
    }
}
```

四：字符串

KMP

```
//HDU1711 PKU1961 PKU2406 PKU2752
//http://www.if-yu.info/2010/08/21/mp-and-kmp.html
//http://www-igm.univ-mlv.fr/~lecroq/string/node8.html#SECTION0080
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int MAXN = 1000001;
const int MAXM = 10001;
int next[MAXN];
char a[MAXN], b[MAXM];
/*
当一个字符串以 0 为起始下标时，next[i]可以描述为"不为自身的最大首尾重复子串长度"
next[i] == max{ j | pattern[0:j) == pattern[i-j+1:i+1) and 0<j<i }
也就是当前位置上保证前缀等于后缀的最大长度
str[0,next[i] - 1] == str[i - next[i] + 1, i];
next[j] != 0 表示在 j 字符前存在一个长度为 next[j]-1 的重复子串
next[n]表示最后一次匹配的位置 求重复可用 d = n - next[n];
*/
void getnext(char s[], int next[]) {
    next[0] = -1;
    for (int i = 0, j = -1; s[i];) {
        while (j != -1 && s[i] != s[j])
            j = next[j];
        i++, j++;
        next[i] = j;//s[i] == s[j] ? next[j] : j;
        //要求前后缀时 next[i] = j;
    }
}
int kmp(char s1[], char s2[], int n, int m) { //s1 中找 s2
    getnext(s2, next); //计算 next 数组
    for (int i = 0, j = 0; i < n;) {
        while (j != -1 && s1[i] != s2[j])
            j = next[j]; // 若某一位不相等，则子串根据 next 数组与母串对齐
        i++, j++;
        if (j >= m) {
            return i - j; //找多次不 return
            j = next[j];
        }
    }
    return -1;
}
int main() {
    while (scanf("%s%s", a, b) != EOF) {
        getnext(b, next);
        for (int i = 0; i < strlen(b); i++)
            printf("%d ", next[i]);
        puts("");
```

```
        printf("%d\n", kmp(a, b, strlen(a), strlen(b)));
    }
}
```

字典树 TRIE

```
#include <iostream>
using namespace std;
const int MAXN = 100010, MAXM = 30, KIND = 26;
int m;
struct node {
    char* s;
    int prefix;
    bool isword;
    node* next[KIND];
    void init() {
        s = NULL;
        prefix = 0;
        isword = false;
        memset(next, 0, sizeof(next));
    }
} a[MAXN*MAXM], *root;//根
void insert(node *root, char *s) {
    node *p = root;
    for (int i = 0; s[i]; i++) {
        int x = s[i] - 'a';
        p->s = s + i;
        if (p->next[x] == NULL) {
            a[m].init();
            p->next[x] = &a[m++];
        }
        p = p->next[x];
        p->prefix++;
    }
    p->isword = true;
}
bool del(node *root, char *s) {
    node *p = root;
    for (int i = 0; s[i]; i++) {
        int x = s[i] - 'a';
        if (p->next[x] == NULL)
            return false;
        p = p->next[x];
    }
    if (p->isword)
        p->isword = false;
    else
        return false;
    return true;
}
bool search(node *root, char* s) {
    node* p = root;
    for (int i = 0; s[i]; i++) {
        int x = s[i] - 'a';
        if (p->next[x] == NULL)
            return false;
        p = p->next[x];
    }
    return p->isword;
}
int count(node *root, char *s) {
    node *p = root;
    for (int i = 0; s[i]; i++) {
        int x = s[i] - 'a';
```

```
        if (p->next[x] == NULL)
            return 0;
        p = p->next[x];
    }
    return p->prefix;
}
int main() {
    m = 0;
    a[m].init();
    root = &a[m++];
    char s[MAXM];
    while (gets(s)) {
        if (strcmp(s, "") == 0) break;
        insert(root, s);
    }
    while (gets(s))
        printf("%d\n", count(root, s));
}
```

扩展 KMP

```
//扩展 KMP
//后缀 A[i]和 B[i]和 A 的最长公共前缀长度
#include<iostream>
using namespace std;
#define maxn 200002
char A[maxn], B[maxn];
int AA_lcp[maxn], BA_lcp[maxn];//分别记录后缀 A[i]和 B[i]和 A 的最长公共前缀长度
void extend_kmp(char *A, char *B, int lenA, int lenB) {
    //先计算 AA_lcp 数组
    int j = 0, k;
    while (A[1+j] == A[2+j] && 2 + j <= lenA) j++;//到了串末 0 是不可能再匹配下去的
    AA_lcp[2] = j, k = 2;
    for (int i = 3; i <= lenA; i++) { //k+AA_lcp[k]是不减的
        int Len = k + AA_lcp[k] - 1, L = AA_lcp[i-k+1];//Len 是 A[k]延伸到的最远处
        if (Len - i + 1 > L)
            AA_lcp[i] = L;
        else {
            j = max(0, Len - i + 1);
            while (A[1+j] == A[i+j] && i + j <= lenA) j++;

            AA_lcp[i] = j;
            k = i;
        }
    }
    //下面计算 BA_lcp 数组
    j = 0;
    while (B[1+j] == A[1+j] && 1 + j <= lenA && 1 + j <= lenB) j++;

    BA_lcp[1] = j, k = 1;
    for (int i = 2; i <= lenB; i++) {
        int Len = k + BA_lcp[k] - 1, L = AA_lcp[i-k+1];
        if (Len - i + 1 > L) {
            BA_lcp[i] = L;
        } else {
            j = max(0, Len - i + 1);
            while (A[1+j] == B[i+j] && 1 + j <= lenA && i + j <= lenB) j++;
            BA_lcp[i] = j;
            k = i;
        }
    }
}
```

```
//HDU2222 给你一个字典,和一个串,找这个串里包含几个单词
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int MAXMEM = 1000001;
const int MAXLEN = 1000001;
struct node {
    int cnt; //该点加了几次
    int id; //该点的 id
    node *fail, *next[26];
    void *operator new(size_t);
} *root, *null, mem[MAXMEM], *top;
void* node::operator new(size_t) {
    memset(top, 0, sizeof(*top));
    return top++;
}
void insert(char* s, int id) {
    node *p = root;
    for (int i;*s;p = p->next[i]) {
        i = *s++ - 'a';
        if (!p->next[i]) {
            p->next[i] = new node;
        }
    }
    p->id = id;
    p->cnt++;
}
void build_ac() {
    typedef node* pnode;
    static pnode queue[MAXMEM], *l, *r;
    node *p,*q;
    l = r = queue;
    *r++ = root;
    for (int i = 0; i < 26; i++) {
        null->next[i] = root;
    }
    root->fail = null;
    while (l != r) {
        p = *l++;
        //if (p->fail->forbid) p->forbid = true;//注意这句，有禁止的要加这个
        for (int i = 0; i < 26; i++) {
            q = p->fail;
            if (p->next[i]) {
                p->next[i]->fail = q->next[i];
                *r++ = p->next[i];
            } else {
                p->next[i] = q->next[i];
            }
        }
    }
}
char s[MAXLEN];
int query(char* s) {
    int ans = 0;
    node *p = root, *q;
    while (*s) {
        int i = *s++ - 'a';
        p = p->next[i];
        for (q = p; q != root && q->cnt != -1; q = q->fail) {
            ans += q->cnt;
            q->cnt = -1;
        }
```

```
    }
    return ans;
}
char word[51];
int main() {
    int n, cas;
    scanf("%d", &cas);
    while (cas--) {
        top = mem;
        null = new node;
        root = new node;
        scanf("%d", &n);
        gets(s);
        for (int i = 0;i < n;i++) {
            gets(word);
            insert(word, i);
        }
        build_ac();
        gets(s);
        printf("%d\n",query(s));
    }
}
```

后缀数组

```
#define M 100005
char data[M], _data[M]; //字符数组 data
//SA 是后缀数组,SA[i]表示排第 i 为的后缀是谁
//rank 是排名数组,rank[i]表示后缀 i 排第几
//height 是高度数组,height[i]为 SA[i]和 SA[i-1]的 LCP,故从 2 开始
int SA[M], rank[M], h[M], height[M];
int n, value;//n 字符串长度
#define _Rank(a)((a)>n?0:rank[a])
bool cmp(int a, int b) {
    return data[a] < data[b];
}
int c[M], a[M], _SA[M], _rank[M];
void Double() {
    memset(c, 0, sizeof(int) * (n + 1));
    int i;
    for (i = 1; i <= n; i++) c[_Rank(i + value)]++;
    a[0] = 1;
    for (i = 1; i <= n; i++) a[i] = a[i - 1] + c[i - 1];
    for (i = 1; i <= n; i++) _SA[a[_Rank(i + value)]++] = i;
    memset(c, 0, sizeof(int) * (n + 1));
    for (i = 1; i <= n; i++) c[_Rank(i)]++;
    a[0] = 1;
    for (i = 1; i <= n; i++) a[i] = a[i - 1] + c[i - 1];
    for (i = 1; i <= n; i++) SA[a[_Rank(_SA[i])]++] = _SA[i];
    _rank[SA[1]] = 1;
    for (i = 2; i <= n; i++)
        if (_Rank(SA[i]) == _Rank(SA[i - 1]) && _Rank(SA[i] + value) == _Rank(SA[i - 1] + value))
            _rank[SA[i]] = _rank[SA[i - 1]];
        else
            _rank[SA[i]] = _rank[SA[i - 1]] + 1;
    memcpy(rank, _rank, sizeof(int) * (n + 1));
    value <<= 1;
}
void make_SA() {
    int i;
    for (i = 1; i <= n; i++) SA[i] = i;
    sort(SA + 1, SA + n + 1, cmp);
    rank[SA[1]] = 1;
    for (i = 2; i <= n; i++)
```

```
        if (data[SA[i]] == data[SA[i - 1]]) rank[SA[i]] = rank[SA[i - 1]];
        else rank[SA[i]] = rank[SA[i - 1]] + 1;
    value = 1;
    while (value < n) Double();
}
void make_Height() {
    int i;
    memset(h, 0, sizeof(int) * (n + 1));
    for (i = 1; i <= n; i++) if (rank[i] == 1) h[i] = 0;
        else if (i == 1 || h[i - 1] <= 1)
            while (data[i + h[i]] == data[SA[rank[i] - 1] + h[i]]) h[i]++;
        else {
            h[i] = h[i - 1] - 1;
            while (data[i + h[i]] == data[SA[rank[i] - 1] + h[i]]) h[i]++;
        }
    for (i = 1; i <= n; i++) height[rank[i]] = h[i];
}
int main() {
    while (gets(data + 1) > 0 ) {
        n = strlen(data + 1);
        make_SA();
        for (int i = 1; i <= n; i++) printf("%s\n", data + SA[i] );
        for (int i = 1; i <= n; i++) printf("%d\n", rank[i]);
    }
}
```

字符串 HASH

```
#include <iostream>
using namespace std;
int MAX;
#define MM 7003
const int PRIME = 124567;//113 2039 4567 124567 999983 3214567 23456789 55566677
int h[MM],c[MM];
int cas;
//BKDRHash
inline unsigned int BKDRHash(char* str) {
    unsigned int seed = 131;  //  31 131 1313 13131 131313 etc..
    unsigned int hash = 0;
    while (*str) {
        hash = hash * seed + (*str++);
    }
    return (hash & 0x7FFFFFFF);
}
//RSHash
inline unsigned int RSHash(char* str) {
    unsigned int b = 50021;
    unsigned int a = 7003;
    unsigned int hash = 0;
    while (*str) {
        hash = hash * a + (*str++);
        a *= b;
    }
    return (hash & 0x7FFFFFFF);
}
//ELFHash
inline unsigned int ELFHash(char* str) {
    unsigned int hash = 0;
    unsigned int x    = 0;
    while (*str) {
        hash = (hash << 4) + (*str++);
        if ((x = hash & 0xF0000000L) != 0) {
            hash ^= (x >> 24);
            hash &= ~x;
```

```
        }
    }
    return (hash & 0x7FFFFFFF);
}
//JSHash
inline unsigned int JSHash(char* str) {
    unsigned int hash = 1315423911;
    while (*str) {
        hash ^= ((hash << 5) + (*str++) + (hash >> 2));
    }
    return (hash & 0x7FFFFFFF);
}
//PJWHash
inline unsigned int PJWHash(char* str) {
    unsigned int BitsInUnignedInt = (unsigned int)(sizeof(unsigned int) * 8);
    unsigned int ThreeQuarters    = (unsigned int)((BitsInUnignedInt * 3) / 4);
    unsigned int OneEighth        = (unsigned int)(BitsInUnignedInt / 8);
    unsigned int HighBits         = (unsigned int)(0xFFFFFFFF) << (BitsInUnignedInt - OneEighth);
    unsigned int hash             = 0;
    unsigned int test             = 0;
    while (*str) {
        hash = (hash << OneEighth) + (*str++);
        if ((test = hash & HighBits) != 0) {
            hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));
        }
    }
    return (hash & 0x7FFFFFFF);
}
//SDBMHash
inline unsigned int SDBMHash(char* str) {
    unsigned int hash = 0;
    while (*str) {
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;
    }
    return (hash & 0x7FFFFFFF);
}

//DJBHash
inline unsigned int DJBHash(char* str) {
    unsigned int hash = 5381;
    while (*str) {
        hash += (hash << 5) + (*str++);
    }
    return (hash & 0x7FFFFFFF);
}

//APHash
inline unsigned int APHash(char* str) {
    unsigned int hash = 0;
    for (int i = 0; *str; i++) {
        if ((i & 1) == 0) {
            hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
        }
        else {
            hash ^= (~((hash << 11) ^ (*str++) ^ (hash >> 5)));
        }
    }
    return (hash & 0x7FFFFFFF);
}
inline void hash(char *str) {
    while (*str=='0') str++;
    int p;
    if (cas == 1) p = ELFHash(str);
    else if(cas % 8 == 0) p = JSHash(str);
    else if(cas % 7 == 0) p = APHash(str);
    else if(cas % 6 == 0) p = DJBHash(str);
```

```
    else if(cas % 5 == 0) p = SDBMHash(str);
    else if(cas % 4 == 0) p = BKDRHash(str);
    else if(cas % 3 == 0) p = PJWHash(str);
    else if(cas % 2 == 0) p = RSHash(str);
    int t = p % MM;
    while (h[t] != -1 && h[t] != p)
        t = (t + 10) % MM;
    if (h[t] == -1) {
        c[t] = 1;
        h[t] = p;
    }
    else {
        c[t]++;
        if (MAX < c[t]) MAX = c[t];
    }
}
/*int hash(char *str) {
    int p = BKDRHash(str);
    while (strcmp(h[p],"") != 0 && strcmp(h[p],str) != 0)
        p = (p + 1) % PRIME;
    strcpy(h[p],str);
    return p;
}*/
int main() {
    int n;
    while (scanf("%d",&n) != EOF) {
        MAX = 1;
        memset(h,-1,sizeof(h));
        char str[100];
        for (int i = 0;i < n;i++) {
            scanf("%s",str);
            hash(str);
        }
        printf("%d ",MAX);
    }
}
```

RABIN-KARP[一维]

```
//HDU3948 O(n)回文算法 + 字符串 hash
//求的是给定长为 n 的 s 串中不同的对称串个数
//使用 unsigned int 自然溢出效果更好
#include <iostream>
#include <cstdio>
#include <cstring>
#include <map>
#include <ctime>
using namespace std;
const int MAXN = 100011;
const int TIME = 3;
long long hash[MAXN][TIME];
long long power[MAXN][TIME];
char s[MAXN];
long long BASE[TIME] = {1231231, 393241, 3145739};
long long MOD[TIME] = {123312323, 25164543, 20132661};
map<long long, int> mp[TIME];
void inithash(char s[], int id) {
    power[0][id] = 1;
    hash[0][id] = (int)(s[0]) % MOD[id];
    int len = strlen(s);
    for (int i = 1; i < len; i++) {
        hash[i][id] = (hash[i - 1][id] * BASE[id] + (s[i])) % MOD[id];
        power[i][id] = (power[i - 1][id] * BASE[id]) % MOD[id];
    }
```

```
}
long long gethash(int l, int r, int id) {
    if (l == 0) return hash[r][id];
    long long ret = (MOD[id] + hash[r][id] - (hash[l - 1][id] * power[r - l + 1][id]) % MOD[id]) %
MOD[id];
    while (ret < 0) ret += MOD[id];
    return ret;
}
int rad[2 * MAXN];
void pal(char s[], int n) {
    int i, j, k;
    for (i = 0,j = 0;i < 2*n;i += k,j = max(j-k, 0)) {
        while (i-j >= 0 && i+j+1 < 2*n && s[(i-j)/2] == s[(i+j+1)/2]) ++j;
        rad[i] = j;
        for (k = 1; i-k >= 0 && rad[i]-k >= 0 && rad[i-k] != rad[i]-k; ++k)
            rad[i+k] = min(rad[i-k],rad[i]-k);
    }
}
//映射关系 TK -> TV, 映射范围_mod, 总节点数目_n
//默认没有重复的 key
const int mod = 111113;
template<class TK,class TV,int _mod=111113,int _n=120000>
class HashTable{
public:
    struct H{ TK key; TV val; int next;  }h[_n];
    int list[_mod],eid;
    int (*hash) (TK);

    HashTable(){ //构造函数
        clear();
    }
    void clear(){ //清 0
        memset(list,-1,sizeof(list));
        eid=0;
    }
    void _setCallBack(int (*hash)(TK) ){ //设置回调函数(哈希函数)
        this->hash=hash;
    }
    int find(TK key){   //查找关键字是否在哈希表中,没有返回-1
        int ind=hash(key);
        for(int p=list[ind];p!=-1;p=h[p].next){
            if( h[p].key == key ) return p;
        }
        return -1;
    }
    void insert(TK key,TV val){ //插入
        int ind=hash(key);
        h[eid].key=key; h[eid].val=val;
        h[eid].next=list[ind]; list[ind]=eid++;
    }
    TV& operator[] (TK key){  //等同于 map 的[],支持没有 key 的查询,没有的话创建一个默认的
        int t=find(key);
        if( t==-1 ) { insert(key,TV()); return h[eid-1].val;}
        else return h[t].val;
    }
    void print(TK *out,int &N){   //0~N-1
        for(int i=0;i<eid;i++)
            out[i]=h[i].key;
        N=eid;
    }
};
inline int h(long long s){
    return int(s%mod);
}
HashTable<long long,int> ht[3];
int main() {
```

```
    int cas;
    scanf("%d", &cas);
    getchar();
    srand(time(NULL));
    for (int T = 1; T <= cas; T++) {
        gets(s);
        for (int i = 0; i < TIME; i++) {
            //BASE[i] = rand() * rand();
            //MOD[i] = rand() * rand();
            ht[i].clear(); ht[i]._setCallBack(h);
            inithash(s, i);
        }
        pal(s, strlen(s));
        int n = strlen(s);
        int ans = 0;
        for (int i = 0; i < 2 * n; i++) {
            int t = i & 1;
            int l = i / 2 - rad[i] / 2 + t, r = i / 2 + rad[i] / 2;
            while (l <= r) {
                bool flag = 1;
                for (int i = 0; i < TIME; i++) {
                    long long h = gethash(l, r, i);
                    if (ht[i].find(h) == -1) {
                        flag = 0;
                        break;
                    }
                }
                if (flag) break;
                for (int i = 0; i < TIME; i++) {
                    long long h = gethash(l, r, i);
                    ht[i].insert(h, 1);
                }
                l++, r--;
                ans++;
            }
        }
        printf("Case #%d: %d\n", T, ans);
    }
}
```

RABIN-KARP[一维可修改]

```
//HDU3973 HDU3948
/*
对于每个字符串，构造如下 hash 函数
S[0] * P^3 +  S[1] * P^2 + S[2] * P^1 + S[3]
对于子串，若 hash 值要跟大矩形的 hash 值匹配上
则要对 hash 值乘上 P^(n - sn) 其中 n, sn 是主串和子串长度
对于主串，若某段 hash 值[l, r]要与子串匹配上
则要对主串 hash 值除以 P^(n - r)，由于有取模操作，所以采用逆元
模板下标都从 0 开始，在树状数组里使用时要+1
*/
//这题求的是可以修改主串的字符，问主串某个子串是否在字典里
#include <iostream>
#include <cstdio>
#include <cstring>
#include <set>
#include <ctime>
using namespace std;
typedef long long LL;

LL inv2(LL a, LL p) {
    LL res = 1;
    for (LL b = p - 2; b; b >>= 1) {
```

```
        if (b & 1) res = res * a % p;
        a = a * a % p;
    }
    return res;
}

struct StrHash {
    typedef long long LL;
    #define MAXN 100011
    #define TIME 1
    static LL BASE[TIME], MOD[TIME];
    static LL power[TIME][MAXN], inv[TIME][MAXN];

    int n;
    char s[MAXN];

    LL h[TIME][MAXN];//hash 值
    LL c[TIME][MAXN];//树状数组求和
    int t[TIME][MAXN], tt;//时间戳，避免 memset

    static void initpower(int id) {
        power[id][0] = 1;
        inv[id][0] = inv2(1, MOD[id]);
        for (int i = 1; i < MAXN; i++) {
            power[id][i] = (power[id][i - 1] * BASE[id]) % MOD[id];
            inv[id][i] = inv2(power[id][i], MOD[id]);
        }
    }

    inline int lowbit(int x) {
        return x & -x;
    }

    void add(int x, LL v, int id) {
        while (x < n + 1) {
            if (t[id][x] != tt) {
                c[id][x] = 0;
                t[id][x] = tt;
            }
            c[id][x] = (c[id][x] + v) % MOD[id];
            x += lowbit(x);
        }
    }

    LL get(int x, int id) {
        LL ret = 0;
        while (x > 0) {
            if (t[id][x] != tt) {
                c[id][x] = 0;
                t[id][x] = tt;
            }
            ret = (ret + c[id][x]) % MOD[id];
            x -= lowbit(x);
        }
        return ret;
    }

    //树状数组维护 hash 值
    void inithash(int id) {
        n = strlen(s);
        for (int i = 0; i < n; i++) {
            add(i + 1, (s[i] * power[id][n - i - 1]) % MOD[id], id);
        }
    }

    //树状数组查询 hash 值
```

```
    LL gethash(int l, int r, int id) {
        return (get(r + 1, id) - get(l, id) + MOD[id]) % MOD[id];
    }

    //子串 hash 值[l, r]->[0..r - l + 1]
    LL resize(int l, int r, int id) {
        return gethash(l, r, id) * inv[id][n - 1 - r] % MOD[id];
    }

    //树状数组修改字符串的第 x 位为 c
    void modify(int x, char c, int id) {
        LL v = (-s[x] * power[id][n - x - 1]) % MOD[id];
        while (v < 0) v += MOD[id];
        add(x + 1, v, id);
        v = (c * power[id][n - x - 1]) % MOD[id];
        add(x + 1, v, id);
    }

    //暴力求整个串的 hash, 无需额外空间
    LL geth(int id) {
        n = strlen(s);
        LL ret = s[0];
        for (int i = 1; i < n; i++) {
            ret = (ret * BASE[id]) % MOD[id] + s[i];
        }
        return ret % MOD[id];
    }
}H;

LL StrHash::BASE[TIME] = {37}, StrHash::MOD[TIME] = {805306457};
LL StrHash::power[TIME][MAXN], StrHash::inv[TIME][MAXN];

set<LL> st[TIME];

int main() {
    int n, m, cas;
    scanf("%d", &cas);
    srand(time(NULL));
    for (int i = 0; i < TIME; i++) {
        StrHash::initpower(i);
    }
    for (int T = 1; T <= cas; T++) {
        scanf("%d", &n);
        getchar();
        for (int i = 0; i < TIME; i++) {
            st[i].clear();
        }
        for (int i = 0; i < n; i++) {
            scanf(" %s", H.s);
            for (int j = 0; j < TIME; j++) {
                st[j].insert(H.geth(j));
            }
        }
        scanf(" %s", H.s);
        for (int i = 0; i < TIME; i++) {
            H.inithash(i);
        }
        scanf("%d", &m);
        char cmd, z;
        int x, y;
        printf("Case #%d:\n", T);
        for (int k = 0; k < m; k++) {
            scanf(" %c", &cmd);
            if (cmd == 'Q') {
                scanf(" %d %d", &x, &y);
                bool flag = 1;
```

```
            for (int i = 0; i < TIME; i++) {
                if (!st[i].count(H.resize(x, y, i))) {
                    flag = 0;
                    break;
                }
            }
            puts(flag ? "Yes" : "No");
        } else {
            scanf(" %d %c", &x, &z);
            for (int i = 0; i < TIME; i++) {
                H.modify(x, z, i);
            }
            H.s[x] = z;
        }
    }
    H.tt++;
  }
}
```

RABIN-KARP[二维]

```
//HDU3531 Tokyo Regional 2010 H
/*
对于每个矩阵，构造如下 hash 函数
a[0][0] * P^3 * Q^2  a[0][1] * P^3 * Q^1  a[0][2] * P^3 * Q^0
a[1][0] * P^2 * Q^2  a[1][1] * P^2 * Q^1  a[1][2] * P^2 * Q^0
a[2][0] * P^1 * Q^2  a[2][1] * P^1 * Q^1  a[2][2] * P^1 * Q^0
对于子矩形，hash 值要跟大矩形的 hash 值匹配上
所以要对 hash 值乘上 P^(n - sn) 和 Q^(m - sm) 其中 n m, sn sm 是大小矩形规格
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <map>
using namespace std;
typedef unsigned int LL;

struct MatHash {
    typedef unsigned int LL;
    #define MAXN 1001
    #define MAXM 1001
    #define TIME 1
    static LL P[TIME], Q[TIME], MOD[TIME];
    static LL powerP[TIME][MAXN], powerQ[TIME][MAXM];

    int n, m;
    int mat[MAXN][MAXM];

    LL h[TIME][MAXN][MAXM];//主矩阵 hash 值

    static void init(int id) {
        powerP[id][0] = 1;
        for (int i = 1; i < MAXN; i++) {
            powerP[id][i] = (powerP[id][i - 1] * P[id]);
        }
        powerQ[id][0] = 1;
        for (int i = 1; i < MAXM; i++) {
            powerQ[id][i] = (powerQ[id][i - 1] * Q[id]);
        }
    }

    void inithash(int id) {
        for (int i = 0; i < n; i++) {
```

```
            for (int j = 0; j < m; j++) {
                //注意下面这句 mat[i][j] + 3 不要出现 0
                h[id][i][j] = (mat[i][j] + 3) * powerP[id][n - 1 - i] * powerQ[id][m - 1 - j];
                if (i) h[id][i][j] = (h[id][i][j] + h[id][i - 1][j]);
                if (j) h[id][i][j] = (h[id][i][j] + h[id][i][j - 1]);
                if (i && j) h[id][i][j] = (h[id][i][j] - h[id][i - 1][j - 1]);
            }
        }
    }

    LL gethash(int x1, int y1, int x2, int y2, int id) {
        LL ret = h[id][x2][y2];
        if (x1) ret = (ret - h[id][x1 - 1][y2]);
        if (y1) ret = (ret - h[id][x2][y1 - 1]);
        if (x1 && y1) ret = (ret + h[id][x1 - 1][y1 - 1]);
        ret = (ret * powerP[id][x1]);
        ret = (ret * powerQ[id][y1]);
        return ret;
    }

    //把当前小矩形的 hash 值匹配到更大的矩形的对应的 hash 值
    LL resize(int x1, int y1, int x2, int y2, int id, int _n, int _m) {
        LL ret = gethash(x1, y1, x2, y2, id);
        ret = ret * powerP[id][_n - n] * powerQ[id][_m - m];
        return ret;
    }

    void input() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                scanf("%d", &mat[i][j]);
            }
        }
    }
}A, B;

LL MatHash::P[TIME], MatHash::Q[TIME], MatHash::MOD[TIME];
LL MatHash::powerP[TIME][MAXN], MatHash::powerQ[TIME][MAXM];

int main() {
    MatHash::P[0] = 393241, MatHash::Q[0] = 784633, MatHash::MOD[0] = 805306457;
    MatHash::P[1] = 784633, MatHash::Q[1] = 111117, MatHash::MOD[1] = 402653189;
    for (int i = 0; i < TIME; i++) {
        MatHash::init(i);
    }
    while (scanf("%d%d", &A.n, &B.n) != EOF) {
        A.m = A.n, B.m = B.n;
        A.input();
        for (int i = 0; i < TIME; i++) {
            A.inithash(i);
        }
        B.input();
        for (int i = 0; i < TIME; i++) {
            B.inithash(i);
        }
        bool yes = false;
        for (int id = 0; id < TIME; id++) {
            B.inithash(id);
            LL val = B.resize(0, 0, B.n - 1, B.m - 1, id, A.n, A.m);
            for (int i = B.n - 1; i < A.n; i++) {
                for (int j = B.m - 1; j < A.m; j++) {
                    if (A.gethash(i - B.n + 1, j - B.m + 1, i, j, id) == val) {
                        yes = true;
                        break;
                    }
                }
```

```
            if (yes) break;
        }
      }
      puts(yes ? "Yes" : "No");
   }
}
```

HASH 后缀数组

```
inithash(){
   foreach i get prehash(i) = prehash(i-1) * x + s[i]
}
hash(start, end) {
    return (prehash(end) - prehash(start) * x^(end - start));
}
int getLCP(int a, int b) {
   binarysearch (l) for largest {
      hash(s+a,s+a+l) == hash(s+b,s+b+l);
   }
   return l;
}
bool cmp(int a, int b) {
    int LCP = getLCP(a, b);
    return s[a+LCP] < s[b+LCP];
}
getSuffixArray() {
    sort(suffix, suffix+n, cmp);
}
```

字符串最小最大表示法

```
int MinimumRepresentation(char *s, int l) { //串s[0~l-1]的最小表示位置
    int i = 0, j = 1, k = 0, t;
    while (i < l && j < l && k < l) { //找不到比它还小的 或者 完全匹配
        t = s[(i+k)%l] - s[(j+k)%l];
        //if (s[(i+k) >= l ? i+k-l : i+k] == s[(j+k) >= l ? j+k-l : j+k])
        if (t == 0)
            k++;//相等的话,检测长度加1
        else {
            if (t > 0)//大于的话,s[i]为首的肯定不是最小表示,最大表示就改<
                i += k + 1;
            else j += k + 1;
            if (i == j) j++;
            k = 0;
        }
    }
    return min(i, j);
}
```

最长回文子串

```
#include <iostream>
using namespace std;
char s[50001];
int pal(char* s, int n) {
    int rad[2*n],i,j,k;
    for (i = 0,j = 0;i < 2*n;i += k,j = max(j-k,0)) {
        while (i-j >= 0 && i+j+1 < 2*n && s[(i-j)/2] == s[(i+j+1)/2]) ++j;
```

```
        rad[i] = j;
        for (k = 1; i-k >= 0 && rad[i]-k >= 0 && rad[i-k] != rad[i]-k; ++k)
            rad[i+k] = min(rad[i-k],rad[i]-k);
    }
    return *max_element(rad,rad+2*n);
}
int main() {
    while (scanf("%s",s) != EOF)
        printf("%d\n",pal(s, strlen(s)));
}
```

五：计算几何

MATRUSH 几何模板

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
using namespace std;
const double eps = 1e-8;
const double pi = acos(-1.0);
const double inf = 1e200;
inline int sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
inline double sqr(double x) {
    return x * x;
}
inline double a2r(double a) { //角度到弧度
    return a * pi / 180;
}
inline double r2a(double r) { //弧度到角度
    return r / pi * 180;
}
struct P {
    double x,y;
    P(double x = 0,double y = 0):x(x), y(y) {};
    friend bool operator ==(const P& p1, const P& p2) {
        return sgn(p1.x - p2.x) == 0 && sgn(p1.y - p2.y) == 0;
    }
    friend bool operator <(const P& p1, const P& p2) {
        return sgn(p1.x - p2.x) == 0 ? sgn(p1.y - p2.y) < 0 : p1.x < p2.x;
    }
    friend bool operator >(const P& p1, const P& p2) {
        return sgn(p1.x - p2.x) == 0 ? sgn(p1.y - p2.y) > 0 : p1.x > p2.x;
    }
    friend P operator +(const P& p1, const P& p2) {
        return P(p1.x + p2.x, p1.y + p2.y);
    }
    friend P operator -(const P& p1, const P& p2) {
        return P(p1.x - p2.x, p1.y - p2.y);
    }
    friend P operator *(const P& p, double a) {
        return P(p.x * a,p.y * a);
    }
    friend P operator /(const P& p, double a) {
        return P(p.x / a,p.y / a);
    }
```

```
    friend inline double operator ^(const P& p1, const P& p2) { //点乘
        return p1.x * p2.x + p1.y * p2.y;
    }
    friend inline double operator *(const P& p1, const P& p2) { //叉乘
        return p1.x * p2.y - p1.y * p2.x;
    }
    friend inline double dot(const P& o, const P& a, const P& b) {
        return (a - o) ^ (b - o);
    }
    friend inline double cross(const P& o, const P& a, const P& b) { //2 倍三角形 oab 有向面积
        return (a - o) * (b - o);
    }
    friend double dist(const P& p1, const P& p2) {
        return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
    }
    friend double dist2(const P& p1, const P& p2) {
        return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
    }
    double len() const {
        return sqrt(x * x + y * y);
    }
    double len2() const {
        return x * x + y * y;
    }
    //求向量 v 的极角，角度范围为 [0,2*pi)
    double angle() const {
        double t = atan2(y,x);
        return sgn(t) >= 0 ? t : t + 2 * pi;
    }
    P trunc(double l) const {
        l /= len();
        return P(x * l, y * l);
    }
    P turn_left() const { //法向量
        return P(-y, x);
    }
    P turn_right() const { //法向量
        return P(y, -x);
    }
    P rotate(double th, const P& p) const { //绕点 p 逆时针转 th 弧度
        double c = cos(th), s = sin(th);
        return P(c * (x - p.x) - s * (y - p.y) + p.x, s * (x - p.x) + c * (y - p.y) + p.y);
    }
    bool in() {
        return scanf("%lf %lf", &x, &y) == 2;
    }
    void out() const {
        printf("%lf %lf\n", x, y);
    }
};

//以下直线相关
struct line {
    P a, b;
    line() {}
    line(P a, P b):a(a), b(b) {}

    //点到直线距离
    friend double dist(const P& p, const line& l) {
        return fabs(cross(p, l.a, l.b)) / dist(l.a, l.b);//到线段所在直线距离
    }

    //判定两直线位置关系，并求出交点(如果存在).
    //完全重合(2),有交点(1),平行无交(0),参数错误(-1)
    friend int intersection(line l1, line l2, P& p) { //直线与直线相交
        if (l1.a == l1.b || l2.a == l2.b) return -1;
```

```
        double c1 = (l1.b - l1.a) * (l2.b - l2.a),//cross(l1.a,l1.b,l2.b);
               c2 = (l2.a - l1.a) * (l2.b - l2.a);//cross(l1.a,l2.a,l2.b);
        if (sgn(c1) == 0) {
            if (sgn(c2) == 0) {
                return 2;//两线重叠
            } else {
                return 0;//两线平行不重叠
            }
        } else {
            p = l1.a + (l1.b - l1.a) * (c2 / c1);
            return 1;
        }
    }
    friend bool sameside(const P& p1, const P& p2, const line& l) { //两个点是否在直线同侧,是则返回 true
        return sgn(cross(p1, l.a, l.b) * cross(p2, l.a, l.b)) > 0;
    }

    //判断点是否在线段上 包含端点
    friend bool onseg(const P& p, const line& s) {
        if (sgn(cross(p, s.a, s.b)) == 0)//如果点保证在线段所在直线上，去掉此条件较好
            if (sgn(p.x - min(s.a.x, s.b.x)) >= 0 && sgn(p.x - max(s.a.x, s.b.x)) <= 0)
                if (sgn(p.y - min(s.a.y, s.b.y)) >= 0 && sgn(p.y - max(s.a.y, s.b.y)) <= 0)
                    return 1;
        return 0;
    }

    //判断点是否在线段上 不包含端点
    friend bool onseg2(const P& p, const line& s) {
        if (sgn(cross(p, s.a, s.b)) == 0)//如果点保证在线段所在直线上，去掉此条件较好
            if (sgn(p.x - min(s.a.x, s.b.x)) >= 0 && sgn(p.x - max(s.a.x, s.b.x)) <= 0)
                if (sgn(p.y - min(s.a.y, s.b.y)) >= 0 && sgn(p.y - max(s.a.y, s.b.y)) <= 0)
                    return !(p == s.a || p == s.b);
        return 0;
    }

    void in() {
        a.in(), b.in();
    }
};

//以下解析几何直线相关
struct L { //直线定义 ax + by + c = 0
    double a, b, c;
    L(const P& p1, const P& p2) { //两点式到一般式
        a = p2.y - p1.y;
        b = p1.x - p2.x;
        c = p2 * p1;
        if (sgn(a) < 0) a *= -1,b *= -1,c *= -1;
    }
    L(double a = 0, double b = 0, double c = 0):a(a), b(b), c(c) {}
    L(line l) {
        *this = L(l.a, l.b);
    }

    //点到直线的距离
    friend double dist(const P& p, const L& l) {
        return fabs(l.a * p.x + l.b * p.y + l.c) / sqrt(sqr(l.a) + sqr(l.b));
    }

    //两个点是否在直线同侧,是则返回 true
    friend bool sameside(const P& p1, const P& p2, const L& l) {
        return sgn((l.a * p1.x + l.b * p1.y + l.c) * (l.a * p2.x + l.b * p2.y + l.c)) > 0;
    }

    //求点 p 关于直线 l 的对称点
    friend P symmerty(const P& p, const L& l) {
```

```
        P t;
        t.x = ((sqr(l.b) - sqr(l.a)) * p.x - 2 * l.a * l.b * p.y - 2 * l.a * l.c) / (sqr(l.a) + sqr(l.b));
        t.y = ((sqr(l.a) - sqr(l.b)) * p.y - 2 * l.a * l.b * p.x - 2 * l.b * l.c) / (sqr(l.a) + sqr(l.b));
        return t;
    }

    //直线 l1,l2 是否相交
    bool intersection(const L& l1, const L& l2, P &p) {
        double d = l1.a * l2.b - l2.a * l1.b;
        if (sgn(d) == 0) return 0; // 不相交
        p.x = (l2.c * l1.b - l1.c * l2.b) / d;
        p.y = (l2.a * l1.c - l1.a * l2.c) / d;
        return 1;
    }

    //返回直线的斜率 k,水平线返回 0,竖直线返回 1e200
    double slope(const L& l) {
        if (abs(l.a) < 1e-20) return 0;
        if (abs(l.b) < 1e-20) return inf;
        return -(l.a / l.b);
    }

    //返回直线的倾斜角 alpha(0~pi)
    double alpha(const L& l) {
        if (sgn(l.a) == 0) return 0;
        if (sgn(l.b) == 0) return pi / 2;
        double k = slope(l);
        if (k > 0) return atan(k);
        else return pi + atan(k);
    }

    void input() {
        P p1, p2;
        p1.in(), p2.in();
        *this = L(p1, p2);
    }

    void out() {
        printf("%lfx", a);
        if (b >= 0) putchar('+');
        printf("%lfy", b);
        if (c >= 0) putchar('+');
        printf("%lf", c);
        printf(" = 0\n");
    }
};

//以下是线段相关
struct seg {
    P a, b;
    seg() {}
    seg(P a, P b):a(a), b(b) {}

    //点到线段距离 可以判点是否落在线段上
    friend double dist(const P& p, const seg& s) {
        if (sgn((s.b - s.a) ^ (p - s.a)) <= 0)
            return dist(s.a, p);
        if (sgn((s.a - s.b) ^ (p - s.b)) <= 0)
            return dist(s.b, p);
        return fabs((p - s.b) * (s.a - s.b)) / (s.a - s.b).len();
        //return cross(p,s.a,s.b) / (s.b - s.a).len();//到线段所在直线距离
    }

    //线段 s1 到线段 s2 最短距离
    //通过<,<=,来处理端点 如果相交可以返回交点
    friend double dist(const seg& s1, const seg& s2) {
```

```
        if (!((sgn(max(s1.a.x, s1.b.x) - min(s2.a.x, s2.b.x)) < 0) ||
              (sgn(max(s2.b.x, s2.a.x) - min(s1.a.x, s1.b.x)) < 0) ||
              (sgn(max(s1.a.y, s1.b.y) - min(s2.a.y, s2.b.y)) < 0) ||
              (sgn(max(s2.b.y, s2.a.y) - min(s1.a.y, s1.b.y)) < 0) )) {
            if (sgn(cross(s1.a, s2.a, s1.b) * cross(s1.a, s2.b, s1.b)) <= 0 &&
                sgn(cross(s2.a, s1.a, s2.b) * cross(s2.a, s1.b, s2.b)) <= 0) {
                /*double t = ((s1.a - s2.a) * (s2.a - s2.b)) / ((s1.a - s1.b) * (s2.a - s2.b));
                P p = (s1.b - s1.a) * t;*/
                return 0;
            }
        }
        //不相交
        return min(min(dist(s1.a, s2), dist(s1.b, s2)), min(dist(s2.a, s1),dist(s2.b, s1)));
    }

    //线段 s 与直线相交 叉积为负
    friend int intersection(seg s, line l, P& p) {
        P v1 = l.b - l.a, v2 = s.b - l.a, v3 = s.a - l.a, v4 = s.b - s.a;
        int sign = sgn((v2 * v1) * (v3 * v1));
        double t = (s.a * v4 + v4 * l.a) / (v1 * v4);
        p = l.a + (l.b - l.a) * t;
        return sign <= 0;
        //sign = 0 交点为端点,sign = 1 同侧不相交 sign -1 异侧相交
    }

    //两线段是否平行
    friend bool isparallel(seg a, seg b) {
        return sgn((a.a - a.b) * (b.a - b.b)) == 0;
    }

    //判定两线段位置关系,并求出交点(如果存在).
    //有重合:完全重合(6),1 个端点重合且共线(5),部分重合(4)
    //无重合:两端点相交(3),交于线上(2),正交(1),无交(0),参数错误(-1)
    friend int intersection(seg a, seg b, P& p) {
        //保证参数 p1!=p2,p3!=p4
        P &p1 = a.a, &p2 = a.b, &p3 = b.a, &p4 = b.b;
        if (p1 == p2 || p3 == p4) return -1; //返回-1 代表至少有一条线段首尾重合,不能构成线段
        if (p1 > p2) swap(p1, p2); //为方便运算,保证各线段的起点在前,终点在后.
        if (p3 > p4) swap(p3, p4);
        P v1 = p2 - p1, v2 = p4 - p3; //求出两线段构成的向量
        double cross = v1 * v2; //求两向量外积,平行时外积为 0
        if (p1 == p3 && p2 == p4) { //判定两线段是否完全重合,返回任意一个端点
            p = p1;
            return 6;
        }
        if (p1 == p3 || p2 == p4) { //起点或终点重合,共线(平行)返回 5;不平行则交于端点,返回 3
            p = p1 == p3 ? p1 : p2;
            return sgn(cross) == 0 ? 5 : 3;
        }
        if (p1 == p4 || p2 == p3) { //如果两线端首尾相连
            p = p1 == p4 ? p1 : p2;
            return 3;
        }
        //经过以上判断,首尾点相重的情况都被排除了
        //将线段按起点坐标排序.若线段 1 的起点较大,则将两线段交换
        if (p1 > p3) {
            swap(a, b);
            swap(v1, v2); //更新原先计算的向量及其外积
            cross = v1 * v2;
        }
        if (sgn(cross) == 0) { //处理两线段平行的情况
            P vs = p3 - p1; //做向量 v1(p1, p2)和 vs(p1,p3)的外积,判定是否共线
            if (sgn(v1 * vs) == 0 && p2 > p3) { //外积为 0 则两平行线段共线,前一条线的终点大于后一条线的起点,
则判定存在重合
                p = p3;
                return 4; //返回值 4 代表线段部分重合
```

```
        }//若三点不共线,则这两条平行线段必不共线.
        return 0; //不共线或共线但无重合的平行线均无交点
    } //以下为不平行的情况,先进行快速排斥试验
    //x 坐标已有序,可直接比较,y 坐标要先求两线段的最大和最小值
    double ymax1 = p1.y, ymin1 = p2.y, ymax2 = p3.y, ymin2 = p4.y;
    if (ymax1 < ymin1) swap(ymax1, ymin1);
    if (ymax2 < ymin2) swap(ymax2, ymin2);
    if (p1.x > p4.x || p2.x < p3.x || ymax1 < ymin2 || ymin1 > ymax2) return 0;//如果以两线段为对角线的矩形不相交,则无交点
    //下面进行跨立试验
    P vs1 = p1 - p3, vs2 = p2 - p3, vt1 = p3 - p1, vt2 = p4 - p1;
    double s1v2 = vs1 * v2, s2v2 = vs2 * v2, t1v1 = vt1 * v1, t2v1 = vt2 * v1;
    //根据外积结果判定否交于线上
    if (sgn(s1v2) == 0 && p4 > p1 && p1 > p3) {
        p = p1;
        return 2;
    }
    if (sgn(s2v2) == 0 && p4 > p2 && p2 > p3) {
        p = p2;
        return 2;
    }
    if (sgn(t1v1) == 0 && p2 > p3 && p3 > p1) {
        p = p3;
        return 2;
    }
    if (sgn(t2v1) == 0 && p2 > p4 && p4 > p1) {
        p = p4;
        return 2;
    } //未交于线上,则判定是否相交
    if(s1v2 * s2v2 > 0 || t1v1 * t2v1 > 0) return 0;
    //以下为相交的情况,计算二阶行列式的两个常数项
    double A = p1 * v1,B = p3 * v2;
    //计算行列式 D1 和 D2 的值,除以系数行列式的值,得到交点坐标
    p.x = (B * v1.x - A * v2.x) / cross;
    p.y = (B * v1.y - A * v2.y) / cross;
    //正交返回 1
    return 1;
}

//判断点是否在线段上 包含端点
friend bool onseg(const P& p, const seg& s) {
    if (sgn(cross(p, s.a, s.b)) == 0)//如果点保证在线段所在直线上,去掉此条件较好
        if (sgn(p.x - min(s.a.x, s.b.x)) >= 0 && sgn(p.x - max(s.a.x, s.b.x)) <= 0)
            if (sgn(p.y - min(s.a.y, s.b.y)) >= 0 && sgn(p.y - max(s.a.y, s.b.y)) <= 0)
                return 1;
    return 0;
}

//判断点是否在线段上 不包含端点
friend bool onseg2(const P& p, const seg& s) {
    if (sgn(cross(p, s.a, s.b)) == 0)//如果点保证在线段所在直线上,去掉此条件较好
        if (sgn(p.x - min(s.a.x, s.b.x)) >= 0 && sgn(p.x - max(s.a.x, s.b.x)) <= 0)
            if (sgn(p.y - min(s.a.y, s.b.y)) >= 0 && sgn(p.y - max(s.a.y, s.b.y)) <= 0)
                return !(p == s.a || p == s.b);
    return 0;
}

//线段垂直平移
//线段向量 ab 往垂线方向左移 r 长度
friend seg left(const seg& s, double r) {
    P p = (s.b - s.a).turn_left().trunc(r);
    return seg(s.a + p,s.b + p);
}

//获得线段 s 的中垂线向量
//s.a 在左边,即 s.a 在半平面内
```

```
    friend seg mid(const seg& s) {
        seg m;
        m.a = (s.a + s.b) / 2;
        m.b.x = m.a.x + s.a.y - s.b.y;
        m.b.y = m.a.y + s.b.x - s.a.x;
        return m;
    }

    //点与线段的关系
    //r = 0:P = A,r = 1:P = B,r < 0:P < AB,r > 1 :P > AB,0 < r < 1, P in AB
    friend double relation(const P& p, const seg& s) {
        return dot(p, s.b, s.a) / dist2(s.a, s.b);
    }

    //求点 p 到线段 s 所在直线的垂足
    friend P perpendicular(const P& p, const seg& s) {
        double r = relation(p, s);
        return s.a + r * (s.b - s.a);
    }

    void in() {
        a.in(), b.in();
    }

    void out() {
        a.out(), b.out();
    }
};

//以下圆相关
struct circle {
    P o;//圆心
    double r;
    circle():r(0) {}
    circle(P o,double r = 0):o(o), r(r) {}
    circle(const P& a, const P& b, const P& c) { //三角形 abc 构造外接圆
        // 先判(sgn((b - a) * (c - a)) != 0) 不共线
        double c1 = (sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y)) / 2;
        double c2 = (sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y)) / 2;
        o.x = (c1 * (a.y - c.y) - c2 * (a.y - b.y)) / ((a - b) * (a - c));
        o.y = (c1 * (a.x - c.x) - c2 * (a.x - b.x)) / ((a - c) * (a - b));
        r = dist2(o, a);
    }

    //求圆外点到圆的切点
    friend void tangent(const P& p, const circle& c, P& p1, P& p2) {
        double d = dist(p, c.o);
        P t = c.o + (p - c.o) * (c.r / d);
        double th = acos(c.r / d);
        p1 = t.rotate(th, c.o);
        p2 = t.rotate(2 * pi - th, c.o);
    }

    //两圆关系
    //相交(1) 外切(2) 内切(3) 相离(4) 内含(5) 有误(0)
    friend int relation(const circle& c1, const circle& c2) {
        double d = dist(c1.o, c2.o);
        if (sgn(d - fabs(c1.r - c2.r)) > 0 && sgn(d - c1.r - c2.r) < 0) return 1; //相交
        if (sgn(d - c1.r - c2.r) == 0) return 2;//外切
        if (sgn(d - fabs(c1.r - c2.r)) == 0) return 3;//内切
        if (sgn(d - c1.r - c2.r) > 0) return 4;//相离
        if (sgn(d - fabs(c1.r - c2.r)) < 0) return 5;//内含
        return 0; //error
    }

    //圆与圆相交(相切)
```

```
//返回交点(切点)个数 相离(0) 重合(3) 相切(1) 相交(2)
friend int intersection(const circle& c1, const circle& c2, P& p1, P& p2) {
    double d = dist(c1.o, c2.o);
    if (sgn(d - (c1.r + c2.r)) > 0 || sgn(d - fabs(c1.r - c2.r)) < 0) { //相离
        return 0;
    }
    if (sgn(d) == 0 && sgn(c1.r - c2.r) == 0) { //重合
        return 3;
    }
    double cosa = (sqr(c1.r) + sqr(d) - sqr(c2.r)) / (2 * c1.r * d);
    double sina = sqrt(max(0., 1. - sqr(cosa)));
    p1 = p2 = c1.o;
    p1.x += c1.r / d * ((c2.o.x - c1.o.x) * cosa + (c2.o.y - c1.o.y) * -sina);
    p1.y += c1.r / d * ((c2.o.x - c1.o.x) * sina + (c2.o.y - c1.o.y) * cosa);
    p2.x += c1.r / d * ((c2.o.x - c1.o.x) * cosa + (c2.o.y - c1.o.y) * sina);
    p2.y += c1.r / d * ((c2.o.x - c1.o.x) * -sina + (c2.o.y - c1.o.y) * cosa);
    if (sgn(p1.x - p2.x) == 0 && sgn(p1.y - p2.y) == 0) {
        return 1; //相切
    } else {
        return 2; //相交
    }
}

//圆与直线是否相交(相切)
//返回交点(切点)个数 相离(0) 相切(1) 相交(2)
friend int intersection(const circle& c, line& l, P& p1, P& p2) {
    if (c.o == l.a) swap(l.a, l.b);
    double d = fabs((c.o - l.a) * (l.b - l.a)) / (l.b - l.a).len();
    P v = (l.b - l.a).turn_right(), p;
    line mid(c.o, c.o + v);
    intersection(mid, l, p);
    int sign = sgn(d - c.r);
    if (sign < 0) { //相交 2 个交点
        double d = dist(c.o, p);
        if (p == l.a) swap(l.a, l.b);
        P v1 = l.a - p;
        p1 = p + v1 * sqrt((sqr(c.r) - sqr(d))) / v1.len();
        p2 = p * 2 - p1; //p2 = p1.rotate(pi,p);
        return 2;
    } else if (sign == 0) { //相切
        p1 = p;
        return 1;
    } else {
        return 0; //相离
    }
}

//圆与线段是否相交(相切)
friend int intersection(const circle& c, const seg& s, P& p1, P& p2) {
    if (sgn(dist(c.o,s.a) - c.r) < 0 && sgn(dist(c.o,s.b) - c.r) < 0) {
        return 0; //线段在圆内(不包括边界)视为相离
    }
    P v = (s.b - s.a).turn_right(), p;
    line l(c.o, c.o + v);
    if (intersection(s, l, p)) { //与垂线相交
        double d1 = dist(c.o,s.a), d2 = dist(c.o,s.b),
               d3 = fabs((c.o - s.b) * (s.a - s.b)) / (s.a - s.b).len();
        int sign = sgn(d3 - c.r);
        if (sign < 0) {
            if (sgn(min(d1,d2) - c.r) < 0) { //只有一个交点
                if (sgn(d1 - c.r) < 0) { //s.a 在圆内
                    P v1 = s.b - p;
                    p1 = p + v1 * (sqrt(sqr(c.r) - sqr(d3)) / v1.len());
                } else { //s.b 在圆内
                    P v1 = s.a - p;
                    p1 = p + v1 * (sqrt(sqr(c.r) - sqr(d3)) / v1.len());
```

```
            }
            return 1;
        } else {
            P v1 = s.a - p;
            p1 = p + v1 * (sqrt(sqr(c.r) - sqr(d3)) / v1.len());
            v1 = s.b - p;
            p2 = p + v1 * (sqrt(sqr(c.r) - sqr(d3)) / v1.len());
            return 2; //有2个交点
        }
    } else if (sign == 0) {
        p1 = p;
        return 3;//相切
    } else {
        return 0;
    }
} else { //与垂线不相交 无相切
    double d1 = dist(c.o,s.a),d2 = dist(c.o,s.b),
          d3 = fabs((c.o - s.b) * (s.a - s.b)) / (s.a - s.b).len();
    int sign = sgn(min(d1,d2) - c.r);
    if (sign < 0) {
        if (sgn(d1 - c.r) < 0) { //s.a在圆内
            P v1 = s.b - s.a;
            p1 = s.b + v1 * (sqrt(sqr(c.r) - sqr(d3)) - sqrt(sqr(d2) - sqr(d3))) / v1.len();
        } else { //s.b在圆内
            P v1 = s.a - s.b;
            p1 = s.a + v1 * (sqrt(sqr(c.r) - sqr(d3)) - sqrt(sqr(d1) - sqr(d3))) / v1.len();
        }
        return 1;//相交,最多一个交点
    }
    if (sign == 0) { //交于一点
        p1 = sgn(d1 - c.r) == 0 ? s.a : s.b;
        return 1;
    } else {
        return 0;
    }
}
}

//圆C1和C2相交面积
friend double intersectionarea(circle& C1, circle& C2) {
    P c1 = C1.o, c2 = C2.o;
    double r1 = C1.r, r2 = C2.r;
    if (r1 > r2) swap(c1, c2);
    double d, a1, a2, p, area;
    d = sqrt(sqr(c1.x - c2.x) + sqr(c1.y - c2.y));
    if (sgn(r1 + d - r2) <= 0) return pi * sqr(r1);
    if (sgn(r1 + r2 - d) <= 0) return 0;
    a1 = acos((sqr(r1) + sqr(d) - sqr(r2)) / (2 * r1 * d));
    a2 = acos((sqr(r2) + sqr(d) - sqr(r1)) / (2 * r2 * d));
    p = (r1 + r2 + d) / 2;
    area = 2 * sqrt(p * (p - r1) * (p - r2) * (p - d));
    area = a1 * sqr(r1) + a2 * sqr(r2) - area;
    return area;
}

//求两个圆的公切线，传入两圆，返回切线条数 * 2，存放在p[]中，(p[2*i], p[2*i+1])构成一条切线
//两圆重合返回-1，无数条切线，内含返回0条切线
//内切返回1条切线，相交返回2条切线，外切返回3条切线，相离返回4条切线
friend int tangentline(circle c1, circle c2, P p[]) {
    int n = 0;
    if (c1.r < c2.r) swap(c1, c2);
    double r = min(c1.r, c2.r), R = max(c1.r, c2.r);
    double d = dist(c1.o, c2.o), th;
    P v = c2.o - c1.o;
    if (sgn(d) == 0 && sgn(R - r) == 0) { //重合，无数条切线
        return -1;
```

```
        }
        if (sgn(d - R + r) < 0) { //内含，0 切线
            return n;
        } else if (sgn(d - R + r) == 0) { //内切，1 切线
            p[n++] = v.trunc(c2.r) + c2.o;
            p[n++] = v.rotate(pi / 2, P(0, 0)) + p[n - 2];
            return n;
        } else if (sgn(d - R + r) > 0) { //相交或相离，都有水平切线
            th = acos((R - r) / d);
            p[n++] = v.trunc(c1.r).rotate(th, P(0, 0)) + c1.o;
            p[n++] = v.trunc(c2.r).rotate(th, P(0, 0)) + c2.o;
            p[n++] = v.trunc(c1.r).rotate(2 * pi - th, P(0, 0)) + c1.o;
            p[n++] = v.trunc(c2.r).rotate(2 * pi - th, P(0, 0)) + c2.o;
        }
        if (sgn(d - R - r) == 0) { //外切，1 条垂直切线
            p[n++] = v.trunc(c1.r) + c1.o;
            p[n++] = v.rotate(pi / 2, P(0, 0)) + p[n - 2];
        } else if (sgn(d - R - r) > 0) { //相离有 2 条交叉切线
            th = acos((R + r) / d);
            p[n++] = v.trunc(c1.r).rotate(th, P(0, 0)) + c1.o;
            p[n++] = v.trunc(c2.r).rotate(th, P(0, 0)) + c2.o;
            p[n++] = v.trunc(c1.r).rotate(2 * pi - th, P(0, 0)) + c1.o;
            p[n++] = v.trunc(c2.r).rotate(2 * pi - th, P(0, 0)) + c2.o;
        }
        return n;
    }

    //点是否在圆内(边上)
    friend int incircle(const P& p, const circle& c) {
        int sign = sgn(dist(p,c.o) - c.r);
        if (sign < 0) return 1; //相交
        else if (sign == 0) return 2; //相切
        else return 0; //相离
    }/*因为圆为凸集,所以判断点集,折线,多边形是否在圆内时,只需要逐一判断点是否在圆内即可*/

    //判断圆是否在多边形内
    friend bool inpolygon(const circle& c, P p[], int n) {
        double mi = inf;
        for (int i = 0; i < n; i++) {
            double t = dist(c.o, seg(p[i], p[(i + 1) % n]));
            mi = min(mi, t);
        }
        return sgn(mi - c.r) >= 0;
    }

    void in() {
        o.in();
        scanf("%lf", &r);
    }
    void out() {
        o.out();
        printf("%lf\n", r);
    }
    double area() {
        return pi * sqr(r);
    }
    double perimeter() {
        return 2 * pi * r;
    }
};

//以下椭圆相关

//椭圆可以利用 2 个参数 d e 得到:(x/d)^2+(y/e)^2=1

//直线与椭圆的交点,返回值为交点个数
```

```
//直线ax+by+c=0, 椭圆(x/d)^2+(y/e)^2=1
//x=(-c*ad*d±be*d*sqrt(ad^2+be^2-c^2))/(ad^2+be^2)
//y=(-c*be*e±ad*e*sqrt(ad^2+be^2-c^2))/(ad^2+be^2)
int line_ellipse_intersection(double d, double e, const L& l, P& p1, P& p2) {
    double ab = sqr(l.a) * sqr(d) + sqr(l.b) * sqr(e), delta = ab - sqr(l.c);
    if (sgn(delta) < 0) {
        return 0;
    } else if (sgn(delta) == 0) {
        p1.x = -l.c * l.a * sqr(d) / ab;
        p1.y = -l.c * l.b * sqr(e) / ab;
        return 1;
    } else {
        double sq = d * e * sqrt(delta);
        p1.x = (-l.c * l.a * sqr(d) + l.b * sq) / ab;
        p1.y = (-l.c * l.b * sqr(e) - l.a * sq) / ab;
        p2.x = (-l.c * l.a * sqr(d) - l.b * sq) / ab;
        p2.y = (-l.c * l.b * sqr(e) + l.a * sq) / ab;
        return 2;
    }
}

//以下矩形相关

//说明：因为矩形的特殊性，常用算法可以化简：
//1.判断矩形是否包含点：只要判断该点的横坐标和纵坐标是否夹在矩形的左右边和上下边之间
//2.判断线段、折线、多边形是否在矩形中：因为矩形是个凸集，所以只要判断所有端点是否都在矩形中就可以了
//3.判断圆是否在矩形中：圆在矩形中的充要条件是：圆心在矩形中且圆的半径小于等于圆心到矩形四边的距离的最小值
struct rect {
    P A, B, C, D;//
    //已知矩形的三个顶点(a, b, c)(可以无序)，计算第四个顶点d的坐标
    friend P rect4th(const P& a, const P& b, const P& c) {
        if (sgn(dot(c, a, b)) == 0) return a + b - c; //说明c点是直角拐角处
        if (sgn(dot(b, a, c)) == 0) return a + c - b; //说明b点是直角拐角处
        if (sgn(dot(a, c, b)) == 0) return c + b - a; //说明a点是直角拐角处
        return P(0, 0);//error
    }
};

//以下三角形相关
struct triangle {
    //三角形外接圆
    friend circle circumcircle(const P& a, const P& b, const P& c) { //三角形的外接圆
        circle C;
        double ab = dist(a,b), bc = dist(b,c), ac = dist(a,c),
               c1 = (sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y)) / 2,
               c2 = (sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y)) / 2;
        C.r = ab * bc * ac / (2 * fabs(cross(a, b, c)));
        C.o.x = (c1 * (a.y - c.y) - c2 * (a.y - b.y)) / ((a - b) * (a - c));
        C.o.y = (c1 * (a.x - c.x) - c2 * (a.x - b.x)) / ((a - c) * (a - b));
        return C;
    }//返回外接圆

    //三角形内切圆
    friend circle inscribedcircle(const P& a, const P& b, const P& c) { //三角形的内切圆
        circle C;
        double ab = dist(a,b), bc = dist(b,c), ac = dist(c,a);
        C.r = fabs(cross(a, b, c)) / (bc + ac + ab);//三角形内接圆的半径公式r = 2S/L
        C.o.x = (bc * a.x + ac * b.x + ab * c.x) / (bc + ac + ab);
        C.o.y = (bc * a.y + ac * b.y + ab * c.y) / (bc + ac + ab);
        return C;
    }//返回内切圆

    //三角形外心
    //三角形三条边的垂直平分线的交点，是外接圆的圆心
    //外心到三角形的三个顶点距离相等
    friend P circumcenter(const P& a, const P& b, const P& c) { //三角形外心
```

```
    line u, v;
    u.a = (a + b) / 2;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a = (a + c) / 2;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.x;
    P p;
    intersection(u, v, p);
    return p;
}

//三角形内心
//内心是三角形三条内角平分线的交点,是内切圆的圆心
//内心到三边距离相等(为内切圆半径)
friend P inscribedcenter(const P& a, const P& b, const P& c) {
    line u, v;
    double m, n;
    u.a = a;
    m = atan2(b.y - a.y, b.x - a.x);
    n = atan2(c.y - a.y, c.x - a.x);
    u.b.x = u.a.x + cos((m + n) / 2);
    u.b.y = u.a.y + sin((m + n) / 2);
    v.a = b;
    m = atan2(a.y - b.y, a.x - b.x);
    n = atan2(c.y - b.y, c.x - b.x);
    v.b.x = v.a.x + cos((m + n) / 2);
    v.b.y = v.a.y + sin((m + n) / 2);
    P p;
    intersection(u, v, p);
    return p;
}

//三角形垂心
//三角形的三条高的交点叫做三角形的垂心
//锐角三角形垂心在三角形内部
//直角三角形垂心在三角形直角顶点
//钝角三角形垂心在三角形外部
friend P perpencenter(const P& a, const P& b, const P& c) {
    line u, v;
    u.a = c;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a = b;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.x;
    P p;
    intersection(u, v, p);
    return p;
}

//三角形重心
//三角形的三条中线相交的交点
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点
//重心到顶点的距离与重心到对边中点的距离之比为 2：1
//重心和三角形 3 个顶点组成的 3 个三角形面积相等
//重心和三角形 3 个顶点的连线的任意一条连线将三角形面积平分
friend P barycenter(const P& a, const P& b, const P& c) {
    line u, v;
    u.a = (a + b) / 2;
    u.b = c;
    v.a = (a + c) / 2;
    v.b = b;
    P p;
    intersection(u, v, p);
```

```
        return p;
    }

    //三角形费马点
    //到三角形三顶点距离之和最小的点
    //(1)若三角形 ABC 的 3 个内角均小于 120°，那么 3 条距离连线正好平分费马点所在的周角，故也称为三角形的等角中心。
    //三内角皆小于 120°的三角形，分别以 AB,BC,CA 为边向三角形外侧做正三角形 ABC1,ACB1,BCA1,
    //然后连接 AA1,BB1,CC1,则三线交于一点 P,则点 P 就是所求的费马点.
    //(2)若三角形有一内角不小于 120 度，则此钝角的顶点就是距离和最小的点。
    //(3)当△ABC 为等边三角形时，此时外心与费马点重合
    friend P fermatpoint(const P& a, const P& b, const P& c) {
        //精度很差, 慎用!!!
        P u, v;
        double step = fabs(a.x) + fabs(a.y) + fabs(b.x) + fabs(b.y) + fabs(c.x) + fabs(c.y);
        u = (a + b + c) / 3;
        while (sgn(step) > 0) {
            for (int k = 0; k < 10; step /= 2, k++) {
                for (int i = -1; i <= 1; i++) {
                    for (int j = -1; j <= 1; j++) {
                        v.x = u.x + step * i;
                        v.y = u.y + step * j;
                        if (dist(u, a) + dist(u, b) + dist(u, c) >
                            dist(v, a) + dist(v, b) + dist(v, c) )
                            u = v;
                    }
                }
            }
        }
        return u;
    }
};

//以下凸包相关
const int MAXN = 10001;

//逆时针排序
bool cmp(const P& p1, const P& p2) {
    return sgn(p1.y - p2.y) < 0 || (sgn(p1.y - p2.y) == 0 && sgn(p1.x - p2.x) < 0);
}

//极角排序
bool polarangle(const P& p1, const P& p2) {
    return sgn((p1 * p2)) >= 0;
}

struct polygon { //从 0..n-1
    P p[MAXN], ch[MAXN];
    int n, num;//点数 n 及凸包中的点数 num

    //凸包 graham 算法
    //对点集 p 求凸包，返回凸包点数，凸包存在 ch 中，凸包的点数存在 num 中，求得点集按逆时针排序
    int graham() {
        sort(p, p + n, cmp);
        if (n == 0) {num = 0; return 0;} ch[0] = p[0];
        if (n == 1) {num = 1; return 1;} ch[1] = p[1];
        if (n == 2) {num = 2; return 2;} ch[2] = p[2];
        int top = 1;
        for (int i = 2;i < n;i++) {
            while (top && sgn(cross(ch[top - 1], p[i], ch[top])) >= 0)
                top--;
            ch[++top] = p[i];
        }
        int len = top;
        ch[++top] = p[n - 2];
        for (int i = n - 3; i >= 0; i--) {
            while (top != len && sgn(cross(ch[top - 1], p[i], ch[top])) >= 0)
```

```
            top--;
        ch[++top] = p[i];
    }
    num = top;
    return top;//返回凸包中点的个数,点逆时针
}

//返回凸包面积
double area() {
    double S = 0;
    for (int i = 1;i < num - 1;i++)
        S += cross(ch[0], ch[i], ch[i + 1]) / 2;
    return S;
}

//返回凸包周长
double perimeter() {
    double L = 0;
    ch[num] = ch[0];
    for (int i = 0; i < num; i++)
        L += dist(ch[i], ch[i + 1]);
    return L;
}

//O(n)返回凸包直径
//利用旋转卡壳O(n)求平面点集上的最远点对
double diameter() {
    int top = 1;
    double d = 0;
    ch[num] = ch[0];
    for (int i = 0; i < num; i++) {
        while (sgn(cross(ch[i], ch[i + 1], ch[top + 1]) - cross(ch[i], ch[i + 1], ch[top])) > 0)
            top = (top + 1) % num;
        d = max(d, dist(ch[i], ch[top]));
        d = max(d, dist(ch[i + 1], ch[top + 1]));//处理平行边
    }
    return d;
}

//凸多边形重心
P gravitycenter() {
    double area = 0;
    P g;
    for (int i = 0; i < num; i++) {
        area += (ch[i] * ch[(i + 1) % num]) / 2;
        g.x += (ch[i] * ch[(i + 1) % num]) * (ch[i].x + ch[(i + 1) % num].x);
        g.y += (ch[i] * ch[(i + 1) % num]) * (ch[i].y + ch[(i + 1) % num].y);
    }
    //注意area不为0
    g.x /= 6 * area;
    g.y /= 6 * area;
    return g;
}

//O(n)判断点是否在凸多边形内
//凸多边形内的点三角剖分后面积和等于凸多边形面积和
friend bool inconvex(P& p, polygon& ch) {
    double area = 0;
    ch.ch[ch.num] = ch.ch[0];
    for (int i = 0; i < ch.num; i++)
        area += fabs(cross(p, ch.ch[i], ch.ch[i + 1]));
    area /= 2;
    return sgn(area - ch.area()) == 0;
}

//O(logn)判断点是否在凸多边形内
```

```
    //注意是否为凸多边形以及点数小于 3 时是否要特殊考虑
    friend bool inconvex2(P p, P ch[], int n) {
        P p1  = ch[0], p2 = ch[1];
        for (int i = n - 1; sgn(cross(ch[0], ch[1], ch[i])) == 0; i--)
            p1 = ch[i];
        for (int i = 2; sgn(cross(ch[0], ch[1], ch[i])) == 0; i++)
            p2 = ch[i];
        double sign = sgn(cross(p1, p2, p));
        if (sign < 0) return false;
        if (sign == 0) return sgn(dot(p, p1, p2)) <= 0;
        int l = 1, r = n - 1;
        while (l != r) {
            int mid = (l + r + 1) / 2;
            if (sgn(cross(ch[0], ch[mid], p)) >= 0) {
                l = mid;
            } else {
                r = mid - 1;
            }
        }
        if (l == n - 1)
            return sgn(cross(p, ch[0], ch[l])) == 0 && dot(p, ch[0], ch[l]) <= 0;
        return sgn(cross(p, ch[l], ch[l + 1])) >= 0;
    }

    //O(n^2)凸多边形 1、2 交出凸多边形 3
    friend void intersection(polygon& CH1, polygon& CH2, polygon &CH3) {
        CH1.ch[CH1.num] = CH1.ch[0];
        CH2.ch[CH2.num] = CH2.ch[0];
        CH3.n = 0;
        P inter;
        for (int i = 0; i < CH1.num; i++) {
            int cnt = 0;
            //插入凸包 1 上的线段与凸包 2 上的线段的交点
            for (int j = 0; j < CH2.num && cnt < 2; j++) {
                int num = intersection(seg(CH1.ch[i], CH1.ch[i + 1]), seg(CH2.ch[j], CH2.ch[j +
1]),inter);
                if (num == 6) {//线段重合
                    CH3.p[CH3.n++] = CH1.ch[i];
                    CH3.p[CH3.n++] = CH1.ch[i + 1];
                    cnt += 2;
                }
                else if (num) {//有交点
                    CH3.p[CH3.n++] = inter;
                    cnt++;
                }
            }
            //插入凸包 1 上的在凸包 2 内的点
            if (inconvex2(CH1.ch[i], CH2.ch, CH2.num))
                CH3.p[CH3.n++] = CH1.ch[i];
            if (inconvex2(CH1.ch[i + 1], CH2.ch, CH2.num))
                CH3.p[CH3.n++] = CH1.ch[i + 1];
        }
        //插入凸包 2 上的在凸包 1 内的点
        for (int i = 0; i < CH2.num; i++) {
            if (inconvex2(CH2.ch[i], CH1.ch, CH1.num))
                CH3.p[CH3.n++] = CH2.ch[i];
            if (inconvex2(CH2.ch[i + 1], CH1.ch, CH1.num))
                CH3.p[CH3.n++] = CH2.ch[i + 1];
        }
    }
}CH;

//以下旋转卡壳相关
struct rotatecalipers {
    //O(n)求凸多边形的直径
    //可以用于对普通点集求最远距离：先求点集凸包，再调用此函数
```

```
double convexdiameter(P ch[], int n) {
    double miny = 1e20, maxy = -1e20;
    int p, q, nextp, nextq;
    for (int i = 0; i < n; i++) {
        if (miny > ch[i].y) {
            miny = ch[i].y;
            p = i;
        }
        if (maxy < ch[i].y) {
            maxy = ch[i].y;
            q = i;
        }
    }
    double d = dist(ch[q], ch[p]);
    for (int i = 0; i <= n; i++) { //是等号=
        nextp = (p + 1) % n;
        nextq = (q + 1) % n;
        int sign = sgn((ch[nextp] - ch[p]) * (ch[nextq] - ch[q]));
        if (sign < 0) {
            p = nextp;
            d = max(d, dist(ch[p], ch[q]));
        }
        else if (sign > 0) {
            q = nextq;
            d = max(d, dist(ch[p], ch[q]));
        }
        else {
            d = max(d, dist(ch[p], ch[nextq]));
            d = max(d, dist(ch[nextp], ch[q]));
            d = max(d,dist(ch[nextp], ch[nextq]));
            p = nextp;
            q = nextq;
        }
    }
    return d;
}

//O(n)求两个凸多边形的最短距离
double mintwoconvexdist(P ch1[], int n, P ch2[], int m) {
    double miny = 1e20, maxy = -1e20;
    int p, q, nextp, nextq;
    for (int i = 0; i < n; i++) {
        if (ch1[i].y < miny) {
            miny = ch1[i].y;
            p = i;
        }
    }
    for (int i = 0; i < m; i++) {
        if (ch2[i].y > maxy) {
            maxy = ch2[i].y;
            q = i;
        }
    }
    double d = dist(ch1[p], ch2[q]);
    for (int i = 0; i <= n + m; i++) { //是等号=
        nextp = (p + 1) % n;
        nextq = (q + 1) % m;
        int sign = sgn((ch1[nextp] - ch1[p]) * (ch2[nextq] - ch2[q]));
        if (sign < 0) {
            seg s(ch1[p], ch1[nextp]);
            p = nextp;
            d = min(d, dist(ch2[q], s));
        } else if (sign > 0) {
            seg s(ch2[q], ch2[nextq]);
            q = nextq;
            d = min(d, dist(ch1[p], s));
```

```
            } else {
                seg s(ch1[p], ch1[nextp]);
                d = min(d, dist(ch2[q], s));
                d = min(d, dist(ch2[nextq], s));
                s = seg(ch2[q], ch2[nextq]);
                d = min(d, dist(ch1[p], s));
                d = min(d, dist(ch1[nextp], s));
                p = nextp;
                q = nextq;
            }
        }
        return d;
    }

    //O(n)求最大面积三角形
    double maxtrianglearea(P ch[], int n) {
        if (n < 3) return 0;
        double S = 0, area, prev;
        ch[n] = ch[0];
        for (int i = 0; i < n; i++) {
            int j = i + 1, k = j + 1;
            if (k >= n) break;
            prev = 0;
            while (j < n - 1) {
                area = cross(ch[i], ch[j], ch[k]);
                while (k < n - 1) {
                    if (sgn(cross(ch[i], ch[j], ch[k + 1]) - area) > 0) {
                        area = cross(ch[i], ch[j], ch[k + 1]);
                        k++;
                    } else {
                        break;
                    }
                }
                if (area < prev) break;
                prev = area;
                S = max(S, area);
                j++;
                if (j >= k) {
                    k = j + 1;
                    if (k >= n) break;
                }
            }
        }
        return S / 2;
    }
}G;

//以下是常用函数

//O(n)判点在任意多边形内,顶点按顺时针或逆时针给出
//修改参数 on_edge 可以判断点是否在多边形上， offset 为多边形坐标上限
int inpolygon(const P& q, int n, P p[], int on_edge = 1) {
    P q2;
    p[n] = p[0];
    int cnt;
    double offset = 1e4 + 4;
    for (int i = 0; i < n; ) {
        for (cnt = i = 0, q2.x = rand() + offset, q2.y = rand() + offset; i < n; i++) {
            if (sgn(cross(q, p[i], p[i + 1])) == 0
                    && sgn((p[i].x - q.x) * (p[i + 1].x - q.x)) <= 0
                    && sgn((p[i].y - q.y) * (p[i + 1].y - q.y)) <= 0) {
                return on_edge;
            } else if (sgn(cross(q, q2, p[i])) == 0) {
                break;
            } else if (sgn(cross(q, p[i], q2) * cross(q, p[i + 1], q2)) < 0
                    && sgn(cross(p[i], q, p[i + 1]) * cross(p[i], q2, p[i + 1])) < 0) {
```

```
                cnt++;
            }
        }
    }
    return cnt & 1;
}

//counterclockwise 可以判别三点关系
int ccw(const P& a, const P& b, const P& c) {
    if (sgn((b - a) * (c - a)) > 0) return 1;        //顺时针
    if (sgn((b - a) * (c - a)) < 0) return -1;       //逆时针
    if (sgn((b - a) ^ (c - a)) < 0) return +2;       //c--a--b 共线
    if (sgn((b - a).len() < (c - a).len())) return -2;//a--b--c 共线
    return 0;                                        //a--c--b 共线
}

//求向量 v 的极角，角度范围为 [0,2*pi)
double angle(const P& v) {
    double t = atan2(v.y, v.x);
    return sgn(t) >= 0 ? t : t + 2 * pi;
}

//求两向量之间的夹角[0,pi] ，乘以 cross(a,b)后便成了有向夹角。
double angle(const P& a, const P& b) {
    return acos((a ^ b) / sqrt(sqr(a.len()) * sqr(b.len())));
    //return (a * b > 0) * (acos(a ^ b / sqrt(sqr(a.len()) * sqr(b.len())));)
}

// 角度修正函数，根据不同的题目有不同写法和功能
void fixangle(double & rad) { //默认转化到(-pi,pi]
    if (sgn(rad - pi) > 0) rad -= 2 * pi;
    if (sgn(rad + pi) < 0) rad += 2 * pi;
}

//将弧度角规范化到[0,2 * pi)
double adjust(double d) {
    d = fmod(d, 2 * pi);
    if (d < 0)
        d += 2 * pi;
    return d;
}

//计算∠bac 余弦值 如果想从余弦求夹角的话,注意反余弦函数的定义域是从 0 到 pi
double cos(const P& a, const P& b, const P& c) { //计算角 bac 余弦值
    double ab = dist(a, b), ac = dist(a, c), bc = dist(b, c);
    return (sqr(ab) + sqr(ac) + sqr(bc)) / (2 * ab * ac);
    //return (b - a) ^ (c - a) / (dist(a, b) * dist(a, c));
}

//返回顶点在 a,起始边为 ab,终止边为 ac 的夹角(弧度)
//角度小于 pi,返回正值 角度大于 pi 返回负值 可以用于求线段之间的夹角
//r = cross(o,a,b) / (dist(o,a) * dist(o,b))
//r' = cross(o,a,b)
//r >= 1 angle = 0;
//r <= -1 angle = -pi;
//-1 < r < 1 && r' > 0 angle = acos(r);
//-1 < r < 1 && r' <= 0 angle = -acos(r);
double angle(const P& o, const P& a, const P& b) { //夹角 aob 的弧度
    double C = (a - o) ^ (b - o) / (dist(o, a) * dist(o, b));
    if (sgn(C - 1) >= 0) return 0;
    if (sgn(C + 1) <= 0) return -pi;
    if (sgn((a - o) * (b - o)) > 0) return acos(C);//oa 在 ob 顺时针方向
    return -acos(C);
}

//求多边形面积 可凹可凸
```

```
double area(P p[], int n) { //求多边形面积逆时针正 顺时针负
    double S = 0;
    for (int i = 0; i < n; i++)
        S += p[i] * p[(i + 1) % n];
    return S / 2;
}

//判断顶点是否按逆时针排列
bool isanticlockwise(P p[], int n) { //如果输入顶点按逆时针排列,返回 true
    return area(p, n) > 0;
}

//判断是否为简单多边形
bool issimple(P p[], int n) { //判断是否为简单多边形
    P t;
    seg s1, s2;
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i - 2; j++) {
            s1.a = p[i], s1.b = p[i - 1];
            s2.a = p[j], s2.b = p[j + 1];
            if (intersection(s1, s2, t)) return 0;
        }
    }
    s2.a = p[0], s2.b = p[n - 1];
    for (int i = 1; i < n - 2; i++) {
        s1.a = p[i], s1.b = p[i + 1];
        if (intersection(s1, s2, t)) return 0;
    }
    return 1;
}

//已知入射线和镜面,求镜面反射线.
//mirror 为镜面直线方程(a x + b y + c = 0 ,下同)系数;
//light 为入射光直线方程系数;
//返回反射光直线方程系数.
//光有方向,使用时注意:入射光向量:<-light.b,light.a>;反射光向量:<ret.b,-ret.a>.
//不要忘记结果中可能会有"negative zeros-0.000"
L reflect(const L& mirror, const L& light) {
    double b = mirror.b * light.b + mirror.a * light.a,
           a = light.a * mirror.b - mirror.a * light.b;
    double m = (b * mirror.b + a * mirror.a) / (sqr(mirror.a) + sqr(mirror.b)),
           n = (a * mirror.b - b * mirror.a) / (sqr(mirror.a) + sqr(mirror.b));
    if (sgn(mirror.a * light.b - light.a * mirror.b - 1e-20) == 0) //平行
        return light;
    P p;//p 是入射线与镜面的交点
    p.x = (mirror.b * light.c - light.b * mirror.c) / (mirror.a * light.b - light.a * mirror.b);
    p.y = (light.a * mirror.c - mirror.a * light.c) / (mirror.a * light.b - light.a * mirror.b);
    return L(n, -m, m * p.y - p.x * n);
}

//判断 2 个共线向量是否同向
bool samedir(const P& a, const P& b) {
    return sgn(a ^ b) > 0;
}

//光的折射
//传入入射光线,界面直线方程 折射率
//返回是否折射,成功折射返回 1,折射光线在 l1 中,发生全反射返回 0
int refraction(line l0, line lp, line &l1, double u) { //入射光线 界面 折射光线 折射率
    P v = l0.b - l0.a, v2, p, o; //入射向量 折射向量
    P vs = (lp.b - lp.a).turn_left(); //法线
    if (!samedir(v, vs)) {
        vs = vs.turn_left().turn_left();
    }
    intersection(l0, lp, p);
    double th = angle(v, vs);
```

```
    if (sgn(sin(th) / u - 1) > 0 || sgn(sin(th) / u + 1) < 0) //全反射
        return 0;
    double th2 = asin(sin(th) / u);
    if (sgn(vs * v) >= 0) {
        v2 = vs.rotate(th2, o);
    } else {
        v2 = vs.rotate(-th2, o);
    }
    l1.a = p;
    l1.b = p + v2;
    return 1;
}

//光的反射
//传入平面 l 及入射点 p 和方向向量 d，返回新 p 和新 d
//返回 1 表示成功反射，0 表示与反射面不相交或相切
int reflection(P& p, P& d, line& l) {
    if (sgn(d * (l.a - l.b)) == 0) return 0;
    line k(p, p + d);
    P x;
    intersection(k, l, x);
    k.a = x, k.b = (l.a - l.b).turn_left() + x;
    double th = atan2(l.a.y - l.b.y, l.a.x - l.b.x);
    th += th - atan2(d.y, d.x);
    p = x;
    d.x = cos(th), d.y = sin(th);
    return 1;
}

//求多边形的重心 可凹可凸
//三角形的重心是中线的交点 平行四边形的重心是对角线的交点
//需要点逆时针(凹的不能直接排序)
P gravitycenter(P p[], int n) {
    double area = 0;
    P g;
    for (int i = 0; i < n; i++) {
        area += (p[i] * p[(i + 1) % n]) / 2;
        g.x += (p[i] * p[(i + 1) % n]) * (p[i].x + p[(i + 1) % n].x);
        g.y += (p[i] * p[(i + 1) % n]) * (p[i].y + p[(i + 1) % n].y);
    }
    //注意 area 不为 0
    g.x /= 6 * area;
    g.y /= 6 * area;
    return g;
}

//传入一个凸多边形以及一条端点不在凸多边形内的直线，传出两个被割开的凸多边形
void cut(const line& li, P p[], int n, P l[], int& ln, P r[], int& rn) {
    ln = rn = 0;
    for (int i = 0; i < n; i++) {
        int sign = sgn((p[i] - li.a) * (li.b - li.a));
        if (sign >= 0) l[ln++] = p[i];
        if (sign <= 0) r[rn++] = p[i];
        P t;
        int flag = intersection(li, line(p[i], p[(i + 1) % n]), t);
        if (flag == 1 && onseg(t, line(p[i], p[(i + 1) % n]))) {//注意要包含端点！
            l[ln++] = r[rn++] = t;
        }
    }
}

//给定凸多边形外一点，求在该点能将凸多边形切割成面积相等的两部分的方向向量
//传入一个凸多边形以及多边形外一点，返回方向向量 复杂度 O(NlogN)
P find_equal(P p[], int n, P o) {
    for (int i = 0; i < n; i++) {//先将基准点平移到原点
        p[i].x -= o.x;
```

```
        p[i].y -= o.y;
    }
    o = P(0.0, 0.0);
    P* all = new P[n], *ll = new P[n], *rr = new P[n];
    int ln, rn;
    for (int i = 0; i < n; i++) all[i] = p[i];
    sort(all, all + n, polarangle);
    double l = 1e100, r = -1e100;
    l = angle(all[0]), r = angle(all[n - 1]);
    if (l > r) r += 2 * pi;
    while (sgn(r - l) > 0) {
        double mid = (l + r) / 2;
        line li(o, P(cos(mid), sin(mid)));
        cut(li, p, n, ll, ln, rr, rn);
        if (sgn(area(ll, ln) - area(rr, rn)) < 0) {
            l = mid;
        } else {
            r = mid;
        }
    }
    delete all; delete ll; delete rr;
    return P(cos(l), sin(l));
}

//主函数
int main() {
    seg s1(P(2,2), P(3,4));
    seg s2 = left(s1, 1);
    s2.out();
    //待写：点在多边形内，线段在多边形内，半平面交
}
/*
公式：
球坐标公式：
直角坐标为 P(x, y, z) 时,对应的球坐标是(rsinφcosθ, rsinφsinθ, rcosφ),
其中 φ 是向量 OP 与 Z 轴的夹角,范围[0,π];是 OP 在 XOY 面上的投影到 X 轴的旋角,范围[0,2π]
直线的一般方程转化成向量方程：
ax+by+c=0
 x-x0    y-y0
----- = ------ // (x0,y0)为直线上一点,m,n 为向量
  m       n
转换关系：
a=n;b=-m;c=m*y0-n*x0;
m=-b;n=a;

三点平面方程：
三点为 P1, P2, P3
设向量  M1=P2-P1; M2=P3-P1;
平面法向量:M=M1 x M2
平面方程:M.i(x-P1.x)+M.j(y-P1.y)+M.k(z-P1.z)=0

经纬度
假设地球是球体，
设地球上某点的经度为 lambda,纬度为 phi，
则这点的空间坐标是
x = cos(phi)*cos(lambda)
y = cos(phi)*sin(lambda)
z = sin(phi)
设地球上两点的空间坐标分别为(x1,y1,z1),(x2,y2,z2)
直线距离即为 R*sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)+(z2-z1)*(z2-z1)),
则它们的夹角为 A = acos(x1 * x2 + y1 * y2 + z1 * z2)
球面距离为 A*R/2(R 即地球半径 6378.0)
扇形的重心 Xc = 2*R*sinA/3/A A 为圆心角的一半
*/
```

圆面积并 1

```
//圆面积并 复杂度 o(n^2logn)
//SPOJ CIRU / VCIRCLES
//参考 http://blog.csdn.net/jasonzhu8/article/details/6010980
//样例输入 3 3 4 5 4 5 6 5 6 7, 输出 159.060
#include <iostream>
#include <cmath>
using namespace std;
const double eps = 1e-15;
const double pi = acos(-1.0);
const int MAXN = 1000;
inline double sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
inline double sqr(double x) { return x * x; }
inline double angle(double x, double y) {//[0, 2 * pi)
    double ret = atan2(y, x);
    if (sgn(ret) < 0) ret += 2 * pi;
    return ret;
}
struct P { double x, y; };
struct circle { P o; double r; } c[MAXN];
struct arc {
    double l, r;
    inline bool operator <(const arc& a) const {
        return r < a.r;
    }
}a[MAXN];
bool cover[MAXN];
double circleunion1(circle c[], int n) {
    for (int i = 0; i < n; i++) cover[i] = false;
    double ans = 0;
    for (int i = 0; i < n; i++) {
        int m = 0;
        for (int j = 0; j < n; j++) if (i != j && !cover[j]) {
            double dist = hypot(c[i].o.x - c[j].o.x, c[i].o.y - c[j].o.y);
            if (sgn(dist + c[i].r - c[j].r) <= 0) {//圆 i 在圆 j 内
                cover[i] = true;
            } else if (sgn(dist + c[j].r - c[i].r) <= 0) {//圆 j 在圆 i 内,无交点
                continue;
            } else if (sgn(dist - c[i].r - c[j].r) < 0) {//不相离,有交点
                m++;
                double dx = (sqr(c[i].r) - sqr(c[j].r) + sqr(dist)) / 2 / dist;//圆 i 圆心到两圆交点连线
距离
                double dy = (sqrt(sqr(c[i].r) - dx * dx));//交点到圆心连线距离
                a[m].r = angle(c[j].o.x - c[i].o.x, c[j].o.y - c[i].o.y);
                a[m].l = a[m].r + angle(dx, dy);//弧 m 的起始弧度
                a[m].r = a[m].r - angle(dx, dy);//弧 m 的终止弧度
                if (sgn(a[m].r) < 0) {
                    a[m].l += 2 * pi;
                    a[m].r += 2 * pi;
                }
            }
        }
        if (m == 0 && !cover[i]) {//与其他圆都相离
            ans += pi * sqr(c[i].r);
        }
        if (m == 0 || cover[i]) {//内含于某个圆
            continue;
        }
        sort(a + 1, a + m + 1);
        int tt = 0, ss = 1;
        for (int j = 1; j <= m; j++) {
            if (tt == 0 || sgn(a[j].r - a[tt].l) > 0) {
```

```
                tt++;
                a[tt].l = a[j].l;
                a[tt].r = a[j].r;
            } else if (sgn(a[tt].l - a[j].l) < 0) {
                a[tt].l = a[j].l;
            }
        }
        while (ss <= tt && sgn(a[tt].l - 2 * pi - a[ss].r) > 0) {
            if (a[tt].l - 2 * pi < a[ss].l) {
                a[tt].l = a[ss].l + 2 * pi;
            }
            ss++;
        }
        for (int j = ss; j <= tt; j++) {
            int k = (j == tt ? ss : j + 1);//k是j的下一条弧
            double dx = a[k].r - a[j].l;
            if (sgn(dx) < 0) dx += 2 * pi;
            ans += sqr(c[i].r) / 2 * (dx - sin(dx));
            double x1 = c[i].o.x + c[i].r * cos(a[j].l);//最后覆盖完的弧j的起点x坐标
            double y1 = c[i].o.y + c[i].r * sin(a[j].l);//最后覆盖完的弧j的起点y坐标
            double x2 = c[i].o.x + c[i].r * cos(a[k].r);//最早未覆盖的弧k的终点x坐标
            double y2 = c[i].o.y + c[i].r * sin(a[k].r);//最早未覆盖的弧k的终点y坐标
            ans += (x1 * y2 - x2 * y1) / 2;
        }
    }
    return ans;
}

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf", &c[i].o.x, &c[i].o.y, &c[i].r);
        }
        printf("%.3lf\n", circleunion1(c, n));
    }
}
```

圆面积并 2

```
//圆面积并 复杂度o(n^2logn)
//SPOJ CIRU / VCIRCLES
//参考http://blog.csdn.net/pouy94/article/details/6425587
//样例输入3 3 4 5 4 5 6 5 6 7, 输出159.060
#include <iostream>
#include <cmath>
using namespace std;
const double eps = 1e-15;
const double pi = acos(-1.0);
const int MAXN = 1000;
inline double sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
inline double sqr(double x) { return x * x; }
inline double angle(double x, double y) {//[0, 2 * pi)
    double ret = atan2(y, x);
    if (sgn(ret) < 0) ret += 2 * pi;
    return ret;
}
struct P { double x, y; };

struct circle { P o; double r; }c[MAXN];
struct arc {
    double l, r;
```

```
    inline bool operator <(const arc& a) const {
        return l == a.l ? r < a.r : l < a.l;
    }
}a[MAXN * 2 + 1];
bool cover[MAXN];
double adjust(double x) {
    while (sgn(x) < 0) x += 2 * pi;
    while (sgn(x - 2 * pi) >= 0) x -= 2 * pi;
    return x;
}
double circleunion2(circle c[], int n) {
    for (int i = 0; i < n; i++) cover[i] = false;
    double ans = 0;
    for (int i = 0; i < n; i++) {
        int m = 0;
        for (int j = 0; j < n; j++) if (i != j && !cover[j]) {
            double dist = hypot(c[i].o.x - c[j].o.x, c[i].o.y - c[j].o.y);
            if (sgn(dist + c[i].r - c[j].r) <= 0) {//圆 i 在圆 j 内
                cover[i] = true;
            } else if (sgn(dist + c[j].r - c[i].r) <= 0) {//圆 j 在圆 i 内,无交点
                continue;
            } else if (sgn(dist - c[i].r - c[j].r) < 0) {//不相离,有交点
                double dx = (sqr(c[i].r) - sqr(c[j].r) + sqr(dist)) / 2 / dist;//圆 i 圆心到两圆交点连线
距离
                double dy = (sqrt(sqr(c[i].r) - dx * dx));//交点到圆心连线距离
                double tmp = angle(c[j].o.x - c[i].o.x, c[j].o.y - c[i].o.y);
                a[m].l = adjust(tmp - angle(dx, dy));//弧 m 的起始弧度
                a[m].r = adjust(tmp + angle(dx, dy));//弧 m 的终止弧度
                m++;
                if (sgn(a[m - 1].r - a[m - 1].l) <= 0) {//拆成 2 个区间
                    a[m].l = 0;
                    a[m].r = a[m - 1].r;
                    a[m - 1].r = 2 * pi;
                    m++;
                }
            }
        }
        if (m == 0 && !cover[i]) {//与其他圆都相离
            ans += pi * sqr(c[i].r);
        }
        if (m == 0 || cover[i]) {//内含于某个圆
            continue;
        }
        sort(a, a + m);
        a[m].l = a[0].l + 2 * pi;
        a[m].r = a[0].r + 2 * pi;
        double st = a[0].r, ed;
        for (int j = 1; j <= m; j++) {
            if (sgn(a[j].l - st) >= 0) {
                ed = a[j].l;
                double dx = adjust(ed - st);
                ans += sqr(c[i].r) / 2 * (dx - sin(dx));
                double x1 = c[i].o.x + c[i].r * cos(st);//弧起点 x 坐标
                double y1 = c[i].o.y + c[i].r * sin(st);//弧起点 y 坐标
                double x2 = c[i].o.x + c[i].r * cos(ed);//弧终点 x 坐标
                double y2 = c[i].o.y + c[i].r * sin(ed);//弧终点 y 坐标
                ans += (x1 * y2 - x2 * y1) / 2;
            }
            st = max(st, a[j].r);
        }
    }
    return ans;
}
int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
```

```
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf", &c[i].o.x, &c[i].o.y, &c[i].r);
        }
        printf("%.3lf\n", circleunion2(c, n));
    }
}
```

圆交 K 次面积

```
//圆交 K 次面积算法，复杂度 O(n^2logn)
//SPOJ CIRUT
//参考 http://hi.baidu.com/aekdycoin/blog/item/c1b28e3711246b3f0b55a95e.html
//参考 http://acm.uestc.edu.cn/bbs/read.php?tid=3316
//样例输入 5 3 4 5 4 5 6 5 6 7 16 9 3 16 5 5, 输出[1] = 110.486 [2] = 60.699 [3] = 73.502 [4] = 0.000
[5] = 0.000
#include <iostream>
#include <cmath>
using namespace std;
const double eps = 1e-8;
const double pi = acos(-1.0);
const int MAXN = 1000;
inline double sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
inline double sqr(double x) { return x * x; }
inline double angle(double x, double y) {//[0, 2 * pi)
    double ret = atan2(y, x);
    if (sgn(ret) < 0) ret += 2 * pi;
    return ret;
}
struct P { double x, y; };
struct circle { P o; double r; }c[MAXN];
struct arc {
    double ag;
    int t;
    inline bool operator <(const arc& a) const {
        return ag == a.ag ? t > a.t : ag < a.ag;
    }
}a[4 * MAXN + 2];
double adjust(double x) {
    if (sgn(x) < 0) x += 2 * pi;
    if (sgn(x - 2 * pi) >= 0) x -= 2 * pi;
    return x;
}
double ans[MAXN];
void circleKunion(circle c[], int n) {
    for (int i = 0; i <= n; i++) ans[i] = 0;
    /*一定要去重才能正确
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (sgn(c[i].o.x - c[j].o.x) == 0 && sgn(c[i].o.y - c[j].o.y) == 0 && sgn(c[i].r - c[j].r)
== 0) {
                swap(c[j], c[n - 1]);
                n--;
            }
        }
    }
    */
    for (int i = 0; i < n; i++) {
        int m = 0, times = 0;
        for (int j = 0; j < n; j++) if (i != j) {
            double dist = hypot(c[i].o.x - c[j].o.x, c[i].o.y - c[j].o.y);
            if (sgn(dist + c[i].r - c[j].r) <= 0) {//圆 i 在圆 j 内
                times++;
```

```
            } else if (sgn(dist + c[j].r - c[i].r) <= 0) {//圆 j 在圆 i 内,无交点
                continue;
            } else if (sgn(dist - c[i].r - c[j].r) < 0) {//不相离,有交点
                double dx = (sqr(c[i].r) - sqr(c[j].r) + sqr(dist)) / 2 / dist;//圆 i 圆心到两圆交点连线
距离
                double dy = (sqrt(sqr(c[i].r) - dx * dx));//交点到圆心连线距离
                double tmp = angle(c[j].o.x - c[i].o.x, c[j].o.y - c[i].o.y);
                a[m].ag = adjust(tmp - angle(dx, dy));//弧 m 的起始弧度
                a[m++].t = 1;
                a[m].ag = adjust(tmp + angle(dx, dy));//弧 m 的终止弧度
                a[m++].t = -1;
                if (sgn(a[m - 1].ag - a[m - 2].ag) <= 0) {//拆成 2 个区间
                    a[m].ag = 0;
                    a[m].t = 1;
                    a[m + 1].ag = a[m - 1].ag;
                    a[m + 1].t = -1;
                    a[m - 1].ag = 2 * pi;
                    m += 2;
                }
            }
        }
        if (m == 0) {//与其他圆都相离
            ans[times] += pi * sqr(c[i].r);
            continue;
        }
        sort(a, a + m);
        a[m].ag = a[0].ag + 2 * pi;
        a[m++].t = 1;
        int s = times + a[0].t;
        for (int j = 1; j < m; j++) {
            int t = s; s += a[j].t;
            double dx = adjust(a[j].ag - a[j - 1].ag);
            ans[t] += sqr(c[i].r) / 2 * (dx - sin(dx));
            double x1 = c[i].o.x + c[i].r * cos(a[j - 1].ag);//弧起点 x 坐标
            double y1 = c[i].o.y + c[i].r * sin(a[j - 1].ag);//弧起点 y 坐标
            double x2 = c[i].o.x + c[i].r * cos(a[j].ag);//弧终点 x 坐标
            double y2 = c[i].o.y + c[i].r * sin(a[j].ag);//弧终点 y 坐标
            ans[t] += (x1 * y2 - x2 * y1) / 2;
        }
    }
    for (int i = 0; i < n; i++) {
        ans[i] -= ans[i + 1];
        printf("[%d] = %.3lf\n", i + 1 , ans[i]);
    }
}
int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf", &c[i].o.x, &c[i].o.y, &c[i].r);
        }
        circleKunion(c, n);
    }
}
```

圆与多边形面积交

```
//圆与简单多边形交 复杂度 O(n)
//利用了将多边形按圆心三角剖分 再计算每个三角形与圆交的有向面积
//PKU3675 ZJU2675 PKU2986 HDU2892
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
```

```
using namespace std;
const double eps = 1e-8;
inline int sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
struct P {
    double x, y;
};
double cross(const P& a, const P& b, const P& c) {
    return (a.x - c.x) * (b.y - c.y) - (a.y - c.y) * (b.x - c.x);
}
double dot(const P& a, const P& b, const P& c) {
    return (a.x - c.x) * (b.x - c.x) + (a.y - c.y) * (b.y - c.y);
}
double dist(const P& a, const P& b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
double Asin(double x) {
    if (sgn(x - 1) >= 0) {
        return asin(1.0);
    } else if (sgn(x + 1) <= 0) {
        return asin(-1.0);
    } else {
        return asin(x);
    }
}

double Acos(double x) {
    if (sgn(x - 1) >= 0) {
        return acos(1.0);
    } else if (sgn(x + 1) <= 0) {
        return acos(-1.0);
    } else {
        return acos(x);
    }
}

double calcarea(const P& a, const P& b, const P& c, double r) {
    if (fabs(cross(a, b, c)) < 1e-8) return 0;
    double A, B, C, x, y, tS;
    A = dist(b, c), B = dist(a, c), C = dist(b, a);
    if (sgn(A - r) < 0 && sgn(B - r) < 0) {
        return cross(a, b, c) / 2;
    } else if (sgn(A - r) < 0 && sgn(B - r) >= 0) {
        x = (dot(a, c, b) + sqrt(fabs(r * r * C * C - cross(a, c, b) * cross(a, c, b)))) / C;
        tS = cross(a, b, c) / 2;
        return Asin(tS * (1 - x / C) * 2 / r / B) * r * r / 2 + tS * x / C;
    } else if (sgn(A - r) >= 0 && sgn(B - r) < 0) {
        y = (dot(b, c, a) + sqrt(fabs(r * r * C * C - cross(b, c, a) * cross(b, c, a)))) / C;
        tS = cross(a, b, c) / 2;
        return Asin(tS * (1 - y / C) * 2 / r / A) * r * r / 2 + tS * y / C;
    } else if (sgn(fabs(cross(a, b, c)) - r * C) >= 0
            || sgn(dot(b, c, a)) <= 0 || sgn(dot(a, c, b)) <= 0) {
        if (sgn(dot(a, b, c)) < 0) {
            if (sgn(cross(a, b, c)) < 0) {
                return (-Acos(-1.0) - Asin(cross(a, b, c) / A / B)) * r * r / 2;
            } else {
                return (Acos(-1.0) - Asin(cross(a, b, c) / A / B)) * r * r / 2;
            }
        } else {
            return Asin(cross(a, b, c) / A / B) * r * r / 2;
        }
    } else {
        x = (dot(a, c, b) + sqrt(fabs(r * r * C * C - cross(a, c, b) * cross(a, c, b)))) / C;
        y = (dot(b, c, a) + sqrt(fabs(r * r * C * C - cross(b, c, a) * cross(b, c, a)))) / C;
        tS = cross(a, b, c) / 2;
```

```
        return (Asin(tS * (1 - x / C) * 2 / r / B) + Asin(tS * (1 - y / C) * 2 / r / A)) * r * r / 2
+ tS * ((y + x) / C - 1);
    }
}
P p[50], o;
int main() {
    o.x = o.y = 0;
    double r, total;
    while (scanf("%lf", &r) != EOF) {
        int n;
        scanf("%d", &n);
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf", &p[i].x, &p[i].y);
        }
        total = 0;
        for (int i = 0; i < n; i++) {
            total += calcarea(p[i], p[(i + 1) % n], o, r);
        }
        printf("%.2lf\n", fabs(total) + eps);
    }
    return 0;
}
```

凸多边形面积并

```
//凸多边形面积并 复杂度O(n^3)
//PKU3733 SPOJ-POLYU BZOJ1845 FZU1407
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cmath>
#define sqr(x) ((x)*(x))
using namespace std;
const int N = 110;
const int M = 20;
const double eps = 1e-8;
int n, n2, vis[N][M];
int sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
struct P {
    double x, y;
    P() {}
    P(double x, double y) : x(x), y(y) {}
    bool operator == (const P &u) const {
        return sgn(x - u.x) == 0 && sgn(y - u.y) == 0;
    }
    void in() {
        scanf("%lf%lf", &x, &y);
    }
}tp[N * M], bp;
struct poly {
    int n;
    P cp[M];
    P& operator [] (int k) {
        return cp[k];
    }
}ply[N], ply2[N];
double cross(P p0, P p1, P p2) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
}
double dot(P p0, P p1, P p2) {
    return (p1.x - p0.x) * (p2.x - p0.x) + (p1.y - p0.y) * (p2.y - p0.y);
```

```
}
int PointOnSegment(P p0, P p1, P p2) {
    return sgn(cross(p0, p1, p2)) == 0 && sgn(dot(p0, p1, p2)) <= 0;
}
int LineInter(P p1, P p2, P p3, P p4, P &cp) {
    double u = cross(p1, p2, p3), v = cross(p2, p1, p4);
    if ( sgn(u + v) ) {
        cp.x = (p3.x * v + p4.x * u) / (u + v);
        cp.y = (p3.y * v + p4.y * u) / (u + v);
        return 1;
    }
    if ( sgn(u) ) return 0;
    if ( sgn(cross(p3, p4, p1)) ) return 0;
    return -1;
}
int SegmentInter(P p1, P p2, P p3, P p4, P &cp) {
    int ret = LineInter(p1, p2, p3, p4, cp);
    if (ret == 1) return PointOnSegment(cp, p1, p2) && PointOnSegment(cp, p3, p4);
    if (ret == -1 && (PointOnSegment(p1, p3, p4) || PointOnSegment(p2, p3, p4)
                || PointOnSegment(p3, p1, p2) || PointOnSegment(p4, p1, p2) ))
      return -1;
    return 0;
}
bool cmp(const P &u, const P &v) {
    return sgn( fabs(u.x - bp.x) + fabs(u.y - bp.y)
             - fabs(v.x - bp.x) - fabs(v.y - bp.y)) < 0;
}
bool check(P u, P v, poly ply[], int id) {
    int i, k, d1, d2, wn = 0;
    P cp, *p1, *p2 = ply[id].cp;
    cp.x = (u.x + v.x) / 2;
    cp.y = (u.y + v.y) / 2;
    for (i = 0; i < ply[id].n; ++i) {
        p1 = p2++;
        if (PointOnSegment(cp, *p1, *p2)) {
            int chk = sgn(u.x - v.x) * sgn(p1->x - p2->x);
            if (chk == 0)
                chk = sgn(u.y - v.y) * sgn(p1->y - p2->y);
            if (chk == -1) return 0;
            return vis[id][i];
        }
        k = sgn( cross(*p1, *p2, cp) );
        d1 = sgn( p1->y - cp.y );
        d2 = sgn( p2->y - cp.y );
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    return wn != 0;
}
double PolyUnion(poly ply[], int &n) {
    double ans = 0;
    int i, j, k, vi, vj, tn = n; n = 0;
    P cp, p1, p2;
    memset(vis, 0, sizeof(vis));
    for (i = 0; i < tn; ++i) {
        if (sgn(cross(ply[i][0], ply[i][1], ply[i][2])) <= 0)
            continue; // 去除共线多边形
        ply[i][ply[i].n] = ply[i][0];
        ply[n++] = ply[i];
    }
    for (i = 0; i < n; ++i) {
        for (vi = 0; vi < ply[i].n; ++vi) {
            tp[0] = p1 = bp = ply[i][vi];
            tp[1] = p2 = ply[i][vi + 1];
            tn = 2;
            for (j = 0; j < n; ++j) if (i != j)
```

```
            for (vj = 0; vj < ply[j].n; ++vj)
                if (SegmentInter(p1, p2, ply[j][vj], ply[j][vj + 1], cp) == 1)
                    tp[tn++] = cp;
            sort(tp, tp + tn, cmp);
            tn = unique(tp, tp + tn) - tp;
            for (k = 1; k < tn; ++k) {
                for (j = 0; j < n; ++j) if (i != j)
                    if (check(tp[k - 1], tp[k], ply, j))
                        break;
                if (j == n)
                    ans += tp[k - 1].x * tp[k].y - tp[k - 1].y * tp[k].x;
            }
            vis[i][vi] = 1;
        }
    }
    return ans / 2;
}
double dissqr(P u, P v) {
    return sqr(u.x - v.x) + sqr(u.y - v.y);
}
int PolarCmp(const P &p1, const P &p2) {
    int u = sgn(cross(bp, p1, p2));
    return u > 0 || (u == 0 && sgn(dissqr(bp, p1)-dissqr(bp, p2)) < 0);
}
void graham(P pin[], int n, P ch[], int &m) {
    int i, j, k, u, v;
    memcpy(ch, pin, n * sizeof(P));
    for (i = k = 0; i < n; ++i) {
        u = sgn(ch[i].x - ch[k].x);
        v = sgn(ch[i].y - ch[k].y);
        if (v < 0 || (v == 0 && u < 0)) k = i;
    }
    bp = ch[k];
    sort(ch, ch + n, PolarCmp);
    n =  unique(ch, ch + n) - ch;
    if (n <= 1) { m = n; return ;}
    if (sgn(cross(ch[0], ch[1], ch[n - 1])) == 0) {
        m = 2; ch[1] = ch[n - 1]; return;
    }
    ch[n++] = ch[0];
    for (i = 1, j = 2; j < n; ++j) {
        while (i > 0 && sgn(cross(ch[i - 1], ch[i], ch[j])) <= 0) i--;
        ch[++i] = ch[j];
    }
    m = i;
}
void solve(int cas) {
    double h, x, y, bh;
    scanf("%lf%lf%lf", &h, &x, &y);
    int m; n = n2 = 0;
    scanf("%d", &m);
    P a[5], tmp[20];
    double ans = 0;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < 3; ++j)
            a[j].in();
        scanf("%lf", &bh);
        bool flag = false;
        for (int j = 0; j < 3; ++j) {
            if (sgn(dot(a[j], a[(j + 1) % 3], a[(j + 2) % 3])) == 0) {
                swap(a[j], a[1]);
                a[3].x = a[0].x + a[2].x - a[1].x;
                a[3].y = a[0].y + a[2].y - a[1].y;
                flag = true;
                break;
            }
```

```
        }
        if (sgn(cross(a[0], a[1], a[2])) < 0)
            reverse(a, a + 4);
        a[4] = a[0];
        int tn = 0;
        for (int j = 0; j < 4; ++j) {
            tmp[tn++] = a[j];
            tmp[tn++] = P(a[j].x + x * bh / h, a[j].y + y * bh / h);
        }
        graham(tmp, tn, ply[n].cp, ply[n].n);
        n++;
        ans -= fabs(cross(a[0], a[1], a[2]));
    }
    ans += PolyUnion(ply, n);
    printf("Case %d: %.3lf\n", cas, ans);
}
int main() {
    int cas;
    scanf("%d", &cas);
    for (int i = 1; i <= cas; ++i) {
        solve(i);
    }
    return 0;
}
```

N 维矩形面积并

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;
#define DIM 10
#define Mod 14121413
struct Hyper {
    int _d, _s[DIM], _e[DIM];
    long long product() const {
        long long ret = 1;
        for (int i = 0; i < _d; ++i)
            ret = ret * (_e[i] - _s[i]) % Mod;
        return ret;
    }
};
vector<Hyper> res, tres;
//------ cut like this ------
//      *---*---*---*
//      |   |___|   |
//      |   |___|   |
//      |   |   |   |
//      *---*---*---*
//----------------------------
//-- n dimensions 2*n cuts --
void cut(Hyper p, const Hyper& q) {
    for (int i = 0; i < p._d; ++i) {
        int s = p._s[i], e = p._e[i];
        if (s < q._s[i] && q._s[i] < e) {
            p._e[i] = q._s[i];
            tres.push_back(p);
        }
        if (s < q._e[i] && q._e[i] < e) {
            p._s[i] = q._e[i];
            p._e[i] = e;
            tres.push_back(p);
        }
```

```
        p._s[i] = max(s, q._s[i]);
        p._e[i] = min(e, q._e[i]);
    }
}
bool collide(const Hyper& p, const Hyper& q) {
    for (int i = 0; i < p._d; ++i)
        if (p._s[i] >= q._e[i] || q._s[i] >= p._e[i])
            return false;
    return true;
}
void do_with(const Hyper& t) {
    tres.clear();
    for (int i = 0; i < res.size(); ++i)
        if (collide(res[i], t)) cut(res[i], t);
        else tres.push_back(res[i]);
    tres.push_back(t);
    tres.swap(res);
}
long long get_union_area(vector<Hyper>& vec) {
    res.clear();
    for (int i = 0; i < vec.size(); ++i) {
        for (int j = 0; j < vec[i]._d; ++j)
            if (vec[i]._s[j] > vec[i]._e[j])
                swap(vec[i]._s[j], vec[i]._e[j]);
        do_with(vec[i]);
    }
    long long sum = 0;
    for (int i = 0; i < res.size(); ++i)
        sum = (sum + res[i].product()) % Mod;
    return sum;
}
int main() {
    int n, m;
    vector<Hyper> v;
    while (scanf("%d%d", &n, &m) != EOF) {
        v.clear();
        Hyper t;
        t._d = m;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j)
                scanf("%d", &t._s[j]);
            for (int j = 0; j < m; ++j) {
                scanf("%d", &t._e[j]);
                if (t._s[j] > t._e[j])
                    swap(t._s[j], t._e[j]);
            }
            v.push_back(t);
        }
        printf("%I64d\n", get_union_area(v));
    }
    return 0;
}
```

三维几何

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
using namespace std;
const double eps = 1e-8;
const double pi = acos(-1.0);
```

```
const double inf = 1e200;
inline int sgn(double x) {
    return x < -eps ? -1 : x > eps ? 1 : 0;
}
inline double sqr(double x) {
    return x * x;
}
inline double a2r(double a) { //角度到弧度
    return a * pi / 180;
}
inline double r2a(double r) { //弧度到角度
    return r / pi * 180;
}
//以下三维点
struct P {
    double x, y, z;
    P(double x = 0, double y = 0, double z = 0): x(x), y(y), z(z) {};
    friend P operator +(const P& p1, const P& p2) {
        return P(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z);
    }
    friend P operator -(const P& p1, const P& p2) {
        return P(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);
    }
    friend P operator *(const P& p, double a) {
        return P(p.x * a, p.y * a, p.z * a);
    }
    friend P operator /(const P& p, double a) {
        return P(p.x / a, p.y / a, p.z / a);
    }
    friend P operator *(const P& a, const P& b) { //叉乘
        return P(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
    }
    friend double operator ^(const P& a, const P& b) { //点乘
        return a.x * b.x + a.y * b.y + a.z * b.z;
    }
    friend double dist(const P& a, const P& b) {
        return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z));
    }
    double len() {
        return sqrt(fabs(*this ^ *this));
    }
    double len2() {
        return *this ^ *this;
    }
    bool in() {
        return scanf("%lf %lf %lf", &x, &y, &z) == 3;
    }
    void out() const {
        printf("%lf %lf %lf\n", x, y, z);
    }
};

//点的旋转
//点 p 绕向量(s->e)顺时针旋转 angle 弧度(右手定则)
P rotate(P p, P s, P e, double angle) {
    e = e - s, p = p - s;
    double Cos = cos(angle), Sin = sin(angle);
    e  = e / e.len();
    double M[3][3] = {
        e.x*e.x*(1 - Cos) + Cos    , e.x*e.y*(1 - Cos) - e.z*Sin, e.x*e.z*(1 - Cos) + e.y*Sin,
        e.y*e.x*(1 - Cos) + e.z*Sin, e.y*e.y*(1 - Cos) + Cos    , e.y*e.z*(1 - Cos) - e.x*Sin,
        e.x*e.z*(1 - Cos) - e.y*Sin, e.y*e.z*(1 - Cos) + e.x*Sin, e.z*e.z*(1 - Cos) + Cos
    };
    P ret;
    ret.x = p.x * M[0][0] + p.y * M[0][1] + p.z * M[0][2] + s.x;
    ret.y = p.x * M[1][0] + p.y * M[1][1] + p.z * M[1][2] + s.y;
```

```
    ret.z = p.x * M[2][0] + p.y * M[2][1] + p.z * M[2][2] + s.z;
    return ret;
}

//坐标系旋转
//把点 p 所在的法向量 v=s->e 的平面转到 xoy 平面后点 p 的平面坐标
P planarizartion(P p, P s, P e) {
    e = e - s;
    double th1 = atan2(e.y, e.z); //把 v 转至 xoz 平面的夹角
    double th2 = 2 * pi - atan2(e.x, sqrt(e.y * e.y + e.z * e.z)); //把 v 从 xoz 平面转至 z 轴
    P o(0, 0, 0), nx(1, 0, 0), ny(0, 1, 0);
    p = rotate(p, o, nx, th1);
    p = rotate(p, o, ny, th2);
    return p;
}

//以下平面相关
struct plane { //平面用一般式表示 ax+by+cz+d=0
    double a, b, c, d;
    plane() {}
    plane(double a, double b, double c, double d): a(a), b(b), c(c), d(d) {}

    //点到平面的距离
    double dist(const P& p, const plane& f) {
        return fabs(f.a * p.x + f.b * p.y + f.c * p.z + f.d) / sqrt(sqr(a) + sqr(b) + sqr(c));
    }
};

//以下四面体相关
struct tetrahedron { //四面体 默认顶点为 A 底面三角形 BCD
    P a, b, c, d;
    tetrahedron() {}
    tetrahedron(const P&a, const P&b, const P&c, const P&d): a(a), b(b), c(c), d(d) {
    }
    P gravitycenter(const P&a, const P&b, const P&c, const P&d) {
        return (a + b + c + d) / 4;
    }
};
int main() {
    P s(-2, -4, 0), e(-2, 4, 0), p(-2, 4, 4);
    //s = P(0,0,0),e = P(0,2,0);
    rotate(p, s, e, pi / 2).output();
}
```

三维几何 [ZJU]

```
//三维几何函数库
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
using namespace std;
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point3 {
    double x, y, z;
};
struct line3 {
    point3 a, b;
};
struct plane3 {
    point3 a, b, c;
```

```
};
struct planef {
    double a, b, c, d;
}; // ax+by+cz+d=0
//平方
inline double sqr(double d) {
    return d*d;
}
//计算cross product U x V
point3 xmult(point3 u, point3 v) {
    point3 ret;
    ret.x = u.y * v.z - v.y * u.z;
    ret.y = u.z * v.x - u.x * v.z;
    ret.z = u.x * v.y - u.y * v.x;
    return ret;
}
//计算dot product U . V
double dmult(point3 u, point3 v) {
    return u.x*v.x + u.y*v.y + u.z*v.z;
}
//矢量差 U - V
point3 subt(point3 u, point3 v) {
    point3 ret;
    ret.x = u.x - v.x;
    ret.y = u.y - v.y;
    ret.z = u.z - v.z;
    return ret;
}
//取平面法向量
point3 pvec(plane3 s) {
    return xmult(subt(s.a, s.b), subt(s.b, s.c));
}
point3 pvec(point3 s1, point3 s2, point3 s3) {
    return xmult(subt(s1, s2), subt(s2, s3));
}
point3 pvec(planef p) {
    point3 ret;
    ret.x = p.a;
    ret.y = p.b;
    ret.z = p.c;
    return ret;
}
//两点距离,单参数取向量大小
double dist(point3 p1, point3 p2) {
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y) + (p1.z - p2.z)*(p1.z - p2.z));
}
//向量大小
double vlen(point3 p) {
    return sqrt(p.x*p.x + p.y*p.y + p.z*p.z);
}
double sqrlen(point3 p) {
    return dmult(p, p);
}
//判三点共线
int dots_inline(point3 p1, point3 p2, point3 p3) {
    return sqrlen(xmult(subt(p1, p2), subt(p2, p3))) < eps;
}
//判四点共面
int dots_onplane(point3 a, point3 b, point3 c, point3 d) {
    return zero(dmult(pvec(a, b, c), subt(d, a)));
}
//判点是否在线段上,包括端点和共线
int dot_online_in(point3 p, line3 l) {
    return zero(sqrlen(xmult(subt(p, l.a), subt(p, l.b)))) && (l.a.x - p.x)*(l.b.x - p.x) < eps &&
           (l.a.y - p.y)*(l.b.y - p.y) < eps && (l.a.z - p.z)*(l.b.z - p.z) < eps;
}
```

```
int dot_online_in(point3 p, point3 l1, point3 l2) {
    return zero(sqrlen(xmult(subt(p, l1), subt(p, l2)))) && (l1.x - p.x)*(l2.x - p.x) < eps &&
           (l1.y - p.y)*(l2.y - p.y) < eps && (l1.z - p.z)*(l2.z - p.z) < eps;
}
//判点是否在线段上,不包括端点
int dot_online_ex(point3 p, line3 l) {
    return dot_online_in(p, l) && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y) || !zero(p.z - l.a.z))
&&
           (!zero(p.x - l.b.x) || !zero(p.y - l.b.y) || !zero(p.z - l.b.z));
}
int dot_online_ex(point3 p, point3 l1, point3 l2) {
    return dot_online_in(p, l1, l2) && (!zero(p.x - l1.x) || !zero(p.y - l1.y) || !zero(p.z - l1.z))
&&
           (!zero(p.x - l2.x) || !zero(p.y - l2.y) || !zero(p.z - l2.z));
}
//判点是否在空间三角形上,包括边界,三点共线无意义
int dot_inplane_in(point3 p, plane3 s) {
    return zero(vlen(xmult(subt(s.a, s.b), subt(s.a, s.c))) - vlen(xmult(subt(p, s.a), subt(p, s.b)))
- vlen(xmult(subt(p, s.b), subt(p, s.c))) - vlen(xmult(subt(p, s.c), subt(p, s.a))));
}
int dot_inplane_in(point3 p, point3 s1, point3 s2, point3 s3) {
    return zero(vlen(xmult(subt(s1, s2), subt(s1, s3))) - vlen(xmult(subt(p, s1), subt(p, s2))) -
              vlen(xmult(subt(p, s2), subt(p, s3))) - vlen(xmult(subt(p, s3), subt(p, s1))));
}
//判点是否在空间三角形上,不包括边界,三点共线无意义
int dot_inplane_ex(point3 p, plane3 s) {
    return dot_inplane_in(p, s) && sqrlen(xmult(subt(p, s.a), subt(p, s.b))) > eps &&
sqrlen(xmult(subt(p, s.b), subt(p, s.c))) > eps && sqrlen(xmult(subt(p, s.c), subt(p, s.a))) > eps;
}
int dot_inplane_ex(point3 p, point3 s1, point3 s2, point3 s3) {
    return dot_inplane_in(p, s1, s2, s3) && sqrlen(xmult(subt(p, s1), subt(p, s2))) > eps &&
sqrlen(xmult(subt(p, s2), subt(p, s3))) > eps && sqrlen(xmult(subt(p, s3), subt(p, s1))) > eps;
}
//判两点在线段同侧,点在线段上返回 0,不共面无意义
int same_side(point3 p1, point3 p2, line3 l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p2, l.b))) > eps;
}
int same_side(point3 p1, point3 p2, point3 l1, point3 l2) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2))) > eps;
}
//判两点在线段异侧,点在线段上返回 0,不共面无意义
int opposite_side(point3 p1, point3 p2, line3 l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p2, l.b))) < -eps;
}
int opposite_side(point3 p1, point3 p2, point3 l1, point3 l2) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2))) < -eps;
}
//判两点在平面同侧,点在平面上返回 0
int same_side(point3 p1, point3 p2, plane3 s) {
    return dmult(pvec(s), subt(p1, s.a))*dmult(pvec(s), subt(p2, s.a)) > eps;
}
int same_side(point3 p1, point3 p2, point3 s1, point3 s2, point3 s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1))*dmult(pvec(s1, s2, s3), subt(p2, s1)) > eps;
}
int same_side(point3 p1, point3 p2, planef s) {
    return (s.a*p1.x + s.b*p1.y + s.c*p1.z + s.d)*(s.a*p2.x + s.b*p2.y + s.c*p2.z + s.d) > eps;
}
//判两点在平面异侧,点在平面上返回 0
int opposite_side(point3 p1, point3 p2, plane3 s) {
    return dmult(pvec(s), subt(p1, s.a))*dmult(pvec(s), subt(p2, s.a)) < -eps;
}
int opposite_side(point3 p1, point3 p2, point3 s1, point3 s2, point3 s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1))*dmult(pvec(s1, s2, s3), subt(p2, s1)) < -eps;
}
int opposite_side(point3 p1, point3 p2, planef s) {
    return (s.a*p1.x + s.b*p1.y + s.c*p1.z + s.d)*(s.a*p2.x + s.b*p2.y + s.c*p2.z + s.d) < -eps;
```

```
}
//判两直线平行
int parallel(line3 u, line3 v) {
    return sqrlen(xmult(subt(u.a, u.b), subt(v.a, v.b))) < eps;
}
int parallel(point3 u1, point3 u2, point3 v1, point3 v2) {
    return sqrlen(xmult(subt(u1, u2), subt(v1, v2))) < eps;
}
//判两平面平行
int parallel(plane3 u, plane3 v) {
    return sqrlen(xmult(pvec(u), pvec(v))) < eps;
}
int parallel(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    return sqrlen(xmult(pvec(u1, u2, u3), pvec(v1, v2, v3))) < eps;
}
int parallel(planef u, planef v) {
    return sqrlen(xmult(pvec(u), pvec(v))) < eps;
}
//判直线与平面平行
int parallel(line3 l, plane3 s) {
    return zero(dmult(subt(l.a, l.b), pvec(s)));
}
int parallel(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return zero(dmult(subt(l1, l2), pvec(s1, s2, s3)));
}
int parellel(line3 l, planef s) {
    return zero(dmult(subt(l.a, l.b), pvec(s)));
}
//判两直线垂直
int perpendicular(line3 u, line3 v) {
    return zero(dmult(subt(u.a, u.b), subt(v.a, v.b)));
}
int perpendicular(point3 u1, point3 u2, point3 v1, point3 v2) {
    return zero(dmult(subt(u1, u2), subt(v1, v2)));
}
//判两平面垂直
int perpendicular(plane3 u, plane3 v) {
    return zero(dmult(pvec(u), pvec(v)));
}
int perpendicular(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    return zero(dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)));
}
int perpendicular(planef u, planef v) {
    return zero(dmult(pvec(u), pvec(v)));
}
//判直线与平面垂直
int perpendicular(line3 l, plane3 s) {
    return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < eps;
}
int perpendicular(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return sqrlen(xmult(subt(l1, l2), pvec(s1, s2, s3))) < eps;
}

int perpendicular(line3 l, planef s) {
    return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < eps;
}
//判两线段相交,包括端点和部分重合
int intersect_in(line3 u, line3 v) {
    if (!dots_onplane(u.a, u.b, v.a, v.b))
        return 0;
    if (!dots_inline(u.a, u.b, v.a) || !dots_inline(u.a, u.b, v.b))
        return !same_side(u.a, u.b, v) && !same_side(v.a, v.b, u);
    return dot_online_in(u.a, v) || dot_online_in(u.b, v) || dot_online_in(v.a, u) || dot_online_in(v.b,
u);
}
int intersect_in(point3 u1, point3 u2, point3 v1, point3 v2) {
```

```
    if (!dots_onplane(u1, u2, v1, v2))
        return 0;
    if (!dots_inline(u1, u2, v1) || !dots_inline(u1, u2, v2))
        return !same_side(u1, u2, v1, v2) && !same_side(v1, v2, u1, u2);
    return dot_online_in(u1, v1, v2) || dot_online_in(u2, v1, v2) || dot_online_in(v1, u1, u2) ||
dot_online_in(v2, u1, u2);
}

//判两线段相交,不包括端点和部分重合
int intersect_ex(line3 u, line3 v) {
    return dots_onplane(u.a, u.b, v.a, v.b) && opposite_side(u.a, u.b, v) && opposite_side(v.a, v.b,
u);
}
int intersect_ex(point3 u1, point3 u2, point3 v1, point3 v2) {
    return dots_onplane(u1, u2, v1, v2) && opposite_side(u1, u2, v1, v2) && opposite_side(v1, v2, u1,
u2);
}

//判线段与空间三角形相交,包括交于边界和(部分)包含
int intersect_in(line3 l, plane3 s) {
    return !same_side(l.a, l.b, s) && !same_side(s.a, s.b, l.a, l.b, s.c) &&
          !same_side(s.b, s.c, l.a, l.b, s.a) && !same_side(s.c, s.a, l.a, l.b, s.b);
}
int intersect_in(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return !same_side(l1, l2, s1, s2, s3) && !same_side(s1, s2, l1, l2, s3) &&
          !same_side(s2, s3, l1, l2, s1) && !same_side(s3, s1, l1, l2, s2);
}

//判线段与空间三角形相交,不包括交于边界和(部分)包含
int intersect_ex(line3 l, plane3 s) {
    return opposite_side(l.a, l.b, s) && opposite_side(s.a, s.b, l.a, l.b, s.c) &&
          opposite_side(s.b, s.c, l.a, l.b, s.a) && opposite_side(s.c, s.a, l.a, l.b, s.b);
}
int intersect_ex(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return opposite_side(l1, l2, s1, s2, s3) && opposite_side(s1, s2, l1, l2, s3) &&
          opposite_side(s2, s3, l1, l2, s1) && opposite_side(s3, s1, l1, l2, s2);
}

//计算两直线交点,注意事先判断直线是否共面和平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point3 intersection(line3 u, line3 v) {
    point3 ret = u.a;
    double t = xmult(subt(u.a, v.a), subt(v.a, v.b)) / xmult(subt(u.a, u.b), subt(v.a, v.b));
    ret.x += (u.b.x - u.a.x) * t;
    ret.y += (u.b.y - u.a.y) * t;
    ret.z += (u.b.z - u.a.z) * t;
    return ret;
}
point3 intersection(point3 u1, point3 u2, point3 v1, point3 v2) {
    point3 ret = u1;
    double t = xmult(subt(u1, v1), subt(v1, v2)) / xmult(subt(u1, u2), subt(v1, v2));
    ret.x += (u2.x - u1.x) * t;
    ret.y += (u2.y - u1.y) * t;
    ret.z += (u2.z - u1.z) * t;
    return ret;
}
//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
//线段和空间三角形交点请另外判断
point3 intersection(line3 l, plane3 s) {
    point3 ret = pvec(s);
    double t = dmult(ret, subt(s.a, l.a)) / dmult(ret, subt(l.b, l.a));
    ret.x = l.a.x + (l.b.x - l.a.x) * t;
    ret.y = l.a.y + (l.b.y - l.a.y) * t;
    ret.z = l.a.z + (l.b.z - l.a.z) * t;
    return ret;
}
```

```
point3 intersection(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    point3 ret = pvec(s1, s2, s3);
    double t = dmult(ret, subt(s1, l1)) / dmult(ret, subt(l2, l1));
    ret.x = l1.x + (l2.x - l1.x) * t;
    ret.y = l1.y + (l2.y - l1.y) * t;
    ret.z = l1.z + (l2.z - l1.z) * t;
    return ret;
}
point3 intersection(line3 l, planef s) {
    point3 ret = subt(l.b, l.a);
    double t = -(dmult(pvec(s), l.a) + s.d) / (dmult(pvec(s), ret));
    ret.x = ret.x * t + l.a.x;
    ret.y = ret.y * t + l.a.y;
    ret.z = ret.z * t + l.a.z;
    return ret;
}
//计算两平面交线,注意事先判断是否平行,并保证三点不共线!
line3 intersection(plane3 u, plane3 v) {
    line3 ret;
    ret.a = parallel(v.a, v.b, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.c) : intersection(v.a,
v.b, u.a, u.b, u.c);
    ret.b = parallel(v.c, v.a, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.c) : intersection(v.c,
v.a, u.a, u.b, u.c);
    return ret;
}
line3 intersection(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    line3 ret;
    ret.a = parallel(v1, v2, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : intersection(v1, v2,
u1, u2, u3);
    ret.b = parallel(v3, v1, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : intersection(v3, v1,
u1, u2, u3);
    return ret;
}
//点到直线距离
double disptoline(point3 p, line3 l) {
    return vlen(xmult(subt(p, l.a), subt(l.b, l.a))) / dist(l.a, l.b);
}
double disptoline(point3 p, point3 l1, point3 l2) {
    return vlen(xmult(subt(p, l1), subt(l2, l1))) / dist(l1, l2);
}
//点到直线最近点
point3 ptoline(point3 p, line3 l) {
    point3 ab = subt(l.b, l.a);
    double t = -dmult(subt(p, l.a), ab) / sqrlen(ab);
    ab.x *= t;
    ab.y *= t;
    ab.z *= t;
    return subt(l.a, ab);
}
//点到平面距离
double disptoplane(point3 p, plane3 s) {
    return fabs(dmult(pvec(s), subt(p, s.a))) / vlen(pvec(s));
}
double disptoplane(point3 p, point3 s1, point3 s2, point3 s3) {
    return fabs(dmult(pvec(s1, s2, s3), subt(p, s1))) / vlen(pvec(s1, s2, s3));
}

double disptoplane(point3 p, planef s) {
    return fabs((dmult(pvec(s), p) + s.d) / vlen(pvec(s)));
}
//点到平面最近点
point3 ptoplane(point3 p, planef s) {
    line3 l;
    l.a = p;
    l.b = pvec(s);
    l.b.x += p.x;
```

```
    l.b.y += p.y;
    l.b.z += p.z;
    return intersection(l, s);
}
//直线到直线距离
double dislinetoline(line3 u, line3 v) {
    point3 n = xmult(subt(u.a, u.b), subt(v.a, v.b));
    return fabs(dmult(subt(u.a, v.a), n)) / vlen(n);
}
double dislinetoline(point3 u1, point3 u2, point3 v1, point3 v2) {
    point3 n = xmult(subt(u1, u2), subt(v1, v2));
    return fabs(dmult(subt(u1, v1), n)) / vlen(n);
}
//直线到直线的最近点对
//p1 在 u 上，p2 在 v 上，p1 到 p2 是 uv 之间的最近距离
//注意，保证两直线不平行
void linetoline(line3 u, line3 v, point3 &p1, point3 &p2) {
    point3 ab = subt(u.b, u.a), cd = subt(v.b, v.a), ac = subt(v.a, u.a);
    double r = (dmult(ab, cd) * dmult(ac, ab) - sqrlen(ab) * dmult(ac, cd)) /
               (sqrlen(ab) * sqrlen(cd) - sqr(dmult(ab, cd)));
    p2.x = v.a.x + r * cd.x, p2.y = v.a.y + r * cd.y,  p2.z = v.a.z + r * cd.z;
    p1 = ptoline(p2, u);
}
//两直线夹角 cos 值
double angle_cos(line3 u, line3 v) {
    return dmult(subt(u.a, u.b), subt(v.a, v.b)) / vlen(subt(u.a, u.b)) / vlen(subt(v.a, v.b));
}
double angle_cos(point3 u1, point3 u2, point3 v1, point3 v2) {
    return dmult(subt(u1, u2), subt(v1, v2)) / vlen(subt(u1, u2)) / vlen(subt(v1, v2));
}
//两平面夹角 cos 值
double angle_cos(plane3 u, plane3 v) {
    return dmult(pvec(u), pvec(v)) / vlen(pvec(u)) / vlen(pvec(v));
}
double angle_cos(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    return dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)) / vlen(pvec(u1, u2, u3)) / vlen(pvec(v1, v2, v3));
}
double angle_cos(planef u, planef v) {
    return dmult(pvec(u), pvec(v)) / (vlen(pvec(u))*vlen(pvec(v)));
}
//直线平面夹角 sin 值
double angle_sin(line3 l, plane3 s) {
    return dmult(subt(l.a, l.b), pvec(s)) / vlen(subt(l.a, l.b)) / vlen(pvec(s));
}
double angle_sin(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return dmult(subt(l1, l2), pvec(s1, s2, s3)) / vlen(subt(l1, l2)) / vlen(pvec(s1, s2, s3));
}
double angle_sin(line3 l, planef s) {
    return dmult(subt(l.a, l.b), pvec(s)) / (vlen(subt(l.a, l.b)) * vlen(pvec(s)));
}
//平面方程形式转化 plane3 -> planef
planef plane3_to_planef(plane3 p) {
    planef ret;
    point3 m = xmult(subt(p.b, p.a), subt(p.c, p.a));
    ret.d = -m.x * p.a.x - m.y * p.a.y - m.z * p.a.z;
    return ret;
}
int main() {
}
```

三维凸包

```
/*************************************
  Name: 3D Convex Hull
```

```
  Author: Isun
  Date: 1-10-10 17:20
  Description: 求三维空间点集凸包 —增量法 O(n^2)
**************************************/
#include <stdio.h>
#include <math.h>
#include <algorithm>
#define eps 1e-8
#define MAXV 1010
using namespace std;
//三维点
struct pt{
    double x, y, z;
    pt(){}
    pt(double _x, double _y, double _z): x(_x), y(_y), z(_z){}
    pt operator - (const pt p1){return pt(x - p1.x, y - p1.y, z - p1.z);}
    pt operator * (pt p){return pt(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);}        //叉乘
    double operator ^ (pt p){return x*p.x+y*p.y+z*p.z;}                            //点乘
};
struct _3DCH{
    struct fac{
        int a, b, c;    //表示凸包一个面上三个点的编号
        bool ok;        //表示该面是否属于最终凸包中的面
    };
    int n;    //初始点数
    pt P[MAXV];    //初始点
    int cnt;    //凸包表面的三角形数
    fac F[MAXV*8]; //凸包表面的三角形
    int to[MAXV][MAXV];
    double vlen(pt a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}    //向量长度
    double area(pt a, pt b, pt c){return vlen((b-a)*(c-a));}    //三角形面积*2
    double volume(pt a, pt b, pt c, pt d){return (b-a)*(c-a)^(d-a);}    //四面体有向体积*6
    //正：点在面同向
    double ptof(pt &p, fac &f){
        pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
        return (m * n) ^ t;
    }
    void deal(int p, int a, int b){
        int f = to[a][b];
        fac add;
        if (F[f].ok){
            if (ptof(P[p], F[f]) > eps)
                dfs(p, f);
            else{
                add.a = b, add.b = a, add.c = p, add.ok = 1;
                to[p][b] = to[a][p] = to[b][a] = cnt;
                F[cnt++] = add;
            }
        }
    }
    void dfs(int p, int cur){
        F[cur].ok = 0;
        deal(p, F[cur].b, F[cur].a);
        deal(p, F[cur].c, F[cur].b);
        deal(p, F[cur].a, F[cur].c);
    }
    bool same(int s, int t){
        pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b, c, P[F[t].b])) < eps &&
fabs(volume(a, b, c, P[F[t].c])) < eps;
    }
    //构建三维凸包
    void construct(){
        cnt = 0;
        if (n < 4)
            return;
```

```
    bool sb = 1;
    //使前两点不共点
    for (int i = 1; i < n; i++){
        if (vlen(P[0] - P[i]) > eps){
            swap(P[1], P[i]);
            sb = 0;
            break;
        }
    }
    if (sb)return;
    sb = 1;
    //使前三点不共线
    for (int i = 2; i < n; i++){
        if (vlen((P[0] - P[1]) * (P[1] - P[i])) > eps){
            swap(P[2], P[i]);
            sb = 0;
            break;
        }
    }
    if (sb)return;
    sb = 1;
    //使前四点不共面
    for (int i = 3; i < n; i++){
        if (fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) > eps){
            swap(P[3], P[i]);
            sb = 0;
            break;
        }
    }
    if (sb)return;
    fac add;
    for (int i = 0; i < 4; i++){
        add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;
        if (ptof(P[i], add) > 0)
            swap(add.b, add.c);
        to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
        F[cnt++] = add;
    }
    for (int i = 4; i < n; i++){
        for (int j = 0; j < cnt; j++){
            if (F[j].ok && ptof(P[i], F[j]) > eps){
                dfs(i, j);
                break;
            }
        }
    }
    int tmp = cnt;
    cnt = 0;
    for (int i = 0; i < tmp; i++){
        if (F[i].ok){
            F[cnt++] = F[i];
        }
    }
}
//表面积
double area(){
    double ret = 0.0;
    for (int i = 0; i < cnt; i++){
        bool nb = 1;
        for (int j = 0; j < i && nb; j++)
            if (same(i, j))
                nb = 0;
        if (nb)
            ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);
    }
    return ret / 2.0;
```

```
    }
    //体积
    double volume(){
        pt O(0, 0, 0);
        double ret = 0.0;
        for (int i = 0; i < cnt; i++){
            bool nb = 1;
            for (int j = 0; j < i && nb; j++)
                if (same(i, j))
                    nb = 0;
            if (nb)
                ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);
        }
        return fabs(ret / 6.0);
    }
    //表面三角形数
    int facetCnt_tri(){
        return cnt;
    }
    //表面多边形数
    int facetCnt(){
        int ans = 0;
        for (int i = 0; i < cnt; i++){
            bool nb = 1;
            for (int j = 0; j < i && nb; j++)
                if (same(i, j))
                    nb = 0;
            ans += nb;
        }
        return ans;
    }
};
_3DCH hull;
int main(){
    int tc;
    scanf("%d", &tc);
    for (int t = 1; t <= tc; t++){
        scanf("%d", &hull.n);
        for (int i = 0; i < hull.n; i++)
            scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y, &hull.P[i].z);
        hull.construct();
        printf("Case %d: %d\n", t, hull.facetCnt());
    }
    return 0;
}
```

三维线段距离[平方分数形式]

```
//返回的是三维空间中两条线段的距离的平方
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef long long T;
inline int sgn(T x) {
    if (x < 0) return -1;
    if (x == 0) return 0;
    return 1;
}
inline T gcd(T a, T b) {
    while (b != 0) {
        T c = a % b;
        a = b;
        b = c;
```

```
    }
    return a;
}
inline T sqr(T x) { return x * x; }
struct Fact {
    T _l, _m;
    Fact(T l, T m): _l(l), _m(m) {
        T g = gcd(_l, _m);
        _l /= g; _m /= g;
        if (_l == 0) _m = 1;
    }
    Fact operator - (const Fact& rh) const {
        return Fact(_l * rh._m - rh._l * _m, _m * rh._m);
    }
    bool operator < (const Fact& rh) const {
        return _l * rh._m < rh._l * _m;
    }
};
struct Vec {
    T _x, _y, _z;
    Vec(T x, T y, T z): _x(x), _y(y), _z(z) {}
    Vec operator + (const Vec& rh) const {
        return Vec(_x + rh._x, _y + rh._y, _z + rh._z);
    }
    Vec operator - (const Vec& rh) const {
        return Vec(_x - rh._x, _y - rh._y, _z - rh._z);
    }
    Vec operator * (const Vec& rh) const {    // cross product
        return Vec(
            _y * rh._z - rh._y * _z,
            _z * rh._x - rh._z * _x,
            _x * rh._y - rh._x * _y
        );
    }
    T operator % (const Vec& rh) const {    // dot product
        return _x * rh._x + _y * rh._y + _z * rh._z;
    }
    T norm() const {
        return _x * _x + _y * _y + _z * _z;
    }
};
inline bool intersect(const Vec& p1, const Vec& f, const Vec& p3, const Vec& p4) {
    return sgn(f % (p3 - p1)) * sgn(f % (p4 - p1)) <= 0;
}
inline Fact dist1(const Vec& f, const Vec& p1, const Vec& p3) {    // project of seg on vector f
    return Fact(sqr((p3 - p1) % f), f.norm());
}
inline Fact dist2(const Vec& p, const Vec& p1, const Vec& p2) {    // point to line
    T l1 = (p - p1).norm(), m1 = 1;
    T l2 = sqr((p - p1) % (p1 - p2)), m2 = (p1 - p2).norm();
    Fact n1(l1, m1), n2(l2, m2);
    return n1 - n2;
}
inline bool project_in(const Vec& p, const Vec& p1, const Vec &p2) {
    return sgn((p - p1) % (p2 - p1)) >= 0 &&
        sgn((p - p1) % (p2 - p1) - (p1 - p2).norm()) <= 0;
}
inline bool share_line(const Vec& p, const Vec& p1, const Vec &p2) {
    return sgn(sqr((p - p1) % (p2 - p1)) - (p - p1).norm() * (p2 - p1).norm()) == 0;
}
Fact segment_distance(const Vec& p1, const Vec &p2, const Vec& p3, const Vec &p4) {
    Vec s1 = p2 - p1, s2 = p4 - p3;
    Vec f = s1 * s2;
    bool flag = intersect(p1, s1 * f, p3, p4) && intersect(p3, s2 * f, p1, p2);
    if (sgn(f.norm()) == 0)       // share plane or p1 = p2 or p3 = p4
        flag = false;
```

```
    if (flag)
        return dist1(f, p1, p3);
    else {
        Fact ans((p1 - p3).norm(), 1);
        ans = min(ans, Fact((p2 - p3).norm(), 1));
        ans = min(ans, Fact((p1 - p4).norm(), 1));
        ans = min(ans, Fact((p2 - p4).norm(), 1));
        if (sgn(s2.norm()) > 0 && project_in(p1, p3, p4))
            ans = min(ans, dist2(p1, p3, p4));
        if (sgn(s2.norm()) > 0 && project_in(p2, p3, p4))
            ans = min(ans, dist2(p2, p3, p4));
        if (sgn(s1.norm()) > 0 && project_in(p3, p1, p2))
            ans = min(ans, dist2(p3, p1, p2));
        if (sgn(s1.norm()) > 0 && project_in(p4, p1, p2))
            ans = min(ans, dist2(p4, p1, p2));
        return ans;
    }
}
int main() {
    int test;
    scanf("%d", &test);
    for (int cas = 1; cas <= test; ++cas) {
        int x1, y1, z1, x2, y2, z2;
        scanf("%d%d%d%d%d%d", &x1, &y1, &z1, &x2, &y2, &z2);
        Vec p1(x1, y1, z1), p2(x2, y2, z2);
        scanf("%d%d%d%d%d%d", &x1, &y1, &z1, &x2, &y2, &z2);
        Vec p3(x1, y1, z1), p4(x2, y2, z2);
        Fact ans = segment_distance(p1, p2, p3, p4);
        printf("%lld %lld\n", ans._l, ans._m);
    }
    return 0;
}
```

两球体积交

```
double Sphere_Intersection(P S1, double r1, P S2, double r2) {
    if (r1 > r2) {
        swap(S1, S2);
        swap(r1, r2);
    }
    double d, h1, h2, volume;
    d = sqrt(sqr(S1.x - S2.x) + sqr(S1.y - S2.y) + sqr(S1.z - S2.z));//球心距
    if (sgn(r1 + d - r2) <= 0)//球 2 包含球 1
        return 4.0 / 3.0 * pi * pow(r1, 3.0);
    if (sgn(r1 + r2 - d) <= 0)//相离
        return 0;
    h1 = (r2 - r1 + d) * (r2 + r1 - d) / (2 * d);
    h2 = (r1 - r2 + d) * (r1 + r2 - d) / (2 * d);
    volume = pi * sqr(h1) * (3 * r1 - h1) / 3.0;
    volume += pi * sqr(h2) * (3 * r2 - h2) / 3.0;
    return volume;
}
```

两圆面积交

```
double Circle_Intersection(P C1, double r1, P C2, double r2) {
    if (r1 > r2) {
        swap(C1, C2);
        swap(r1, r2);
    }
    double d, a1, a2, p, area;
```

```
    d = sqrt(sqr(C1.x - C2.x) + sqr(C1.y - C2.y));//圆心距
    if (sgn(r1 + d - r2) <= 0)//圆 2 包含圆 1
        return pi * sqr(r1);
    if (sgn(r1 + r2 - d) <= 0)//相离
        return 0;
    a1 = acos((sqr(r1) + sqr(d) - sqr(r2)) / (2 * r1 * d));
    a2 = acos((sqr(r2) + sqr(d) - sqr(r1)) / (2 * r2 * d));
    p = (r1 + r2 + d) / 2;
    area = 2 * sqrt(p * (p - r1) * (p - r2) * (p - d));
    area = a1 * sqr(r1) + a2 * sqr(r2) - area;
    return area;
}
```

两圆公切线

```
/*
传入两圆，返回切线条数 * 2，存放在 p[]中，(p[2*i], p[2*i+1])构成一条切线
两圆重合返回-1，无数条切线，内含返回 0 条切线
内切返回 1 条切线，相交返回 2 条切线，外切返回 3 条切线，相离返回 4 条切线
*/
int tangentline(circle c1, circle c2, P p[]) {
    int n = 0;
    if (c1.r < c2.r) swap(c1, c2);
    double r = min(c1.r, c2.r), R = max(c1.r, c2.r);
    double d = dist(c1.o, c2.o), th;
    P v = c2.o - c1.o;
    if (sgn(d) == 0 && sgn(R - r) == 0) {//重合，无数条切线
        return -1;
    }
    if (sgn(d - R + r) < 0) {//内含，0 切线
        return n;
    } else if (sgn(d - R + r) == 0) {//内切，1 切线
        p[n++] = v.trunc(c2.r) + c2.o;
        p[n++] = v.rotate(pi / 2, P(0, 0)) + p[n - 2];
        return n;
    } else if (sgn(d - R + r) > 0) {//相交或相离，都有水平切线
        th = acos((R - r) / d);
        p[n++] = v.trunc(c1.r).rotate(th, P(0, 0)) + c1.o;
        p[n++] = v.trunc(c2.r).rotate(th, P(0, 0)) + c2.o;
        p[n++] = v.trunc(c1.r).rotate(2 * pi - th, P(0, 0)) + c1.o;
        p[n++] = v.trunc(c2.r).rotate(2 * pi - th, P(0, 0)) + c2.o;
    }
    if (sgn(d - R - r) == 0) {//外切，1 条垂直切线
        p[n++] = v.trunc(c1.r) + c1.o;
        p[n++] = v.rotate(pi / 2, P(0, 0)) + p[n - 2];
    } else if (sgn(d - R - r) > 0) {//相离有 2 条交叉切线
        th = acos((R + r) / d);
        p[n++] = v.trunc(c1.r).rotate(th, P(0, 0)) + c1.o;
        p[n++] = v.trunc(c2.r).rotate(th, P(0, 0)) + c2.o;
        p[n++] = v.trunc(c1.r).rotate(2 * pi - th, P(0, 0)) + c1.o;
        p[n++] = v.trunc(c2.r).rotate(2 * pi - th, P(0, 0)) + c2.o;
    }
    return n;
}
```

半平面交

```
//HDU3982
//给出一个圆，以及圆上一点，再给出若干条切割线，问该点所在区域面积
//先半平面交求出该点所在凸多边形，再凸多边形与圆求交，复杂度 O(n^2 + n)
#include <iostream>
```

```cpp
#include <string>
#include <cmath>
#include <cassert>
#include <algorithm>
using namespace std;
const int MAXN = 2011;
const double eps = 1e-8;
const double inf = 1e10;
const double pi = acos(-1.0);

inline int sgn(double x) {
    return x < -eps ? -1 : x > eps;
}

struct P {
    double x, y;
    P(double x = 0, double y = 0):x(x), y(y) {}
    friend bool operator ==(const P& a, const P& b) {
        return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;
    }
    friend bool operator <(const P& a, const P& b) {
        return sgn(a.x - b.x) == 0 ? sgn(a.y - b.y) < 0 : sgn(a.x - b.x) < 0;
    }

    friend P operator -(const P& a, const P& b) {
        return P(a.x - b.x, a.y - b.y);
    }
    friend P operator +(const P& a, const P& b) {
        return P(a.x + b.x, a.y + b.y);
    }
    friend P operator *(const P& a, double s) {
        return P(a.x * s, a.y * s);
    }
    friend double operator *(const P& a, const P& b) {
        return a.x * b.y - a.y * b.x;
    }
    friend double dist(const P& a, const P& b) {
        return sqrt(fabs((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y)));
    }
    P turn_right() const {
        return P(y, -x);
    }
    double len() const {
        return sqrt(fabs(x * x + y * y));
    }
}p[MAXN], now;//, np[MAXN];
double cross(const P& a, const P& b, const P& c) {
    return (a.x - c.x) * (b.y - c.y) - (a.y - c.y) * (b.x - c.x);
}
double dot(const P& a, const P& b, const P& c) {
    return (a.x - c.x) * (b.x - c.x) + (a.y - c.y) * (b.y - c.y);
}
struct line {
    P a, b;
    line() {}
    line(P _a, P _b):a(_a), b(_b) {}
}l[MAXN];
struct circle {
    P o;
    double r;
    circle() {}
    circle(P o, double r = 0):o(o), r(r) {}
}c;
bool sameside(const P& p1, const P& p2, const P& l1, const P& l2) {
    return sgn(cross(l1, p1, l2) * cross(l1, p2, l2)) > 0;
}
```

```
P intersection(const P& u1, const P& u2, const P& v1, const P& v2) {
    P ret = u1;
    double t = ((u1.x - v1.x) * (v1.y - v2.y) - (u1.y - v1.y) * (v1.x - v2.x)) /
               ((u1.x - u2.x) * (v1.y - v2.y) - (u1.y - u2.y) * (v1.x - v2.x));
    ret.x += (u2.x - u1.x) * t;
    ret.y += (u2.y - u1.y) * t;
    return ret;
}
P pp[MAXN];
//点的个数 0~n-1,逆时针的点集 p, 插入的直线上的两个点 l1,l2,和判断是要哪一边的 side 点
void polycut(int &n, P p[], const P& l1, const P& l2, const P& side) {
    int m = 0, i;
    for (i = 0; i < n; i++) {
        if (sameside(p[i], side, l1, l2)) {
            pp[m++] = p[i];
        }
        if (!sameside(p[i], p[(i + 1) % n], l1, l2) &&
            !(sgn(cross(p[i], l1, l2)) == 0 && sgn(cross(p[(i + 1) % n], l1, l2)) == 0)) {
            pp[m++] = intersection(p[i], p[(i + 1) % n], l1, l2);
        }
    }
    for (n = i = 0; i < m; i++) {
        if (!i || sgn(pp[i].x - p[i - 1].x) != 0 || sgn(pp[i].y - pp[i - 1].y) != 0) {
            p[n++] = pp[i];
        }
    }
    if (sgn(p[n - 1].x - p[0].x) == 0 && sgn(p[n - 1].y - p[0].y) == 0) {
        n--;
    }
    if (n < 3) {
        n = 0;
    }
}
int main() {
    int n, m, T = 0;
    c.o.x = c.o.y = 0.0;
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%lf%d", &c.r, &n);
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf%lf", &l[i].a.x, &l[i].a.y, &l[i].b.x, &l[i].b.y);
        }
        scanf("%lf%lf", &now.x, &now.y);
        m = 0;
        l[n++] = line(P(2 * c.r, 0), P(2 * c.r, 2 * c.r));
        l[n++] = line(P(0, 2 * c.r), P(2 * c.r, 2 * c.r));
        l[n++] = line(P(-2 * c.r, 0), P(-2 * c.r, 2 * c.r));
        l[n++] = line(P(0, -2 * c.r), P(2 * c.r, -2 * c.r));
        int all = 4;
        p[0] = P(-inf, -inf);
        p[1] = P(inf, -inf);
        p[2] = P(inf, inf);
        p[3] = P(-inf, inf);
        for (int i = 0; i < n; i++) {
            polycut(all, p, l[i].a, l[i].b, now);
        }
        double total = 0;
        for (int i = 0; i < all; i++)
            total += calcarea(p[i], p[(i + 1) % all], c.o, c.r);
        printf("Case %d: %.5lf%%\n", ++T, fabs(total) / pi / c.r / c.r * 100);
    }
}
```

N 维最近点对

```
//HDU1007 ZJU2107 UVa10245 UVa10750 UVa11378(需要重定义距离)
//可以扩展到 N 维
//复杂度近似 O(n)
#include <iostream>
#include <cmath>
#include <ctime>
using namespace std;
typedef long long LL;
const double eps = 1e-8;
const int MAXN = 100002;
int n;
struct point {
    double x, y, nx;
    inline bool operator <(const point& d) const {
        return nx < d.nx;
    }
}p[MAXN];
inline double sqr(double x) {
    return x * x;
}
int main() {
    srand(time(NULL));
    while (scanf("%d", &n) != EOF && n) {
        double vx = rand() % 1000 + 1, vy = rand() % 1000 + 1;
        double v = 1.0 / (vx * vx + vy * vy);
        for (int i = 1; i <= n; i++) {
            scanf("%lf%lf", &p[i].x, &p[i].y);
            p[i].nx = p[i].x * vx + p[i].y * vy;
        }
        sort(p + 1, p + n + 1);
        double ans = 1LL << 50;
        for (int i = 1; i <= n; i++) {
            for (int j = i + 1; j <= n; j++) {
                double tmp = sqr(p[i].x - p[j].x) + sqr(p[i].y - p[j].y);
                if (tmp < ans) {
                    ans = tmp;
                }
                if (v * sqr(p[j].nx - p[i].nx) - ans > -eps) {
                    break;
                }
            }
        }
        printf("%.2lf\n", sqrt((double)ans) / 2);
    }
}
```

多边形费马点

```
//PKU2420 HDU3694
//模拟退火求费马点 复杂度不固定
//最好枚举每个顶点作为起点
#include <iostream>
#include <cmath>
using namespace std;
struct Point {
    double x, y;
    Point() {}
    Point(double x, double y): x(x), y(y) {}
};
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 allDis(Point *ps, int n, Point o) {
    double sum = 0;
    for (int i = 0; i < n; i ++)
        sum += dis(o, ps[i]);
    return sum;
}
double fermat(Point *ps, int n, int m) {
    int dx[8] = {1, -1, 0, 0, 1, 1, -1, -1};
    int dy[8] = {0, 0, 1, -1, 1, -1, 1, -1};
    Point o = ps[m];
    double res = allDis(ps, n, o);
    for (double step = 2048.0; step >= 1e-5; step *= 0.80) {
        for (int i = 0; i < 8; i ++) {
            Point no(o.x + step * dx[i], o.y + step * dy[i]);
            double tmp = allDis(ps, n, no);
            if (tmp < res) {
                res = tmp;
                o = no;
                i = -1;    ///    again!!
            }
        }
    }
    return res;
}
Point ps[110];
int n;
int main() {
    Point o, no;
    int i;
    while (1) {
        int n = 4;
        bool end = 1;
        for (i = 0; i < n; i ++) {
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
            if (ps[i].x != -1 || ps[i].y != -1) {
                end = 0;
            }
        }
        if (end) break;
        double ans = 1e30;
        for (int i = 0; i < n; i++) {
            ans = min(ans, fermat(ps, n, i));
        }
        printf("%.4f\n", ans);
    }
    return 0;
}
```

面积最大三角形

```
    //先求凸包，再旋转卡壳
     double maxtrianglearea(P ch[],int n) {
        if (n < 3) return 0;
        double S = 0,area,prev;
        ch[n] = ch[0];
        for (int i = 0;i < n;i++) {
            int j = i+1,k = j+1;
            if (k >= n) break;
            prev = 0;
            while (j < n-1) {
                area = cross(ch[i],ch[j],ch[k]);
                while (k < n-1) {
```

```
                if (sgn(cross(ch[i],ch[j],ch[k+1]) - area) > 0) {
                    area = cross(ch[i],ch[j],ch[k+1]);
                    k++;
                }
                else break;
            }
            if (area < prev) break;
            prev = area;
            S = max(S,area);
            j++;
            if (j >= k) {
                k = j+1;
                if (k >= n) break;
            }
        }
    }
    return S / 2;
}
double maxtrianglearea1(P ch[],int n) {
    if (n < 3) return 0;
    double S = 0;
    for (int i = 0;i < n;i++) {
        int j = (i+1) % n;
        int k = (j+1) % n;
        while (sgn(cross(ch[i],ch[j],ch[(k+1)%n]) - cross(ch[i],ch[j],ch[k])) > 0) //旋转 k
            k = (k+1) % n;
        //int t = (k+1) % n;
        while (j != i) {//j != k,j != t
            S = max(S,cross(ch[i],ch[j],ch[k]));
            while (sgn(cross(ch[i],ch[j],ch[k]) - cross(ch[i],ch[j],ch[(k+1)%n])) < 0) //旋转 j 和
k
                k = (k+1) % n;
            j = (j+1) % n;
        }
    }
    return S / 2;
}
double maxtrianglearea2(P ch[],int n) {
    if (n < 3) return 0;
    double S = 0;
    ch[n] = ch[0];
    for (int i = 0;i < n;i++) {
        int j = (i+1) % n;
        int k = (j+1) % n;
        while (j != i) {
            S = max(S,cross(ch[i],ch[j],ch[k]));
            while (sgn(cross(ch[i],ch[j],ch[k+1]) - cross(ch[i],ch[j],ch[k])) > 0)
                k = (k+1) % n;
            j = (j+1) % n;
        }
    }
    return S / 2;
}
```

周长最短三角形

```
//周长最小三角形 分治算法 复杂度 O(NlogN)
//GCJ2009 Final Problem B HDU3868
//设当前集合的答案为 r，对于新加入的点，如果这个点能更新答案，那么它与另外的两个点的距离一定不超过 r/2
//假设当前新加入点为 C，且 C 与之前集合中的点 A 和 B 构成了新的最近的三个点，设|AC|>= ans / 2，
//由于三角形两边之和大于第三边，必有|AB| + |BC| > |AC|，即 |AB|+ |BC| + |AC| > 2|AC| > ans，假设不成立
#include <iostream>
#include <cmath>
using namespace std;
```

```
typedef pair<double, double> PDD;
const int MAXN = 20000 + 5;
int n;
PDD P[MAXN];
double R;
double dist(const PDD& a, const PDD& b) {
    return hypot(a.first - b.first, a.second - b.second);
}
void checkIt(int i, int j, int k) {
    double t = dist(P[i], P[j]) + dist(P[j], P[k]) + dist(P[k], P[i]);
    if (t < R) R = t;
}
void solve(int H, int T) {
    if (T - H + 1 <= 5) {
        for (int i = H; i <= T; i++)
            for (int j = i + 1; j <= T; j++)
                for (int k = j + 1; k <= T; k++)
                    checkIt(i, j, k);
        return;
    }
    int M = (H + T) / 2;
    solve(H, M);
    solve(M + 1, T);
    double sp = (P[M].first + P[M + 1].first) / 2.0;
    vector<pair<double, int> > QL, QR;
    for (int i = H; i <= T; i++) {
        if (fabs(P[i].first - sp) <= R / 2.0) {
            if (i <= M) {
                QL.push_back(make_pair(P[i].second, i));
            } else {
                QR.push_back(make_pair(P[i].second, i));
            }
        }
    }
    sort(QL.begin(), QL.end());
    sort(QR.begin(), QR.end());
    for (int step = 0; step < 2; step++) {
        for (int i = 0, k = 0; i < QL.size(); i++) {
            for (; k < QR.size() && QR[k].first < QL[i].first - R / 2.0; k++);
            int d = k;
            for (; d < QR.size() && QR[d].first <= QL[i].first + R / 2.0; d++);
            for (int a = k; a < d; a++)
                for (int b = a + 1; b < d; b++)
                    checkIt(QL[i].second, QR[a].second, QR[b].second);
        }
        if (step == 0) {
            swap(QL, QR);
        }
    }
}
int main() {
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%d", &n);
        for (int i = 0; i < n; i++) {
            double x, y;
            scanf("%lf%lf", &x, &y);
            P[i] = make_pair(x, y);
        }
        sort(P, P + n);
        R = 1e100;
        solve(0, n - 1);
        printf("%.3lf\n", R);
    }
}
```

```
//http://blog.renren.com/share/306601516/4075309816
#include <iostream>
const int MAXN = 300;
using namespace std;
const double eps = 1e-8;
const double pi = acos(-1.0);
#define sqr(x) ((x) * (x))
double R;//定长
struct point {
    double x, y;
    void read() {
        scanf("%lf%lf", &x, &y);
    }
    void print() {
        printf("%lf%lf\n", x, y);
    }
    double friend dis(const point &a, const point &b) {
        return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
    }
} p[MAXN + 5];
struct alpha {
    double v;
    bool flag;
    bool friend operator <(const alpha &a, const alpha &b) { //排序专用偏序关系
        return a.v < b.v;
    }
} alp[MAXN * MAXN + 5];//C(N,2)*2 的可能交点（可能极角）
void solve(int n) {
    R = 1.0;//本题内为单位圆
    for (int i = 0; i < n; i++) p[i].read();
    int MAX = 0;
    for (int i = 0; i < n; i++) {
        int t = 0;
        for (int j = 0; j < n; j++) {
            if (i == j) continue;
            double theta, phi, D;
            D = dis(p[i], p[j]);
            if (D > 2.0 * R) continue; //距离超界直接秒杀
            //关键部分
            theta = atan2(p[j].y - p[i].y, p[j].x - p[i].x);
            if (theta < 0)
                theta += 2 * pi;
            phi = acos(D / (2.0 * R));
            alp[t].v = theta - phi + 2 * pi;
            alp[t].flag = true;
            alp[t + 1].v = theta + phi + 2 * pi;
            alp[t + 1].flag = false;
            t += 2;
        }
        sort(alp, alp + t);
        int sum = 0;
        for (int j = 0; j < t; j++) {
            if (alp[j].flag)
                sum ++;
            else
                sum --;
            if (sum > MAX)
                MAX = sum;
        }
    }
    printf("%d\n", MAX + 1);
}
int main() {
```

```
    int n;
    while (scanf("%d", &n) != EOF && n) {
        solve(n);
    }
}
```

最小圆覆盖[随机增量法]

```
//最小圆覆盖
//包含点集所有点的最小圆的算法，复杂度O(n)
//ZJU1450 HDU3007 HDU3932
//mincircle(p, n, 0, Tri, ans);
#include <iostream>
#include <cmath>
using namespace std;
const double eps = 1e-8;
const double pi = acos(-1.0);
const double inf = 1e200;
inline int sgn(double x) {
    return x < -eps ? -1 : x > eps ? 1 : 0;
}
inline double sqr(double x) {
    return x * x;
}
struct P {
    double x, y;
}
//以下圆相关
struct circle {
    P o;//圆心
    double r;
    circle(): r(0) {}
    circle(P o, double r = 0): o(o), r(r) {}
};
circle circumcircle(const P& a, const P& b, const P& c) { //三角形的外接圆
    circle C;
    double ab = dist(a, b), bc = dist(b, c), ac = dist(a, c),
           c1 = (sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y)) / 2,
           c2 = (sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y)) / 2;
    C.r = ab * bc * ac / (2 * fabs(cross(a, b, c)));
    C.o.x = (c1 * (a.y - c.y) - c2 * (a.y - b.y)) / ((a - b) * (a - c));
    C.o.y = (c1 * (a.x - c.x) - c2 * (a.x - b.x)) / ((a - c) * (a - b));
    return C;
}//返回外接圆
struct triangle {
    P t[3];
} Tri;
circle mincircle2(int m, triangle ce) {
    circle tmp;
    if (m == 0) tmp.r = -2;//bug;
    else if (m == 1) {
        tmp.o = ce.t[0];
        tmp.r = 0;
    } else if (m == 2) {
        tmp.o = (ce.t[0] + ce.t[1]) / 2;
        tmp.r = dist(ce.t[0], ce.t[1]) / 2;
    } else if (m == 3)
        tmp = circumcircle(ce.t[0], ce.t[1], ce.t[2]);
    return tmp;
}
void mincircle(P p[], int n, int m, triangle ce, circle& c) {
    P tmp;
    c = mincircle2(m, ce);
    if (m == 3) return;
```

```
    for (int i = 0; i < n; i++) {
        if (dist(p[i], c.o) > c.r) {
            ce.t[m] = p[i];
            mincircle(p, i, m + 1, ce, c);
            tmp = p[i];
            for (int j = i; j > 0; j--)
                p[j] = p[j - 1];
            p[0] = tmp;
        }
    }
}
P p[501];
int main() {
    int n;
    while (scanf("%d", &n) != EOF && n) {
        for (int i = 0; i < n; i++)
            p[i].in();
        circle ans;
        mincircle(p, n, 0, Tri, ans);
        printf("%.2lf %.2lf %.2lf\n", ans.o.x, ans.o.y, ans.r);
    }
}

//随机增量另一个版本
circle circum(const P& a, const P& b) {
    circle c;
    c.o.x = (a.x + b.x) / 2;
    c.o.y = (a.y + b.y) / 2;
    c.r = dist(a, b) / 2;
    return c;
}
circle circum(const P& a, const P& b, const P& c) { //三角形的外接圆
    circle C;
    double ab = dist(a,b),bc = dist(b,c),ac = dist(a,c),
          c1 = (sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y)) / 2,
          c2 = (sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y)) / 2;
    C.r = ab * bc * ac / (2 * fabs(cross(a, b, c)));
    C.o.x = (c1 * (a.y - c.y) - c2 * (a.y - b.y)) / ((a - b) * (a - c));
    C.o.y = (c1 * (a.x - c.x) - c2 * (a.x - b.x)) / ((a - c) * (a - b));
    return C;
}//返回外接圆
bool out(const P& p, const circle& c) {
    return sgn(dist(p, c.o) - c.r) > 0;
}
int main() {
    int n;
    while (scanf("%d", &n) != EOF && n) {
        for (int i = 0; i < n; i++) {
            p[i].input();
        }
        random_shuffle(p, p + n);
        //n >= 3
        ans.o = p[0];
        ans.r = 0;
        if (n >= 2) ns = circum(p[0], p[1]);
        for (int i = 2; i < n; i++) {
            if (out(p[i], ans)) {//若第 i 个点在圆外
                ans = circum(p[0], p[i]);
                for (int j = 1; j < i; j++) {
                    if (out(p[j], ans)) {//若 j 不在 i 构成的圆内，则 i,j 确定一个圆
                        ans = circum(p[i], p[j]);
                        for (int k = 0; k < j; k++) {
                            if (out(p[k], ans)) {
                                ans = circum(p[i], p[j], p[k]);
                            }
                        }
```

```
                }
            }
        }
    }
    printf("%.10lf\n%.10lf %.10lf\n", ans.r, ans.o.x, ans.o.y);
  }
}
```

最小球覆盖[随机增量法]

```
//最小圆覆盖
//包含点集所有点的最小球的算法，复杂度O(n)
//UVa10095 PKU2069 CodeForces106E
#include <iostream>
#include <cstdio>
#include <sstream>
#include <vector>
#include <algorithm>
#include <valarray>
#include <stack>
#include <list>
using namespace std;
const int MAXN = 10086;
const double EPS = 1e-7;
struct P {
    static const int D = 3 ;
    double x, y, z;
    P(double _x = 0.0 , double _y = 0.0, double _z = 0.0):x(_x), y(_y), z(_z) {
    }
    friend double dot(const P& p1 , const P& p2) {
        return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
    }
    friend P det(const P& p1 , const P& p2) {
        return P(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x);
    }
    double norm() const {
        return dot(*this, *this) ;
    }
    double abs() const {
        return sqrt(norm());
    }
};
struct sphere {
    P pos;
    double r ;
    sphere(P pos = P(0, 0, 0) , double r = 0.0) : pos(pos) , r(r) {}
    bool isIn(P q) {
        if (P(pos - q).abs() < r + EPS) return true;
        return false;
    }
};
sphere b_md(vector<P>& R) {
    int n = (int)R.size();
    if (n == 1) return sphere(R[0], 0.0);
    if (n > 1) {
        static P v[4], c[4];
        static double r[4];
        r[0] = 0.0;
        c[0] = R[0];
        for (int i = n - 1; i < n; i++) { // 1..n-2 所有结果都相同
            v[i] = P(R[i] - c[0]);
            for (int j = 1; j < i; j++) {
                v[i] = v[i] - v[j] * dot(P(v[j]), P(R[i] - c[0])) ;
```

```
            }
            double d = P(R[i] - c[i - 1]).abs();
            double e = (d - r[i - 1] * r[i - 1] / d) / 2.0 ;
            c[i] = P(c[i - 1] + v[i] / v[i].norm() * e * d);
            r[i] = P(c[i] - R[0]).abs();
            v[i] = v[i] / v[i].abs();
        }
        return sphere(c[n - 1], r[n - 1]);
    }
    return sphere(P(0, 0, 0), -1);
}

/**
 * @see http://citeseer.ist.psu.edu/welzl91smallest.html
 * @see http://www.inf.ethz.ch/personal/gaertner/texts/own_work/esa99_final.pdf
 * 因为前者栈溢出所以采用后者(Page 3)
 */
sphere b_miniball(P p[], int pos, vector<P>& R) {
    sphere D = b_md(R);
    if ((int)R.size() == 4) return D;
    for (int k = 0; k < pos; k++) {
        P *now = &p[k];
        if (!D.isIn(*now)) {
            R.push_back(*now) ;
            D = b_miniball(p, k, R) ;
            R.pop_back();
            P t = p[k];
            for (int j = k; j > 0; j--) {
                p[j] = p[j - 1];
            }
            p[0] = t;
        }
    }
    return D ;
}
sphere miniball(P p[], int n) {
    vector<P> R;
    return b_miniball(p, n, R);
}
P p[MAXN];
int main() {
    int n ;
    while (cin >> n && n) {
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
        }
        sphere ans = miniball(p, n);
        printf("%.4f %.4f %.4f %.4f\n", ans.r, ans.pos.x , ans.pos.y , ans.pos.z) ;
    }
    return 0;
}
```

最小球覆盖[三分法]

```
//最小球覆盖
//包含点集所有点的最小球的算法，试用于100个点左右，精度较够
//CodeForces106E
#include <iostream>
#include <cmath>
using namespace std;
const int MAXN = 10086;
const double EPS = 1e-7;
inline int sgn(double x) {
    return x < -EPS ? -1 : x > EPS;
```

```
}
struct P {
    double x, y, z;
    friend double dist(const P& a, const P& b) {
        return ((a.x - b.x) * (a.x - b.x) + (a.y - b.y) *  (a.y - b.y) + (a.z - b.z) * (a.z - b.z));
    }
}p[MAXN], re;
int n;
double ans;
double mi[3], ma[3];
double calc(double x, double y, double z) {
    P t;
    double res = 0;
    t.x = x, t.y = y, t.z = z;
    for (int i = 0; i < n; i++) {
        res = max(res, dist(t, p[i]));
    }
    return res;
}
double getZ(double x, double y, double z) {
    double l = mi[2], r = ma[2], mid, midmid, midv, midmidv;
    while (sgn(r - l) > 0) {
        mid = (l + r) / 2;
        midmid = (mid + r) / 2;
        midv = calc(x, y, mid);
        midmidv = calc(x, y, midmid);
        // 假设求解最小极值.
        if (midv <= midmidv) {
            r = midmid;
        } else {
            l = mid;
        }
    }
    double dis = calc(x, y, l);
    if (sgn(dis - ans) < 0) {
        ans = dis;
        re.x = x, re.y = y, re.z = l;
    }
    return dis;
}
double getY(double x, double y, double z) {
    double l = mi[1], r = ma[1], mid, midmid, midv, midmidv;
    while (sgn(r - l) > 0) {
        mid = (l + r) / 2;
        midmid = (mid + r) / 2;
        midv = getZ(x, mid, z);
        midmidv = getZ(x, midmid, z);
        // 假设求解最小极值.
        if (midv <= midmidv) {
            r = midmid;
        } else {
            l = mid;
        }
    }
    return getZ(x, l, z);
}
double getX(double x, double y, double z) { //三分 CD
    double l = mi[0], r = ma[0], mid, midmid, midv, midmidv;
    while (sgn(r - l) > 0) {
        mid = (l + r) / 2;
        midmid = (mid + r) / 2;
        midv = getY(mid, y, z);
        midmidv = getY(midmid, y, z);
        // 假设求解最小极值.
        if (midv <= midmidv) {
            r = midmid;
```

```
        } else {
            l = mid;
        }
    }
    return getY(l, y, z);
}
int main() {
    while (scanf("%d", &n) != EOF && n) {
        fill(mi, mi + 3, 1e20);
        fill(ma, ma + 3, -1e20);
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
            mi[0] = min(mi[0], p[i].x), ma[0] = max(ma[0], p[i].x);
            mi[1] = min(mi[1], p[i].y), ma[1] = max(ma[1], p[i].y);
            mi[2] = min(mi[2], p[i].z), ma[2] = max(ma[2], p[i].z);
        }
        ans = 1e20;
        getX(0, 0, 0);
        printf("%.6lf %.6lf %.6lf\n", re.x + EPS, re.y + EPS, re.z + EPS);
    }
}
```

点集最小外接矩形

```
//点击最小外接矩形 旋转卡壳 复杂度O(n)
//UVa10173
//最小外接矩形的四条边上，其中一条边有至少两个点，其他边上至少有一个点。
//然后沿着凸包的边旋转，维护矩形另外三条边上的点
#include <cstdio>
#include <cmath>
using namespace std;
const double eps = 1e-8;
struct P {
    double x, y;
};
int sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
double cross(P p0, P p1, P p2) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}
double dot(P p0, P p1, P p2) {
    return (p1.x - p0.x) * (p2.x - p0.x) + (p1.y - p0.y) * (p2.y - p0.y);
}
double sqr(double x) {return x * x;}
double dissqr(P a, P b) {
    return sqr(a.x - b.x) + sqr(a.y - b.y);
}
bool PointEqual(P a, P b) {
    return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;
}
P bp;
int cmp(const P &p1, const P &p2) {
    int u = sgn(cross(bp, p1, p2));
    return u > 0 || (u == 0 && sgn(dissqr(bp, p1) - dissqr(bp, p2)) < 0);
}
void graham(P pin[], int n, P ch[], int &m) {
    int i, j, k, u, v;
    memcpy(ch, pin, n * sizeof(P));
    for (i = k = 0; i < n; ++i) {
        u = sgn(ch[i].x - ch[k].x);
        v = sgn(ch[i].y - ch[k].y);
        if (v < 0 || (v == 0 && u < 0)) k = i;
    }
```

```
    bp = ch[k];
    sort(ch, ch + n, cmp);
    n = unique(ch, ch + n, PointEqual) - ch;
    if (n <= 1) {m = n; return ;}
    if (sgn(cross(ch[0], ch[1], ch[n - 1])) == 0) {
        m = 2; ch[1] = ch[n - 1]; return;
    }
    ch[n++] = ch[0];
    for (i = 1, j = 2; j < n; ++j) {
        while (i > 0 && sgn(cross(ch[i - 1], ch[i], ch[j])) <= 0) i--;
        ch[++i] = ch[j];
    }
    m = i;
}
const int N = 1010;
int n;
P cp[N], ch[N];
void solve() {
    for (int i = 0; i < n; ++i) scanf("%lf%lf", &cp[i].x, &cp[i].y);
    int m, q, p, r;
    graham(cp, n, ch, m);
    if (m < 3) {
        puts("0.0000");
        return;
    }
    double res = 1e99;
    ch[m] = ch[0];
    p = q = 1;
    for (int i = 0; i < m; ++i) {
        while (sgn(cross(ch[i], ch[i + 1], ch[p + 1])
            - cross(ch[i], ch[i + 1], ch[p])) > 0 )
            p = (p + 1) % m;
        while (sgn(dot(ch[i], ch[i + 1], ch[q + 1])
            - dot(ch[i], ch[i + 1], ch[q])) > 0)
            q = (q + 1) % m;
        if (i == 0) r = q;
        while (sgn(dot(ch[i], ch[i + 1], ch[r + 1])
            - dot(ch[i], ch[i + 1], ch[r])) <= 0)
            r = (r + 1) % m;
        double d = dissqr(ch[i], ch[i + 1]);
        res = min(res, cross(ch[i], ch[i + 1], ch[p])
                    * (dot(ch[i], ch[i + 1], ch[q]) - dot(ch[i], ch[i + 1], ch[r])) / d);
    }
    printf("%.4lf\n", res);
}
int main() {
    while (scanf("%d", &n), n) solve();
    return 0;
}
```

扫描线判断某点可见线段数

```
//给定一个点和一堆不重合的线段，问在该点有几条线段可见
//利用线段大小关系不变性质，用扫描线排序，利用 set 维护复杂度 O(nlogn)
//默认同一极角上不能有多个线段端点
//HDU3867
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#include <cmath>
#include <set>
#include <algorithm>
using namespace std;
```

```
const int MAXN = 40086;
const double EPS = 1e-8;
const double PI = acos(-1.0);
inline int sgn(double x) {
    return x < -EPS ? -1 : x > EPS;
}
struct P {
    double x, y, ag;
    P(double _x = 0, double _y = 0):x(_x), y(_y) {}
    inline double len() const {
        return x * x + y * y;
    }
    friend P operator -(const P& a, const P& b) {
        return P(a.x - b.x, a.y - b.y);
    }
    friend double operator *(const P& a, const P& b) {
        return a.x * b.y - a.y * b.x;
    }
    double norm() const {
        return x * x + y * y;
    }
}p[MAXN], o;
void intersection(P a, P b, P c, P d, P& p) { //直线 ab 与直线 cd 相交，假设都有交点
    double c1 = (b - a) * (d - c), c2 = (c - a) * (d - c);
    p.x = a.x + (b.x - a.x) * (c2 / c1);
    p.y = a.y + (b.y - a.y) * (c2 / c1);
}
struct seg {
    P a, b;
}s[MAXN];
double getx(const P& p1, const P& p2) {
    return (p2.y * p1.x - p2.x * p1.y) / (p2.y - p1.y);
}
P line;
int idx;
struct sweep {
    P *s, *t;
    int type;//1 是起点，-1 终点
    int id;
    sweep(P *_s = NULL, P *_t = NULL, int _type = 0, int _id = 0):s(_s), t(_t), type(_type), id(_id)
{
    }
    bool operator <(const sweep& d) const {
        P p1, p2;
        intersection(*s, *t, P(0, 0), line, p1);
        intersection(*(d.s), *(d.t), P(0, 0), line, p2);
        return p1.x * p1.x + p1.y * p1.y < p2.x * p2.x + p2.y * p2.y;
    }
}sl[MAXN];
bool cmp(const sweep& a, const sweep& b) {
    return sgn(a.s->ag - b.s->ag) == 0 ? a.s->norm() < b.s->norm() : a.s->ag < b.s->ag;
}
bool ok[MAXN], is[MAXN];
set<sweep> st;
int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        scanf("%lf%lf", &o.x, &o.y);
        int m = 0;
        idx = 0;
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf%lf", &s[i].a.x, &s[i].a.y, &s[i].b.x, &s[i].b.y);
            s[i].a.x -= o.x, s[i].a.y -= o.y;
            s[i].b.x -= o.x, s[i].b.y -= o.y;
            s[i].a.ag = atan2(s[i].a.y, s[i].a.x);
            s[i].b.ag = atan2(s[i].b.y, s[i].b.x);
```

```
            is[i] = 0;
            if (sgn(s[i].a.ag - s[i].b.ag) > 0) swap(s[i].a, s[i].b);
            if (sgn(s[i].b.ag - s[i].a.ag - PI) > 0) {
                P t = P(getx(s[i].a, s[i].b), 0);
                p[m] = t; p[m++].ag = -PI;
                p[m++] = s[i].a;
                p[m++] = s[i].b;
                p[m] = t; p[m++].ag = PI;
                sl[idx++] = sweep(&p[m - 4], &p[m - 3], 1, i);
                sl[idx++] = sweep(&p[m - 3], &p[m - 4], -1, i);
                sl[idx++] = sweep(&p[m - 2], &p[m - 1], 1, i);
                sl[idx++] = sweep(&p[m - 1], &p[m - 2], -1, i);
            } else {
                p[m++] = s[i].a;
                p[m++] = s[i].b;
                sl[idx++] = sweep(&p[m - 2], &p[m - 1], 1, i);
                sl[idx++] = sweep(&p[m - 1], &p[m - 2], -1, i);
            }
        }
        sort(sl, sl + idx, cmp);
        st.clear();
        for (int i = 0; i < idx; i++) {
            line = *sl[i].s;
            if (sgn(sl[i].s->ag - PI) == 0) {
                is[st.begin()->id] = 1;
                break;
            }
            if (sl[i].type == 1) {
                st.insert(sl[i]);
            } else {
                st.erase(sweep(sl[i].t, sl[i].s, 1, sl[i].id));
            }
            if (!st.empty()) {
                is[st.begin()->id] = 1;
            }
        }
        printf("%d\n", count(is, is + n, 1));
    }
}
```

扫描线最近圆对

```
//最近圆对 扫描线算法 复杂度O(NlogN)
//二分半径增量，扫描线判断是否有交
//HDU3124 ZJU3521
#include <cstdio>
#include <cmath>
#include <set>
#define sqr(x) ((x)*(x))
using namespace std;
const int N = 50010;
const double eps = 1e-8;
int n, lft[N], rht[N];
set<int> s;
struct circle {
    double x, y, r;
}cir[N];
int sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
double dis(circle a, circle b) {
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)) - a.r - b.r;
}
bool cmp(circle a, circle b) {
```

```
    return a.y < b.y;
}
bool cmp2(int i, int j) {
    return cir[i].x - cir[i].r < cir[j].x - cir[j].r;
}
bool cmp3(int i, int j) {
    return cir[i].x + cir[i].r < cir[j].x + cir[j].r;
}
bool inter(circle a, circle b, double delta) {
    return sgn(dis(a, b) - delta * 2) < 0;
}
bool insert(int i, double mid) {
    set<int>::iterator it = s.insert(lft[i]).first;
    if (it != s.begin()) {
        if (inter(cir[*--it], cir[lft[i]], mid))
            return false;
        ++it;
    }
    if (++it != s.end() && inter(cir[*it], cir[lft[i]], mid))
        return false;
    return true;
}
bool remove(int j, double mid) {
    set<int>::iterator it = s.find(rht[j]);
    if (it != s.begin() && it != --s.end()) {
        int a = *--it;  ++it;
        int b = *++it;
        if (inter(cir[a], cir[b], mid))
            return false;
    }
    s.erase(rht[j]);
    return true;
}
bool check(double mid) {
    s.clear();
    for (int i = 0, j = 0; i < n || j < n;) {
        if (j == n) {
            if (!insert(i++, mid)) return false;
        } else if (i == n) {
            if (!remove(j++, mid)) return false;
        } else if (sgn(cir[lft[i]].x - cir[lft[i]].r - mid
            - cir[rht[j]].x - cir[rht[j]].r - mid) <= 0) {
                if (!insert(i++, mid)) return false;
        } else {
            if (!remove(j++, mid)) return false;
        }
    }
    return true;
}
void solve() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%lf%lf%lf", &cir[i].x, &cir[i].y, &cir[i].r);
        lft[i] = rht[i] = i;
    }
    sort(cir, cir + n, cmp);
    sort(lft, lft + n, cmp2);
    sort(rht, rht + n, cmp3);
    double mid, low = 0, high = dis(cir[0], cir[1]) / 2;
    while (sgn(high - low)) {
        mid = (high + low) / 2;
        if (check(mid)) low = mid;
        else high = mid;
    }
    printf("%lf\n", low + high);
}
```

```
int main() {
    int cas;
    scanf("%d", &cas);
    while (cas--) solve();
    return 0;
}
```

直线切割凸多边形

```
//SGU345 直线切割凸多边形
//N=50000 个点，M=50000 个询问，每次给出一条直线，复杂度 O(MlogN);
//可以有三点共线，代码里有去重
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
const double eps = 1e-8;
inline int sgn(double x) {
    return x < -eps ? -1 : x > eps;
}
struct P {
    double x, y;
    P(double _x = 0, double _y = 0): x(_x), y(_y) {
    }
    void input() {
        scanf("%lf%lf", &x, &y);
    }
};
P operator +(const P& p1, const P& p2) {
    return P(p1.x + p2.x, p1.y + p2.y);
}
P operator -(const P& p1, const P& p2) {
    return P(p1.x - p2.x, p1.y - p2.y);
}
double operator ^(const P& p1, const P& p2) {
    return p1.x * p2.x + p1.y * p2.y;
}
double operator *(const P& p1, const P& p2) {
    return p1.x * p2.y - p1.y * p2.x;
}
struct edge {
    int id;
    P v;
    edge() {}
    edge(int _id, const P& _v): id(_id), v(_v) {}
};
bool operator <(const edge& e1, const edge& e2) {
    return sgn(atan2(e1.v.y, e1.v.x) - atan2(e2.v.y, e2.v.x)) < 0;
}
const int MAXN = 50000 + 10;
int n, m;
P pol[MAXN * 2], buf[MAXN], l1, l2;
edge e[MAXN];
double area_sum[MAXN * 2];
void solve();
void pre_compute();
double get_min_area(const P&, const P&);
bool is_less(const P&, const P&);
P get_intersection(const P&, const P&, const P&, const P&);
double get_area(int, int);
int main() {
    solve();
    return 0;
}
```

```
void solve() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        buf[i].input();
    pre_compute();
    scanf("%d", &m);
    for (int i = 0; i < m; ++i) {
        P p1, p2;
        p1.input();
        p2.input();
        printf("%.8lf\n", get_min_area(p1, p2));
    }
}
void pre_compute() {
    double area = 0;
    for (int i = 0; i < n; ++i)
        area += buf[i] * buf[(i + 1) % n];
    if (sgn(area) < 0)
        reverse(buf, buf + n);
    int cnt = 0;
    for (int i = 0; i < n; ++i)
        if (sgn((buf[i] - buf[(i + n - 1) % n]) * (buf[(i + 1) % n] - buf[i]) != 0))
            pol[cnt++] = buf[i];
    n = cnt;
    for (int i = 0; i < n; ++i) {
        e[i] = edge(i, pol[(i + 1) % n] - pol[i]);
        pol[n + i] = pol[i];
    }
    sort(e, e + n);
    area_sum[0] = pol[n - 1] * pol[0];
    for (int i = 1; i < n * 2; ++i)
        area_sum[i] = area_sum[i - 1] + pol[i - 1] * pol[i];
}
double get_min_area(const P& p1, const P& p2) {
    int p_l = e[(lower_bound(e, e + n, edge(-1, p1 - p2)) - e) % n].id;
    int p_r = e[(lower_bound(e, e + n, edge(-1, p2 - p1)) - e) % n].id;
    if (sgn((p2 - p1) * (pol[p_l] - p1)) * sgn((p2 - p1) * (pol[p_r] - p1)) >= 0)
        return 0;
    l1 = p2, l2 = p1;
    int k1 = (lower_bound(pol + p_l, pol + (p_r < p_l ? p_r + n : p_r) + 1, p1, is_less) - pol) % n;
    l1 = p1, l2 = p2;
    int k2 = (lower_bound(pol + p_r, pol + (p_l < p_r ? p_l + n : p_l) + 1, p2, is_less) - pol) % n;
    P c1 = get_intersection(p1, p2, pol[k1], pol[(k1 + n - 1) % n]);
    P c2 = get_intersection(p1, p2, pol[k2], pol[(k2 + n - 1) % n]);
    double area1 = c1 * pol[k1] + get_area(k1, (k2 + n - 1) % n) + pol[(k2 + n - 1) % n] * c2 + c2
* c1;
    double area2 = c2 * pol[k2] + get_area(k2, (k1 + n - 1) % n) + pol[(k1 + n - 1) % n] * c1 + c1
* c2;
    return min(abs(area1 / 2.0), abs(area2 / 2.0));
}
bool is_less(const P& p1, const P& p2) {
    return sgn((l1 - p1) * (l2 - p1) - (l1 - p2) * (l2 - p2)) < 0;
}
P get_intersection(const P& p1, const P& p2, const P& p3, const P& p4) {
    double d1 = (p2 - p1) * (p3 - p1), d2 = (p2 - p1) * (p4 - p1);
    return P((p3.x * d2 - p4.x * d1) / (d2 - d1), (p3.y * d2 - p4.y * d1) / (d2 - d1));
}
double get_area(int k1, int k2) {
    if (k1 == k2)
        return 0;
    if (k2 < k1)
        return area_sum[n - 1] - (area_sum[k1] - area_sum[k2]);
    return area_sum[k2] - area_sum[k1];
}
```

PICK 定理

```
/*
Pick 定理
给定顶点坐标均是整点（或正方形格点）的简单多边形
皮克定理说明了其面积 A 和内部格点数目 i、边上格点数目 b 的关系：A = i + b/2 - 1.
先求出边界上的点，然后求出面积，根据 pick 公式，求出内部点的个数.
以格子点为顶点的线段，覆盖的点的个数为 GCD(dx,dy)
其中 dxdy 分别为线段横向占的点数和纵向占的点数 如果 dx 或 dy 为 0 则覆盖的点数为 dy 或 dx
*/
```

欧拉公式

```
/*
简单多面体的顶点数 V、面数 F 及棱数 E 间有关系 V+F-E=2
这个公式叫欧拉公式。公式描述了简单多面体顶点数、面数、棱数特有的规律.
平面图欧拉公式：V+F-E=1
给定平面上的 N 个圆，问整个平面被分成了几个部分。

如果用圆离散化来解就太大自然了。这里可以用传说中的欧拉公式(Euler's formula)，χ=V-E+F 来计算，V-vertices 表示顶点数，E-edges 表示边数，F-faces 表示围成的面数。于是面的个数可以这样计算：对于几个相连的圆，其内有 F=E-V+1 个面，其中 1 是平面图情况的欧拉欧拉示性数(Euler characteristic)；对于一个独立的圆，显然 F = 1；最后答案还要加上外部那个无穷大的面。通过求圆与圆的交点可以很容易的得到顶点数，边数和圆与圆是否相连，从而求得答案。
*/
```

六：动态规划和贪心

斜率优化 DP

```
#define maxn 30005
typedef long long LL;
struct Point{
    int id; LL x,y;
    Point(int _id=0,LL _x=0,LL _y=0):id(_id),x(_x),y(_y){};
    friend double slope(const Point &a,const Point &b){
        if( a.x == b.x ){
            if( b.y > a.y ) return 9999999999999.0;
            else return -999999999999.0;
        }
        return (double)(a.y-b.y) / (a.x-b.x);
    }
};
struct ConvexQueue{ //维护一个下凸队列
    Point que[maxn]; int s,t;
    void clear(){ s=t=0; }
    void push(Point p){
        while( t>=s+2 ){
            if( slope(que[t-2],que[t-1]) > slope(que[t-1],p) ) //这里决定下凸
                t--;
            else break;
        }
        que[t++]=p;
    }
    Point getFirst(){
        if( s==t ) return Point(0,0,(LL)1 <<60 );//这个要保证不给力，看题目定数字
        return que[s];
```

```
    }
    void pop(double _slope){ //斜率不够的弹出
        while(t>=s+2){
            if( slope(que[s],que[s+1]) < _slope ) s++;
            else break;
        }
    }
    void pop(int ID){ //去掉下标小于 ID 的
        while( s<t ){
            if( que[s].id < ID ) s++;
            else break;
        }
    }
}cq;
```

四边形不等式

```
/*
状态转移方程形如：
dp[i][j]=min{ dp[i][k]+dp[k+1][j]+w[i][j] }
然后对于 k1 < k2 < k3 < k4 有：
w[k2][k3] <= w[k1][k4]                          //区间包含单调性
w[k1][k3] + w[k2][k4] <= w([k2][k3] + w[k1][k4]  //四边形不等式
*/
//hdu 3516 Tree construction 四边形不等式
/*
dp[i][j]=min{ dp[i][k]+dp[k+1][j] + x[k+1]-x[i] + y[k]-y[j] } (i<=k<j )  ***注意变量范围
令 dp[i][j]+x[i]+y[j] = m[i][j]
则 m[i][j]=min{ m[i][k]+m[k+1][j] - (x[i]+y[j]) } (i<=k<j) 发现权值和 k 无关了****
*/
#include<iostream>
using namespace std;
int m[1005][1005],sel[1005][1005];
int x[1005],y[1005];

int main() {
    int n;
    while( scanf("%d",&n)!=EOF ) {
        for(int i=1;i<=n;i++) scanf("%d%d",x+i,y+i);
        for(int i=1;i<=n;i++) {
            m[i][i]=x[i]+y[i];
            sel[i][i]=i;
        }
        for(int len=2;len<=n;len++) {
            for(int i=1;i+len-1 <=n;i++)  {
                int j=i+len-1;

                int mi=1000000000,s;

                for(int k=sel[i][j-1];k<=sel[i+1][j] && k<j;k++)//这里 k<j 一定要加上去,推导必须是严谨的
                {
                    if( m[i][k] + m[k+1][j] < mi)
                    {
                        mi=m[i][k]+m[k+1][j];
                        s=k;
                    }
                }

                m[i][j]=mi-x[i]-y[j];
                sel[i][j]=s;
            }
        }
        int ans=m[1][n]-x[1]-y[n];
        cout<<ans<<endl;
```

```
    }
}
```

最长上升子序列 LIS

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAXN = 200001;
const int INF = 2000000000;
int x,n,m,a[MAXN],f[MAXN];
bool cmp(int a,int b) {
    return a > b;//下降>不升>=,上升<不降<=且初值足够大
}
int LIS(int a[],int n,int f[]) {
    //f[0]=2147483647;//上升或不降时赋初值
    int m = 0;
    for (int i = 0;i < n;i++) {
        int *p = lower_bound(a,a + m,a[i],cmp);
        if (p == a+m) //不在有序数组中
            m++;
        *p = a[i]; //更新有序数组
        f[i] = p - a + 1; //以 i 结尾的最长序列
    }
    return m;
}
int main() {
    while (scanf("%d",&n) != EOF) {
        for (int i = 0;i < n;i++)
            scanf("%d",&a[i]);
        printf("%d\n",LIS(a,n,f));
    }
}
```

最长公共子串 O(NLOGN)

```
/*
设序列 AB,记序列 A 中各个元素在 B 中的位置(降序排列)
然后按在 A 中的位置依次列,最后求 A 的最长递增子序列
如：有 A={a,b,a,c,x}，B={b,a,a,b,c,a}
有 a={6,3,2},b={4,1},c={5};x=/;(注意降序排列)
然后按 A 中次序排出{a(6,3,2),b(4,1),a(6,3,2),c(5),x()}={6,3,2,4,1,6,3,2,5}；
对此序列求最长递增子序列即可
*/
int sq[100005],inds;//0~inds-1
int y[10005][12];//
int iy[10005]; //存放下标
int a[100010],b[100010];
int LCS(int n,int m){//最长公共子序列 nlogn
    int len=n*m;
    memset(iy,0,sizeof(iy));
    for(int i=1;i<=len;i++){
        int v=a[i];
        y[ v ][ iy[v]++ ]=i;
    }
    for(int i=1;i<n;i++){
        sort(y[i],y[i]+m);
    }
    inds=0;
    for(int i=1;i<=len;i++){
```

```
        for(int j=m-1;j>=0;j--) {
            int v=y[ b[i] ][j];//1~100000
            if( inds==0 ){
                sq[inds++]=v;
                continue;
            }
            int left=0,right=inds-1;
            if(sq[left]>=v){
                sq[left]=min(sq[left],v);
                continue;
            }
            if( sq[right]<v){
                sq[inds++]=v;
                continue;
            }
            while( left+1<right){
                int mid=(left+right)>>1;
                if( sq[mid] >= v) right=mid;
                else left=mid;
            }
            sq[right]=min(sq[right],v);
        }
    }
    return inds;
}
/*
Longest Common Subsequence 最长公共子序列
利用字符个数较少转化为 NlogN
MAXN 是长串长,MAXM 是每种字符最多出现次数
*/
const int MAXN = 100001,MAXM = 11,MAXV = 1061109567;
int n,m,e;
int a[MAXN],b[MAXN];
int head[MAXN],cnt[MAXN];//边表
int f[MAXN*MAXM],g[MAXN*MAXM],next[MAXN*MAXM],pos[MAXN*MAXM];//LIS

inline void add(int u,int p) {
    pos[e] = p;next[e] = head[u];head[u] = e++;cnt[u]++;
}

int LCS(int a[],int b[],int n) {
    e = 0;
    memset(head,-1,sizeof(head));
    memset(g,63,sizeof(g));
    for (int i = 0;i < n;i++)
        add(b[i],i);
    int p = 0,len = 0;
    for (int i = 0;i < n;i++)
        for (int j = head[a[i]];j != -1;j = next[j]) {
            int k = lower_bound(g,g+(len++),pos[j]) - g;
            f[p++] = k+1;
            g[k] = pos[j];
            //cout << pos[j] << ' ';
        }
    return *max_element(f,f+p);
}
```

多重背包

```
//多重背包 O(NV) HDU3591
#include <iostream>
using namespace std;
const int MAXN = 101;
int v[MAXN],c[MAXN],dp[20001],dp2[20001];
```

```
const int MAXM = 20001;
//v、n、w：当前所处理的这类物品的体积、个数、价值
//V：背包体积， MAXM：背包的体积上限值
//f[i]：体积为 i 的背包装前几种物品，能达到的价值上限。
inline void pack(int dp[],int V,int v,int n,int w) {
    if (n == 0 || v == 0) return;
    if (n == 1) { //01 背包
        for (int i = V; i >= v; --i)
            if (dp[i] > dp[i - v] + w) dp[i] = dp[i - v] + w;
        return;
    }
    if (n * v >= V - v + 1) { //完全背包(n >= V / v)
        for (int i = v; i <= V; ++i)
            if (dp[i] > dp[i - v] + w) dp[i] = dp[i - v] + w;
        return;
    }
    int va[MAXM], vb[MAXM]; //va/vb: 主/辅助队列
    for (int j = 0; j < v; ++j) { //多重背包
        int *pl = va, *pr = va - 1; //pl/pr 分别指向队列首/末元素
        int *ql = vb, *qr = vb - 1; //ql/qr 分别指向辅助队列首/末元素
        for (int k = j,i = 0;k <= V;k += v,i++) {
            if (pr == pl + n) { //若队列大小达到指定值，第一个元素 X 出队
                if (*pl == *ql) ++ql; //若辅助队列第一个元素等于 X，该元素也出队
                ++pl;
            }
            int now = dp[k] - i * w;
            *++pr = now; //元素 X 进队
            //删除辅助队列所有大于 X 的元素，ql 到 qr 单调递增，也可以用二分法
            while (qr >= ql && *qr > now) --qr;
            *++qr = now;                          //元素 X 也存放入辅助队列
            dp[k] = *ql + i * w;           //辅助队列首元素恒为指定队列所有元素的最大值
        }
    }
}
int main() {
    int n,m,cas = 0;
    while (scanf("%d%d",&n,&m) != EOF && (n || m)) {
        for (int i = 1;i <= n;i++)
            scanf("%d",&v[i]);
        for (int i = 1;i <= n;i++)
            scanf("%d",&c[i]);
        memset(dp,63,sizeof(dp));
        dp[0] = 0;
        for (int i = 1;i <= n;i++)
            pack(dp,20000,v[i],c[i],1);
        int ans = 0x3F3F3F3F;
        for (int i = m;i <= 20000;i++)
            ans = min(ans,dp[i]+dp[i-m]);
        if (ans == 0x3F3F3F3F)
            ans = -1;
        printf("Case %d: %d\n",++cas,ans);
    }
}
```

双调路径

```
/*
HDU2686
dp[i][j][k]表示从(0,0)出发走了 i 步，且 1 号路径横着走了 j 步，2 号路径横着走了 k 步时的最大值，
那么显然 1 号路径竖着走了 i-j 步，2 号路径竖着走了 i-k 步，这样本来要五维的状态降到了三维。
dp[i][j][k] = max(dp[i-1][j][k],dp[i-1][j-1][k],dp[i-1][j][k-1],dp[i-1][j-1][k-1]) + v;
其中 v = map[i-j][j] + (j == k ? 0 : mat[i-k][k]);
*/
#include <iostream>
```

```
using namespace std;
const int dx[] = {-1,-1,0,0};
const int dy[] = {-1,0,-1,0};
const int MAXN = 601;
int mat[MAXN][MAXN],dp[2][MAXN][MAXN];
int main() {
    int n;
    while (scanf("%d",&n) != EOF) {
        memset(mat,0,sizeof(mat));
        for (int i = 0;i < n;i++)
            for (int j = 0;j < n;j++)
                scanf("%d",&mat[i][j]);
        memset(dp,-1,sizeof(dp));
        int p = 0;
        dp[p][0][0] = mat[0][0];
        for (int i = 1;i <= n + n - 2;i++) {
            p = 1-p;
            int x = i;
            if (x > n)
                x = n-1;
            for (int j = 0;j <= x;j++)
                for (int k = 0;k <= x;k++) {
                    dp[p][j][k] = -1;
                    int v = mat[i-j][j] + (j == k ? 0 : mat[i-k][k]);
                    for (int l = 0;l < 4;l++)
                        if (j+dx[l] >= 0 && j+dy[l] >= 0)
                            if (dp[1-p][j+dx[l]][k+dy[l]]> dp[p][j][k])
                                dp[p][j][k] = dp[1-p][j+dx[l]][k+dy[l]];
                    dp[p][j][k] += v;
                }
        }
        printf("%d\n",dp[p][n-1][n-1]);
    }
}
```

DIGITCOUNT

```
/*
http://hi.baidu.com/sunshine_0316/blog/item/e34ade3c01c986c19f3d6235.html
http://topic.csdn.net/u/20071101/23/03c40abb-cc82-450c-8781-fc121e616ba6.html
http://blog.csdn.net/shiren_Bod/archive/2010/08/10/5802827.aspx
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
typedef long long LL;
LL f[] = {0, 1, 20, 300, 4000, 50000, 600000, 7000000, 80000000, 900000000};
LL ten[] = {0, 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000};
LL cnt[10];
/*
因为[0,10^x-1]中每个数字出现的次数相同，设 f[x]为该范围内 y 的个数，则有 f[x] = 10^(x-1) + 10 * f[x-1];
前者代表这一位是 y 共有几个数，后者代表这一位[0,9]任取再乘上前面几位中 y 的个数，解得 f[x] = x * 10^(x-1);
34567 为例子
34560~34567 中：末 0 位的[0,10^0-1]中每个数字共 7 * f[0]=0 个    第 1 位放[0,6]共 10^1 第 1 位放 7 共[0,0] = 1 个
34500~34567 中：末 1 位的[0,10^1-1]中每个数字共 6 * f[1]=6 个    第 2 位放[0,5]共 10^2 第 2 位放 6 共[0,7] = 8 个
34000~34567 中：末 2 位的[0,10^2-1]中每个数字共 5 * f[2]=100 个   第 3 位放[0,4]共 10^3 第 3 位放 5 共[0,67] = 68 个
30000~34567 中 末 3 位的[0,10^3-1]中每个数字共 4 * f[3]=1200 个  第 4 位放[0,3]共 10^4 第 4 位放 4 共[0,567] = 568 个
```

```
00000~34567中：末4位的[0,10^4-1]中每个数字共3 * f[4]=12000个 第5位放[0,2]共10^5 第5位放3共[0,4567] = 4568个
*/
void get(LL x, LL cnt[]) {
    if (x < 0) return;
    char s[11];
    sprintf(s, "%lld", x);
    int n = strlen(s);
    LL left = 0;//末位后的余数
    for (int i = 0; i < n; ++i) {
        int d = s[n - i - 1] - '0';//末位
        for (int j = 0; j < 10; ++j) {
            cnt[j] += d * f[i];
        }
        for (int j = 0; j < d; ++j) {
            cnt[j] += ten[i + 1];
        }
        cnt[d] += left + 1;
        cnt[0] -= ten[i + 1];
        left += d * ten[i + 1];
    }
}
int main() {
    LL l, r;
    while (scanf("%lld%lld", &l, &r) != EOF) {
        if (l + r <= 0) break;
        if (l > r) swap(l, r);
        for (int i = 0; i < 10; ++i) {
            cnt[i] = 0;
        }
        get(l - 1, cnt);
        for (int i = 0; i < 10; ++i) {
            cnt[i] = -cnt[i];
        }
        get(r, cnt);
        for (int i = 0;i < 10;i++) {
            if (i) putchar(' ');
            printf("%lld", cnt[i]);
        }
        puts("");
        //printf("%lld\n", cnt[0] + (l == 0));
    }
}
```

树上最长路径 树形 DP

```
#include <iostream>
using namespace std;
const int MAXN = 10086;
vector<pair<int, int> > G[MAXN];
int diameter, num;
pair<int, int> dfs(int fa, int u) {
    int h1 = 0, h2 = 0, cnt1 = 1, cnt2 = 1, add = 0; //最高和次高
    for (int i = 0; i < G[u].size(); i++) {
        int v = G[u][i].first;
        if (v != fa) {
            pair<int, int> p = dfs(u, v);
            int h = p.first + G[u][i].second;
            if (h == h1) {
                add += cnt1 * p.second;
                cnt1 += p.second;
            }
            if (h == h2) {
                cnt2 += p.second;
```

```
            }
            if (h > h1) {
                h2 = h1;
                h1 = h;
                cnt2 = cnt1;
                cnt1 = p.second;
                add = 0;
            } else if (h > h2) {
                h2 = h;
                cnt2 = p.second;
            }
        }
    }
    if (add == 0) {
        add = cnt1 * cnt2;
    }
    if (h1 + h2 == diameter) {
        num += add;
    } else if (h1 + h2 > diameter) {
        diameter = h1 + h2;
        num = add;
    }
    return make_pair(h1, cnt1);
}
int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; i++) G[i].clear();
        int x,y,z;
        for (int i = 0; i < n - 1; i++) {
            scanf("%d%d%d", &x, &y, &z);
            x--, y--;
            G[x].push_back(make_pair(y, z));
            G[y].push_back(make_pair(x, z));
        }
        num = 0;
        diameter = -1<<30;
        dfs(-1, 0);
        printf("%d %d\n", diameter, num);
    }
}
```

最小区间覆盖

```
//work(p,n)为答案
const int MAXN = 50010;
struct data {int l,r;}p[MAXN];
bool cmp(const data& a,const data& b) {
    return a.l == b.l? a.r < b.r : a.l < b.l;
}
int work(data p[],int n) {
    sort(p,p+n,cmp);
    int Min = 1<<30,Max = -1<<30;
    for (int i = 0;i < n;i++){
        Min = min(Min,p[i].l);
        Max = max(Max,p[i].r);
    }
    int cur = Min,i = 0,sum = 0;
    while (cur <= Max && i < n) {
        int tmp = cur;
        for (;p[i].l <= cur;i++)
            if(p[i].r > tmp)
                tmp = p[i].r;
        cur = tmp + 1;
```

```
        sum++;
    }
    return sum;
}
```

七：其他

高精度

```
#define SIZE(X) ((int)(X.size()))//NOTES:SIZE(
#define LENGTH(X) ((int)(X.length()))//NOTES:LENGTH(
#define MP(X,Y) make_pair(X,Y)//NOTES:MP(
const int maxlength=2000;
class bigint
{
public:
    int oper,length,a[maxlength];
    bigint(int=0);
    ~bigint();
    int max(int a,int b);
    void check();
    void operator=(bigint m);
    void operator=(int m);
    void operator=(char *s);
    bool operator<(bigint m);
    bool operator<=(bigint m);
    bool operator>(bigint m);
    bool operator>=(bigint m);
    bool operator==(bigint m);
    bool operator!=(bigint m);
    bigint operator-();
    bigint operator+(bigint m);
    void operator+=(bigint m);
    bigint operator-(bigint m);
    void operator-=(bigint m);
    bigint operator*(bigint m);
    bigint operator*(int m);
    void operator*=(bigint m);
    void operator*=(int m);
    bigint operator/(bigint m);
    bigint operator/(int m);
    void operator/=(bigint m);
    void operator/=(int m);
    bigint operator%(bigint m);
    bigint operator%(int m);
    void operator%=(bigint m);
    void operator%=(int m);
};
bigint abs(bigint m);
bool read(bigint &m);
void write(bigint m);
void swrite(char *s,bigint m);
void writeln(bigint m);
bigint sqr(bigint m);
bigint sqrt(bigint m);
bigint gcd(bigint a,bigint b);
bigint lcm(bigint a,bigint b);
int bigint::max(int a,int b) {
```

```
    return (a>b)?a:b;
}
bigint::bigint(int v) {
    (*this)=v;
    this->check();
}
bigint::~bigint() {}
void bigint::check() {
    for (;length>0 && a[length]==0;length--);
    if (length==0)
        oper=1;
}
void bigint::operator=(bigint m) {
    oper=m.oper;
    length=m.length;
    memcpy(a,m.a,maxlength*sizeof(int));
    this->check();
}
void bigint::operator=(int m) {
    oper=(m>0)?1:-1;
    m=abs(m);
    memset(a,0,maxlength*sizeof(int));
    for (length=0;m>0;m=m/10000)
        a[++length]=m%10000;
    this->check();
}
void bigint::operator=(char *s) {
    int i,L;
    (*this)=0;
    if (s[0]=='-' || s[0]=='+') {
        if (s[0]=='-')
            oper=-1;
        L=strlen(s);
        for (i=0;i<L;i++)
            s[i]=s[i+1];
    }
    L=strlen(s);
    length=(L+3)/4;
    for (i=0;i<L;i++)
        a[(L-i+3)/4]=a[(L-i+3)/4]*10+(s[i]-48);
    this->check();
}
bool bigint::operator<(bigint m) {
    if (oper!=m.oper)
        return oper<m.oper;
    if (length!=m.length)
        return oper*length<m.length*oper;
    for (int i=length;i>=1;i--)
        if (a[i]!=m.a[i])
            return a[i]*oper<m.a[i]*oper;
    return false;
}
bool bigint::operator<=(bigint m) {
    return !(m<(*this));
}
bool bigint::operator>(bigint m) {
    return m<(*this);
}
bool bigint::operator>=(bigint m) {
    return !((*this)<m);
}
bool bigint::operator==(bigint m) {
    return (!((*this)<m)) && (!(m<(*this)));
}
bool bigint::operator!=(bigint m) {
    return ((*this)<m) || (m<(*this));
```

```
}
bigint bigint::operator-() {
    bigint c=(*this);
    c.oper=-c.oper;
    c.check();
    return c;
}
bigint abs(bigint m) {
    bigint c=m;
    c.oper=abs(c.oper);
    c.check();
    return c;
}
bigint bigint::operator+(bigint m) {
    if (m.length==0)
        return (*this);
    if (length==0)
        return m;
    if (oper==m.oper) {
        bigint c;
        c.oper=oper;
        c.length=max(length,m.length)+1;
        for (int i=1,temp=0;i<=c.length;i++)
            c.a[i]=(temp=(temp/10000+a[i]+m.a[i]))%10000;
        c.check();
        return c;
    }
    return (*this)-(-m);
}
bigint bigint::operator-(bigint m) {
    if (m.length==0)
        return (*this);
    if (length==0)
        return (-m);
    if (oper==m.oper) {
        bigint c;
        if (abs(*this)>=abs(m)) {
            c.oper=oper;
            c.length=length;
            for (int i=1,temp=0;i<=length;i++)
                c.a[i]=((temp=(-int(temp<0)+a[i]-m.a[i]))+10000)%10000;
            c.check();
            return c;
        }
        return -(m-(*this));
    }
    return (*this)+(-m);
}
bool read(bigint &m) {
    char s[maxlength*4+10];
    if (scanf("%s",s)==-1)
        return false;
    for (int i=0;s[i];i++)
        if (!(s[i]>='0' && s[i]<='9' || (s[i]=='+' || s[i]=='-') && i==0))
            return false;
    m=s;
    return true;
}
void swrite(char *s,bigint m) {
    int L=0;
    if (m.oper==-1)
        s[L++]='-';
    sprintf(s+L,"%d",m.a[m.length]);
    for (;s[L]!=0;L++);
    for (int i=m.length-1;i>=1;i--) {
        sprintf(s+L,"%04d",m.a[i]);
```

```
        L+=4;
    }
    s[L]=0;
}
void write(bigint m) {
    if (m.oper==-1)
        printf("-");
    printf("%d",m.a[m.length]);
    for (int i=m.length-1;i>=1;i--)
        printf("%04d",m.a[i]);
}
void writeln(bigint m) {
    write(m);
    printf("\n");
}
bigint bigint::operator*(bigint m) {
    bigint c;
    c.oper=oper*m.oper;
    c.length=length+m.length;
    for (int i=1;i<=m.length;i++) {
        int number=m.a[i],j,temp=0;
        for (j=1;j<=length;j++)
            c.a[i+j-1]+=number*a[j];
        if (i%10==0 || i==m.length)
            for (j=1;j<=c.length;j++)
                c.a[j]=(temp=(temp/10000)+c.a[j])%10000;
    }
    c.check();
    return c;
}
bigint bigint::operator*(int m) {
    if (m<0)
        return -((*this)*(-m));
    if (m>100000)
        return (*this)*bigint(m);
    bigint c;
    c.length=length+2;
    c.oper=oper;
    int t=0;
    for (int i=1;i<=c.length;i++)
        c.a[i]=(t=(t/10000+a[i]*m))%10000;
    c.check();
    return c;
}
bigint bigint::operator/(bigint m) {
    if (m.length==0) {
        printf("Division by zero.\n");
        exit(0);
    }
    if (abs(*this)<abs(m))
        return 0;
    bigint c,left;
    c.oper=oper/m.oper;
    m.oper=1;
    c.length=length-m.length+1;
    left.length=m.length-1;
    memcpy(left.a+1,a+length-left.length+1,left.length*sizeof(int));
    for (int i=c.length;i>=1;i--) {
        left=left*10000+a[i];
        int head=0,tail=10000,mid;
        while (head+1<tail) {
            mid=(head+tail)/2;
            if (m*mid<=left)
                head=mid;
            else
                tail=mid;
```

```
        }
        c.a[i]=head;
        left-=m*head;
    }
    c.check();
    return c;
}
bigint bigint::operator/(int m) {
    if (m<0)
        return -((*this)/(-m));
    if (m>100000)
        return (*this)/bigint(m);
    bigint c;
    c.oper=oper;
    c.length=length;
    int t=0;
    for (int i=c.length;i>=1;i--)
        c.a[i]=(t=(t%m*10000+a[i]))/m;
    c.check();
    return c;
}
bigint bigint::operator%(bigint m) {
    return (*this)-((*this)/m)*m;
}
bigint bigint::operator%(int m) {
    if (m<0)
        return -((*this)%(-m));
    if (m>100000)
        return (*this)%bigint(m);
    int t=0;
    for (int i=length;i>=1;i--)
        t=(t*10000+a[i])%m;
    return t;
}
bigint sqr(bigint m) {
    return m*m;
}
bigint sqrt(bigint m) {
    if (m.oper<0 || m.length==0)
        return 0;
    bigint c,last,now,templast;
    c.length=(m.length+1)/2;
    c.a[c.length]=int(sqrt((double)m.a[c.length*2]*10000+m.a[c.length*2-1])+1e-6);
    templast.length=c.length*2;
    templast.a[c.length*2-1]=(c.a[c.length]*c.a[c.length])%10000;
    templast.a[c.length*2]=(c.a[c.length]*c.a[c.length])/10000;
    templast.check();
    for (int i=c.length-1;i>=1;i--) {
        last=templast;
        int head=0,tail=10000,mid,j,temp;
        while (head+1<tail) {
            mid=(head+tail)/2;
            now=last;
            now.a[2*i-1]+=mid*mid;
            for (j=i+1;j<=c.length;j++)
                now.a[i+j-1]+=mid*c.a[j]*2;
            now.length++;
            for (j=2*i-1,temp=0;j<=now.length;j++)
                now.a[j]=(temp=(temp/10000+now.a[j]))%10000;
            now.check();
            if (now<=m) {
                templast=now;
                head=mid;
            }
            else
                tail=mid;
```

```
        }
        c.a[i]=head;
    }
    c.check();
    return c;
}
bigint gcd(bigint a,bigint b) {
    return (b==0)?a:gcd(b,a%b);
}
bigint lcm(bigint a,bigint b) {
    return a*b/gcd(a,b);
}
void bigint::operator+=(bigint m) {
    (*this)=(*this)+m;
}
void bigint::operator-=(bigint m) {
    (*this)=(*this)-m;
}
void bigint::operator*=(bigint m) {
    (*this)=(*this)*m;
}
void bigint::operator/=(bigint m) {
    (*this)=(*this)/m;
}
void bigint::operator%=(bigint m) {
    (*this)=(*this)%m;
}
void bigint::operator*=(int m) {
    (*this)=(*this)*m;
}
void bigint::operator/=(int m) {
    (*this)=(*this)/m;
}
void bigint::operator%=(int m) {
    (*this)=(*this)%m;
}
int main() {
    bigint a,b;
    while (read(a)) {
        read(b);
        writeln(a+b);
    }
}
```

LL 高精度

```
#include <iostream>
using namespace std;
typedef long long LL;
LL gcd(LL a, LL b) {
    return b ? gcd(b, a % b) : a;
}
struct BigInt {
    static const LL MOD = 10000000000000000000LL;
    LL h, l;
    BigInt() : h(0), l(0) {
    }
    void add(LL x) {
        l += x;
        while (l >= MOD) {
            l -= MOD;
            ++h;
        }
    }
```

```
    void print() {
        if (h > 0) {
            printf("%lld%018lld\n", h, l);
        } else {
            printf("%lld\n", l);
        }
    }
    void reduction(LL x) { //num / x 通分
        if (x == 0) {
            printf("NaN");
            return;
        }
        LL g = gcd((h % x) * (MOD % x) + l, x);
        LL hh = h / g;
        LL ll = (h % g) * (MOD / g) + ((h % g) * (MOD % g) + l) / g;
        if (hh > 0) {
            printf("%lld%018lld/%lld", hh, ll, x / g);
        } else {
            printf("%lld/%lld", ll, x / g);
        }
    }
};
int main() {
    BigInt a, b;
    int t = 1111111;
    while (t--) {
        a.add(98765434123LL);
    }
    a.print();
    a.reduction(1111111);
}
```

DLX 精确覆盖

```
//Dancing Links 精确覆盖 PKU2676 解数独
#include <iostream>
#include <cstdio>
#include <cstring>
const int MAXR = 1000; //row
const int MAXC = 1000; //column
const int TOTAL = (MAXR+1)*(MAXC+1);
const int MAXV = 0x3F3F3F3F;
using namespace std;

int n,m,mat[MAXR][MAXC];//初始矩阵,0 或 1,从 1,1 开始用;
int cid[TOTAL],rid[TOTAL],s[MAXC]; //显示行号，列号，每列的总个数
int res[MAXR]; //每次取的哪行
int l[TOTAL],r[TOTAL],u[TOTAL],d[TOTAL];
int lastc[MAXC]; //第 i 列的最下面一个元素

void remove(const int &col) { //删掉
    l[r[col]] = l[col];
    r[l[col]] = r[col];
    for(int i = d[col];i != col;i = d[i]) { //枚举每一行
        for(int j = r[i];j != i;j = r[j]) { //枚举每一列
            d[u[j]] = d[j];
            u[d[j]] = u[j];
            s[cid[j]]--;
        }
    }
}

void resume(const int &col) { //恢复
```

```
    for(int i = u[col];i != col;i = u[i]) {
        for(int j = l[i];j != i;j = l[j]) {
            d[u[j]] = j;
            u[d[j]] = j;
            s[cid[j]]++;
        }
    }
    l[r[col]] = col;
    r[l[col]] = col;
}

inline void addlr(const int &ln,const int &rn) { //完了后还是成环的
    l[r[ln]] = rn;
    r[rn] = r[ln];
    r[ln] = rn;
    l[rn] = ln;
}

inline void addud(const int &un,const int &dn) {
    u[d[un]] = dn;
    d[dn] = d[un];
    d[un] = dn;
    u[dn] = un;
}

bool search(int k) { //将开始第几层搜索
    if(r[0] == 0) {
        return true; //找到一组解
    }
    int c,minv = MAXV;
    for (int i = r[0];i != 0;i = r[i]) {
        if (s[i] < minv) {
            minv = s[i];
            c = i;
        }
    }
    if (minv == 0)//可以不要，包括在下面这个 for 里的
        return false;
    remove(c);
    for (int i = d[c];i != c;i = d[i]) {
        res[k] = rid[i]; //选择了哪一行，这个数组可以用来输出所选择的行
        for (int j = r[i];j != i;j = r[j])
            remove(cid[j]);
        if (search(k+1))
            return true;
        for (int j = l[i];j != i;j = l[j])
            resume(cid[j]);
    }
    resume(c);
    return false;
}

void adapt(int n,int m) { //传入 01 矩阵 mat 的规格,适配函数(由 mat 构造 dancing link 表)
    memset(s,0,sizeof(s)); //每列元素个数清 0
    l[0] = r[0] = 0; //head 是 0
    for(int i = 1;i <= m;i++) { //总共有 m 列
        addlr(i-1,i);
        u[i] = d[i] = i; //自成循环
        lastc[i] = i; //最下面一个元素号码
    }
    int ind = m+1;//从 m+1 开始用
    for (int i = 1;i <= n;i++) {
        bool isfirst = true;
        for (int j = 1;j <= m;j++) {
            if (mat[i][j]) {
                if (isfirst) {
```

```
                isfirst = false;
                l[ind] = r[ind] = ind;
            }
            else {
                addlr(ind-1,ind);
            }
            addud(lastc[j],ind);
            lastc[j] = ind;
            rid[ind] = i;
            cid[ind] = j;
            s[j]++;
            ind++;
        }
    }
}
}
char sudoku[10][10];
int ans;
void build() {
    memset(mat,0,sizeof(mat));
    int l,r;
    for (int i = 1;i <= 9;i++)
        for (int j = 1;j <= 9;j++) {
            if (sudoku[i][j] != '0')
                l = sudoku[i][j] - '0',r = sudoku[i][j] - '0';
            else
                l = 1,r = 9;
            for (int k = l;k <= r;k++) {
                mat[(k-1)*81+(i-1)*9+j][243+(i-1)*9+j] = 1;//1 blocks 1 number
                mat[(k-1)*81+(i-1)*9+j][(0+(k-1)*3)*9+i] = 1;//row i contains k
                mat[(k-1)*81+(i-1)*9+j][(1+(k-1)*3)*9+j] = 1;//column j contains k
                mat[(k-1)*81+(i-1)*9+j][(2+(k-1)*3)*9+(i-1)/3*3+(j-1)/3+1] = 1;//blocks(i,j) contains
k
            }
        }
}
void print() {
    for (int i = 1;i <= 81;i++) {
        int x = res[i];
        int k = (x-1) / 81 + 1;
        x -= (k-1) * 81;
        int r = (x-1) / 9 + 1;
        x -= (r-1) * 9;
        int c = x;
        sudoku[r][c] = k + '0';
    }
    for (int i = 1;i <= 9;i++) {
        for (int j = 1;j <= 9;j++)
            printf("%c",sudoku[i][j]);
        printf("\n");
    }
}
int main() {
    int cas;
    scanf("%d",&cas);
    while (cas--) {
        for(int i = 1;i <= 9;i++) {
            getchar();
            for(int j = 1;j <= 9;j++)
                scanf("%c",&sudoku[i][j]);
        }
        build();
        n = 729,m = 324;
        adapt(n,m);
        ans = 0;
        if (search(1)) print();
```

```
    }
}
```

DLX 重复覆盖

```
//Dancing links 重复覆盖问题 NUAA 1507
#include <iostream>
#include <cstdio>
#include <cstring>
const int MAXR = 62; //row
const int MAXC = 62; //column
const int TOTAL = (MAXR + 1) * (MAXC + 1);
const int MAXV = 0x3F3F3F3F;
using namespace std;

int n, m, mat[MAXR][MAXC];//初始矩阵,0 或 1,从 1,1 开始用
int cid[TOTAL], rid[TOTAL], s[MAXC]; //显示行号，列号，每列的总个数
int res[MAXR]; //每次取的哪行
int l[TOTAL], r[TOTAL], u[TOTAL], d[TOTAL];
int lastc[MAXC]; //第 i 列的最下面一个元素

void remove(const int& col) {//这一整列没了，包括列头**********
    for (int i = d[col]; i != col; i = d[i]) {
        r[l[i]] = r[i];
        l[r[i]] = l[i];
    }
}
void resume(const int& col) {
    for (int i = u[col]; i != col; i = u[i]) {
        r[l[i]] = i;
        l[r[i]] = i;
    }
}
inline void addlr(const int& ln, const int& rn) { //完了后还是成环的
    l[r[ln]] = rn;
    r[rn] = r[ln];
    r[ln] = rn;
    l[rn] = ln;
}
inline void addud(const int& un, const int& dn) {
    u[d[un]] = dn;
    d[dn] = d[un];
    d[un] = dn;
    u[dn] = un;
}
//******************************************************
bool visit[MAXC];
int H() { //估价函数，表示要重复覆盖全部还至少需要几行
    int res = 0;
    memset(visit, false, sizeof(visit));
    for (int c = r[0]; c != 0; c = r[c]) {
        if (!visit[c]) {
            res++;
            visit[c] = true;
            for (int i = d[c]; i != c; i = d[i])
                for (int j = r[i]; j != i; j = r[j])
                    visit[cid[j]] = true;
        }
    }
    return res;
}
//******************************************************
int mi;
void search(int k, int sum) {//将开始第几层搜索,k 是输出答案数组的下表,sum 是当前已经多少层了
```

```
    if (sum + H() >= mi)
        return;
    if (r[0] == 0) {
        mi = min(mi, sum);
        return;//找到一组解
    }
    int c, mi = 1000000;
    for (int i = r[0]; i != 0; i = r[i]) {
        if (s[i] < mi) {
            mi = s[i];
            c = i;
        }
    }
    for (int i = d[c]; i != c; i = d[i]) {
        res[k] = rid[i];
        remove(i);
        for (int j = r[i]; j != i; j = r[j])
            remove(j);
        search(k + 1, sum + 1);
        for (int j = l[i]; j != i; j = l[j])
            resume(j);
        resume(i);
    }
}

void adapt(int n, int m) { //传入 01 矩阵 mat 的规格,适配函数(由 mat 构造 dancing link 表)
    memset(s, 0, sizeof(s)); //每列元素个数清 0
    l[0] = r[0] = 0; //head 是 0
    for (int i = 1; i <= m; i++) { //总共有 m 列
        addlr(i - 1, i);
        u[i] = d[i] = i; //自成循环
        lastc[i] = i; //最下面一个元素号码
    }
    int ind = m + 1;//从 m+1 开始用
    for (int i = 1; i <= n; i++) {
        bool isfirst = true;
        for (int j = 1; j <= m; j++) {
            if (mat[i][j]) {
                if (isfirst) {
                    isfirst = false;
                    l[ind] = r[ind] = ind;
                } else {
                    addlr(ind - 1, ind);
                }
                addud(lastc[j], ind);
                lastc[j] = ind;
                rid[ind] = i;
                cid[ind] = j;
                s[j]++;
                ind++;
            }
        }
    }
}
int main() {
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF) {
        memset(mat, 0, sizeof(mat));
        int x, k;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &x);
            for (int j = 1; j <= x; j++)
                scanf("%d", &k), mat[i][k] = 1;
        }
        adapt(n, m);
        mi = m;
```

```
        search(1, 0);
        printf("%d\n", mi);
    }
}
```

表达式计算

```
#include<iostream>
using namespace std;
map<char,int>mp;
void preprocess() { //出现新的操作符可以在这里添加，值的大小为优先级的高低
    mp['|'] = 0;mp['&'] = 0;//位或 位与运算
    mp['+'] = 1;mp['-'] = 1;
    mp['*'] = 2;mp['/'] = 2;
    mp['%'] = 2;mp['^'] = 3;//取余 乘方运算
    mp['('] = 10000;//')'特殊考虑的
}
void cal(stack<int> &val,stack<char> &opt) {
    int val1,val2,ans;
    char ch;
    val1 = val.top(),val.pop();
    val2 = val.top(),val.pop();
    ch = opt.top(),opt.pop();
    switch(ch) {
        case '&':ans=val2&val1;break;
        case '|':ans=val2|val1;break;
        case '+':ans=val2+val1;break;
        case '-':ans=val2-val1;break;
        case '*':ans=val2*val1;break;
        case '/':ans=val2/val1;break;
        case '%':ans=val2%val1;break;
        case '^':
            ans = 1;
            for (;val1;val1>>=1) {//整数幂的二分加速
                if (val1&1)
                    ans*=val2;
                val2*=val2;
            }
            break;
    }
    val.push(ans);
    return;
}
int calculate(string &s) {
    int len = s.length(),ind = 0;
    for (int i = 0;i < len;i++) //过滤空格
        if (s[i] != ' ')
            s[ind++] = s[i];
    len = ind;
    if(s[0] == '-') {
        s = '0' + s;
        len++;
    }
    stack<int> val;
    stack<char> opt;
    for (int i = 0;i < len;i++) {
        if (s[i] == '(') {
            opt.push(s[i]);
            continue;
        }
        else if (s[i] == ')') {
            while (opt.top() != '(')
                cal(val,opt);
            opt.pop();//弹出'('
```

```
        }
        else if (isdigit(s[i])) {
            int value = 0;
            while (i < len && isdigit(s[i])) {
                value = 10 * value + (s[i]-'0');
                i++;
            }
            i--;//返回一个
            val.push(value);
        }
        else {
            char op = s[i];
            while (!opt.empty() && opt.top() != '(' && mp[opt.top()] >= mp[op])
                cal(val,opt);
            opt.push(op);
        }
    }
    while (!opt.empty())
        cal(val,opt);
    return val.top();
}
int main() {
    preprocess();//这里面定义运算的优先级
    string s;
    cout << "请输入中缀表达式:\n";
    while (getline(cin,s)) {
        int ans = calculate(s);
        cout << "the result is " << ans << endl;
    }
}
```

跳马问题

```
//CII4970
//可以用公式或用附近若干个点预处理 BFS
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <queue>
using namespace std;
typedef long long LL;
const int LIMIT = 500;
const LL INF = 1LL << 50;
const int dx[] = { -2, -1, 1, 2, 2, 1, -1, -2};
const int dy[] = {1, 2, 2, 1, -1, -2, -2, -1};
LL step[LIMIT][LIMIT];

bool isvalid(int x, int y) {
    return x >= 0 && x < LIMIT && y >= 0 && y < LIMIT;
}
queue<int> q;
void bfs() {
    memset(step, 63, sizeof(step));
    q.push(0);
    q.push(0);
    step[0][0] = 0;
    int x, y, nx, ny;
    while (!q.empty()) {
        x = q.front(); q.pop();
        y = q.front(); q.pop();
        for (int i = 0; i < 8; i++) {
            nx = x + dx[i];
            ny = y + dy[i];
```

```
            if (nx < 0) nx = -nx;
            if (ny < 0) ny = -ny;
            if (!isvalid(nx, ny)) continue;
            if (step[x][y] + 1 < step[nx][ny]) {
                step[nx][ny] = step[x][y] + 1;
                q.push(nx);
                q.push(ny);
            }
        }
    }
}

LL get(LL x, LL y) {//返回从(0, 0)到(x, y)所需的最少步数
    x = x > 0 ? x : -x;
    y = y > 0 ? y : -y;
    if (x > y)swap(x, y);
    if (x == 0 && y == 1)return 3;
    if (x == 2 && y == 2)return 4;
    LL r = max((y + 1) / 2, (x + y + 2) / 3);
    if ((r - x - y) & 1)r++;
    return (LL)r;
}

LL go(LL x, LL y) {
    if (x < 0) x = -x;
    if (y < 0) y = -y;
    if (x > y) swap(x, y);
    if (2 * y < x || 2 * x < y) {
        if ((y & 1) || (y / 2 + x) % 2) return INF;
        return y / 2;
    }
    if ((2 * y - x) % 3 != 0 || (2 * x - y) % 3 != 0) return INF;
    return (2 * y - x) / 3 + (2 * x - y) / 3;
}

LL dfs(LL x, LL y) {
    if (x < 0) x = -x;
    if (y < 0) y = -y;
    if (x > y) swap(x, y);
    if (x < LIMIT && y < LIMIT) return step[(int)x][(int)y];
    LL ans = INF;
    for (int i = -10; i < 10; i++) {
        for (int j = -10; j < 10; j++) {
            LL tmp = step[abs(i)][abs(j)] + go(x + i, y + j);
            ans = min(ans, tmp);
        }
    }
    return ans;
}

LL inf = 1LL << 40;
struct graph {
    int n;
    LL a[20][20], lx[20], ly[20];
    int visx[20];
    int visy[20];
    LL slack[20];
    int match[20];
    void init(int _n) {
        n = _n;
        memset(a, 0, sizeof(a));
        memset(slack, 0, sizeof(slack));
    }
    int dfs(int u) {
        int i;
        visx[u] = 1;
```

```
        for (i = 0; i < n; i++) {
            if (visy[i])continue;
            if (lx[u] + ly[i] == a[u][i]) {
                visy[i] = 1;
                if (match[i] == -1 || dfs(match[i])) {
                    match[i] = u;
                    return 1;
                }
            } else {
                slack[i] = min(slack[i], lx[u] + ly[i] - a[u][i]);
            }
        }
        return 0;
    }
    LL km() {
        int i, j;
        memset(lx, 0, sizeof(lx));
        memset(ly, 0, sizeof(ly));
        memset(match, -1, sizeof(match));
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                lx[i] = max(lx[i], a[i][j]);
            }
        }
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++)slack[j] = inf;
            while (1) {
                memset(visx, 0, sizeof(visx));
                memset(visy, 0, sizeof(visy));
                if (dfs(i))break;
                LL d = inf;
                for (j = 0; j < n; j++) {
                    if (!visy[j])d = min(d, slack[j]);
                }
                for (j = 0; j < n; j++) {
                    if (visx[j])lx[j] -= d;
                    if (visy[j])ly[j] += d;
                    else slack[j] -= d;
                }
            }
        }
        LL ans = 0;
        for (i = 0; i < n; i++) {
            ans += lx[i] + ly[i];
        }
        return ans;
    }
} g;
int x1[16], y1[16], x2[16], y2[16];
int main() {
    bfs();
    int n, T = 1;
    while (scanf("%d", &n) != EOF && n) {
        for (int i = 0; i < n; i++) {
            scanf("%d%d", &x1[i], &y1[i]);
        }
        for (int i = 0; i < n; i++) {
            scanf("%d%d", &x2[i], &y2[i]);
        }
        g.init(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                LL t = -dfs((LL)x1[i] - x2[j], (LL)y1[i] - y2[j]);
                g.a[i][j] = t;
            }
        }
```

```
        printf("%d. %lld\n", T++, -g.km());
    }
}
```

二分查找

```
#include <iostream>
using namespace std;
const int MAXN = 100;
int find(int x,int a[],int n) {
    int mid,l = 0,r = n - 1;
    while (l <= r) {
        mid=(l + r) >> 1;//mid=low+((high-low)/2);防溢出
        if (x == a[mid]) return mid;
        if (x < a[mid]) r = mid - 1;
        if (x > a[mid]) l = mid + 1;
    }
    return -1;//更新第一个小于等于 x 的最大值用 r,第一个大于等于 x 的最小值用 l;
}
int BinarySearchLower(int l,int r,bool a[]) {//找出第一个出现表达式 true 的位置
    int mid;
    while (l < r) {
        mid = l+ ((r-l)>>1);//取中间位置
        if (a[mid]) r = mid;//中间为 true，丢弃 mid 右边的 true
        else l = mid + 1;//中间为 false，丢弃 mid 及其左边的 false
    }
    if (!a[l]) return -1;//不存在
    else return l;//第一个出现表达式 true 的位置
}
int BinarySearchUpper(int l,int r,bool a[]) {//找出 false 的最大的元素位置。
    int mid;
    while (l < r) {
        mid = l + ((r-l+1)>>1);//取中间坐标，这里是唯一的不同
        if (a[mid]) r=mid-1;//a[mid]为真，我们删除 mid 以及其右边的一半
        else l=mid;//a[mid]为假，我们删除 mid 左边的一半
    }
    if (a[l]) return -1;//没找到
    else return l;//最后一个出现表达式 false 的位置
}
int main() {
    int x,num[MAXN];
    for (int i = 0;i < MAXN;i++)
        num[i] = i;//MAXN - i + 1;
    cin >> x;
    cout << find(x,num,MAXN) << endl;
    return 0;
}
```

1X2 矩形完美覆盖

```
//8x8->12988816
#include <iostream>
#include <cmath>
using namespace std;
#define sqr(x) ((x)*(x))
const double pi=acos(-1.0);//NOTES:pi
int main() {
    int n,m;//n*m 棋盘
    while (cin >> n >> m && n && m) {
        double ans = 1;
        for (int i = 1;i <= m / 2;i++)
```

```
            for (int j = 1;j <= n / 2;j++)
                ans *= (4*(sqr(cos(i*pi/(m+1)))+sqr(cos(j*pi/(n+1)))));
        if (n % 2 && m % 2)
            ans = 0;
        cout << (long long)ans << endl;
    }
}
```

矩形切割

```
#include <iostream>
using namespace std;
const int MAXN = 1001;
int n,m,c,x1[MAXN],y1[MAXN],x2[MAXN],y2[MAXN],color[MAXN];
void cover(int l,int r,int d,int u,int col,int k) { //左右下上 颜色 序号
    while (k <= c && (x1[k] >= r || x2[k] <= l || y1[k] >= u || y2[k] <= d))
        k++;
    if (k > c) {color[col]++;return;}
    if (x1[k] > l) {cover(l,x1[k],d,u,col,k+1);l = x1[k];}
    if (x2[k] < r) {cover(x2[k],r,d,u,col,k+1);r = x2[k];}
    if (y1[k] > d) {cover(l,r,d,y1[k],col,k+1);d = y1[k];}
    if (y2[k] < u) {cover(l,r,y2[k],u,col,k+1);u = y2[k];}
}
int main() {
    while (scanf("%d%d%d",&n,&m,&c) != EOF) {
        memset(color,0,sizeof(color));
        for (int i = 1;i <= c;i++) {
            scanf("%d%d%d%d",&x1[i],&y1[i],&x2[i],&y2[i]);
            ++x2[i];
            ++y2[i];
        }
        for (int i = c;i >= 1;i--)
            cover(x1[i],x2[i],y1[i],y2[i],i,i+1);
        int ans = 0;
        for (int i = 1;i <= c;i++)
            if (color[i] == 0)
                ans++;
        printf("%d\n",ans);
    }
}
```

日期函数

```
//日期函数
int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
struct date{
    int year, month, day;
};
//判闰年
inline int leap(int year){
    return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
}
//判合法性
inline int legal(date a){
    if (a.month < 0 || a.month > 12)
        return 0;
    if (a.month == 2)
        return a.day > 0 && a.day <= 28 + leap(a.year);
    return a.day > 0 && a.day <= days[a.month-1];
}
//比较日期大小
```

```
inline int datecmp(date a, date b){
    if (a.year != b.year)
        return a.year - b.year;
    if (a.month != b.month)
        return a.month - b.month;
    return a.day - b.day;
}
//返回指定日期是星期几
int weekday(date a){
    int tm = a.month >= 3 ? a.month - 2 : a.month + 10;
    int ty = a.month >= 3 ? a.year : a.year - 1;
    return (ty+ty/4-ty/100+ty/400+(int)(2.6*tm-0.2)+a.day)%7;
}
//日期转天数偏移
int date2int(date a){
    int ret = a.year * 365 + (a.year-1) / 4-(a.year-1)/100+(a.year-1)/400;
    days[1] += leap(a.year);
    for (int i = 0;i < a.month-1;ret += days[i++]);
    days[1] = 28;
    return ret + a.day;
}
//天数偏移转日期
date int2date(int a){
    date ret;
    ret.year = a/146097*400;
    for (a%=146097;a>=365+leap(ret.year);a-=365+leap(ret.year),ret.year++);
    days[1] += leap(ret.year);
    for (ret.month=1;a>=days[ret.month-1];a-=days[ret.month-1],ret.month++);
    days[1] = 28;
    ret.day = a+1;
    return ret;
}
```

带缓冲读入

```
int readT() {
    char c;int ret;
    while(c=getchar(),c<'0'||c>'9');
    ret=c-'0';
    while(c=getchar(),c>='0'&&c<='9')ret=ret*10+c-'0';
    return ret;
}
char buf[10000],*o;
inline int getint() {
    while (!isdigit(*o) && *o!='-') ++o;
    int r = 0,f = *o=='-'?++o,1:0;
    while (isdigit(*o)) r = r*10+*o++-'0';
    return (f?-r:r);
}

inline int nextInt() {
    char c;c=getchar();
    while(c!='-'&&(c<'0'||c>'9'))c=getchar();
    int n=0,s=1;if(c=='-')s=-1,c=getchar();
    while(c>='0'&&c<='9')n*=10,n+=c-'0',c=getchar();
    return n*s;
}

int A[20],k;
inline void print_int(int x) {
    if(x<0)putchar('-'),x=-x;
    k=0;while(x)A[k++]=x%10,x/=10;
    for(int i=k-1;i>=0;i--)putchar('0'+A[i]);
    putchar('\n');
```

```
}
/*
这个代码提供了一个 getint()的函数来加快读入整数的速度
我们使用 o = gets(buf)，时候就可以进行相关的操作了
在这个函数的基础上稍作改进还可以读入实数等等。
*/
```

精度问题

```
inline int sgn(double a){return a < -eps ? -1 : a < eps ? 0 : 1;}
/*
a == b   sgn(a - b) == 0    a != b   sgn(a - b) != 0
a < b    sgn(a - b) < 0     a <= b   sgn(a - b) <= 0
a > b    sgn(a - b) > 0     a >= b   sgn(a - b) >= 0
3. eps 带来的函数越界
如果 sqrt(a), asin(a), acos(a) 中的 a 是你自己算出来并传进来的，那就得小心了。
如果 a 本来应该是 0 的，由于浮点误差，可能实际是一个绝对值很小的负数(比如 1e-12),这样 sqrt(a)应得 0 的，直接因 a 不在定义域而出错。
类似地，如果 a 本来应该是±1,则 asin(a)、acos(a)也有可能出错。
因此，对于此种函数，必需事先对 a 进行校正。
(int)a;  将 a 靠进 0 取整 (int)-2.2 = -2
现在考虑一种情况,题目要求输出保留两位小数。有个 case 的正确答案的精确值是 0.005,按理应该输出 0.01,
但你的结果可能是 0.005000000001(恭喜)，也有可能是 0.004999999999(悲剧)，
如果按照 printf("%.2lf", a)输出，那你的遭遇将和括号里的字相同。
解决办法是，如果 a 为正，则输出 a+eps，否则输出 a-eps
ceil(a); floor(a); 顾名思义，向上取证、向下取整。需要注意的是，这两个函数都返回 double，而非 int
ICPC 题目输出有个不成文的规定(有时也成文)，不要输出：-0.000
那我们首先要弄清，什么时候按 printf("%.3lf\n", a)输出会出现这个结果。
直接给出结果好了：a∈(-0.000499999……, -0.000……1)
所以,如果你发现 a 落在这个范围内,请直接输出 0.000。更保险的做法是用 sprintf 直接判断输出结果是不是-0.000 再予处理。
关于 set<T>
有时候我们可能会有这种需求 ,对浮点数进行 插入、查询是否插入过 的操作。手写 hash 表是一个方法(hash 函数一样要小心设计)，
但 set 不是更方便吗。但 set 好像是按==来判重的呀？貌似行不通呢。经观察,set 不是通过==来判断相等的，是通过<来进行的，
具体说来
只要 a<b 和 b<a 都不成立，就认为 a 和 b 相等，可以发现，
如果将小于定义成： bool operator < (const Dat dat)const{return val < dat.val - eps;}就可以解决问题了
(基本类型不能重载运算符，所以封装了下)
*/
#pragma comment(linker, "/STACK:16777216")
```

错排公式为 M(n)=n!(1/2!-1/3!+…..+(-1)^n/n!)

JAVA 单关键字排序

```
import java.util.Arrays;
class Node implements Comparable {
   int a;
   Node(int a) {
       this.a = a;
   }
   @Override
   public int compareTo(Object arg0) {
       int a = ((Node)arg0).a;
       if(this.a > a)
           return -1;
       else if(this.a < a)
           return 1;
       return 0;
   }
   @Override
   public String toString() {
       return "" + a;
```

```
    }

}
public class Main {
    public static void main(String[]args) {
        Node[]node=new Node[4];
        node[0]=new Node(9);
        node[1]=new Node(8);
        node[2]=new Node(100);
        node[3]=new Node(0);
        Arrays.sort(node);
        System.out.println(Arrays.toString(node));
     }
}
```

JAVA 多关键字排序

```
import java.util.Arrays;
import java.util.Comparator;
class Person {
    private String name;
    private int age;
    public Person(String name,int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return this.name;
    }
    public int getAge() {
        return this.age;
    }
    @Override
    public String toString() {
        return "" + this.name + " " + this.age;
    }
}
class Cmp1 implements Comparator {
    @Override
    public int compare(Object arg0,Object arg1) {
        Person a = (Person)arg0;
        Person b = (Person)arg1;
        if (a.getName().compareTo(b.getName()) != 0)
            return a.getName().compareTo(b.getName());
        else {
            if (a.getAge() < b.getAge())
                return -1;
            else if (a.getAge() > b.getAge())
                return 1;
            else
                return 0;
        }
    }
}
class Cmp2 implements Comparator {
    @Override
    public int compare(Object arg0,Object arg1) {
        Person a = (Person)arg0;
        Person b = (Person)arg1;
        if (a.getAge() < b.getAge())
            return -1;
        else if (a.getAge() > b.getAge())
            return 1;
        else
```

```
        return a.getName().compareTo(b.getName());
    }
}
public class Main{
    public static void main(String[]args) {
        Person[] p = new Person[4];
        p[0] = new Person("ZZZ",19);
        p[1] = new Person("AAA",109);
        p[2] = new Person("AAA",19);
        p[3] = new Person("YYY",100);
        Arrays.sort(p,new Cmp1());
        System.out.print("Cmp1:");
        System.out.println(Arrays.toString(p));
        Arrays.sort(p,new Cmp2());
        System.out.print("Cmp2:");
        System.out.println(Arrays.toString(p));
    }
}
```

JAVA 开根号

```
import java.util.*;
import java.math.*;
public class Main{
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        BigInteger a;
        int n;
        while (in.hasNext()) {
            n = in.nextInt();
            a = in.nextBigInteger();
            int l = 1, r = 1000000000;
            while (l < r) {
                int mid = (l + r) / 2;
                //System.out.println(mid);
                BigInteger mul = BigInteger.valueOf(1);
                BigInteger t = BigInteger.valueOf(mid);
                for (int m = n; m-- != 0; ) {
                    mul = mul.multiply(t);
                }
                if (mul.equals(a)) {
                    System.out.println(t);
                    break;
                }
                if (mul.equals(mul.max(a))) {
                    r = mid;
                } else {
                    l = mid;
                }
            }
        }
    }
}
```