1 MATHEMATICS

Math Conclusions

```
五边形定理
五边形数 n * (3 * n +- 1) / 2
(1-x)*(1-x^2)*(1-x^3)....=sigma{(-1)^k * x^(n * (3 * n (+-) 1) / 2)}
即f[n] = f[n - 1] + f[n - 2] - f[n - 5] - f[n - 7] + f[n - 12] + f[n - 15] - .....

fibonacci数性质:
f[n] = f[n - 1] + f[n - 2]
f[n + m + 1] = f[n] * f[m] + f[n + 1] * f[m + 1]
gcd(f[n], f[n + 1]) = 1
gcd(f[n], f[n + 2]) = 1
gcd(f[n], f[m]) = f[gcd(n, m)]
f[n+1]*f[n+1]-f[n]*f[n+2] = (-1)^n
sigma{f[i]^2, 1<=i<=n} = f[n]*f[n+1]
sigma{f[i], 0<=i<=n} = f[n+2] - 1
sigma{f[2*i-1],1<=i<=n} = f[2*n]
sigma{f[2*i],1<=i<=n} = f[2*n+1]-1
sigma{(-1)^i*f[i],0<=i<=n} = (-1)^n*(f[n+1]-f[n])+1
f[2*n-1]=f[n]^2-f[n-2]^2
f[2*n+1]=f[n]^2+f[n+1]^2
3*f[n]=f[n+2]+f[n-2]
f[n]=c(n-1,0)+c(n-2,1)+..c(n-1-m,m) (m<=n-1-m)
sigma{f[i]*i,1<=i<=n}=n*f[n+2]-f[n+3]+2

catalan数性质:
凸多边形三角剖分数
简单有序根树的计数
(0,0) 走到 (n,n)经过的点(a,b)满足a<=b的路径数
乘法结合问题
c[n+1] = (4 * n - 2) / (n + 1) * c[n]
c[n] = (2*n)!/(n!)/((n+1)!)

第一类stirling数性质
有正有负, 其绝对值是n个元素的项目分作k个环排列的数量,s[n,k]
(n个人分成k组, 每组再按特定顺序围圈)
s[n][0] = 0, s[1][1] = 1;
s[n+1][k]= = s[n][k - 1] + n * s[n][k]
|s[n][1]| = (n-1)!
s[n][k] = (-1)^(n+k)*|s[n][k]|
s[n][n-1] = -C(n,2)
x*(x-1)*(x-2)..(x-n+1) = sigma{s[n][k] * x ^k}

第二类stirling数性质
n个元素的集定义k个等价类的方法数目(n个人分成k组的方法数)
s[n][n] = s[n][1] = 1
s[n][k] = s[n - 1][k - 1] + k * s[n - 1][k]
s[n][n - 1] = C(n, 2)
s[n][2] = 2^(n-1)-1
s[n][k] = 1/(k!)sigma{(-1)^k-j * C(k, j) * j ^n, 1<=j<=k}

bell数性质
B[n] = sigma{s[n][k], 1<=k<=n}
B[n+1] = simga{C(n,k)*B[k], 0<=k<=n}
B[p+n] = B[n] + B[n + 1] (mod p)
B[p^m+n] = B[n] + B[n+1] (mod p)

多项式性质
f(x)不存在重根<=>gcd(f(x), f '(x))的次数小于1次
多项式gcd可以用来判断两多项式是否有公共根

多项式取模
f[x] = 0 (mod m)
m = m1 * m2 * m3 ... mk
Ti 表示 f[x] = 0 (mod mi)的解数, 则T = T1 * T2 * T3...Tk

数论
a^n % b = a^(n % phi(b) + phi(b)) % b (n >= phi(b))
lucas定理 c(n, m) = c(n % p, m % p) * c(n / p, m / p) % p
lucas函数 满足 f(n, m) = f(n % p, m % p) * f(n / p, m / p) % p, 可以猜测满足

原根
2,4,p^k,2*p^k存在原根, 存在原根则原根数量为phi(phi(n))
验证原根x = phi(n), x = p1^a1*p2^a2..pk^ak
原根满足t ^ (x / pi) != 1 (mod n)

x*x+y*y==n的整数解:
x*x+y*y==n的整数解个数num = 4 * sigma{H(d), d | n}
H(d) =
(1) 奇数 : (-1)^((d-1)/2)
(2) 偶数 : 0

平方和定理:
(1)费马平方和定理:
    奇质数能表示为两个平方数之和的充分必要条件是该素数被4除余1
(2)费马平方和定理的拓展定理:
    正整数能表示为两平方数之和的充要条件是在它的标准分解式中, 形如素因子的指数是偶数
(3)Brahmagupta-Fibonacci identity
    如果两个整数都能表示为两个平方数之和, 则它们的积也能表示为两个平方数之和。
```

公式：$(a^2+b^2)(c^2+d^2) = (ac-bd)^2 + (ad+bc)^2 = (ac+bd)^2 + (ad-bc)^2$

拓展：$(a^2+n*b^2)(c^2+n*d^2) = (ac-n*bd)^2 + n*(ad+bc)^2 = (ac+n*bd)^2 + n(ad-bc)^2$

```
    推论: 如果不能表示为三个数的平方和, 那么也就不能表示为两个数的平方和。
(4)四平方和定理:
    每个正整数都可以表示成四个整数的平方数之和
(5)表为3个数的平方和条件:
    正整数能表示为三个数的平方和的充要条件是不能表示成的形式, 其中和为非负整数。

连分数
连分数(a+(n^0.5)) / b
开始时, i满足, (a+i)/b=floor((a+(n^0.5))/b),之后过程一样
如果不成功, 则可以变换为(ab+((nb^2)^0.5))/(b^2), 之后再来

杨氏矩阵
(1)如果格子(i,j)没有元素, 则它右边和上边的相邻格子也一定没有元素。
(2)如果格子(i,j)有元素a[i][j], 则它右边和上边的相邻格子要么没有元素, 要么有元素且比a
    [i][j]大。
1 ~ n所组成杨氏矩阵的个数可以通过下面的递推式得到:
f[1] = 1; f[2] = 2; f[n] = f[n - 1] + (n - 1) * f[n - 2];

钩子公式:
对于给定形状, 不同的杨氏矩阵的个数为: n!除以每个格子的钩子长度加1的积。
其中钩子长度定义为该格子右边的格子数和它上边的格子数之和。
```

Chinese remainder theorem

```
LL ex_gcd(LL a,LL b,LL &x,LL &y) {
    if (!a) return x = 0, y = 1, b;
    LL g = ex_gcd(b % a, a, x, y);
    LL t = y;
    y = x;
    x = t - (b / a) * y;
    return g;
}

LL china(const vector<LL>& m, const vector<LL>& b) {
    bool flag = false;
    LL x, y, i, d, result, a1, m1, a2, m2;
```

```
    m1 = m[0]; a1 = b[0];
    for(i = 1; i < m.size(); ++i){
        m2 = m[i]; a2 = b[i];
        d = ex_gcd(m1, m2, x, y);
        if((a2 - a1) % d != 0) flag = true;
        result = (x * ((a2 - a1) / d) % m2 + m2) % m2;
        a1 = a1 + m1 * result; //对于求多个方程
        m1 = (m1 * m2) / d;    //lcm(m1,m2)最小公倍数
        a1 = (a1 % m1 + m1) % m1;
    }
    if (flag) return -1;
    else return a1;
}
```

Pollard rho Factorization

```
int miller_rabin(ll n, int k = 10) {
    if (n <= 3) return n > 1;
    while (k--) {
        ll a = rand() % (n - 3) + 2;
        if (qmod(a, n - 1, n) != 1) return 0;
    }
    return 1;
}

ll f(ll x, ll m, ll c) { return (mult(x, x, m) + c) % m; }

ll pollard_rho(ll n) {
    if (!(n & 1)) return 2;
    while (1) {
        ll x = rand() % n, y = x, c = rand() % n, d = 1;
        while (1) {
            x = f(x, n, c);
            y = f(f(y, n, c), n, c);
            d = gcd(y > x ? y - x : x - y, n);
            if (d == n) break;
            if (d > 1) return d;
        }
    }
}

void fac(ll n, vector<ll> &r) {
    if (miller_rabin(n)) {
        if (n != 1) r.push_back(n);
    } else {
        ll d = pollard_rho(n);
        fac(d, r), fac(n / d, r);
    }
}
```

FFT

```
void fft(Comp a[], int n, bool invert){
    for(int i=1,j=0; i<n; i++){
        int bit=n>>1;
        for(; j>=bit; bit>>=1)j-=bit;
        j+=bit;
        if(i<j)swap(a[i],a[j]);
    }
    for(int len=2; len<=n; len<<=1){
        double ang=2*PI/len*(invert?-1:1);
        Comp wlen(cos(ang),sin(ang));
        for(int i=0; i<n; i+=len){
            Comp w(1,0);
            for(int j=0; j<len/2; j++){
                Comp u=a[i+j],v=a[i+j+len/2]*w;
                a[i+j]=u+v; a[i+j+len/2]=u-v;
                w=w*wlen;
            }
        }
    }
    if(invert)for(int i=0; i<n; i++)a[i]=a[i]/n;
}
```

NNT

```
const int MOD[] = {998244353, 995622913, 786433};
const int ROOT[] = {3, 5, 10};
const LL M1 = 397550359381069386LL;
const LL M2 = 596324591238590904LL;
const LL MM = 993874950619660289LL;

LL mul(LL x,LL y,LL z){
    return (x * y - (LL)(x / (long double) z * y + 1e-3) * z + z) % z;
}

class NNT {
    public:
        NNT(int n, int mod, int root);
        void forward(int a[]) {
            work(a, r);
        }
        void reverse(int a[]) {
            work(a, ir);
            for (int i = 0; i < n; ++i) a[i] = 1LL * a[i] * n_rev % mod;
        }
    private:
        int n, p, mod, n_rev;
        vector<int> rb;
        int r[20];
        int ir[20];
        void work(int a[], int* roots);
};

NNT::NNT(int n, int mod, int root) : n(n) , mod(mod), rb(n) , p(0) {
    n_rev = qmod(n, mod - 2, mod);
    while ((1 << p) < n) ++p;
    for(int i = 0; i < n; i++){
        int x = i, y = 0;
        for (int j = 0; j < p; ++j) {
            y = (y << 1) | (x & 1);
            x >>= 1;
        }
        rb[i] = y;
    }
    int inv = qmod(root, mod - 2, mod);
    r[p - 1] = qmod(root, (mod - 1) / (1 << p), mod);
    ir[p - 1] = qmod(inv, (mod - 1) / (1 << p), mod);
    for(int i = p - 2; i >= 0; i--){
        r[i] = 1LL * r[i + 1] * r[i + 1] % mod;
        ir[i] = 1LL * ir[i + 1] * ir[i + 1] % mod;
    }
}

void NNT::work(int a[], int* r) {
    for (int i = 0; i < n; ++i) if (rb[i] > i) swap(a[i], a[rb[i]]);
    for (int len = 2; len <= n; len <<= 1) {
        int root = *r++;
        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; ++j) {
                int u = a[i + j];
                int v = 1LL * a[i + j + len / 2] * w % mod;
                a[i + j] = u + v < mod ? u + v : u + v - mod;
                a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;
                w = 1LL * w * root % mod;
            }
        }
```

```
    }
}

int merge(int a, int b, int c){
    int ret;
    long long m1 = china(a, b);
    int m2 = c;
    int z = 1LL * ((m2 - m1) % MOD[2]) * iv % MOD[2];
    z = (z % MOD[2] + MOD[2]) % MOD[2];
    ret = (1LL * z * MOD[0] % MO * MOD[1] + m1) % MO;
    return (ret % MO + MO) % MO;
}
```

FWT

```
// xor:
// FWT: a[i] = x + y; a[i + len] = x - y;
// unFWT: a[i] = (x + y) / 2; a[i + len] = (x - y) / 2;

// and:
// FWT: a[i] = x + y; a[i + len] = y;
// unFWT: a[i] = x - y; a[i + len] = y;

// or:
// FWT: a[i] = x; a[i + len] = y + x;
// unFWT: a[i] = x; a[i + len] = y - x;

void FWT(LL a[], int l, int r){
  if (l == r - 1) return;
  int len = (r - l) >> 1;
  int mid = l + len;
  FWT(a, l, mid);
  FWT(a, mid, r);
  for(int i = l; i < mid; i++){
    LL x = a[i], y = a[i + len];
    a[i] = x + y;
    a[i + len] = x - y;
  }
}

void unFWT(LL a[], int l, int r){
  if (l == r - 1) return;
  int len = (r - l) >> 1;
  int mid = l + len;
  for(int i = l; i < mid; i++){
    LL x = a[i], y = a[i + len];
    a[i] = (x + y) / 2;
    a[i + len] = (x - y) / 2;
  }
  unFWT(a, l, mid);
  unFWT(a, mid, r);
}
```

Jacobi Symbol

```
int Jacobi(int a, int n) {
    if (a == 0) return 0;
    if (a == 1) return 1;
    int s, n1, a1 = a, e = 0;
    while (!(a1 & 1)) a1 >>= 1, ++e;
    if (!(e & 1)) s = 1;
    else {
        int u = n % 8;
        if (u == 1 || u == 7) s = 1;
        else s = -1;
    }
    if (n % 4 == 3 && a1 % 4 == 3) s = -s;
    n1 = n % a1;
    if (a1 == 1) return s;
    return s * Jacobi(n1, a1);
}
```

Newton Polynomial

```
int n, q;
ll fs[N][N], xs[N], ys[N];

ll f(ll x) {
    ll y = ys[0];
    ll p = 1;
    for (int i = 1; i < n; i++) {
        p = modular(p * (x - xs[i - 1]), R);
        y = modular(y + p * fs[0][i], R);
    }
    return y;
}

int main() {
    scanf("%d %d", &n, &q);
    for (int i = 0; i < n; i++)
        scanf("%lld %lld", &xs[i], &ys[i]);
    for (int i = 0; i < n; i++)
        fs[i][i] = ys[i];
    for (int d = 1; d < n; d++)
        for (int i = 0, j = d; j < n; i++, j++)
            fs[i][j] = modular(fs[i+1][j] - fs[i][j-1], R) * invert(d, R) % R;
    while (q--) {
        ll x; scanf("%lld", &x);
        printf("%lld\n", f(x));
    }
    return 0;
}
```

Nth Element from $\Sigma a_i x^i$

```
// calc the Nth element from f[0] ~ f[m] (f[x] = sigma(ai * x ^ i, 0 <= i <= m);
int calc(LL n, int m, int f[]){
    static int pre[N], suf[N];
    if (n <= m) return f[n];
    pre[0] = n % MOD;
    for(int i = 1; i <= m; i++) pre[i] = 1LL * pre[i-1] * ((n-i)%MOD) % MOD;
    suf[m] = (n - m) % MOD;
    for(int i = m - 1; i >= 0; i--) suf[i] = 1LL * suf[i+1] * ((n-i)%MOD) % MOD;
    int ret = 0;
    int now = (m & 1) ? -1 : 1;
    for(int i = 0; i <= m; i++){
        int tmp = 1LL * now * f[i] * inv[i] % MOD * inv[m - i] % MOD;
        if (i) tmp = 1LL * tmp * pre[i - 1] % MOD;
        if (i < m) tmp = 1LL * tmp * suf[i + 1] % MOD;
        ret = (0LL + ret + tmp) % MOD;
        now = -now;
    }
    return ret;
}

// calc 0! ~ m! and 1/0! ~ 1/m!
void init(){
    fac[0] = 1;
    for(int i = 1; i < N; i++) fac[i] = 1LL * fac[i - 1] * i % MOD;
    inv[N - 1] = qmod(fac[N - 1], MOD - 2);
    for(int i = N - 2; i >= 0; i--){
        inv[i] = 1LL * inv[i + 1] * (i + 1) % MOD;
    }
}
```

Pell Equation

```
// x * x - D * y * y = 1, xn+yn*sqrt(d) = (x0+y0*sqrt(d))^n
```

```
// x * x - D * y * y = -1, xn+yn*sqrt(d) = (x0+y0*sqrt(d))^(2*n+1)
//
// x * x - D * y * y = -1, D为质数, 有解即D!=3(mod 4)
// 当D==0(mod 4)时, 无解
//
// a * x * x - b * y * y = c
// get x0, y0 from x * x - a * b * y = 1
// get x1, y1 from a * x * x - b * y * y = c
// [xk] = [x0, by0] ^ k-1 * [x1]
// [yk] = [ay0, x0]         [y1]
bool pell( int D, int& x, int& y ) {
    int sqrtD = sqrt(D + 0.0);
    if( sqrtD * sqrtD == D ) return false;
    int c = sqrtD, q = D - c * c, a = (c + sqrtD) / q;
    int step = 0;
    int X[] = { 1, sqrtD };
    int Y[] = { 0, 1 };
    while( true ) {
        X[step] = a * X[step^1] + X[step];
        Y[step] = a * Y[step^1] + Y[step];
        c = a * q - c;
        q = (D - c * c) / q;
        a = (c + sqrtD) / q;
        step ^= 1;
        if( c == sqrtD && q == 1 && step ) {
            x = X[0], y = Y[0];
            return true;
        }
    }
}

// pell x*x-d*y*y = -1
struct Matrix{
    int n, m;
    LL v[2][2];
}c, tmp, ans;

Matrix operator*(const Matrix &a, const Matrix &b){
    c.n = a.n, c.m = b.m;
    for(int i = 0; i < c.n; i++){
        for(int j = 0; j < c.m; j++){
            c.v[i][j] = 0;
            for(int k = 0; k < a.m; k++){
                c.v[i][j] = (c.v[i][j] + 1LL * a.v[i][k] * b.v[k][j]);
            }
        }
    }
    return c;
}

int n, l, base, a[N];

bool build(int n){
    base = 0;
    while(base * base <= n) base++;
    base--;
    if (base * base == n) return false;
    int k = base;
    int n_k = n - k * k;
    l = 0;
    a[l++] = k;
    while(true){
        int i1 = n_k - k % n_k;
        i1 += ((base - i1) / n_k) * n_k;
        a[l++] = (i1 + k) / n_k;
        if (a[l - 1] == 2 * base) break;
        k = i1;
        n_k = (n - k * k) / n_k;
    }
    return true;
}

void solve(){
    ans.n = 2, ans.m = 2;
    ans.v[0][0] = a[0], ans.v[0][1] = 1;
    ans.v[1][0] = 1, ans.v[1][1] = 0;
    for(int i = 1; i < l - 1; i++){
        tmp.n = 2, tmp.m = 2;
        tmp.v[0][0] = a[i], tmp.v[0][1] = 1;
        tmp.v[1][0] = 1, tmp.v[1][1] = 0;
        ans = ans * tmp;
    }
    if (ans.v[0][0] * ans.v[0][0] - n * ans.v[1][0] * ans.v[1][0] == -1){
        printf("%d %d\n", ans.v[0][0], ans.v[1][0]);
    }else{
        puts("No Solution");
    }
}
```

Prime Count

```
const int N = 2 * 5000000;
LL n, m;
int tot, prime[N];
bool vis[N];
int v[N];
unordered_map<int, LL> s;

int main(){
    tot = 0;
    int tmp;
    for(int i = 2; i < N; i++){
        if (!vis[i]) prime[tot++] = i;
        for(int j = 0; j < tot; j++){
            tmp = i * prime[j];
            if (tmp >= N) break;
            vis[tmp] = true;
            if (i % prime[j] == 0) break;
        }
    }
    while(~scanf("%lld", &n)){
        int i;
        for(i = 1; i * i <= n; i++) v[i] = n / i;
        i--;
        while(v[i]) { i++; v[i] = v[i - 1] - 1; }
        int len = i;
        s.clear();
        for(int i = 1; i < len; i++){
            s[v[i]] = 1LL * v[i] * (v[i] + 1) / 2 - 1;
            //s[v[i]] = v[i] - 1;
        }
        for(int i = 0; prime[i] * prime[i] <= n; i++){
            int p = prime[i];
            LL sp = s[p - 1];
            int p2 = 1LL * p * p;
            for(int i = 1; i < len; i++){
                if (v[i] < p2) break;
                //s[v[i]] -= 1 * (s[v[i]/p] - sp);
                s[v[i]] -= p * (s[v[i]/p] - sp);
            }
        }
        printf("%lld\n", s[n]);
    }
}
```

Baby-step Giant-step

```
// get the number of x
// a ^ x = d (mod p) where x >= 0 && x <= m
int log_mod(int a, int b, int p, int &len){
    a %= p, b %= p;
    if (a == 0){
        if (b == 0) return len = 1, 0;
        return -1;
    }
    int m = ceil(sqrt(p)), iv = qmod(invert(a, p), m, p), now = 1;
    unordered_map<int, int> dict;
    dict[1] = 0, len = -1;
    for(int i = 1; i < m; i++) {
        now = 1LL * now * a % p;
        if (now == 1 && len == -1) len = i;
        if (!dict.count(now)) dict[now] = i;
    }
    int ans = -1;
    for(int i = 0; i <= m; i++, b = 1LL * b * iv % p){
        if (!dict.count(b)) continue;
        ans = i * m + dict[b];
        break;
    }
    b = iv;
    for(int i = 1; i <= m; i++, b = 1LL * b * iv % p){
        if (!(dict.count(b) && len == -1)) continue;
        len = i * m + dict[b];
        break;
    }
    return ans;
}

ULL work(){
    int cnt = 0, now = 1;
    for(;;cnt++){
        int g = gcd(a, p);
        if (g == 1){
            int iv = invert(now, p), len, pos = log_mod(a, 1LL*d*iv%p, p, len);
            if (pos < 0 || pos + cnt > m) return 0;
            return (ULL)1 + (m - pos - cnt) / len;
        }
        if (now % p == d) return 1;
        if (d % g) return 0;
        d /= g, p /= g;
        now = 1LL * now * (a / g) % p;
    }
}
```

Continue Fraction

```
void solve(){
    int len = l - 1;
    now.n = 2, now.m = 2;
    now.v[0][0] = now.v[1][1] = 1;
    now.v[0][1] = now.v[1][0] = 0;
    for(int i = 1; i < l; i++){
        tmp.n = 2, tmp.m = 2;
        tmp.v[0][0] = a[i], tmp.v[0][1] = 1;
        tmp.v[1][0] = 1, tmp.v[1][1] = 0;
        now = now * tmp;
    }
    ans.n = 2, ans.m = 2;
    ans.v[0][0] = a[0], ans.v[0][1] = 1;
    ans.v[1][0] = 1, ans.v[1][1] = 0;
    int t = m / len;
    while(t){
        if (t & 1) ans = ans * now;
        now = now * now;
        t >>= 1;
    }
    t = m % len;
    for(int i = 0; i < t; i++){
        tmp.n = 2, tmp.m = 2;
        tmp.v[0][0] = a[i + 1], tmp.v[0][1] = 1;
        tmp.v[1][0] = 1, tmp.v[1][1] = 0;
        ans = ans * tmp;
    }
    printf("%d/%d\n", ans.v[0][0], ans.v[1][0]);
}
```

2 GRAPH/TREE

Bridge/Cutvertex-Finding

```
void tarjan(int u, int fa)
{
    dfn[u] = low[u] = ++stamp;
    int ch = 0;
    for (int v: e[u]) {
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (u ? low[v] >= dfn[u] : ++ch > 1) cut[u] = true;
            if (low[v] > dfn[u]) bridge.emplace_back(u, v);
        } else if (v != fa) {
            low[u] = min(low[u], dfn[v]);
        }
    }
}
```

Strongly Connected Components

```
void tarjan(int u)
{
    dfn[u] = low[u] = ++stamp;
    sta[top++] = u; ins[u] = true;
    for (int v: e[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (ins[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        int v;
        do {
            v = sta[--top];
            ins[v] = false;
            scc[v] = cnt;
        } while (v != u);
        ++cnt;
    }
}
```

Lowest Common Ancestor

```
void tarjan(int u)
{
    anc[u] = u; vis[u] = 1;
    for (int v: e[u]) {
        if (!vis[v]) {
            tarjan(v);
            join(u, v);
            anc[find(u)] = u;
        }
    }
    vis[u] = 2;
    for (auto i: q[u]) if (vis[i.v] == 2) lca[i.id] = anc[find(i.v)];
}
```

Maximum Flow

```
int esz, psz, s, t;
int h[MAXV], vh[MAXV + 1];

int aug(int u, int m)
{
    if (u == t) return m;
    int d = m;
    for (edge *i = e[u]; i; i = i->next) {
        if (i->u && h[u] == h[i->t] + 1) {
            int f = aug(i->t, min(i->u, d));
            i->u -= f, i->pair->u += f, d -= f;
            if (h[s] == esz || !d) return m - d;
        }
    }
    int w = d < m ? min(esz, h[u] + 2) : esz;
    for (edge *i = e[u]; i; i = i->next) {
        if (i->u) w = min(w, h[i->t] + 1);
    }
    ++vh[w];
    --vh[h[u]] ? h[u] = w : h[s] = esz;
    return m - d;
}

void maxflow()
{
    flow = 0;
    memset(h, 0, sizeof(h));
    memset(vh, 0, sizeof(vh));
    vh[0] = esz;
    while (h[s] != esz) flow += aug(s, INT_MAX);
}
```

网络流模型变换

无源无汇上下界可行流:
  (1) 建立附加源s和汇t，添加t->s容量为无穷大
  (2) 对u->v，下界b上界c，拆成3条：(s, v, b)，(u, v, c-b)，(u, t, b)
  (3) 对每个点i，合并下界流量：(s, i, Σb(u, i))，(i, v, Σb(i, v))
  (4) 求s-t最大流，当且仅当所有附加弧满载时原网络有可行流

有源有汇上下界最大/最小流:
  先用上述做法求可行流，然后用传统的s-t增广路算法即可得到最大流
  把t看成源，s看成汇求t-s最大流就是s-t最小流
  注意，原先每条弧u->v的反向弧容量为0，而在有容量上下界情形下，应等于下界

Minimum Cost Maximum Flow

```
int psz, s, t;
int cost, dist, d[MAXV];
bool vis[MAXV];

int aug(int u, int m)
{
    if (u == t) return cost += dist * m, m;
    int d = m; vis[u] = true;
    for (edge *i = e[u]; i; i = i->next) {
        if (i->u && !i->c && !vis[i->t]) {
            int f = aug(i->t, min(d, i->u));
            i->u -= f, i->pair->u +=f, d -= f;
            if (!d) return m;
        }
    }
    return m - d;
}

bool modlabel()
{
    deque<int> q;
    memset(vis, 0, sizeof(vis));
    memset(d, 0x3f, sizeof(d));
    q.push_back(s); d[s] = 0; vis[s] = true;
    while (!q.empty()) {
        int u = q.front(); q.pop_front(); vis[u] = false;
        for (edge *i = e[u]; i; i = i->next) {
            int v = i->t;
            if (i->u && d[u] + i->c < d[v]) {
                d[v] = d[u] + i->c;
                if (vis[v]) continue;
                vis[v] = true;
                if (q.size() && d[v] < d[q[0]]) q.push_front(v);
                else q.push_back(v);
            }
        }
    }
    for (edge *i = epool; i < epool + psz; ++i) {
        i->c -= d[i->t] - d[i->pair->t];
    }
    dist += d[t];
    return d[t] < inf;
}

void costflow()
{
    cost = dist = 0;
    while (modlabel()) {
        do memset(vis, 0, sizeof(vis));
        while (aug(s, INT_MAX));
    }
}
```

Minimum Cost Maximum Flow (Cycle Canceling)

```
int psz, s, t;
int d[MAXV];
bool vis[MAXV];
edge *fa[MAXV];

void cancelcycle(int u)
{
    int i = u;
    do {
        --fa[i]->u, ++fa[i]->pair->u, cost += fa[i]->c;
        i = fa[i]->pair->t;
    } while (i != u);
}

bool aug(int u)
{
    vis[u] = true;
    for (edge *i = e[u]; i; i = i->next) {
        int v = i->t;
        if (i->u && d[u] + i->c < d[v]) {
            d[v] = d[u] + i->c;
            fa[v] = i;
            if (vis[v]) cancelcycle(v);
            if (vis[v] || aug(v)) return true;
        }
    }
    vis[u] = false;
    return false;
}

void costflow()
{
    cost = 0;
```

```
    for (;;) {
        memset(d, 0, sizeof(d));
        memset(vis, 0, sizeof(vis));
        bool flag = false;
        for (int i = 0; i < esz; ++i) {
            if (aug(i)) { flag = true; break; }
        }
        if (!flag) return;
    }
}

/*
Initialize:
addedge(t, s, inf, -inf);
CAUTION: maybe OVERFLOW
*/
```

Maximum Bipartite Matching

```
int n, m;
bool g[MAXN][MAXM];
int match[MAXM];
bool v[MAXN];

bool dfs(int i)
{
    for (int j = 0; j < m; ++j) {
        if (g[i][j] && !v[j]) {
            v[j] = true;
            if (match[j] < 0 || dfs(match[j])) {
                match[j] = i;
                return true;
            }
        }
    }
    return false;
}

int hungarian()
{
    int c = 0;
    memset(match, -1, sizeof(match));
    for (int i = 0; i < n; ++i) {
        memset(v, 0, sizeof(v));
        if (dfs(i)) ++c;
    }
    return c;
}
```

Maximum Weight Perfect Biparite Matching

```
int n;
int w[MAXN][MAXN], lx[MAXN], ly[MAXN], match[MAXN], slack[MAXN];
bool vx[MAXN], vy[MAXN];

bool dfs(int i)
{
    vx[i] = true;
    for (int j = 0; j < n; ++j) {
        if (lx[i] + ly[j] > w[i][j]) {
            slack[j] = min(slack[j], lx[i] + ly[j] - w[i][j]);
        } else if (!vy[j]) {
            vy[j] = true;
            if (match[j] < 0 || dfs(match[j])) {
                match[j] = i;
                return true;
            }
        }
    }
    return false;
}

void km()
{
    memset(match, -1, sizeof(match));
    memset(ly, 0, sizeof(ly));
    for (int i = 0; i < n; ++i) lx[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) {
        for (;;) {
            memset(vx, 0, sizeof(vx));
            memset(vy, 0, sizeof(vy));
            memset(slack, 0x3f, sizeof(slack));
            if (dfs(i)) break;
            int d = inf;
            for (int i = 0; i < n; ++i) {
                if (!vy[i]) d = min(d, slack[i]);
            }
            for (int i = 0; i < n; ++i) {
                if (vx[i]) lx[i] -= d;
                if (vy[i]) ly[i] += d;
            }
        }
    }
}
```

Maximum Matching on General Graph

```
int n;
int next[MAXN], match[MAXN], v[MAXN], f[MAXN];
int que[MAXN], head, tail;

int find(int p) { return f[p] < 0 ? p : f[p] = find(f[p]); }

void join(int x, int y)
{
    x = find(x); y = find(y);
    if (x != y) f[x] = y;
}

int lca(int x, int y)
{
    static int v[MAXN], stamp = 0;
    ++stamp;
    for (;;) {
        if (x >= 0) {
            x = find(x);
            if (v[x] == stamp) return x;
            v[x] = stamp;
            if (match[x] >= 0) x = next[match[x]];
            else x = -1;
        }
        swap(x, y);
    }
}

void group(int a, int p)
{
    while (a != p) {
        int b = match[a], c = next[b];
        if (find(c) != p) next[c] = b;
        if (v[b] == 2) v[que[tail++] = b] = 1;
        if (v[c] == 2) v[que[tail++] = c] = 1;
        join(a, b); join(b, c);
        a = c;
    }
}

void aug(int s)
```

```
{
    memset(v, 0, sizeof(v));
    memset(f, -1, sizeof(f));
    memset(next, -1, sizeof(next));
    que[0] = s; head = 0; tail = 1; v[s] = 1;
    while (head < tail && match[s] < 0) {
        int x = que[head++];
        for (edge *i = e[x]; i; i = i->next) {
            int y = i->t;
            if (match[x] == y || v[y] == 2 || find(x) == find(y)) {
                continue;
            } else if (v[y] == 1) {
                int p = lca(x, y);
                if (find(x) != p) next[x] = y;
                if (find(y) != p) next[y] = x;
                group(x, p);
                group(y, p);
            } else if (match[y] < 0) {
                next[y] = x;
                while (~y) {
                    int z = next[y];
                    int p = match[z];
                    match[y] = z; match[z] = y;
                    y = p;
                }
                break;
            } else {
                next[y] = x;
                v[que[tail++] = match[y]] = 1;
                v[y] = 2;
            }
        }
    }
}

void blossom()
{
    memset(match, -1, sizeof(match));
    for (int i = 0; i < n; ++i) {
        if (match[i] < 0) aug(i);
    }
}
```

Maximum Weight Perfect Matching on General Graph

```
int n;
int w[MAXN][MAXN];
int match[MAXN], p[MAXN], d[MAXN];
int path[MAXN], len;
bool v[MAXN];
const int inf = 0x3f3f3f3f;

bool dfs(int i)
{
    path[len++] = i;
    if (v[i]) return true;
    v[i] = true;
    for (int j = 0; j < n; ++j) {
        if (i != j && match[i] != j && !v[j]) {
            int k = match[j];
            if (d[k] < d[i] + w[i][j] - w[j][k]) {
                d[k] = d[i] + w[i][j] - w[j][k];
                if (dfs(k)) return true;
            }
        }
    }
    --len;
    v[i] = false;
    return false;
}

int matching()
{
    for (int i = 0; i < n; ++i) p[i] = i, match[i] = i^1;
    int cnt = 0;
    for (;;) {
        len = 0;
        bool flag = false;
        memset(d, 0, sizeof(d));
        memset(v, 0, sizeof(v));
        for (int i = 0; i < n; ++i) {
            if (dfs(p[i])) {
                flag = true;
                int t = match[path[len - 1]], j = len - 2;
                while (path[j] != path[len - 1]) {
                    match[t] = path[j];
                    swap(t, match[path[j]]);
                    --j;
                }
                match[t] = path[j];
                match[path[j]] = t;
                break;
            }
        }
        if (!flag) {
            if (++cnt >= 3) break;
            random_shuffle(p, p + n);
        }
    }
}
```

2-SAT

```
struct TwoSAT {
    int n;
    vector<int> G[maxn*2];
    bool mark[maxn*2];
    int S[maxn*2], c;

    bool dfs(int x) {
        if (mark[x^1]) return false;
        if (mark[x]) return true;
        mark[x] = true;
        S[c++] = x;
        for (int i = 0; i < G[x].size(); i++)
            if (!dfs(G[x][i])) return false;
        return true;
    }

    void init(int n) {
        this->n = n;
        for (int i = 0; i < n*2; i++) G[i].clear();
        memset(mark, 0, sizeof(mark));
    }

    // x = xval or y = yval
    void add_clause(int x, int xval, int y, int yval) {
        x = x * 2 + xval;
        y = y * 2 + yval;
        G[x^1].push_back(y);
        G[y^1].push_back(x);
    }

    bool solve() {
        for(int i = 0; i < n*2; i += 2)
            if(!mark[i] && !mark[i+1]) {
                c = 0;
                if(!dfs(i)) {
```

```
                    while(c > 0) mark[S[--c]] = false;
                    if(!dfs(i+1)) return false;
                }
            }
        return true;
    }
};
```

Divide and Conquer for Tree

```
int getsize(int u, int fa)
{
    size[u] = 1;
    for (edge *i = e[u]; i; i = i->next) {
        if (i->t != fa) size[u] += getsize(i->t, u);
    }
    return size[u];
}

int divide(int u)
{
    for (edge *i = e[u]; i; i = i->next) {
        if (size[i->t] > size[u] / 2) {
            size[u] -= size[i->t], size[i->t] += size[u];
            return divide(i->t);
        }
    }
    return u;
}

void solve(int u)
{
    u = divide(u);
    size[u] = 0; // delete
    for (edge *i = e[u]; i; i = i->next) {
        if (size[i->t]) {
            dfs1(i->t, u); // calculate answer
            dfs2(i->t, u); // update
        }
    }
    // calculate answer with root
    for (edge *i = e[u]; i; i = i->next) {
        if (size[i->t]) solve(i->t);
    }
}
```

Heavy-Light Decomposition

```
int fa[MAXN], dep[MAXN], size[MAXN], hson[MAXN], top[MAXN], dfn[MAXN], stamp;

void dfs1(int u)
{
    size[u] = 1, hson[u] = 0;
    for (edge *i = e[p]; i; i = i->next) {
        int v = i->t;
        if (v == fa[u]) continue;
        fa[v] = u;
        dep[v] = dep[u] + 1;
        dfs1(v);
        size[u] += size[v];
        if (!hson[u] || size[v] > size[hson[u]]) hson[u] = v;
    }
}

void dfs2(int u, int anc)
{
    dfn[u] = stamp++;
    top[u] = anc;
    if (hson[u]) dfs2(hson[u], anc);
    for (edge *i = e[p]; i; i = i->next) {
        int v = i->t;
        if (v != fa[u] && v != hson[u]) dfs2(v, v);
    }
}

int lca(int u, int v)
{
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        // query(dfn[top[u]], dfn[u])
        u = fa[top[u]];
    }
    if (dep[u] > dep[v]) swap(u, v);
    // query(dfn[u], dfn[v]) -- include LCA
    // if (u != v) query(dfn[u] + 1, dfn[v]) -- exclude LCA
    return u;
}
```

Virtual Tree

```
bool cmp(int i, int j) { return dfn[i] < dfn[j]; }

int vtree(int h[], int m, int T[], int fa[])
{
    static int sta[MAXN];
    int tot = 0, top = 0;
    sort(h, h + m, cmp);
    sta[top++] = 0; // d[0] == -1
    for (int i = 0; i < m; ++i) {
        if (top <= 1) {
            sta[top++] = h[i];
            fa[h[i]] = 0;
        } else {
            int g = lca(h[i], sta[top - 1]);
            while (d[sta[top - 1]] > d[g]) {
                --top;
                if (d[sta[top - 1]] <= d[g]) fa[sta[top]] = g;
            }
            if (sta[top - 1] != g) {
                T[tot++] = g;
                fa[g] = sta[top - 1];
                sta[top++] = g;
            }
            fa[h[i]] = g;
            sta[top++] = h[i];
        }
        T[tot++] = h[i];
    }
    sort(T, T + tot, cmp);
    return tot;
}
// return the number of nodes in virtual tree
// T[] -- nodes in vtree, fa[] -- father in vtree
```

Degree Limited MST

```
// Find a minimum spanning tree whose vertex 1 has a degree limit D
struct Tedge { int v, w, next; }edge[MAXM * 2], mst_edge[MAXM * 2];
int first[MAXN], mst_first[MAXN], dist[MAXN], heap[MAXN], pos[MAXN], maxw[MAXN],
     path[MAXN], prev[MAXN];
bool used[MAXN];
int N, M, D, cnt, num, ans;

inline void add_edge(Tedge& e, int& first, int i, int v, int w) {
    e.v = v; e.w = w; e.next = first; first = i;
}

void init() {
```

```
    memset(first, -1, sizeof(first));
    scanf("%d%d%d", &N, &M, &D);
    for (int i = 0; i < M; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u; --v;
        add_edge(edge[i * 2], first[u], i * 2, v, w);
        add_edge(edge[i * 2 + 1], first[v], i * 2 + 1, u, w);
    }
}

inline void moveup(int i) {
    int key = heap[i];
    while (i > 1 && dist[heap[i >> 1]] > dist[key])    heap[i] = heap[i >> 1],
         pos[heap[i]] = i, i >>= 1;
    heap[i] = key; pos[key] = i;
}

inline void movedown(int i) {
    int key = heap[i];
    while ((i << 1) <= num) {
        int j = i << 1;
        if (j < num && dist[heap[j + 1]] < dist[heap[j]]) ++j;
        if (dist[key] <= dist[heap[j]]) break;
        heap[i] = heap[j]; pos[heap[i]] = i; i = j;
    }
    heap[i] = key; pos[key] = i;
}

void Prim(int u) {
    int minw = INF, s;
    num = 0;
    while (1) {
        used[u] = 1;
        for (int i = first[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v, w = edge[i].w;
            if (!used[v] && (dist[v] == -1 || w < dist[v])) {
                dist[v] = w;
                prev[v] = u;
                if (pos[v] == -1) pos[v] = ++num, heap[num] = v;
                moveup(pos[v]);
            }
            else if (used[v] && v == 0 && w < minw) minw = w, s = i;
        }
        if (!num) break;
        u = heap[1]; heap[1] = heap[num--]; movedown(1);
        ans += dist[u];
        add_edge(mst_edge[cnt], mst_first[u], cnt, prev[u], dist[u]); ++cnt;
        add_edge(mst_edge[cnt], mst_first[prev[u]], cnt, u, dist[u]); ++cnt;
    }
    if (minw == INF) return;
    edge[s].w = -1; edge[s ^ 1].w = -1;
    s = edge[s ^ 1].v; ans += minw; --D;
    add_edge(mst_edge[cnt], mst_first[0], cnt, s, minw); ++cnt;
    add_edge(mst_edge[cnt], mst_first[s], cnt, 0, minw); ++cnt;
}

void DFS(int u) {
    used[u] = 1;
    for (int i = mst_first[u]; i != -1; i = mst_edge[i].next) {
        int v = mst_edge[i].v, w = mst_edge[i].w;
        if (w > -1 && !used[v]) {
            if (w > maxw[v]) maxw[v] = w, path[v] = i;
            if (maxw[u] > maxw[v]) maxw[v] = maxw[u], path[v] = path[u];
            DFS(v);
        }
    }
}

void work() {
    ans = cnt = 0;
    memset(mst_first, -1, sizeof(mst_first));
    memset(dist, -1, sizeof(dist));
    memset(pos, -1, sizeof(pos));
    memset(used, 0, sizeof(used));
    used[0] = 1;
    for (int i = first[0]; i != -1; i = edge[i].next)
        if (!used[edge[i].v]) Prim(edge[i].v);
    if (D < 0) {
        printf("NONE\n");
        return;
    }
    for (int i = 1; i < N; ++i)
        if (!used[i]) {
            printf("NONE\n");
            return;
        }
    memset(maxw, -1, sizeof(maxw));
    memset(used, 0, sizeof(used));
    used[0] = 1;
    for (int i = mst_first[0]; i != -1; i = mst_edge[i].next) DFS(mst_edge[i].v);
    for (int i = 0; i < D; ++i) {
        int minw = INF, s, x, y;
        for (int j = first[0]; j != -1; j = edge[j].next) {
            int v = edge[j].v, w = edge[j].w;
            if (w > -1 && maxw[v] > -1 && w - maxw[v] < minw) {
                minw = w - maxw[v]; s = v;
                x = path[v]; y = j;
            }
        }
        if (minw >= 0) break;
        ans += minw;
        mst_edge[x].w = mst_edge[x ^ 1].w = -1;
        add_edge(mst_edge[cnt], mst_first[0], cnt, s, edge[y].w); ++cnt;
        add_edge(mst_edge[cnt], mst_first[s], cnt, 0, edge[y].w); ++cnt;
        edge[y].w = edge[y ^ 1].w = -1;
        memset(used, 0, sizeof(used));
        used[0] = 1;
        for (int u = 0; u < N; ++u)
            if (path[u] == x) maxw[u] = -1;
        DFS(s);
    }

    printf("%d\n", ans);
}
```

Minimum Directed Spanning Tree

```
int n;
int w[maxn][maxn]; // 边权
int vis[maxn];     // 访问标记，仅用来判断无解
int ans;           // 计算答案
int removed[maxn]; // 每个点是否被删除
int cid[maxn];     // 所在圈编号
int pre[maxn];     // 最小入边的起点
int iw[maxn];      // 最小入边的权值
int max_cid;       // 最大圈编号

// 从s出发能到达多少个结点
int dfs(int s) {
    vis[s] = 1;
    int ans = 1;
    for(int i = 0; i < n; i++)
        if(!vis[i] && w[s][i] < INF) ans += dfs(i);
    return ans;
}
```

```
// 从u出发沿着pre指针找圈
bool cycle(int u) {
    max_cid++;
    int v = u;
    while(cid[v] != max_cid) { cid[v] = max_cid; v = pre[v]; }
    return v == u;
}

// 计算u的最小入弧，入弧起点不得在圈c中
void update(int u) {
    iw[u] = INF;
    for(int i = 0; i < n; i++)
        if(!removed[i] && w[i][u] < iw[u]) {
            iw[u] = w[i][u];
            pre[u] = i;
        }
}

// 根结点为s，如果失败则返回false
bool solve(int s) {
    memset(vis, 0, sizeof(vis));
    if(dfs(s) != n) return false;

    memset(removed, 0, sizeof(removed));
    memset(cid, 0, sizeof(cid));
    for(int u = 0; u < n; u++) update(u);
    pre[s] = s; iw[s] = 0; // 根结点特殊处理
    ans = max_cid = 0;
    for(;;) {
        bool have_cycle = false;
        for(int u = 0; u < n; u++) if(u != s && !removed[u] && cycle(u)){
            have_cycle = true;
            // 以下代码缩圈，圈上除了u之外的结点均删除
            int v = u;
            do {
                if(v != u) removed[v] = 1;
                ans += iw[v];
                // 对于圈外点i，把边i->v改成i->u（并调整权值）；v->i改为u->i
                // 注意圈上可能还有一个v'使得i->v'或者v'->i存在，因此只保留权值最
                    小的i->u和u->i
                for(int i = 0; i < n; i++) if(cid[i] != cid[u] && !removed[i]) {
                    if(w[i][v] < INF) w[i][u] = min(w[i][u], w[i][v]-iw[v]);
                    w[u][i] = min(w[u][i], w[v][i]);
                    if(pre[i] == v) pre[i] = u;
                }
                v = pre[v];
            } while(v != u);
            update(u);
            break;
        }
        if(!have_cycle) break;
    }
    for(int i = 0; i < n; i++)
        if(!removed[i]) ans += iw[i];
    return true;
}
```

Stoer-Wagner Minimum Cut

```
const int MAXN = 501, MAXV = 0x3f3f3f3f;
int res,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++) {
            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) {
                res = min(res,dis[v[maxp]]);
                for (int j = 0; j < n; j++) {
                    mat[v[prev]][v[j]] += mat[v[j]][v[maxp]];
                    mat[v[j]][v[prev]] = mat[v[prev]][v[j]];
                }
                v[maxp] = v[--n];
            }
            vis[v[maxp]] = true;  prev = maxp; maxp = -1;
            for (int j = 1;j < n;j++) if (!vis[v[j]]) {
                    dis[v[j]] += mat[v[prev]][v[j]];
                    if (maxp == -1 || dis[v[maxp]] < dis[v[j]]) maxp = j;
            }
        }
    }
    return res;
}

void init() {
    scanf("%d%d", &n, &m);
    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;
    }
}
```

3 DATA STRUCTURES

RMQ

```
int rmq[MAXN][LOGN];

void initRMQ(int a[], int n)
{
    for (int i = 0; i < n; ++i) rmq[i][0] = a[i];
    for (int j = 1; (1<<j) <= n; ++j) {
        for (int i = 0; i + (1<<j) <= n; ++i) {
            rmq[i][j] = min(rmq[i][j-1], rmq[i+(1<<(j-1))][j-1]);
        }
    }
}

int RMQ(int l, int r) // query [l, r]
{
    int k = sizeof(int) * 8 - __builtin_clz(r - l + 1) - 1;
    return min(rmq[l][k], rmq[r-(1<<k)+1][k]);
}
```

Segment Tree

```
int n, h;
S T[MAXN * 2]; // val
int d[MAXN * 2]; // lazy flag

void push(int p)
{
    for (int s = h, k = 1<<(h-1); s; --s, k >>= 1) {
        int i = p >> s;
        if (d[i]) {
            apply(i<<1,   k, d[i]);
            apply(i<<1|1, k, d[i]);
```

```
            d[i] = 0;
        }
    }
}

S query(int l, int r)
{
    S L, R;
    push(l += n), push(r += n);
    for (; l <= r; l >>= 1, r >>= 1) {
        if ( l&1) L = merge(L, T[l++]);
        if (~r&1) R = merge(T[r--], R);
    }
    return merge(L, R);
}

void modify(int l, int r, int x)
{
    bool cl = false, cr = false;
    push(l += n), push(r += n);
    for (int k = 1; l <= r; l >>= 1, r >>= 1, k <<= 1) {
        if (cl) update(l - 1);
        if (cr) update(r + 1);
        if ( l&1) apply(l++, k, x), cl = true;
        if (~r&1) apply(r--, k, x), cr = true;
    }
    for (--l, ++r; r; l >>= 1, r >>= 1) {
        if (cl) update(l);
        if (cr && (!cl || l != r)) update(r);
    }
}
// h = sizeof(int) * 8 - __builtin_clz(n);
```

Treap

```
struct node {
    int k, w;
    node *l, *r;
};

void split(node *t, int k, node *&l, node *&r)
{
    if (!t) { l = r = 0; return; }
    if (k <= t->k) {
        r = t, split(t->l, k, l, t->l);
    } else {
        l = t, split(t->r, k, t->r, r);
    }
}

node *merge(node *l, node *r)
{
    if (!l) return r;
    if (!r) return l;
    if (l->w > r->w) {
        l->r = merge(l->r, r); return l;
    } else {
        r->l = merge(l, r->l); return r;
    }
}
```

Splay

```
struct node_t *null, *root;
struct node_t {
    node_t *ch[2], *fa;
    int size;

    int dir() { return fa->ch[0] == this ? 0 : 1; }
    void setc(node_t *c, int d) { ch[d] = c; if (c != null) c->fa = this; }
    void update() { size = ch[0]->size + ch[1]->size + 1; }
    void sink() {}

    void rot()
    {
        node_t *p = fa;
        int d = dir();
        if (p->fa == null) fa = null, root = this;
        else p->fa->setc(this, p->dir());
        p->setc(ch[d^1], d), setc(p, d^1);
        p->update(), update();
    }

    void splay(node_t *header = null)
    {
        for (; fa != header; rot()) {
            if (fa->fa != header) {
                if (dir() == fa->dir()) fa->rot();
                else rot();
            }
        }
    }

    node_t *select(int k)
    {
        node_t *t = this;
        while (t->sink(), k != t->ch[0]->size + 1) {
            if (k <= t->ch[0]->size) t = t->ch[0];
            else k -= t->ch[0]->size + 1, t = t->ch[1];
        }
        t->splay(fa);
        return t;
    }

    node_t *select(int l, int r)
    {
        return select(r + 1)->ch[0]->select(l - 1)->ch[1];
    }
}node[MAXN];
```

Link-cut Tree

```
struct node_t {
    node_t *ch[2], *fa;
    int val, mx;
    bool rev;

    bool isroot() { return !fa || (fa->ch[0] != this && fa->ch[1] != this); }
    int dir() { return fa->ch[0] == this ? 0 : 1; }
    void setc(node_t *c, int d) { ch[d] = c; if (c) c->fa = this; }
    void reverse() { rev ^= 1; swap(ch[0], ch[1]); }

    void init(int v)
    {
        ch[0] = ch[1] = fa = 0;
        rev = false;
        val = mx = v;
    }

    void update()
    {
        mx = val;
        if (ch[0]) mx = max(mx, ch[0]->mx);
        if (ch[1]) mx = max(mx, ch[1]->mx);
    }

    void sink()
    {
```

```
        if (rev) {
            if (ch[0]) ch[0]->reverse();
            if (ch[1]) ch[1]->reverse();
            rev = 0;
        }
    }

    void rot()
    {
        node_t *p = fa;
        int d = dir();
        if (p->isroot()) fa = p->fa;
        else p->fa->setc(this, p->dir());
        p->setc(ch[d^1], d), setc(p, d^1);
        p->update(), update();
    }

    void sinkdown() { if (!isroot()) fa->sinkdown(); sink(); }

    void splay()
    {
        sinkdown();
        for (; !isroot(); rot()) {
            if (!fa->isroot()) {
                if (dir() == fa->dir()) fa->rot();
                else rot();
            }
        }
    }

    node_t *expose()
    {
        node_t *u = 0, *t = this;
        for (; t; u = t, t = t->fa) {
            t->splay();
            t->ch[1] = u;
            t->update();
        }
        return u;
    }

    node_t *root()
    {
        node_t *t = expose();
        while (t->sink(), t->ch[0]) t = t->ch[0];
        return t;
    }

    void setroot() { expose()->reverse(); }

    void link(node_t *p)
    {
        setroot(); // for un-rooted tree
        expose()->fa = p;
    }

    void cut(node_t *p)
    {
        p->setroot(); // for un-rooted tree
        expose();
        splay();
        ch[0] = ch[0]->fa = 0;
        update();
    }

    int query(node_t *t) { t->setroot(); return expose()->mx; }

    int query(node_t *t) // without setroot
    {
        expose();
        t = t->expose(); // lca
        int ret = t->val; // analysis lca
        if (t->ch[1]) ret = max(ret, t->ch[1]->mx); // lca -> v
        if (t != this) {
            splay();
            ret = max(ret, mx); // lca -> u
        }
        return ret;
    }
}node[MAXN];
```

Euler Tour Tree

```
struct node_t {
    // splay tree ...

    node_t *walkdown(int d)
    {
        node_t *t = this;
        while (t->ch[d] != null) t = t->ch[d];
        return t;
    }

    node_t *adj(int d) // 0 -- prev, 1 -- succ
    {
        if (ch[d] != null) return ch[d]->walkdown(d^1);
        node_t *t = this;
        while (t->dir() == d) t = t->fa;
        return t->fa;
    }
}node[MAXN * 2]; // each node split into 2 nodes. i --> (i<<1) && (i<<1)^1

void cut(int t)
{
    node_t *x = node[t<<1].adj(0), *y = node[t<<1^1].adj(1);
    x->splay(), y->splay(x);
    y->ch[0]->fa = null, y->setc(null, 0);
    y->update(), x->update();
}

void link(int t, int p) // link subtree t to p
{
    node_t *x = &node[p<<1], *y = &node[t<<1];
    x->splay(), x->adj(1)->splay(x), y->splay();
    x->ch[1]->setc(y, 0);
    x->ch[1]->update(), x->update();
}
```

Leftist Tree

```
int n;
int key[MAXN], left[MAXN], right[MAXN], dist[MAXN];
// dist[0] = -1

int merge(int a, int b)
{
    if (!a) return b;
    if (!b) return a;
    if (key[b] > key[a]) swap(a, b);
    right[a] = merge(right[a], b);
    if (dist[left[a]] < dist[right[a]]) {
        swap(left[a], right[a]);
    }
    dist[a] = dist[right[a]] + 1;
    return a;
}
```

4 STRINGOLOGY

KMP Algorithm

```
void getf(const char *s, int f[])
{
    int n = strlen(s);
    f[0] = 0, f[1] = 0;
    for (int i = 1; i < n; ++i) {
        int j = f[i];
        while (j && s[i] != s[j]) j = f[j];
        f[i + 1] = s[i] == s[j] ? j + 1 : 0;
    }
}

int match(const char *s, const char *p, int f[])
{
    int n = strlen(s), m = strlen(p), j = 0;
    for (int i = 0; i < n; ++i) {
        while (j && s[i] != p[j]) j = f[j];
        if (s[i] == p[j]) ++j;
        if (j == m) return i - m + 1;
    }
}
```

Extend-KMP Algorithm

```
void getf(const char *s, int f[])
{
    int n = strlen(s), j = 0, k = 1;
    while (j + 1 < n && s[j] == s[j + 1]) ++j;
    f[0] = n, f[1] = j;
    for (int i = 2; i < n; ++i) {
        int len = k + f[k] - 1, t = f[i - k];
        if (i + t <= len) {
            f[i] = t;
        } else {
            j = max(0, len - i + 1);
            while (i + j < n && s[i + j] == s[j]) ++j;
            f[i] = j; k = i;
        }
    }
}

void match(const char *s, const char *p, int f[], int ex[])
{
    int n = strlen(s), j = 0, k = 0;
    while (j < n && s[j] == p[j]) ++j;
    ex[0] = j;
    for (int i = 1; i < n; ++i) {
        int len = k + ex[k] - 1, t = f[i - k];
        if (i + t <= len) {
            ex[i] =  t;
        } else {
            j = max(0, len - i + 1);
            while (i + j < n && s[i + j] == p[j]) ++j;
            ex[i] = j; k = i;
        }
    }
}
```

Aho-Corasick Automation

```
const int PSZ = MAXN * LEN;
struct trie {
    trie *ch[SIGMA], *f; // trie *last;
    int val;
}pool[PSZ], *dict;
int psz;
int head, tail;
trie *que[PSZ];

void insert(trie *t, const char *s)
{
    for (; *s; ++s) {
        int c = *s - 'a';
        if (!t->ch[c]) memset(t->ch[c] = pool + psz++, 0, sizeof(trie));
        t = t->ch[c];
    }
    ++t->val;
}

void build_fail(trie *t)
{
    head = tail = 0;
    for  (int i = 0; i < SIGMA; ++i) {
        if (t->ch[i]) (que[tail++] = t->ch[i])->f= t;
        else t->ch[i] = t;
    }
    while (head < tail) {
        t = que[head++];
        t->val += t->f->val;                          // 重复计数
        t->last = t->f->val ? t->f : t->f->last;  // 不重复计数
        for (int i = 0; i < SIGMA; ++i) {
            if (t->ch[i]) (que[tail++] = t->ch[i])->f = t->f->ch[i];
            else t->ch[i] = t->f->ch[i];
        }
    }
}

int find(trie *t, const char *s)
{
    int sum = 0;
    for (; *s; ++s) {
        int c = *s - 'a';
        t = t->ch[c];
        sum += i->val;                                          // 重复计数
        for (trie *i = t; i && i->val != -1; i = i->last) { // 不重复计数
            sum += i->val, i->val = -1; // -1为访问标记
        }
    }
    return sum;
}
```

Suffix Array

```
int sa[MAXN], rank[MAXN], height[MAXN], c[MAXN], wx[MAXN], wy[MAXN];

void build_sa(int m)
{
    int *x = wx, *y = wy;
    for (int i = 0; i < m; ++i) c[i] = 0;
    for (int i = 0; i < n; ++i) ++c[x[i] = s[i]];
    for (int i = 1; i < m; ++i) c[i] += c[i - 1];
    for (int i = n - 1; i >= 0; --i) sa[--c[x[i]]] = i;
    for (int k = 1; k <= n; k <<= 1) {
        int p = 0;
        for (int i = n - k; i < n; ++i) y[p++] = i;
        for (int i = 0; i < n; ++i) if (sa[i] >= k) y[p++] = sa[i] - k;
        for (int i = 0; i < m; ++i) c[i] = 0;
        for (int i = 0; i < n; ++i) ++c[x[y[i]]];
        for (int i = 1; i < m; ++i) c[i] += c[i - 1];
        for (int i = n - 1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
        swap(x, y);
        p = 1; x[sa[0]] = 0;
        for (int i = 1; i < n; ++i) {
            x[sa[i]] = y[sa[i - 1]] == y[sa[i]] &&
                y[sa[i - 1] + k] == y[sa[i] + k] ?
```

```
                p - 1 : p++;
        }
        if (p == n) break;
        m = p;
    }
}

void build_height()
{
    for (int i = 0; i < n; ++i) rank[sa[i]] = i;
    for (int i = 0, k = 0; i < n; ++i) {
        if (k) --k;
        if (!rank[i]) continue;
        int j = sa[rank[i] - 1];
        while (s[i + k] == s[j + k]) ++k;
        height[rank[i]] = k;
    }
}
// height[i] == lcp(suffix(sa[i-1]), suffix(sa[i]))
// REMEBER: add '$' after the string
```

Suffix Automation

```
struct sam {
    int l;
    sam *f, *ch[SIGMA];
}pool[LEN * 2], *root, *tail;
int psz;

sam *init_node(sam *p)
{
    memset(p->ch, 0, sizeof(p->ch));
    p->f = 0, p->l = 0;
    return p;
}

void sam_add(int v)
{
    sam *p = init_node(pool + psz++), *i;
    p->l = tail->l + 1;
    for (i = tail; i && !i->ch[v]; i = i->f) i->ch[v] = p;
    if (!i) {
        p->f = root;
    } else if (i->ch[v]->l == i->l + 1) {
        p->f = i->ch[v];
    } else {
        sam *q = pool + psz++, *r = i->ch[v];
        *q = *r;
        q->l = i->l + 1;
        p->f = r->f = q;
        for (; i && i->ch[v] == r; i = i->f) i->ch[v] = q;
    }
    tail = p;
}

int match(sam *root, const char *s)
{
    int k = 0, ret = 0;
    sam *p = root;
    for (; *s; ++s) {
        int c = *s - 'a';
        if (p->ch[c]) {
            ++k, p = p->ch[c];
        } else {
            while (p && !p->ch[c]) p = p->f;
            if (p) k = p->l + 1, p = p->ch[c];
            else p = root; k = 0;
        }
        ret = max(ret, k);
        // p->match = max(p->match, k);
    }
    return ret;
}
```

Longest Palindorme Substring

```
void manacher(const char *s, int len[])
// len[i]: longest palindrome center at i/2
{
    n = strlen(s);
    for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0)) {
        while (i - j >= 0 && i + j + 1 < n * 2
            && s[(i - j) / 2] == s[(i + j + 1) / 2]) ++j;
        len[i] = j;
        for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; ++k) {
            len[i + k] = min(len[i - k], j - k);
        }
    }
}
```

Palindromic Tree

```
int n;
char s[MAXN];
int ans[MAXN][2];
struct node {
    int len, diff;
    node *ch[26], *fail, *sfail;
    int dp[2];

    int get_min(int n, int i)
    {
        dp[i] = ans[n - sfail->len - diff][i];
        if (diff == fail->diff) dp[i] = min(dp[i], fail->dp[i]);
        return dp[i] + 1;
    }
}pool[MAXN], *root, *last;
int psz;

node *newnode(int len)
{
    node *t = pool + psz++;
    t->len = len, t->diff = 0;
    t->fail = t->sfail = 0;
    memset(t->ch, 0, sizeof(t->ch));
    return t;
}

node *find(node *t, int i)
{
    while (i <= t->len || s[i - t->len - 1] != s[i]) t = t->fail;
    return t;
}

bool add(int i)
{
    int c = s[i] - 'a';
    node *t = find(last, i);
    if (t->ch[c]) { last = t->ch[c]; return false; }
    last = t->ch[c] = newnode(t->len + 2);
    last->fail = t->fail ? find(t->fail, i)->ch[c] : root;
    last->diff = last->len - last->fail->len;
    last->sfail = last->diff != last->fail->diff ? last->fail : last->fail->sfail
        ;
    return true;
}

void dp() // palindromes factorizations
```

```
{
    scanf("%s", s + 1);
    n = strlen(s + 1);
    psz = 0;
    node *t0 = newnode(-1), *t1 = newnode(0);
    t0->dp[0] = t1->dp[1] = MAXN;
    t1->fail = t0;
    root = last = t1;
    ans[0][1] = MAXN;
    for (int i = 1; i <= n; ++i) {
        add(i);
        ans[i][0] = ans[i][1] = MAXN; // 0 - even factor, 1 - odd factor
        for (node *t = last; t->len > 0; t = t->sfail) {
            ans[i][0] = min(ans[i][0], t->get_min(i, 1));
            ans[i][1] = min(ans[i][1], t->get_min(i, 0));
        }
    }
}
```

Minimum Representation

```
int minrep(char *s)
{
    int n = strlen(s), i = 0, j = 1, k = 0, t;
    while (i < n && j < n && k < n) {
        t = s[(i + k) % n] - s[(j + k) % n];
        if (!t) { ++k; continue; }
        if (t > 0) i += k + 1; else j += k + 1;
        if (i == j) ++j;
        k = 0;
    }
    return min(i, j);
}
```

5 GEOMETRY

Geometry Conclusions

椭圆面积：πab
球冠面积：$2\pi Rh$，体积：$\pi h^2(R-h/3)$或$\pi h(3r^2+h^2)/6$，R为球半径，h为球冠高，r为底面半径
圆台与棱台体积：$(S_1+S_2+\sqrt{S_1S_2})h/3$，$S_1,S_2$为两底面面积，$h$为高
扇形重心：角平分线上距离圆心$4r\sin(\alpha/2)/(3\alpha)$
圆锥重心：底面圆心与顶点连线上离顶点$0.75h$

旋转体与线积分：
$S=2\pi\int_a^b y\sqrt{1+y'^2}dx$
$V=\pi\int_a^b y^2dx$
$L=\int_a^b f(x,y(x))\sqrt{1+y'^2}dx$

正四面体顶点座标：$(1,1,1),(1,-1,-1),(-1,1,-1),(-1,-1,1)$
正六面体顶点座标：$(\pm1,\pm1,\pm1)$
正八面体顶点座标：$(\pm1,0,0),(0,\pm1,0),(0,0,\pm1)$
正十二面体顶点座标：$(0,\pm1/\phi,\pm\phi),(\pm1/\phi,\pm\phi,0),(\pm\phi,0,\pm1/\phi),(\pm1,\pm1,\pm1)$，$\phi=(1+\sqrt{5})/2$
正二十面体顶点座标：$(0,\pm1,\pm\phi),(\pm1/,\pm\phi,0),(\pm\phi,0,\pm1)$，$\phi=(1+\sqrt{5})/2$

反演：给定点O，常数k，点P的变换对应以O开始的射线上的一点P'使得$|OP||OP'|=k^2$
反演的结果：
(1) 过O的直线：直线
(2) 过O的圆：不过O的直线
(3) 过O的球：不过O的平面
反演圆的半径：$r'=(1/(OC-r)-1/(OC+r))r^2/2$

Geometry

```
vec   proj(vec v, vec n)    { return n * dot(v, n) / norm(n); }
vec   reflect(vec v, vec n) { return proj(v, n) * 2.0 - v;    }
point proj(point p, line ln)
{ return ln.s + proj(p - ln.s, dir(ln)); }
point reflect(point p, line ln)
{ return ln.s + reflect(p - ln.s, dir(ln)); }

vec    rotate(vec v, double a) { return v * polar(1.0, a); }
double angle(vec a, vec b)     { return arg(b / a);        }
```

Line

```
double dis(point p, line ln) { return fabs(cross(p, ln.s, ln.t)) / len(ln); }

bool onseg(point p, line ln)
{ return dcmp(cross(p, ln.s, ln.t)) == 0 && dcmp(dot(p, ln.s, ln.t)) <= 0; }

double dtoseg(point p, line ln)
{
    if (dcmp(dot(ln.s, ln.t, p)) <= 0) return dis(p, ln.s);
    if (dcmp(dot(ln.t, ln.s, p)) <= 0) return dis(p, ln.t);
    return dis(p, ln);
}

bool inter(line a, line b, point &p)
{
    double s1 = cross(a.s, a.t, b.s);
    double s2 = cross(a.s, a.t, b.t);
    if (!dcmp(s1 - s2)) return false;
    p = (s1 * b.t - s2 * b.s) / (s1 - s2);
    return true;
}

bool seginter(line a, line b, point &p) // segment intersection(strict)
{
    double s1 = cross(a.s, a.t, b.s);
    double s2 = cross(a.s, a.t, b.t);
    if ((dcmp(s1) ^ dcmp(s2)) != -2) return false;
    double s3 = cross(b.s, b.t, a.s);
    double s4 = cross(b.s, b.t, a.t);
    if ((dcmp(s3) ^ dcmp(s4)) != -2) return false;
    p = (s1 * b.t - s2 * b.s) / (s1 - s2);
    return true;
}
```

Triangle

```
double area(double a, double b, double c) // Heron's Formula
{
    double p = (a + b + c) * 0.5;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}

double angle(double a, double b, double c) // Law of Cosines
{
    return acos((sqr(a) + sqr(b) - sqr(c)) / (2 * a * b));
}

point center(point A, point B, point C) // Circumcenter
{
    double d1 = dot(A, B, C), d2 = dot(B, C, A), d3 = dot(C, A, B);
    double c1 = d2 * d3, c2 = d1 * d3, c3 = d1 * d2, c = c1 + c2 + c3;
    if (!dcmp(c)) return A; // coincident
    return ((c2 + c3) * A + (c1 + c3) * B + (c1 + c2) * C) / (2 * c);
}

point incenter(point A, point B, point C)
{
    double a = abs(B - C), b = abs(C - A), c = abs(A - B);
    if (!dcmp(a + b + c)) return A; // coincident
```

```
    return (a * A + b * B + c * C) / (a + b + c);
}

point centroid(point A, point B, point C)
{
    return (A + B + C) / 3;
}

point orthocenter(point A, point B, point C)
{
    double d1 = dot(A, B, C), d2 = dot(B, C, A), d3 = dot(C, A, B);
    double c1 = d2 * d3, c2 = d1 * d3, c3 = d1 * d2, c = c1 + c2 + c3;
    if (!dcmp(c)) return A; // coincident
    return (c1 * A + c2 * B + c3 * C) / c;
}

point fermat(point A, point B, point C)
{
    double a = abs(B - C), b = abs(C - A), c = abs(A - B);
    if (dot(A, B, C) / b / c < -0.5) return A;
    if (dot(B, C, A) / c / a < -0.5) return B;
    if (dot(C, A, B) / a / b < -0.5) return C;
    if (cross(A, B, C) < 0) swap(B, C);
    point CC = (B - A) * polar(1.0, -pi / 3) + A;
    point BB = (C - A) * polar(1.0,  pi / 3) + A;
    return inter(line(B, BB), line(C, CC));
}
```

Circle

```
bool inter(circle c, line ln, point &p1, point &p2)
{
    point p = proj(c.c, ln);
    double d = dis(p, c.c);
    if (dcmp(d - c.r) > 0) return false;
    vec v = sqrt(c.r * c.r - d * d) * unit(dir(ln));
    p1 = p - v; p2 = p + v;
    return true;
}

bool inter(circle c, line ln, double &a1, double &a2)
{
    point p = proj(c.c, ln);
    double d = dis(p, c.c);
    if (dcmp(d - c.r) > 0) return false;
    double a = arg(p - c.c), b = acos(d / c.r);
    a1 = remainder(a - b, 2 * pi), a2 = remainder(a + b, 2 * pi);
    return true;
}

bool inter(circle a, circle b, point &p1, point &p2)
{
    double d = dis(a.c, b.c);
    if (dcmp(d - (a.r + b.r)) > 0) return false;
    if (!dcmp(d) || dcmp(d - fabs(a.r - b.r)) < 0) return false;
    double d1 = (sqr(d) + sqr(a.r) - sqr(b.r)) / (2 * d), d2 = d - d1;
    point p = (d1 * b.c + d2 * a.c) / d;
    vec v = sqrt(sqr(a.r) - sqr(d1)) * unit(normal(b.c - a.c));
    p1 = p - v; p2 = p + v;
    return true;
}

bool inter(circle a, circle b, double &a1, double &a2)
{
    double d = dis(a.c, b.c);
    if (dcmp(d - (a.r + b.r)) > 0) return false;
    if (!dcmp(d) || dcmp(d - fabs(a.r - b.r)) < 0) return false;
    double a = arg(b.c - a.c), b = angle(a.r, d, b.r);
    a1 = remainder(a - b, 2 * pi), a2 = remainder(a + b, 2 * pi);
    return true;
}

bool tan(circle c, point p, point &p1, point &p2)
{
    double d = dis(p, c.c);
    if (dcmp(d - c.r) < 0) return false;
    double d1 = c.r * c.r / d, d2 = d - d1;
    point p0 = (d1 * p + d2 * c.c) / d;
    vec v = sqrt(sqr(c.r) - sqr(d1)) * unit(normal(p - c.c));
    p1 = p0 - v; p2 = p0 + v;
    return true;
}

bool tan(circle c, point p, double &a1, double &a2)
{
    double d = dis(p, c.c);
    if (dcmp(d - c.r) < 0) return false;
    double a = arg(p - c.c), b = acos(c.r / d);
    a1 = remainder(a - b, 2 * pi), a2 = remainder(a + b, 2 * pi);
    return true;
}

bool outertan(circle a, circle b, double &a1, double &a2)
{
    double d = dis(a.c, b.c);
    if (!dcmp(d) || dcmp(d - fabs(a.r - b.r)) < 0) return false;
    double a = arg(b.c - a.c), b = acos((a.r - b.r) / d);
    a1 = remainder(a - b, 2 * pi), a2 = remainder(a + b, 2 * pi);
    return true;
}

bool innertan(circle a, circle b, double &a1, double &a2)
{
    double d = dis(a.c, b.c);
    if (!dcmp(d) || dcmp(d - (a.r + b.r)) < 0) return false;
    double a = arg(b.c - a.c), b = acos((a.r + b.r) / d);
    a1 = remainder(a - b, 2 * pi), a2 = remainder(a + b, 2 * pi);
    return true;
}
```

Point in Polygon Problem

```
bool inpoly(point a, point *p, int n)
{
    int wn = 0;
    for (int i = 0; i < n; ++i) {
        point p1 = p[i], p2 = p[(i + 1) % n];
        int d = dcmp(cross(a, p1, p2));
        if (!s && dot(a, p1, p2) <= 0) return true;
        int d1 = dcmp(p1.Y - a.Y);
        int d2 = dcmp(p2.Y - a.Y);
        if (d > 0 && d1 <= 0 && d2 > 0) ++wn;
        if (d < 0 && d2 <= 0 && d1 > 0) --wn;
    }
    return wn != 0;
}
```

Convex Hull

```
bool cmpx(point a, point b) { return dcmp(a.X - b.X) ? a.X < b.X : a.Y < b.Y; }

int graham(point p[], int n, point h[])
{
    int m = 0;
    sort(p, p + n, cmpx);
    for (int i = 0; i < n; ++i) {
        while (m > 1 && dcmp(cross(h[m - 2], h[m - 1], p[i])) <= 0) --m;
        h[m++] = p[i];
```

```
    }
    int k = m;
    for (int i = n - 2; i >= 0; --i) {
        while (m > k && dcmp(cross(h[m - 2], h[m - 1], p[i])) <= 0) --m;
        h[m++] = p[i];
    }
    if (n > 1) --m;
    return m;
}
```

Dynamic Convex Hull

```
struct cmpx {
    bool operator()(const point &a, point &b) { return dcmp(a.X - b.X) < 0; }
}
set<point, cmpx> lower, upper;

double insert(set<point, cmpx> &h, point p)
{
    double s = 0;
    set<point, cmpx>::iterator it = h.lower_bound(p);
    if (it != h.end() && !dcmp(p.X - it->X)) {
        if (dcmp(p.Y - it->Y) >= 0) return 0;
        if (it != h.begin())     s += cross(p, *it, *prev(it));
        if (next(it) != h.end()) s += cross(p, *next(it), *it);
        h.erase(it);
    } else if (it != h.begin() && it != h.end()) {
        double ds = cross(p, *it, *prev(it));
        if (dcmp(ds) <= 0) return 0;
        s += ds;
    }
    it = h.insert(p).first;
    while (it != h.begin() && prev(it) != h.begin()) {
        double ds = cross(p, *prev(it), *prev(prev(it)));
        if (dcmp(ds) < 0) break;
        h.erase(prev(it));
        s += ds;
    }
    while (next(it) != h.end() && next(next(it)) != h.end()) {
        double = cross(p, *next(it), *next(next(it)));
        if (dcmp(ds) > 0) break;
        h.erase(next(it));
        s -= ds;
    }
    return s * 0.5;
}

double insert(point p) // return area increment
{
    double s = 0;
    if (lower.size()) {
        s += max(0.0, cross(p, *lower.begin(), conj(*upper.begin())));
        s += max(0.0, cross(p, conj(*upper.rbegin()), *lower.rbegin()));
    }
    s += insert(lower, p);
    s += insert(upper, conj(p));
    return s;
}
```

Half-plane Intersection

```
bool inhp(point p, line hp) { return dcmp(cross(hp.s, hp.t, p)) >= 0; }

bool cmpang(line a, line b)
{ return dcmp(a.a - b.a) ? a.a < b.a : cross(a.s, a.t, b.s) < 0; }

int hpinter(line q[], int n, point h[])
{
    // line q[i] represent the half-plane on its left
    int head = 0, tail = 0, m = 0;
    for (int i = 0; i < n; ++i) q[i].a = arg(dir(q[i]));
    sort(q, q + n, cmpang);
    for (int i = 1; i < n; ++i) {
        if (!dcmp(q[i].a - q[i - 1].a)) continue;
        while (head < tail && !inhp(h[tail - 1], q[i])) --tail;
        while (head < tail && !inhp(h[head], q[i])) ++head;
        q[++tail] = q[i];
        if (head < tail) h[tail - 1] = inter(q[tail - 1], q[tail]);
    }
    while (head < tail && !inhp(h[tail - 1], q[head])) --tail;
    if (head < tail) h[tail] = inter(q[tail], q[head]);
    for (int i = head; i <= tail; ++i) h[m++] = h[i];
    return m;
}

line makehp(double a, double b, double c) // ax + by + c > 0
{
    point p1 = fabs(a) > fabs(b) ? point(-c / a, o) : point(0, -c / b);
    point p2 = p1 + vec(b, -a);
    return line(p1, p2);
}
```

Closest Pair

```
bool cmpx(point a, point b) { return a.X < b.X; }
bool cmpy(point a, point b) { return a.Y < b.Y; }

double mindis(point p[], int l, int r)
{
    static point t[MAXN];
    if (r - l <= 1) return inf;
    int mid = (l + r) >> 1, m = 0;
    double x = p[mid].X;
    double d = min(mindis(l, mid), mindis(mid, r));
    inplace_merge(p + l, p + mid, p + r, cmpy());
    for (int i = l; i < r; ++i) {
        if (fabs(x - p[i].X) < d) t[m++] = p[i];
    }
    for (int i = 0; i < m; ++i) {
        for (int j = i + 1; j < m; ++j) {
            if (t[j].Y - t[i].Y >= d) break;
            d = min(d, abs(t[i] - t[j]));
        }
    }
    return d;
}
double mindis() { sort(p, p + n, cmpx); return mindis(0, n); }
```

Farthest Pair

```
double maxdis(point *p, int n)
{
    int m = graham(p, n, h);
    if (m == 2) return abs(h[0] - h[1]);
    h[m] = h[0];
    double d = 0;
    for (int i = 0, j = 1; i < m; ++i) {
        while (dcmp(cross(h[i + 1] - h[i], h[j + 1] - h[j])) > 0) {
            j = (j + 1) % m;
        }
        d = max(d, abs(h[i] - h[j]));
    }
    return d;
}
```

Minimum Distance Between Convec Hull

```
void mindis(point *p1, int n, point *p2, int m)
```

```
{
    int i = 0, j = 0;
    for (int k = 1; k < n; ++k) if (cmpx(p1[k], p1[i])) i = k;
    for (int k = 1; k < m; ++k) if (cmpx(p2[j], p2[k])) j = k;
    for (int t = 0; t < n + m; ++t) {
        if (dcmp(cross(p1[i + 1] - p1[i], p2[j + 1] - p2[j])) < 0) {
            ans = min(ans, dtoseg(p2[j], line(p1[i], p1[i + 1])));
            i = (i + 1) % n;
        } else {
            ans = min(ans, dtoseg(p1[i], line(p2[j], p2[j + 1])));
            j = (j + 1) % m;
        }
    }
}
```

Union Area of a Circle and a Polygon

```
double area(circle c, point a, point b)
{
    a -= c.c; b -= c.c;
    if (zero(a) || zero(b)) return 0;
    double s1 = .5 * arg(b / a) * sqr(c.r);
    double s2 = .5 * cross(a, b);
    return fabs(s1) < fabs(s2) ? s1 : s2;
}

double unionarea(circle c, point p[], int n)
{
    double s = 0;
    for (int i = 0; i < n; ++i) {
        point A = p[i], B = p[(i + 1) % n], p1, p2;
        line AB = line(A, B);
        if (inter(c, AB, p1, p2) && (onseg(p1, AB) || onseg(p2, AB))) {
            s += area(c, A, p1) + area(c, p1, p2) + area(c, p2, B);
        } else {
            s += area(c, A, B);
        }
    }
    return fabs(s);
}
```

Union Area of Polygons

```
double pos(point p, line ln) { return dot(p - ln.s, dir(ln)) / norm(dir(ln)); }

void unionarea(vector<point> p[], int n, double tot[])
{
    for (int i = 0; i <= n; ++i) tot[i] = 0;
    for (int i = 0; i < n; ++i)
    for (int ii = 0; ii < p[i].size(); ++ii) {
        point A = p[i][ii], B = p[i][(ii + 1) % p[i].size()];
        line AB = line(A, B);
        vector<pair<double, int> > c;
        for (int j = 0; j < n; ++j) if (i != j)
        for (int jj = 0; jj < p[j].size(); ++jj) {
            point C = p[j][jj], D = p[j][(jj + 1) % p[j].size()];
            line CD = line(C, D);
            int f1 = dcmp(cross(A, B, C));
            int f2 = dcmp(cross(A, B, D));
            if (!f1 && !f2) {
                if (i < j && dcmp(dot(dir(AB), dir(CD))) > 0) {
                    c.push_back(make_pair(pos(C, AB),  1));
                    c.push_back(make_pair(pos(D, AB), -1));
                }
                continue;
            }
            double s1 = cross(C, D, A);
            double s2 = cross(C, D, B);
            double t = s1 / (s1 - s2);
            if (f1 >= 0 && f2 < 0) c.push_back(make_pair(t,  1));
            if (f1 < 0 && f2 >= 0) c.push_back(make_pair(t, -1));
        }
        c.push_back(make_pair(0., 0));
        c.push_back(make_pair(1., 0));
        sort(c.begin(), c.end());
        double s = .5 * cross(A, B), z = min(max(c[0].s, 0.), 1.);
        for (int j = 1, k = c[0].second; j < c.size(); ++j) {
            double w = min(max(c[j].first, 0.), 1.);
            tot[k] += s * (w - z);
            k += c[j].second;
            z = w;
        }
    }
}
```

Union Area of Circles

```
bool same(circle a, circle b)  { return zero(a.c - b.c) && !dcmp(a.r - b.r);  }
bool incir(circle a, circle b) { return dcmp(dis(a.c, b.c) + a.r - b.r) <= 0; }

void unionarea(circle c[], int n, double tot[])
{
    static pair<double, int> a[MAXN * 2];
    for (int i = 0; i <= n; ++i) tot[i] = 0;
    for (int i = 0; i < n; ++i) {
        int m = 0, k = 0;
        for (int j = 0; j < n; ++j) if (i != j) {
            double a1, a2;
            if (same(c[i], c[j]) && i < j) continue;
            if (incir(c[i], c[j])) { ++k; continue; }
            if (!inter(c[i], c[j], a1, a2)) continue;
            a[m++] = make_pair(a1,  1);
            a[m++] = make_pair(a2, -1);
            if (a1 > a2) ++k;
        }
        sort(a, a + m);
        double a1 = a[m - 1].first - 2 * pi, a2, rad;
        for (int j = 0; j < m; ++j) {
            a2 = a[j].first, rad = a2 - a1;
            tot[k] += .5 * sqr(c[i].r) * (rad - sin(rad));
            tot[k] += .5 * cross(c[i].p(a1), c[i].p(a2));
            k += a[j].second;
            a1 = a2;
        }
        if (!m) tot[k] += pi * sqr(c[i].r);
    }
}
```

Minimum Enclosing Circle

```
circle mincir(point *p, int n)
{
    point c; double r;
    random_shuffle(p, p + n);
    c = p[0]; r = 0;
    for (int i = 1; i < n; ++i) {
        if (dcmp(abs(p[i] - c) - r) <= 0) continue;
        c = p[i]; r = 0;
        for (int j = 0; j < i; ++j) {
            if (dcmp(abs(p[j] - c) - r) <= 0) continue;
            c = (p[i] + p[j]) * 0.5; r = dis(p[j], c);
            for (int k = 0; k < j; ++k) {
                if (dcmp(abs(p[k] - c) - r) <= 0) continue;
                c = center(p[i], p[j], p[k]); r = dis(p[k], c);
            }
        }
    }
    return circle(c, r);
}
```

Ellipse Circumference

```
double cal(double a, double b) { // a >= b
    double e2 = 1.0 - b * b / a / a;
    double e = e2;
    double ret = 1.0;
    double xa = 1.0, ya = 2.0;
    double t = 0.25;
    for (int i = 1; i <= 10000; ++i) {
        ret -= t * e;
        t = t * xa * (xa + 2) / (ya + 2) / (ya + 2);
        xa += 2.0;
        ya += 2.0;
        e *= e2;
    }
    return 2.0 * pi * a * ret;
}
```

Planar Strainght-line Graph

```
int pcnt, ecnt, fcnt;
point p[MAXN];
struct edge { int t; double ang; };
edge E[MAXN * MAXN * 2];
vector<int> G[MAXN * MAXN];
int co[MAXN * MAXN * 2];
int pre[MAXN * MAXN * 2];
vector<point> face[MAXN * MAXN * 2];

void addedge(int u, int v)
{
    G[u].push_back(ecnt);
    E[ecnt++] = (edge){v, arg(p[v] - p[u])};
    G[v].push_back(ecnt);
    E[ecnt++] = (edge){u, arg(p[u] - p[v])};
}

bool cmpang(int i, int j) { return E[i].ang < E[j].ang; }

void build_pslg()
{
    for (int u = 0; u < pcnt; ++u) {
        sort(G[u].begin(), G[u].end(), cmpang);
        int n = G[u].size();
        for (int j = 0; j < n; ++j) pre[G[u][(j + 1) % n]] = G[u][j];
    }
    memset(co, -1, sizeof(co));
    for (int u = 0; u < pcnt; ++u) {
        for (int i = 0; i < G[u].size(); ++i) {
            int e = G[u][i];
            if (co[e] != -1) continue;
            while (co[e] == -1) {
                co[e] = fcnt;
                face[fcnt].push_back(p[E[e].t]);
                e = pre[e^1];
            }
            face[++fcnt].clear();
        }
    }
}
```

3D Geometry

```
dot(a, b)   { return a.x * b.x + a.y * b.y + a.z * b.z; }
cross(a, b) { return vec3(a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z, a.x*b.y-a.y*b.x); }

area2(a, b, c)   { return abs(cross(b - a, c - a)); }
vol6(a, b, c, d) { return dot(cross(b - a, c - a), d - a); }

vec3 proj(vec3 v, vec3 d) { return d * dot(v, d) / dot(d, d); }
point3 proj(point3 p, line3 ln) { return ln.s + proj(p - ln.s, dir(ln)); }
point3 proj(point3 p, point3 p0, vec3 n) { return p - proj(p - p0, n); }

vec3 reflect(vec3 v, vec3 n) { return proj(v, n) * 2 - v; }
point3 reflect(point3 p, line3 ln) { return ln.s + reflect(p - ln.s, dir(ln)); }
point3 reflect(point3 p, point3 p0, vec3 n) { return p - proj(p - p0, n) * 2; }

double angle(vec3 a, vec3 b) { return acos(dot(a, b) / abs(a) / abs(b)); }
vec3 rotate(vec3 v, vec3 n, double a)
{
    n = unit(n);
    double cosa = cos(a), sina = sin(a);
    return v * cosa + cross(n, v) * sina + n * dot(n, v) * (1 - cosa);
}
```

Line in 3D

```
double dis(point3 p, line3 ln)
{ return area2(p, ln.s, ln.t) / len(ln); }

double dtoseg(point3 p, line3 ln)
{
    if (dcmp(dot(p - ln.s, dir(ln))) <= 0) return dis(p, ln.s);
    if (dcmp(dot(p - ln.t, dir(ln))) >= 0) return dis(p, ln.t);
    return dis(p, ln);
}

bool onseg(point3 p, line3 ln)
{
    return zero(cross(p - ln.s, p - ln.t))
        && dcmp(dot(p - ln.s, p - ln.t)) <= 0;
}

bool inter(line3 ln, point3 p0, vec3 n, point3 &p) // line & plane intersection
{
    double d1 = dot(ln.s - p0, n);
    double d2 = dot(ln.t - p0, n);
    if (!dcmp(d1 - d2)) return false;
    p = (ln.t * d1 - ln.s * d2) / (d1 - d2);
    return true;
}

double dis(line3 a, line3 b)
{
    vec3 n = cross(dir(a), dir(b));
    if (zero(n)) return dis(a.s, b);
    return fabs(dot(a.s - b.s, n)) / abs(n);
}

bool approach(line3 a, line3 b, point3 &p) // clost approach point of 2 lines
{
    vec3 u = dir(a), v = dir(b), w = a.s - b.s;
    double d = dot(u, u) * dot(v, v) - dot(u, v) * dot(u, v);
    if (!dcmp(d)) return false; // parallel
    double c = dot(u, v) * dot(v, w) - dot(v, v) * dot(u, w);
    p = a.s + u * (c / d);
    return true;
}
```

Sphere

```
bool inter(sphere s, line3 ln, point3 &p1, point3 &p2)
{
    point3 p = proj(s.c, ln);
    double d = abs(p - s.c);
    if (dcmp(d - s.r) > 0) return false;
    vec3 v = unit(dir(ln)) * sqrt(s.r * s.r - d * d);
    p1 = p - v; p2 = p + v;
```

```
        return true;
}
```

Convex Hull in 3D

```
struct face {
    int v[3];
    face(int a, int b, int c) { v[0] = a; v[1] = b; v[2] = c; }
    int operator[](int i) const { return v[i % 3]; }
};

bool visible(point3 p[], face f, int i)
{ return dcmp(vol6(p[f[0]], p[f[1]], p[f[2]], p[i])) > 0; }

vector<face> ch3d(point3 p[], int n)
{
    static bool v[MAXN][MAXN];
    int i, j, k;
    for (i = 2; i < n && !dcmp(area2(p[0], p[1], p[i])); ++i) {}
    swap(p[2], p[i]);
    for (i = 3; i < n && !dcmp(vol6(p[0], p[1], p[2], p[i])); ++i) {}
    swap(p[3], p[i]);
    vector<face> cur;
    cur.push_back(face(0, 1, 2));
    cur.push_back(face(2, 1, 0));
    for (i = 3; i < n; ++i) {
        vector<face> next;
        for (j = 0; j < cur.size(); ++j) {
            face f = cur[j];
            bool vis = visible(p, f, i);
            if (!vis) next.push_back(f);
            for (int k = 0; k < 3; ++k) v[f[k]][f[k + 1]] = vis;
        }
        for (j = 0; j < cur.size(); ++j) {
            for (k = 0; k < 3; ++k) {
                int a = cur[j][k], b = cur[j][k + 1];
                if (v[a][b] && !v[b][a]) {
                    next.push_back(face(a, b, i));
                }
            }
        }
        cur.swap(next);
    }
    return cur;
}
```

Half-space Intersection

```
struct plane {
    point3 p; vec3 n; // represent the half-space that n direct to
    plane() {}
    plane(point3 p, point3 n): p(p), n(n) {}
};

point3 inter(line3 ln, plane f)
{
    double d1 = dot(ln.s - f.p, f.n);
    double d2 = dot(ln.t - f.p, f.n);
    return (ln.t * d1 - ln.s * d2) / (d1 - d2);
}

struct face {
    point3 p[3];
    face(point3 a, point3 b, point3 c) { p[0] = a, p[1] = b, p[2] = c; }
    point3 &operator[](int i) { return p[i]; }
    vec3 normal() { return cross(p[1] - p[0], p[2] - p[0]); }
    void adjust(vec3 n) { if (dot(n, normal()) < 0) swap(p[0], p[1]); }
};

vector<face> cut(vector<face> h, plane f)
{
    vector<face> ans;
    point3 p0; int m = 0;
    for (int i = 0; i < h.size(); ++i) {
        vector<point3> c0, c1, c2;
        for (int j = 0; j < 3; ++j) {
            int d = dcmp(dot(h[i][j] - f.p, f.n));
            if (d == 0) c0.push_back(h[i][j]);
            else if (d > 0) c1.push_back(h[i][j]);
            else c2.push_back(h[i][j]);
        }
        if (c0.size() == 3) {
            if (dot(f.n, h[i].normal()) < 0) h.clear();
            return h;
        }
        if (c0.size() == 1) {
            if (c1.size() > c2.size()) c1.push_back(c0[0]);
            else c2.push_back(c0[0]);
        }
        if (c0.size() == 2) c2.push_back(c0[0]), c2.push_back(c0[1]);
        if (c1.size() == 3) ans.push_back(h[i]);
        if (c1.size() == 3 || c2.size() == 3) continue;
        point3 p1, p2; vec3 n = h[i].normal();
        if (c1.size() == 1) {
            p1 = inter(line3(c1[0], c2[0]), f);
            p2 = inter(line3(c1[0], c2[1]), f);
            ans.push_back(face(c1[0], p1, p2));
            ans.back().adjust(n);
        } else {
            p1 = inter(line3(c1[0], c2[0]), f);
            p2 = inter(line3(c1[1], c2[0]), f);
            ans.push_back(face(c1[0], p1, p2));
            ans.back().adjust(n);
            ans.push_back(face(c1[0], c1[1], p2));
            ans.back().adjust(n);
        }
        if (m++) {
            ans.push_back(face(p0, p1, p2));
            ans.back().adjust(f.n);
        } else p0 = p1;
    }
    return ans;
}
```

3D Transformation Matrix

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

沿 $\overrightarrow{(x, y, z)}$ 方向平移

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

以 (x, y, z) 为比例缩放

$$\begin{bmatrix} -1 & 0 & 0 & 2x \\ 0 & -1 & 0 & 2y \\ 0 & 0 & -1 & 2z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

关于点 (x, y, z) 对称

$$\begin{bmatrix} 2x^2-1 & 2xy-1 & 2xz-1 & 0 \\ 2yx-1 & 2y^2-1 & 2yz-1 & 0 \\ 2zx-1 & 2zy-1 & 2z^2-1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

关于原点到 (x, y, z) 的直线轴对称

$$\begin{bmatrix} -2x^2 & -2xy & -2xz & 0 \\ -2yx & -2y^2 & -2yz & 0 \\ -2zx & -2zy & -2z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

关于过原点以 $\overrightarrow{(x, y, z)}$ 为法向量的平面对称

$$\begin{bmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ zx(1-c)-ys & zy(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, s = sin(\alpha), c = cos(\alpha)$$

以 $\overrightarrow{(x, y, z)}$ 为轴旋转 α 弧度

变换一个点, 相当于右乘列向量 (x, y, z, 1)。注意轴对称、面对称和旋转矩阵需要单位化 (x, y, z)。

6 OTHERS

Exact Cover

```
int N, S[COL + 1], L[NODE], R[NODE], U[NODE], D[NODE], row[NODE], C[NODE];

void dlxinit(int c) // c Cumns, numbered from 1
{
    for (int i = 0; i <= c; ++i) {
        U[i] = D[i] = i;
        L[i] = i - 1; R[i] = i + 1;
        S[i] = 0;
    }
    L[0] = c; R[c] = 0; N = c + 1;
}

void addrow(const vector<int> &c)
{
    int h = N;
    for (int i = 0; i < c.size(); ++i) {
        U[N] = U[c[i]]; D[N] = c[i];
        D[U[N]] = U[D[N]] = N;
        L[N] = N - 1; R[N] = N + 1;
        ++S[C[N++] = c[i]];
    }
    L[h] = N - 1; R[N - 1] = h;
}

void remove(int c)
{
    L[R[c]] = L[c];
    R[L[c]] = R[c];
    for (int i = D[c]; i != c; i = D[i]) {
        for (int j = R[i]; j != i; j = R[j]) {
            U[D[j]] = U[j];
            D[U[j]] = D[j];
            --S[C[j]];
        }
    }
}

void resume(int c)
{
    for (int i = U[c]; i != c; i = U[i]) {
        for (int j = L[i]; j != i; j = L[j]) {
            U[D[j]] = j;
            D[U[j]] = j;
            ++S[C[j]];
        }
    }
    L[R[c]] = c;
    R[L[c]] = c;
}

bool dance(int d)
{
    if (R[0] == 0) return true;
    int c = R[0];
    for (int i = R[0]; i; i = R[i]) {
        if (S[i] < S[c]) c = i;
    }
    remove(c);
    for (int i = D[c]; i != c; i = D[i]) {
        // select row[i]
        for (int j = R[i]; j != i; j = R[j]) remove(C[j]);
        if (dance(d + 1)) return true;
        for (int j = L[i]; j != i; j = L[j]) resume(C[j]);
    }
    resume(c);
    return false;
}
```

Fuzzy Cover

```
void remove(int i)
{
    for (int j = D[i]; j != i; j = D[j]) {
        R[L[j]] = R[j];
        L[R[j]] = L[j];
    }
}

void resume(int i)
{
    for (int j = U[i]; j != i; j = U[j]) {
        R[L[j]] = j;
        L[R[j]] = j;
    }
}

int h()
{
    static int v[COL + 1], m;
    int s = 0; ++m;
    for (int i = R[0]; i; i = R[i]) {
        if (v[i] == m) continue;
        ++s; v[i] = m;
        for (int j = D[i]; j != i; j = D[j]) {
            for (int k = R[j]; k != j; k = R[k]) {
                v[C[k]] = m;
            }
        }
    }
    return s;
}

bool dance(int d)
{
    if (!R[0]) return true;
    if (d + h() > limit) return false;
    int c = R[0];
    for (int i = R[c]; i; i = R[i]) {
        if (S[i] < S[c]) c = i;
    }
    for (int i = D[c]; i != c; i = D[i]) {
        remove(i);
        for (int j = R[i]; j != i; j = R[j]) remove(j);
        if (dance(d + 1)) return true;
        for (int j = L[i]; j != i; j = L[j]) resume(j);
        resume(i);
    }
    return false;
}
```

Linear Programming

```
int n, m;
double a[MAXM][MAXN];
double x[MAXN];
int N[MAXN], B[MAXM];
const double eps = 1e-10, inf = 1e100;

// a[i][0]*x[0] + a[i][1]*x[1] + ... <= a[i][n]
// max(a[m][0]*x[0] + a[m][1]*x[1] + ... - a[m][n])
// x[i] >= 0

void pivot(int r, int c)
{
```

```
    swap(N[c], B[r]);
    a[r][c] = 1 / a[r][c];
    for (int i = 0; i <= n; ++i) if (i != c) a[r][i] *= a[r][c];
    for (int i = 0; i <= m; ++i) if (i != r) {
        for (int j = 0; j <= n; ++j) if (j != c) {
            a[i][j] -= a[i][c] * a[r][j];
        }
        a[i][c] *= -a[r][c];
    }
}

bool feasible()
{
    for (;;) {
        int r, c;
        double p = inf;
        for (int i = 1; i < m; ++i) if (a[i][n] < p) p = a[r = i][n];
        if (p > -eps) return true;
        p = 0;
        for (int i = 1; i < n; ++i) if (a[r][i] < p) p = a[r][c = i];
        if (p > -eps) return false;
        p = a[r][n] / a[r][c];
        for (int i = r + 1; i < m; ++i) if (a[i][c] > eps) {
            double v = a[i][n] / a[i][c];
            if (v < p) r = i, p = v;
        }
        pivot(r, c);
    }
}

int simplex() // 0 - no solution, -1 - infinity, 1 - has a solution
{
    for (int i = 0; i < n; ++i) N[i] = i;
    for (int i = 0; i < m; ++i) B[i] = n + i;
    if (!feasible()) return 0;
    for (;;) {
        int r, c;
        double p = 0;
        for (int i = 0; i < n; ++i) if (a[m][i] > p) p = a[m][c = i];
        if (p < eps) break;
        p = inf;
        for (int i = 0; i < m; ++i) if (a[i][c] > eps) {
            double v = a[i][n] / a[i][c];
            if (v < p) r = i, p = v;
        }
        if (p == inf) return -1;
        pivot(r, c);
    }
    for (int i = 0; i < n; ++i) if (N[i] < n) x[N[i]] = 0;
    for (int i = 0; i < m; ++i) if (B[i] < n) x[B[i]] = a[i][n];
    ans = -a[m][n];
    return 1;
}
```

3D Partial Order

```
// 三维偏序最长上升子序列
// 改用注释代码则变为最长不下降子序列
struct triple {
    int x, y, z;
    bool operator<(const triple &b) const
    {
        return x != b.x ? x < b.x : y > b.y;
        // return x != b.x ? x < b.x : y < b.y;
    }
}v[MAXN];
int f[MAXN];

void solve(int l, int r)
{
    if (r - l == 1) f[l] = max(f[l], 1);
    if (r - l <= 1) return;
    static int p[MAXN];
    int mid = (l + r) / 2;
    solve(l, mid);
    for (int i = l; i < r; ++i) p[i] = i;
    sort(p + l, p + r, [](int i, int j) {
            return v[i].y != v[j].y ? v[i].y < v[j].y : i > j;
            // return v[i].y != v[j].y ? v[i].y < v[j].y : i < j;
            });
    for (int i = l; i < r; ++i) {
        if (p[i] < mid) bit.add(v[p[i]].z, f[p[i]]); // maintain maximum
        else f[p[i]] = max(f[p[i]], bit.query(v[p[i]].z - 1) + 1);
        // f[p[i]] = max(f[p[i]], bit.query(v[p[i]].z) + 1);
    }
    for (int i = l; i < mid; ++i) bit.clear(v[i].z);
    solve(mid, r);
}

void solve()
{
    sort(v, v + n);
    static int z[MAXN];
    for (int i = 0; i < n; ++i) z[i] = v[i].z;
    sort(z, z + n);
    int tot = unique(z, z + n) - z;
    for (int i = 0; i < n; ++i) {
        v[i].z = lower_bound(z, z + tot, v[i].z) - z + 1;
        f[i] = 0;
    }
    solve(0, n);
    return *max_element(f, f + n);
}
```

Adaptive Simpson's Method

```
double simpson(double a, double b) {
    double c = a + (b-a)/2;
    return (F(a)+4*F(c)+F(b))*(b-a)/6;
}

double asr(double a, double b, double eps, double A) {
    double c = a + (b-a)/2;
    double L = simpson(a, c), R = simpson(c, b);
    if(fabs(L+R-A) <= 15*eps) return L+R+(L+R-A)/15.0;
    return asr(a, c, eps/2, L) + asr(c, b, eps/2, R);
}

double asr(double a, double b, double eps) {
    return asr(a, b, eps, simpson(a, b));
}
```

Connect DP

```
const LL bit=(LL)(1000000000)*(LL)(1000000000);
const int MAXD=15, HASH=30007, STATE=1000010, MAXN=3;

int N,M,n,m;
int maze[MAXD][MAXD], code[MAXD], ch[MAXD];
int ex,ey;

struct HASHMAP  {
    int head[HASH],next[STATE],size;
    LL state[STATE];
    LL f[STATE];
    void init()  {
        size=0;
        memset(head,-1,sizeof(head));
```

```
    }
    void push(LL st,LL ans)  {
        int h=st%HASH;
        for(int i=head[h];i!=-1;i=next[i])
            if(state[i]==st)  {
                f[i]=f[i]+ans;
                return;
            }
        state[size]=st;
        f[size]=ans;
        next[size]=head[h];
        head[h]=size++;
    }
}hm[2];

void decode(int *code,int m,LL  st)  {
    for(int i=m;i>=0;i--)  {
        code[i]=st&7;
        st>>=3;
    }
}

LL encode(int *code,int m)  {
    int cnt=1;
    memset(ch,-1,sizeof(ch));
    ch[0]=0;
    LL st=0;
    for(int i=0;i<=m;i++)  {
        if(ch[code[i]]==-1)ch[code[i]]=cnt++;
        code[i]=ch[code[i]];
        st<<=3;
        st|=code[i];
    }
    return st;
}

void shift(int *code,int m)  {
    for(int i=m;i>0;i--)code[i]=code[i-1];
    code[0]=0;
}

bool jud(int *code,int m)  {
    int cnt=1;
    memset(ch,-1,sizeof(ch));
    ch[0]=0;
    for(int i=0;i<=m&&cnt<3;i++)
        if(ch[code[i]]==-1)ch[code[i]]=cnt++;
    return cnt==2;
}

//12 34567
//  |-----
//--|
//01234567
bool con(int *code,int x,int y)  {
    memset(ch,0,sizeof(ch));
    for(int i=1;i<=M;i++)  {
        if(i==y)continue;
        if(i<y&&maze[x+1][i])  ch[code[i-1]]=1;
        else if(i>y&&maze[x][i])  ch[code[i]]=1;
    }
    for(int i=0;i<=M;i++) if(code[i]!=0&&ch[code[i]]==0) return false;
    return true;
}

void dpblank(int i,int j,int cur)  {
    int left,up;
    for(int k=0;k<hm[cur].size;k++)  {
        int cod[MAXD];
        decode(code,M,hm[cur].state[k]);
        memcpy(cod,code,sizeof(code));
        left=code[j-1];
        up=code[j];
        tmp=hm[cur].f[k];
        if(left&&up)  {
            if(left==up)  {
                if(j==M)shift(code,M);
                hm[cur^1].push(encode(code,M),hm[cur].f[k]);
                if(jud(code,M)) ans=ans+tmp;
            }
            else  {
                code[j-1]=code[j]=left;
                for(int t=0;t<=M;t++) if(code[t]==up)code[t]=left;
                if(j==M)shift(code,M);
                hm[cur^1].push(encode(code,M),hm[cur].f[k]);
                if(jud(code,M)) ans=ans+tmp;
            }
        }
        else if((left&&(!up))||((!left)&&up))  {
            int t;
            if(!left)t=up;
            else t=left;
            code[j-1]=code[j]=t;
            if(j==M)shift(code,M);
            hm[cur^1].push(encode(code,M),hm[cur].f[k]);
            if(jud(code,M)) ans=ans+tmp;
        }
        else  {
            code[j-1]=code[j]=13;
            if(j==M)shift(code,M);
            hm[cur^1].push(encode(code,M),hm[cur].f[k]);
            if(jud(code,M))ans=ans+tmp;
        }
        memcpy(code,cod,sizeof(cod));
        maze[i][j]=0;
        if(!con(code,i,j))  continue;
        code[j]=code[j-1]=0;
        if(j==M)shift(code,M);
        hm[cur^1].push(encode(code,M),hm[cur].f[k]);
    }
}

void dpblock(int i,int j,int cur)  {
    for(int k=0;k<hm[cur].size;k++)  {
        decode(code,M,hm[cur].state[k]);
        if(!con(code,i,j))  continue;
        code[j-1]=code[j]=0;
        if(j==M)shift(code,M);
        hm[cur^1].push(encode(code,M),hm[cur].f[k]);
    }
}

void dp()  {
    int cur=0;
    ans=0;
    hm[cur].init();
    hm[cur].push(0,1);
    for(int i=1;i<=N;i++)
        for(int j=1;j<=M;j++)  {
            hm[cur^1].init();
            if(maze[i][j])dpblank(i,j,cur);
            else  dpblock(i,j,cur);
            cur^=1;
        }
}
```

Rectangular Cut

```
int n, m;
LL ans;
vi v[N][2];

void work(int lev, vi &a, vi &b);

bool is_in(vi &a, vi &b, vi &a2, vi &b2){
    rep(i, n){
        if (!(a2[i] <= a[i] && b[i] <= b2[i])) return false;
    }
    return true;
}

void dfs(vi &a, vi &b, vi &a2, vi &b2, int now_d, int lev){
    rep(i, n){
        if (a[i] == b[i]) return;
    }
    if (is_in(a, b, a2, b2)) return;
    if (now_d == n) return;
    int l = max(a[now_d], a2[now_d]);
    int r = min(b[now_d], b2[now_d]);
    int tmp_l = a[now_d], tmp_r = b[now_d];

    a[now_d] = l, b[now_d] = r;
    dfs(a, b, a2, b2, now_d + 1, lev);
    a[now_d] = tmp_l, b[now_d] = tmp_r;

    b[now_d] = l;
    work(lev + 1, a, b);
    b[now_d] = tmp_r;

    a[now_d] = r;
    work(lev + 1, a, b);
    a[now_d] = tmp_l;
}

void work(int lev, vi &a, vi &b){
    rep(i, n){
        if (a[i] == b[i]) return;
    }
    if (lev == m){
        LL ret = 1;
        rep(i, n) ret = ret * (b[i] - a[i]) % MD;
        ans = (ans + ret) % MD;
        return;
    }
    vi &a2 = v[lev][0], &b2 = v[lev][1];
    bool no_cover = false;
    rep(i, n){
        int l = max(a[i], a2[i]);
        int r = min(b[i], b2[i]);
        if (l >= r) no_cover = true;
    }
    if (no_cover) work(lev + 1, a, b);
    else dfs(a, b, a2, b2, 0, lev);
}

int main(){
    while(~scanf("%d%d", &m, &n)){
        rep(i, m){
            v[i][0].clear();
            rep(j, n){
                int x;
                scanf("%d", &x);
                v[i][0].PB(x);
            }
            v[i][1].clear();
            rep(j, n){
                int x;
                scanf("%d", &x);
                v[i][1].PB(x);
            }
            rep(j, n) if (v[i][0][j] > v[i][1][j]) swap(v[i][0][j], v[i][1][j]);
        }
        ans = 0;
        rep(i, m){
            work(i + 1, v[i][0], v[i][1]);
        }
        ans = (ans % MD + MD) % MD;
        printf("%d\n", (int)ans);
    }
    return 0;
}
```

Total Monotonicity DP

```
struct data{
    int l,r,p;
}q[maxn];

int find(data t,int q){
    int l=t.l,r=t.r,mid;
    while(l<=r){
        mid=(l+r)>>1;
        if(cal(q,mid)<cal(t.p,mid)) r=mid-1;
        else l=mid+1;
    }
    return l;
}

void dp(){
    int head=1,tail=0;
    q[++tail]=(data){0,n,0};
    for(int i=1;i<=n;i++){
        if(i>q[head].r) head++;
        f[i]=cal(q[head].p,i);
        if(head>tail||cal(i,n)<cal(q[tail].p,n)){
            while(head<=tail&&cal(i,q[tail].l)<cal(q[tail].p,q[tail].l))
                tail--;
            if(head<=tail){
                int t=find(q[tail],i);
                q[tail].r=t-1;
                q[++tail]=(data){t,n,i};
            }
            else q[++tail]=(data){i,n,i};
        }
    }
}
```

Vim Configuration

```
set nocp nu cin ts=4 sw=4
syn on
set mp=g++\ -g\ -o\ %<\ %\ -Wall\ -std=c++11
map mk :make<cr>
map mr :!./%<<cr>
map mw :!./%< < %<.in<cr>
map mi :sp %<.in<cr>
```