杂项

forgottencsc

Sept 19, 2019

杂项

排列，置换，群

按字典序遍历所有排列

```
int p[N]; iota(p + 1, p + n + 1, 1);
do {
    // ...
} while (next_permutation(p + 1, p + n + 1));
// 改成 prev_permutation 并将 p 初始化为 n~1 即可从大到小遍历
```

Cantor 展开

```
// Cantor expansion when S is huge, 1 based
ll cexp(int* a, int n) {
    static int b[N] = {0};
    fill(b+1,b+n+1,0);
    ll res = 0;
    for (int i = n; i >= 1; --i) {
        ll cnt = 0;
        for (int j=a[i]; j; j-=j&-j) cnt += b[j];
        res = M(res + cnt * f[n-i]);
        for (int j=a[i]; j<=n; j+=j&-j) b[j]++;
    }
    return res + 1;
}

// Cantor expansion when S is small, 0 based
ll cexp(array<int, S> a) {
    int res = 0;
    for (int i = 0; i != S - 1; ++i)
        for (int j = i + 1; j != S; ++j)
            if (a[i] > a[j]) res += f[S - 1 - i];
    return res;
}

array<int, S> icexp(ll x) {
    array<int, S> res;
    vector<int> v(S); iota(v.begin(), v.end(), 0);
    for (int i = 0; i != S; ++i) {
        int t = f[S - i - 1];
        res[i] = v[x / t];
        v.erase(v.begin() + x / t);
        x %= t;
    }
    return res;
}
```

循环分解

Burnside 引理/**Polya** 定理（见组合部分）

数值算法

拉格朗日插值

已知 $f(x)$ 为次数不超过 $n-1$ 次的多项式与 $y_1 = f(x_1), y_2 = f(x_2), ..., y_n = f(x_n)$ 的值，则 $f(x)$ 在其他任意位置的值可以在 $O(n^2)$ 内求出。

考虑具有特殊性质的多项式

$$p_k(x) = \prod_{i=1,i!=k}^{n} \frac{x - x_i}{x_k - x_i}$$

易知 $p_k(x)$ 在 x_k 处取值为 1，在其他 x_i 处取值为 0。那么我们可以直接构造出与 $f(x)$ 恒等的多项式

$$f(x) = \sum_{k=1}^{n} y_k p_k(x) = \sum_{k=1}^{n} y_k \prod_{i=1,i!=k}^{n} \frac{x - x_i}{x_k - x_i}$$

当 $x_i = x_{i-1} + 1$ 时到 p_k 的分母分子均可快速求出，此时插值可在 $O(n)$ 内完成。

```
// Lagrangian Interpolation, P(x[0])==y[0], P(x[1])==y[1], ...
ll lintp(const vector<ll>& y, const vector<ll>& x, ll m) {
    int n = y.size(); ll res = 0;
    for (int i = 0; i != n; ++i) {
        ll t = y[i];
        for (int j = 0; j != n; ++j)
            if (j != i)
                t = M(t * M((P+m-x[j]) * inv(P+x[i]-x[j])));
        res = M(res + t);
    }
    return res;
}

// Lagrangian Interpolation, P(0)==y[0], P(1)==y[1], ...
ll lp[N], ls[N];
ll lintp(const vector<ll>& y, ll x) {
    int n = y.size(); ll res = 0; lp[0] = x; ls[n - 1] = x - (n - 1);
    for (int i = 1; i != n; ++i) lp[i] = M(lp[i - 1] * (x - i));
    for (int i = n - 2; i >= 0; --i) ls[i] = M(ls[i + 1] * (x - i));
    for (int i = 0; i != n; ++i) {
        ll t = (n - i + 1) & 1 ? P - y[i] : y[i];
        if (i)          t = M(t * M(lp[i - 1] * fi[i]));
        if (i != n - 1) t = M(t * M(ls[i + 1] * fi[n - 1 - i]));
        res = M(res + t);
    }
    return res;
}
```

高斯消元，零空间

```
#include <bits/stdc++.h>
#define R(m) (int)(m).size()
#define C(m) (int)((m)[0].size())
const dbl eps = 1e-8;
typedef vector<dbl> vec;
typedef vector<vec> mat;

//  Gaussian Elimination, O(n^3)
int row_reduction(mat& a) {
    const int& n = R(a), m = C(a);
    for (int i = 0, j = 0; i != n; ++i) {
        do for (int k = i + 1; k != n; ++k)
            if (fabs(a[i][j]) < fabs(a[k][j]))
                swap(a[i], a[k]);
        while (!dc(a[i][j]) && ++j != n);
        if (j == n) return i; //  rk(a)
        for (int l = m - 1; l >= i; --l)
            a[i][l] /= a[i][j];
        for (int k = 0; k != n; ++k) {
            if (k != i) for (int l = m - 1; l >= i; --l)
                a[k][l] -= a[k][j] * a[i][l];
    }
    return n;
}

//  Get a base of null space of a, O(n^2)
mat nsp(mat& a) {
    const int n = C(a);
    while(R(a) < n) a.push_back(vec(n, 0));
    int w = row_reduction(a); mat b(n, vec(n - w, 0));
    vector<int> p, vis(n, 0);
    for (int c = 0, r = 0; c != n; ++c)
        if (abs(a[r][c]) > eps) ++r, vis[c] = 1;
    for (int i = 0; i != n; ++i) if (vis[i]) p.push_back(i);
    for (int i = 0; i != n; ++i) if (!vis[i]) p.push_back(i);
    for (int i = 0; i != n - w; ++i) {
        for (int j = 0; j != n; ++j)
            b[j][i] = a[p[j]][p[i + w]];
        b[p[i + w]][i] = -1;
    }
    return b;
}
```

Graham-Schmidt 正交化，**QR** 分解

```
//  Gram-Schmidt Orthogonalization, O(n^3)
void gso(mat& a) {
    const int& n = R(a), m = C(a);
    for (int i = 0; i != n; ++i) {
        for (int j = 0; j != i; ++j) {
            dbl l = 0;
            for (int k = 0; k != m; ++k) l += a[i][k] * a[j][k];
            for (int k = 0; k != m; ++k) a[i][k] -= a[j][k] * l;
        }
        dbl l = 0;
        for (int k = 0; k != m; ++k) l += a[i][k] * a[i][k];
        l = sqrt(l);
        for (int k = 0; k != m; ++k) a[i][k] /= l;
    }
}

pair<mat, mat> QR(const mat& a) {
    mat q = a; gso(q);
    return { q, transpose(q) * a };
}

vector<dbl> eigenvalues(mat a) {
    pair<mat, mat> qr;
    for (int i = 0; i != 10; ++i) {
        qr = QR(a);
        a = qr.second * qr.first;
    }
    vector<dbl> res;
    for (int i = 0; i != R(a); ++i)
        res.push_back(a[i][i]);
    return res;
}
```

实数的分数逼近（**Farey** 序列与 **Stern-Brocott** 树）

定义：Farey 序列

最开始左侧为 $\frac{0}{1}$，右侧为 $\frac{1}{0}$

之后向序列中每对相邻的分数 $\frac{p_1}{q_1}$ 与 $\frac{p_2}{q_2}$ 之间插入 $\frac{p_1+p_2}{q_1+q_2}$

以 $\frac{1}{1}$ 为根，将除了 $\frac{0}{1}$ 与 $\frac{1}{0}$ 之外的所有分数向生成它的两个分数之中最深的那一个连边即可获得 Stern-Brocott 树。这里的深是指迭代法雷序列的迭代深度。

若在每次迭代中将所有分母大于当前迭代次数的分数都去掉，则每层新产生的分数为所有分母为当前迭代层数的真分数。

```
pair<ll, ll> get_frac(dbl x, ll p1, ll q1, ll p2, ll q2) {
    ll p3 = (p1 + p2), q3 = (q1 + q2), d = gcd(p3, q3);
    p3 /= d; q3 /= d;
    if (fabs(q3 * x - p3) < eps) return { p3, q3 };
    else if (x * q3 < p3) return get_frac(x, p1, q1, p3, q3);
    else return get_frac(x, p3, q3, p2, q2);
}
```

单峰函数求极值

```
typedef double dbl;
const dbl eps = 1e-8, phi = (sqrt(5) - 1) / 2;
pair<dbl, dbl> gss0(const function<dbl(dbl)>& f,
         dbl l1, dbl l2, dbl r2, dbl r1,
         dbl l1v, dbl l2v, dbl r2v, dbl r1v) {
    if (l2 + eps >= r2) {
        if (l2v < r2v) return { l2, l2v };
        else return { r2, r2v };
    }
    if (l2v > r2v)  //  < for minimum, > for maximum
        return gss0(f, l1, r2 - phi * (r2 - l1), l2, r2,
```

```cpp
                    l1v, f(r2 - phi * (r2 - l1)), l2v, r2v);
    else
        return gss0(f, l2, r2, l2 + phi * (r1 - l2), r1,
                    l2v, r2v, f(l2 + phi * (r1 - l2)), r1v);
}

pair<dbl, dbl> gss(const function<dbl(dbl)>& f,
                   dbl l1, dbl r1) {
    dbl t = phi * (r1 - l1), l2 = r1 - t, r2 = l1 + t;
    return gss0(f, l1, l2, r2, r1, f(l1), f(l2), f(r2), f(r1));
}

//Example:
//Gym101981D(2018 南京现场赛 D) 最小球覆盖，三分套三分套三分，nlogM^3
//这里的 M 是 1e5/1e-8=1e13，至少有个 1.5 倍的常数，因为 log 的底数是 phi+1
int main() {
    cout << fixed << setprecision(10) << gss([&](dbl x) {
            return gss([&](dbl y) {
                return gss([&](dbl z) {
                    dbl res = 0;
                    for (vec w : v)
                        res = max(res, dis(w, {x,y,z}));
                    return res;
                }, -1e5, 1e5).second;
            }, -1e5, 1e5).second;
        }, -1e5, 1e5).second <<endl;
    }
```

自适应辛普森积分

```cpp
template<class F> dbl sps0(const F& f,dbl l, dbl r) {
    return (f(l) + 4 * f((l + r) / 2) + f(r)) * (r - l) / 6;
}

template<class F> dbl sps1(const F& f, dbl l, dbl r, dbl eps, dbl s) {
    dbl m = (l + r) / 2, lv = sps0(f, l, m), rv = sps0(f, m, r), d = lv + rv - s;
    if (fabs(d) < 15 * eps) return lv + rv + 15 * d;
    else return sps1(f, l, m, eps / 2, lv) + sps1(f, m, r, eps / 2, rv);
}

template<class F> dbl spsint(const F& f, dbl l, dbl r, dbl eps) {
    return sps1(f, l, r, eps, sps0(f, l, r));
}
```

三次/四次方程求复根

```cpp
#define sq(x) ((x) * (x))

#define cb(x) (sq(x) * (x))

typedef complex<dbl> cplx;
void cubic(dbl a0, dbl a1, dbl a2, dbl a3,
    cplx& x1, cplx& x2, cplx& x3) {
    a0 /= a3; a1 /= a3; a2 /= a3;
    dbl p = sq(a2) / 9 - a1 / 3, q = a2 * a1 / 6 - cb(a2) / 27 - a0/ 2;
    cplx del = sqrt((cplx)(sq(q) - cb(p))), w(-0.5, sqrt(3) / 2),
        f1 = pow(q + del, 1. / 3), f2 = pow(q - del, 1. / 3);
    x1 = f1 + f2 - a2 / 3;
    x2 = w * f1 + sq(w) * f2 - a2 / 3;
    x3 = sq(w) * f1 + w * f2 - a2 / 3;
}

void quadr(dbl a0, dbl a1, dbl a2, dbl a3, dbl a4,
    cplx& x1, cplx& x2, cplx& x3, cplx& x4) {
    a0 /= a4; a1 /= a4; a2 /= a4; a3 /= a4; a3 /= 4;
    dbl b = a2 - 6 * sq(a3), c = 3 * (a1 - 2 * a2 * a3 + cb(2 * a3)),
        d = 12 * (a0 - a1 * a3 + a2 * sq(a3) - 3 * sq(sq(a3))),
        p = sq(b) + d, q = 2 * cb(b) + 3 * sq(c) - 6 * b * d, k = 2 * sqrt(3);
    cplx del = sqrt((cplx(sq(q) - 4 * cb(p)))),
        f = pow((q + del) / 2., 1. / 3) + pow((q - del) / 2., 1. / 3),
        f1 = sqrt(f - 2 * b), f2 = k * c / f1, f3 = -4 * b - f;
    x1 = (f1 - sqrt(f3 - f2)) / k - a3; x2 = (f1 + sqrt(f3 - f2)) / k - a3;
    x3 = (-f1 - sqrt(f3 + f2)) / k - a3; x4 = (-f1 + sqrt(f3 + f2)) / k - a3;
}
```

多项式方程求实根

```
const dbl eps = 1e-8;
int sgn(dbl d) { return d < -eps ? -1 : d > eps ? 1 : 0; }

dbl val(const vector<dbl>& p, dbl x) {
    dbl w = 1, r = 0;
    for(int i = 0; i != p.size(); ++i, w *= x)
        r += p[i] * w;
    return r;
}

dbl fr(const vector<dbl>& p, dbl lb, dbl ub, bool flg) {
    dbl x = (lb + ub) / 2;
    while(sgn(ub - lb)) {
        dbl res = val(p, x);
        if (!sgn(res)) break;
        else if (flg ^ (sgn(res) == 1)) lb = x;
        else ub = x; x = (ub + lb) / 2;
    }
    return x;
}

vector<dbl> peq(vector<dbl> p) {
    if (p.size() < 2) return {};
    else if (p.size() == 2) return { -p[0] / p[1] };
    else {
        vector<dbl> p_, res, sx, sy; dbl b = 0;
        for (int i = 0; i != p.size(); ++i) b = max(b, abs(p[i] / p[0]) + 1);
        for (int i = 1; i != p.size(); ++i) p_.push_back(p[i] * i);
        sx = peq(p_); sx.insert(sx.begin(), -b); sx.push_back(b);
        for (dbl x : sx) sy.push_back(val(p, x));
        for (int i = 0; i != sx.size() - 1; ++i)
            if (sgn(sy[i] * sy[i + 1]) == -1)
                res.push_back(fr(p, sx[i], sx[i + 1], sgn(sy[i]) == -1));
        return res;
    }
}
```

线性规划与对偶

单纯形法

```
// Simple simplex method, maximize cTx, s.t.Ax<=B;
int n, m, c[5001]; dbl a[4001][201], x[201], z;

int dcmp(dbl d) { return d < -eps ? -1 : d <= eps ? 0 : 1; }

void pivot(int u, int v) {
    swap(c[n + u], c[v]);
    dbl k = a[u][v]; a[u][v] = 1;
    for (int j = 0; j <= n; ++j) a[u][j] /= k;
    for (int i = 0; i <= m; ++i) {
        if (i == u || !dcmp(a[i][v])) continue;
        k = a[i][v]; a[i][v] = 0;
        for (int j = 0; j <= n; ++j)
            a[i][j] -= a[u][j] * k;
    }
}

bool init() {
    for (int i = 1; i <= n; ++i) c[i] = i;
    while (1) {
        int u = 0, v = 0;
        for (int i = 1; i <= m; ++i)
            if (dcmp(a[i][0]) == -1 && (!u || dcmp(a[u][0] - a[i][0]) == 1)) u = i;
        if (!u) return 1;
        for (int j = 1; j <= n && !v; ++j)
            if (dcmp(a[u][j]) == -1) v = j;
        if (!v) return 0;
        pivot(u, v);
    }
}

int simplex() {
    if (!init()) return 0;
    while (1) {
        int u = 0, v = 0;
        for (int j = 1; j <= n; ++j)
            if (dcmp(a[0][j]) == 1 && (!v || a[0][j] > a[0][v])) v = j;

        if (!v) {
            z = -a[0][0];
            for (int i = 1; i <= m; ++i)
                x[c[n + i]] = a[i][0];
            return 1;
        }

        dbl w = 1e20;
        for (int i = 1; i <= m; ++i)
            if (dcmp(a[i][v]) == 1 &&
                dcmp(w - a[i][0] / a[i][v]) == 1) {
                w = a[i][0] / a[i][v];
                u = i;
            }
        if (!u) return 2;
        pivot(u, v);
    }
}
```

单纯形法 **II**

```
int dcmp(const dbl &x) { return x < -eps ? -1 : x > eps; }
struct simplex_t {

    struct cstr_t {
        vector<pair<int, dbl>> a;
        dbl b; int t, r;
    };

    vector<cstr_t> cstrs;
    int m, n; vector<int> c;
    vector<vector<dbl>> a;
    vector<dbl> x;

    // -1 for coeff, 0 for less, 1 for equal, 2 for greater
    void add_cstr(const vector<pair<int, dbl>>& a, dbl b, int t) {
        cstrs.push_back({ a, b, t });
    }

    void pivot(int u, int v) {
        swap(c[n + u], c[v]);
        dbl k = a[u][v]; a[u][v] = 1;
        for (int j = 0; j <= n; ++j) a[u][j] /= k;
        for (int i = 0; i <= m; ++i) {
            if (i == u || !dcmp(a[i][v])) continue;
            k = -a[i][v]; a[i][v] = 0;
            for (int j = 0; j <= n; ++j)
                a[i][j] += a[u][j] * k;
        }
    }

    bool simplex0() {
        while (1) {
            int u = 0, v = 0;
            for (int j = 1; !v && j <= n; ++j)
                if (dcmp(a[0][j]) == 1) v = j;
            if (!v)
                return true;
            dbl w = 1e100;
            for (int i = 1; i <= m; ++i)
                if (dcmp(a[i][v]) == 1 &&
                    dcmp(w - a[i][0] / a[i][v]) == 1) {
                    w = a[i][0] / a[i][v];
                    u = i;
                }
            if (!u)
                return false;
            pivot(u, v);
        }
    }

    //  0 for Infeasible, 2 for Unbounded
    int simplex() {
        int r = 0;
        m = cstrs.size() - 1; n = 0;
        for (cstr_t& cstr : cstrs) {
            for (const pair<int, dbl>& p : cstr.a)
                n = max(n, p.first);
            if (cstr.t == 0 || cstr.t == 2)
                cstr.r = ++r;
            if (cstr.t == -1)
                swap(cstr, cstrs.front());
        }

        c.resize(1 + n + r + m, 0);
        a.resize(m + 1, vector<dbl>(n + r + 1, 0));
        x.resize(1 + n + r, 0);

        for (const pair<int, dbl>& p : cstrs[0].a)
            a[0][p.first] = p.second;
        for (int i = 1; i <= m; ++i) {
            const cstr_t& cstr = cstrs[i];
            for (const pair<int, dbl>& p : cstr.a)
                a[i][p.first] = p.second;
            if (cstr.t == 0) a[i][cstr.r + n] = 1;
            if (cstr.t == 2) a[i][cstr.r + n] = -1;
            a[i][0] = cstr.b;
        }

        n += r;

        for (int i = 1; i <= n + m; ++i) c[i] = i;

        vector<dbl> a0(n + 1, 0); swap(a0, a[0]);
        for (int i = 1; i <= m; ++i) {
            if (a[i][0] < 0)
                for (int j = 0; j <= n; ++j)
                    a[i][j] = -a[i][j];
            for (int j = 0; j <= n; ++j)
                a[0][j] += a[i][j];
        }

        simplex0();

        if (dcmp(a[0][0])) return 0;

        vector<int> cv(1, 0), rv(1, 0);
        for (int i = 1; i <= m; ++i) {
            bool del = 0;
            if (c[n + i] > n) {
                int p = i, q = 0;
                for (int j = 1; !q && j <= n; ++j)
                    if (dcmp(a[i][j]) && c[j] <= n) q = j;
                if (!q) del = 1;
```

```
                else pivot(p, q);
            }
            if (!del) rv.push_back(i);
        }
        for (int j = 1; j <= n + m; ++j)
            if (c[j] <= n) cv.push_back(j);

        m = rv.size() - 1; n = n - m;
        vector<vector<dbl>> a1(m + 1, vector<dbl>(n + 1, 0));

        for (int j = 1; j <= n; ++j) a1[0][j] = a0[c[cv[j]]];
        for (int i = 1; i <= m; ++i)
            for (int j = 0; j <= n; ++j)
                a1[i][j] = a[rv[i]][cv[j]];

        for (int i = 1; i <= m; ++i)
            for (int j = 0; j <= n; ++j)
                a1[0][j] -= a0[c[cv[n + i]]] * a1[i][j];

        for (int i = 1; i <= m + n; ++i) cv[i] = c[cv[i]];
        swap(c, cv);
        swap(a, a1);

        if (!simplex0()) return 2;

        x[0] = -a[0][0];
        for (int i = 1; i <= m; ++i)
            x[c[n + i]] = a[i][0];

        return 1;
    }

};
```

多项式

等幂求和

```
ll invs[N], f[N], fi[N];
ll bi[N][N], be[N], ep[N][N];
ll inv(ll x) { return x == 1 ? 1 : M(inv(P % x) * (P - P / x)); }
void ginv() {
    invs[1] = 1; f[0] = fi[0] = 1;
    for (int i = 2; i != N; ++i) invs[i] = M(invs[P % i] * (P - P / i));
    for (int i = 1; i != N; ++i) f[i] = M(f[i - 1] * i);
    for (int i = 1; i != N; ++i) fi[i] = M(fi[i - 1] * invs[i]);
    // Binomial coefficient
    for (int i = 0; i != N; ++i) {
        bi[i][0] = bi[i][i] = 1;
        for (int j = 1; j < i; ++j)
            bi[i][j] = M(bi[i - 1][j - 1] + bi[i - 1][j]);
    }
    be[0] = 1;  //  Bernoulli numbers
    for (int i = 1; i != N; ++i)
        for (int j = 0; j != i; ++j)
            be[i] = M(be[i] - M(bi[i][j] * M(be[j] * invs[i - j + 1])));
    //  Equal Power Sum Coef
    for (int i = 1; i != N; ++i)
        for (int j = 0; j <= i; ++j)
            ep[i][i+1-j]=M(M(j&1?-invs[i+1]:invs[i+1])*M(bi[i + 1][j]*be[j]));
}

//  \sum_{i=1}^{n}{i^k}
ll epsum(ll n, ll k) {
    ll w = n, r = 0;
    for (int i = 1; i <= k + 1; ++i, w = M(w * n))
        r = M(r + M(w * ep[k][i]));
    return r;
}
```

初等对称多项式

对于变量集 $X=\{x_1,x_2,...,x_m\}$，其上的 n 阶初等对称多项式 σ_n 被定义为

$$\sigma_n=\sum_{|S|=n,S\subseteq X}\prod_{x_i\in S}x_i$$

特别的，恒有 $\sigma_0=1$。

即

$$\sigma_1=x_1+x_2+...+x_n$$

$$\sigma_2=x_1x_2+x_1x_3+...+x_{n-1}x_n$$

$$...$$

$$\sigma_m=x_1x_2...x_m$$

向 X 中加入元素 x' 后对 σ_k 的影响：

$$\sigma_1'=\sigma_1+x'=\sigma_1+\sigma_0x'$$

$$\sigma_2'=\sigma_2+x_1x'+x_2x'+...+x_mx'=\sigma_2+\sigma_1x'$$

$$...$$

$$\sigma_k'=\sigma_k+\sigma_{k-1}x'$$

从 X 中删去元素 x' 后对 σ_k 的影响：

$$\sigma_1'=\sigma_1-\sigma_0'x'$$

$$\sigma_2'=\sigma_2-\sigma_1'x'$$

$$\cdots$$

$$\sigma'_k = \sigma_k - \sigma'_{k-1} x'$$

由定义易得对称多项式的 OGF：

$$F(x) = \prod_{x_i \in X} (1 + x_i x)$$

若设

$$s_n = \sum_{x \in X} x^n$$

则存在恒等式（牛顿公式）：

$$\sigma_k = \frac{(-1)^{k-1}}{k} \sum_{i=0}^{k-1} (-1)^i \sigma_i s_{k-i}$$

多项式操作

```
#include <bits/stdc++.h>
#define N (1<<19)
#define P 998244353
#define M(x) (((x) + P) % P)
typedef long long ll;

using namespace std;

ll invs[N], f[N], fi[N];
ll inv(ll x) { return x == 1 ? 1 : M(inv(P % x) * (P - P / x)); }
ll binom(ll n, ll k) { return M(f[n] * M(fi[n - k] * fi[k])); }
void ginv() {
    invs[1] = 1; f[0] = fi[0] = 1;
    for (int i = 2; i != N; ++i) invs[i] = M(invs[P % i] * (P - P / i));
    for (int i = 1; i != N; ++i) f[i] = M(f[i - 1] * i);
    for (int i = 1; i != N; ++i) fi[i] = M(fi[i - 1] * invs[i]);
}

ll qp(ll a, ll b) {
    ll r = 1;
    do if (b & 1) r = M(r * a);
    while (a = M(a * a), b >>= 1);
    return r;
}

ll inv(ll x, ll p) { return x == 1 ? 1 : inv(p % x, p) * (p - p / x) % p; }
ll qpm(ll a, ll b, ll q) {
    ll r = 1;
    do if (b & 1) r = r * a % q;
    while (a = a * a % q, b >>= 1);
    return r;
}
```

```
ll msqrt(ll n, ll p) {
    if (!n) return 0;
    ll q = p - 1, s = 0, z = 2;
    //while (~q & 1) q >>= 1, s++;
    q >>= (s = __builtin_ctzll(q));
    if (s == 1) return qpm(n, (p + 1) / 4, p);
    while(qpm(z, (p - 1) / 2, p) == 1) ++z;
    ll c = qpm(z, q, p), t = qpm(n, q, p),
       r = qpm(n, (q + 1) / 2, p), m = s;
    while(t % p != 1) {
        ll i = 1; while(qpm(t, 1ll << i, p) != 1) ++i;
        ll b = qpm(c, 1ll << (m - i - 1), p);
        r = r * b % p; c = (b * b) % p;
        t = (t * c) % p; m = i;
    }
    return min(r, p - r); //    r^2=(p-r)^2=n
}

typedef unsigned long long ull;
typedef long double ld;
ull mul(ull a, ull b, ull p) {
    return ((__int128)a * b) % p;
    ll res = a * b - p * (ull)((ld)a * (ld)b / (ld)p);
    return res + p * (res < 0) - p * (res >= (ll)p);
}

namespace poly {

    istream& operator>>(istream& is, vector<ll>& p) {
        for (ll& w : p) is >> w;
        return is;
    }

    ostream& operator<<(ostream& os, const vector<ll>& p) {
        for (ll w : p) os << w << ' ';
        return os;
    }

    int fr[N], fs;
    ll pa[N], pb[N], pc[N], pd[3][N];

    void init(int s) {
        for (fs = 1; fs < s; fs <<= 1);
        for (int i = 0; i != fs; ++i)
            fr[i] = (fr[i >> 1] >> 1) | (i & 1 ? (fs >> 1) : 0);
    }

    vector<ll> add(const vector<ll>& p1, const vector<ll>& p2) {
        int n1 = p1.size(), n2 = p2.size(), n3 = max(n1, n2);
        vector<ll> pr(n3, 0);
        for (int i = 0; i != n3; ++i) {
            if (i < n1) pr[i] = M(pr[i] + p1[i]);
            if (i < n2) pr[i] = M(pr[i] + p2[i]);
        }
        return pr;
    }
    vector<ll> sub(const vector<ll>& p1, const vector<ll>& p2) {
        int n1 = p1.size(), n2 = p2.size(), n3 = max(n1, n2);
        vector<ll> pr(n3);
        for (int i = 0; i != n3; ++i) {
            if (i < n1) pr[i] = M(pr[i] + p1[i]);
            if (i < n2) pr[i] = M(pr[i] - p2[i]);
        }
        return pr;
    }

    vector<ll> mul(const vector<ll>& p1, ll k) {
        int n1 = p1.size();
        vector<ll> p2(n1);
        for (int i = 0; i != n1; ++i) p2[i] = M(k * p1[i]);
        return p2;
    }

//  MTT
//    void ntt(ll* a, int f, ll q) {
//        static const ll g = 3;
//        for (int i = 0; i != fs; ++i) if (i < fr[i]) swap(a[i], a[fr[i]]);
//        for (int i = 1; i != fs; i <<= 1) {
//            ll e = (q - 1) / (i << 1), w0 = qpm(g, f == 1 ? e : q - 1 - e, q);
//            for (int j = 0; j != fs; j += (i << 1)) {
//                ll w = 1;
//                for (int k = 0; k != i; k++, w = w * w0 % q) {
//                    ll u = a[j + k], v = w * a[i + j + k] % q;
//                    a[j + k] = (u + v) % q; a[i + j + k] = (u - v + q) % q;
//                }
//            }
//        }
//        if (f == -1) {
//            ll d = inv(fs, q);
//            for (ll i = 0; i != fs; ++i)
//                a[i] = a[i] * d % q;
//        }
//    }
//    vector<ll> mul(const vector<ll>& p1, const vector<ll>& p2, int n = 0) {
//        ll q[4] = { 469762049, 998244353, 1004535809 }; q[3] = q[0] * q[1];
//        int n1 = p1.size(), n2 = p2.size(), n3 = n1 + n2 - 1;
//        init(n3 + 1); vector<ll> pr(n3);
//        for (int j = 0; j != 3; ++j) {
//            copy_n(p1.begin(), n1, pa); fill(pa + n1, pa + fs, 0);
//            copy_n(p2.begin(), n2, pb); fill(pb + n2, pb + fs, 0);
//            ntt(pa, 1, q[j]); ntt(pb, 1, q[j]);
//            for (int i = 0; i != fs; ++i) pd[j][i] = pa[i] * pb[i] % q[j];
//            ntt(pd[j], -1, q[j]);
//        }
//        ll i1 = inv(q[1] % q[0], q[0]), i2 = inv(q[0] % q[1], q[1]),
//                i3 = inv(q[3] % q[2], q[2]);
//        for (int i = 0; i != n3; ++i) {
//            ll A = (::mul(pd[0][i] * q[1] % q[3], i1, q[3])
//                    + ::mul(pd[1][i] * q[0] % q[3], i2, q[3])) % q[3];
//            ll k = (((pd[2][i] - A) % q[2] + q[2]) % q[2]) * i3 % q[2];
//            pr[i] = M(M(M(k) * M(q[3])) + M(A));
//          }
//          if (n) pr.resize(n, 0);
//          return pr;
//      }

    void ntt(ll* p, int f) {
        static const ll g = 3;
        for (int i = 0; i != fs; ++i) if (i < fr[i]) swap(p[i], p[fr[i]]);
        for (int i = 1; i != fs; i <<= 1) {
            ll e = (P - 1) / (i << 1), w0 = qp(g, f == 1 ? e : P - 1 - e);
            for (int j = 0; j != fs; j += (i << 1)) {
                ll w = 1;
                for (int k = 0; k != i; k++, w = M(w * w0)) {
                    ll u = p[j + k], v = M(w * p[i + j + k]);
                    p[j + k] = M(u + v); p[i + j + k] = M(u - v);
                }
            }
        }
        if (f == -1)
            for (ll i = 0; i != fs; ++i)
                p[i] = M(p[i] * invs[fs]);
    }

    vector<ll> mul(const vector<ll>& p1, const vector<ll>& p2, int n = 0) {
        int n1 = p1.size(), n2 = p2.size(), n3 = n1 + n2 - 1;
        init(n3 + 1); vector<ll> pr(n3);
        copy_n(p1.begin(), n1, pa); fill(pa + n1, pa + fs, 0);
        copy_n(p2.begin(), n2, pb); fill(pb + n2, pb + fs, 0);
        ntt(pa, 1); ntt(pb, 1);
        for (int i = 0; i != fs; ++i) pc[i] = M(pa[i] * pb[i]);
        ntt(pc, -1); copy(pc, pc + n3, pr.begin());
        if (n) pr.resize(n, 0);
        return pr;
    }


    vector<ll> inv(const vector<ll>& p1) {
        int n1 = p1.size(), n2 = (n1 + 1) >> 1;
        if (n1 == 1) return { ::inv(p1[0]) };
        else {
            vector<ll> p2 = inv(vector<ll>(p1.begin(), p1.begin() + n2));
            return sub(mul(p2, 2), mul(p1, mul(p2, p2, n1), n1));
        }
    }


    pair<vector<ll>, vector<ll>> div(const vector<ll>& p1, const vector<ll>& p2) {
        int n1 = p1.size(), n2 = p2.size(), n3 = n1 - n2 + 1;
        vector<ll> p1r = p1, p2r = p2;
        reverse(p1r.begin(), p1r.end());
        reverse(p2r.begin(), p2r.end());
        p1r.resize(n3, 0); p2r.resize(n3, 0);
        vector<ll> p3 = mul(p1r, inv(p2r), n3);
        reverse(p3.begin(), p3.end());
        vector<ll> p4 = sub(p1, mul(p2, p3));
        p4.resize(n2 - 1, 0);
        return { p3, p4 };
    }

    vector<ll> deriv(const vector<ll>& p1) {
        int n1 = p1.size();
        vector<ll> p2(n1 - 1);
        for (int i = 1; i != n1; ++i) p2[i - 1] = M(i * p1[i]);
        return p2;
    }

    vector<ll> integ(const vector<ll>& p1) {
        int n1 = p1.size();
        vector<ll> p2(n1 + 1, 0);
        for (int i = 0; i != n1; ++i) p2[i + 1] = M(p1[i]*invs[i + 1]);
        return p2;
    }

    vector<ll> log(const vector<ll>& p1) {
        return integ(mul(deriv(p1), inv(p1), p1.size() - 1));
    }

    vector<ll> exp(const vector<ll>& p1) {
        if (p1.size() == 1) return { 1 };
        else {
            vector<ll> p2 = exp({p1.begin(),p1.begin()+(p1.size()+1>>1)});
            p2.resize(p1.size(), 0);
            return mul(p2, add(sub({ 1 }, log(p2)), p1), p1.size());
        }
    }

    vector<ll> sqrt(const vector<ll>& p1) {
        int n1 = p1.size(), n2 = (n1 + 1) >> 1;
        if (n1 == 1) return { ::msqrt(p1[0], P) };
        else {
            vector<ll> p2 = sqrt(vector<ll>(p1.begin(), p1.begin() + n2));
            vector<ll> p3 = mul(p2, 2); p3.resize(n1);
            p3 = inv(p3);
            return mul(add(mul(p2, p2, n1), p1), p3, n1);
        }
    }
    //  k mod P, not P - 1
    vector<ll> pow(const vector<ll>& p1, int k) {
        int n1 = p1.size(), n2 = n1;
        while (n2 && !p1[n1 - n2]) n2--;
        int n3 = max(n1 - 1ll * (n1 - n2) * k, 0ll);
        if (!n2 || !n3) return vector<ll>(n1, 0);
        vector<ll> p2(p1.begin() + n1 - n2, p1.begin() + n1 - n2 + n3);
        ll c = p2[0]; p2 = mul(exp(mul(log(mul(p2, ::inv(c))), k)), qp(c, k));
        p2.resize(n1, 0); rotate(p2.begin(), p2.begin() + n3, p2.end());
        return p2;
    }


    //  f[i] = \sum f[i-j]g[j]
    vector<ll> conv(const vector<ll>& g) {
        return inv(sub({ 1 }, g));
    }

    vector<ll> egf(const vector<ll>& g) {
        vector<ll> r(g.size());
        for (int i = 0; i != g.size(); ++i)
            r[i] = M(g[i] * fi[i]);
        return r;
    }

    vector<ll> iegf(const vector<ll>& g) {
        vector<ll> r(g.size());
        for (int i = 0; i != g.size(); ++i)
            r[i] = M(g[i] * f[i]);
        return r;
    }

}

using namespace poly;

vector<ll> catalan(int n) {
    vector<ll> a = { 1, P - 4 }; a.resize(n);
    a = sub({1}, sqrt(a)); a.erase(a.begin());
    return mul(a, inv(2));
}

vector<ll> bell(int n) {
    vector<ll> a(n, 0); a[1] = 1;
    return iegf(exp(sub(exp(a), {1})));
}

vector<ll> bernoulli(int n) {
    vector<ll> a(n + 1, 1);
    a = sub(egf(a), { 1 });
    rotate(a.begin(), a.begin() + 1, a.end());
    a.pop_back();
    return iegf(inv(a));
}

ll epsum(const vector<ll>& b, ll n, ll k) {
    n = M(n);
    ll ans = 0; ll w = M(n);
    for (int i = 1; i <= k + 1; ++i) {
        ll t = M(binom(k + 1, k + 1 - i) * b[k + 1 - i]);
        if ((k + 1 - i) & 1) ans = M(ans - M(w * t));
        else ans = M(ans + M(w * t));
        w = M(w * n);
    }
    return M(ans * invs[k + 1]);
}

vector<ll> connected_graph(int n) {
    vector<ll> a(n);
    for (int i = 0; i != n; ++i)
        a[i] = qp(2, 1ll * i * (i - 1) / 2);
    return iegf(log(egf(a)));
```

```
}

vector<ll> eulerian_graph(int n) {
    vector<ll> a(n);
    a[0] = 1;
    for (int i = 1; i != n; ++i)
        a[i] = qp(2, 1ll * (i - 1) * (i - 2) / 2);
    return iegf(log(egf(a)));
}

vector<ll> colored_bipartite(int n) {
    vector<ll> b1(n), b2(n), c;
    int sqrt2 = msqrt(2, P);
    for (int i = 0; i != n; ++i) {
        b1[i] = qp(sqrt2, 1ll * i * i);
        b2[i] = inv(b1[i]);
    }
    b1 = iegf(b1);
    b2 = egf(b2);
    c = mul(b2, b2, n);
    for (int i = 0; i != n; ++i) c[i] = M(c[i] * b1[i]);
    return c;
}

vector<ll> bipartite(int n) {
    return iegf(sqrt(egf(colored_bipartite(n))));
}

vector<ll> connected_bipartite(int n) {
    return iegf(mul(log(egf(colored_bipartite(n))),invs[2]));
}
```

线性基

```
typedef bitset<W> bv;
typedef array<bv, W> bs;

bool ins(bs& b, bv v, bool flg = 1) {
    for (int i = W - 1; i >= 0; --i) {
        if (!v[i]) continue;
        if (!b[i][i]) {
            if (flg) b[i] = v;
            return true;
        }
        else v ^= b[i];
    }
    return false;
}

bs bunion(const bs& b1, const bs& b2) {
    bs r = b1;
    for (int i = 0; i != W; ++i)
        if (b2[i].any()) ins(r,b2[i]);
    return r;
}
```

```
bs bitsc(const bs& b1, const bs& b2) {
    bs b = b1, t, r;
    for (int i = W - 1; i >= 0; --i) {
        if (!b2[i][i]) continue;
        bv v = b2[i], k; k[i] = 1; bool f = 1;
        for (int j = W - 1; j >= 0; --j) {
            if (!v[j]) continue;
            if (b[j].any()) v ^= b[j], k ^= t[j];
            else { b[j] = v; t[j] = k; f = 0; break; }
        }
        if (!f) continue;
        bv w;
        for (int j = 0; j != W; ++j)
            if (k[j]) w ^= b2[j];
        ins(r, w);
    }
    return r;
}

bv qmax(const bs& b) {
    bv v;
    for (int i = W - 1; i >= 0; --i)
        if (!v[i] && b[i][i]) v ^= b[i];
    return v;
}
```

快速变换与卷积

FFT

```
typedef double dbl;
typedef complex<dbl> cplx;
const dbl pi = acos(-1);

int fr[N], fs;
void init(int s) {
    for (fs = 1; fs <= s; fs <<= 1);
    for (int i = 0; i != fs; ++i)
        fr[i] = (fr[i >> 1] >> 1) | (i & 1 ? (fs >> 1) : 0);
}

void fft(cplx* p, int f) {
    for (int i = 0; i != fs; ++i) if (i < fr[i]) swap(p[i], p[fr[i]]);
    for (int i = 1; i != fs; i <<= 1) {
        cplx w0{ cos(pi / i), f * sin(pi / i) };
        for (int j = 0; j != fs; j += (i << 1)) {
            cplx w{ 1, 0 };
            for (int k = 0; k != i; k++) {
                cplx u = p[j + k], v = w * p[i + j + k];
                p[j + k] = u + v; p[i + j + k] = u - v;
                w *= w0;
            }
        }
    }
}
```

```
void fft_res(cplx* p) {
    for (int i = 0; i != fs; ++i)
        p[i] = p[i] * (1. / fs);
}

cplx p1[N], p2[N], p3[N];
int conv(int* a, int n, int* b, int m, int* c) {
    init(n + m + 1);
    for (int i = 0; i <= n; ++i) p1[i] = a[i];
    fill(p1 + n + 1, p1 + fs, cplx{});
    for (int i = 0; i <= m; ++i) p2[i] = b[i];
    fill(p2 + m + 1, p2 + fs, cplx{});
    fft(p1, 1); fft(p2, 1);
    for (int i = 0; i != fs; ++i) p3[i] = p1[i] * p2[i];
    fft(p3, -1); fft_res(p3);
    for (int i = 0; i != fs; ++i) c[i] = round(p3[i].real());
    return fs;
}
```

NTT

FWT

$$h(U) = \sum_{S \circ T = U} f(S)g(T)$$

FWT	&	|	^	IFWT	&	|	^
a[i+j]=	x+y	x	x+y	a[i+j]=	x-y	x	(x+y)/2
a[i+j+k]=	y	x+y	x-y	a[i+j+k]=	y	y-x	(x-y)/2

```
void fwt(ll* a, int n) {
    for (int k = 1; k < n; k <<= 1)
        for (int m = k << 1, i = 0; i < n; i += m)
            for (int j = 0; j != k; j++) {
                ll x = a[i + j], y = a[i + j + k];
                //a[i + j] = ...
                //a[i + j + k] = ...
            }
}
```

FZT/FMT

FZT(ζ 变换):

$$g(S) = \sum_{T \subseteq S} f(T)$$

FMT(μ 变换):

$$g(S) = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T)$$

```
void fzt(ll* f, ll* g, int w) {
    copy_n(f, 1 << w, g);
    for (int i = 0; i != w; ++i)
        for (int j = 0; j != (1 << w); ++j)
            if (j & (1 << i)) g[j] += g[j ^ (1 << i)];
}
```

```
void fmt(ll* f, ll* g, int w) {
    copy_n(f, 1 << w, g);
    for (int i = 0; i != w; ++i)
        for (int j = 0; j != (1 << w); ++j)
            if (j & (1 << i)) g[j] -= g[j ^ (1 << i)];
}
```

概率

随机游走

有限图上的随机游走　定义：

1. p_{uv} 表示当前在 u，下一步在 v 的概率。
2. $p_{uv}^{(k)}$ 表示当前在 u，k 步后在 v 的概率。
3. $p_{uv}^{(k)} = \sum_{w\in N(u)} p_{uw}p_{wv}^{k-1}$
4. 转移矩阵 P 的第 i 行第 j 列元素为 p_{ji}。
5. k 步转移矩阵 P^k 第 i 行第 j 列元素为 $p_{ji}^{(k)}$。
6. $X^{(0)}$ 为初始态列向量，$X_u^{(0)}$ 为初始状态时点在 u 的概率。
7. $X^{(k)} = P^k X^{(0)}$ 为进行 k 次转移后的状态。

期望问题：计算从一些点（给定概率）出发到达某个终末点 t 的期望步数/消耗。

注：在这个问题中，无论原图如何，都需要将 t 点的出边删去并指向虚空，即 $\forall u \in V, p_{tu} = 0$。

但特别的，有 $p_{tt}^{(0)} = 1$

设 $P(u)$ 为从 u 出发，经过无限步后到达 t 的概率。特别的 $P(t) = 1$。

$$
\begin{aligned}
P(u) = \sum_{k=1}^{\infty} p_{ut}^{(k)} &= \sum_{k=1}^{\infty}\sum_{v\in N(u)} p_{uv}p_{vt}^{(k-1)} = \sum_{v\in N(u)} p_{uv}\sum_{k=1}^{\infty} p_{vt}^{(k-1)} \\
&= p_{vt} + \sum_{t\neq v\in N(u)} p_{uv}P(v) = \sum_{v\in N(u)} p_{uv}P(v)
\end{aligned}
$$

设 $E(u)$ 为从 u 出发，到达 t 的期望转移次数。特别的，$E(t) = 0$。

$$
E(u) = \sum_{k=1}^{\infty} kp_{ut}^{(k)} = \sum_{k=1}^{\infty} k\sum_{v\in N(u)} p_{uv}p_{vt}^{(k-1)} = \sum_{v\in N(u)} p_{uv}\sum_{k=1}^{\infty} kp_{vt}^{(k-1)}
$$

$$
= p_{ut} + \sum_{t\neq v\in N(u)} p_{uv}\sum_{k=1}^{\infty}(k+1)p_{vt}^{(k)} = p_{ut} + \sum_{t\neq v\in N(u)} p_{uv}\left(\sum_{k=1}^{\infty} kp_{vt} + \sum_{k=1}^{\infty} p_{vt}\right)
$$

$$
= p_{ut} + \sum_{t\neq v\in N(u)} p_{uv}(E(v)+P(v)) = \sum_{v\in N(u)} p_{uv}(P(v)+E(v))
$$

设 $F(u)$ 为从 u 出发，到达 t 的期望消耗。其中第 k 步的消耗为 k。特别的，$F(t) = 0$。

$$
F(u) = \sum_{k=1}^{\infty}\frac{k(k+1)}{2}p_{ut}^{(k)} = \sum_{k=1}^{\infty}\frac{k(k+1)}{2}\sum_{v\in N(u)} p_{uv}p_{vt}^{(k-1)} = \sum_{v\in N(u)} p_{uv}\sum_{k=1}^{\infty}\frac{k(k+1)}{2}p_{vt}^{(k-1)}
$$

$$
= p_{ut} + \sum_{n\neq v\in N(u)} p_{uv}\sum_{k=1}^{\infty}\frac{k(k+1)}{2}p_{vt}^{(k-1)} = p_{ut} + \sum_{n\neq v\in N(u)} p_{uv}\sum_{k=1}^{\infty}\frac{(k+1)(k+2)}{2}p_{vt}^{(k)}
$$

$$= p_{ut} + \sum_{n \neq v \in N(u)} p_{uv} \sum_{k=1}^{\infty} \left[\frac{k(k+1)}{2} + k + 1 \right] p_{vt}^{(k)} = p_{ut} + \sum_{n \neq v \in N(u)} p_{uv} \left[\sum_{k=1}^{\infty} \frac{k(k+1)}{2} p_{vt}^{(k)} + \sum_{k=1}^{\infty} k p_{vt}^{(k)} + \sum_{k=1}^{\infty} p_{vt}^{(k)} \right]$$

$$= p_{ut} + \sum_{n \neq v \in N(u)} p_{uv} [F(v) + E(v) + P(v)] = \sum_{v \in N(u)} p_{uv} [F(v) + E(v) + P(v)]$$

推广：

设 $F_m(u)$ 表示从 u 出发，到达 t 的期望消耗，其中第 k 步消耗为 k^m。特别的，$F_m(t) = 0$，$F_0(u) = P(u)$，$F_1(u) = E(u)$。

$$F_m(u) = \sum_{v \in N(u)} p_{uv} \sum_{k=1}^{\infty} k^m p_{vt}^{(k-1)} = p_{ut} + \sum_{t \neq v \in N(u)} p_{uv} \sum_{k=1}^{\infty} (k+1)^m p_{vt}$$

$$= p_{ut} + \sum_{t \neq v \in N(u)} p_{uv} \sum_{k=1}^{\infty} \sum_{i=0}^{m} \binom{m}{i} k^i p_{vt} = p_{ut} + \sum_{t \neq v \in N(u)} p_{uv} \sum_{i=0}^{m} \binom{m}{i} \sum_{k=1}^{\infty} k^i p_{vt}$$

$$= \sum_{v \in N(u)} p_{uv} \sum_{i=0}^{m} \binom{m}{i} F_i(v)$$

设 $E_P(u)$ 表示从 u 出发，到达 t 的期望消耗，其中第 k 步消耗为 $P(k) = \sum_{i=0}^{\infty} a_i k^i$。特别的，$E_P(t) = 0$。

$$E_P(u) = \sum_{k=1}^{\infty} \sum_{i=0}^{\infty} \sum_{v \in N(u)} a_i k^i p_{uv} p_{vt}^{(k-1)} = \sum_{v \in N(u)} p_{uv} \sum_{k=1}^{\infty} \sum_{i=0}^{\infty} a_i k^i p_{vt}^{(k-1)}$$

$$= \sum_{v \in N(u)} p_{uv} \sum_{i=0}^{\infty} a_i \sum_{k=1}^{\infty} (k+1)^i p_{vt}^{(k)} = \sum_{v \in N(u)} p_{uv} \sum_{i=0}^{\infty} a_i \sum_{j=0}^{i} \binom{i}{j} \sum_{k=1}^{\infty} k^j p_{vt}^k$$

$$= \sum_{v \in N(u)} p_{uv} \sum_{j=0}^{\infty} F_j(v) \sum_{i=j}^{\infty} a_i \binom{i}{j}$$

无限图上的随机游走

一维直线上的随机游走　无限制：

从原点出发，每次必须往左/右走一步，则时刻 t 在位置 x 的概率为 0 当且仅当 $x > t$ 或 $2 \nmid (x+t)$，否则为 $\frac{1}{2^t}\binom{t}{\frac{t-x}{2}}$。

从原点出发，

有限制：

从原点出发，每次可以选择往左右走，但不允许越过原点。默慈金数（允许不动）/卡塔兰数（必须动）。

二维平面网格上的随机游走　无限制：

从原点出发，每次必须往四个方向 $(-1,0), (1,0), (0,-1), (0,1)$ 等概率走一步：

进行坐标变换 $(u,v) = (x+y, x-y)$，则相当于每次必须往 $(-1,-1), (-1,1), (1,-1), (1,1)$ 等概率走一步，发现两维互相独立，于是时刻 t 在位置 (u,v) 的概率为 0 当且仅当 $u > t$ 或 $v > t$ 或 $2 \nmid u$ 或 $2 \nmid v$，否则为 $\frac{1}{2^{2t}}\binom{t}{\frac{t-u}{2}}\binom{t}{\frac{t-v}{2}}$。逆变换得 $\frac{1}{2^{2t}}\binom{t}{\frac{t-x-y}{2}}\binom{t}{\frac{t-x+y}{2}}$。

有限制：

考虑通过坐标变换转化为两维互相独立的一维随机游走。

更高维的情况　三维空间中，从原点出发，每次必须往八个方向

$$(-1,0,0),(1,0,0),(0,-1,0),(0,1,0),(0,0,-1),(0,0,1),(-1,-1,-1),(1,1,1)$$

等概率走一步

进行坐标变换 $(u,v,w)=(x+y-z,y+z-x,z+x-y)$

注：有逆变换 $(x,y,z)=(\frac{u+w}{2},\frac{u+v}{2},\frac{v+w}{2})$

方向	$(1,1,1)$	$(0,1,0)$	$(1,0,0)$	$(0,0,-1)$	$(0,0,1)$	$(-1,0,0)$	$(0,-1,0)$	$(-1,-1,-1)$
u	1	1	1	1	-1	-1	-1	-1
v	1	1	-1	-1	1	1	-1	-1
w	1	-1	1	-1	1	-1	1	-1

观察到三维互相独立，因此 t 时刻在 (x,y,z) 的概率为

$$\frac{1}{2^{3t}}\binom{t}{\frac{t-(x+y-z)}{2}}\binom{t}{\frac{t-(y+z-x)}{2}}\binom{t}{\frac{t-(z+x-y)}{2}}$$

注意无法整除时为 0。

高精度类

高精度整数

```cpp
#define pb push_back
#define mp make_pair
typedef pair<int,int> pii;

typedef double ld;
typedef vector<int> vi;
#define fi first
#define se second

// base and base_digits must be consistent
constexpr int base = 1000000000;
constexpr int base_digits = 9;

struct bigint {
  // value == 0 is represented by empty z
  vector<int> z;  // digits

  // sign == 1 <==> value >= 0
  // sign == -1 <==> value < 0
  int sign;

  bigint() : sign(1) {}

  bigint(long long v) { *this = v; }

  bigint& operator=(long long v) {
    sign = v < 0 ? -1 : 1;
    v *= sign;
    z.clear();
    for (; v > 0; v = v / base) z.push_back((int)(v % base));
    return *this;
```

```
}

bigint(const string& s) { read(s); }

bigint& operator+=(const bigint& other) {
  if (sign == other.sign) {
    for (int i = 0, carry = 0; i < other.z.size() || carry; ++i) {
      if (i == z.size()) z.push_back(0);
      z[i] += carry + (i < other.z.size() ? other.z[i] : 0);
      carry = z[i] >= base;
      if (carry) z[i] -= base;
    }
  } else if (other != 0 /* prevent infinite loop */) {
    *this -= -other;
  }
  return *this;
}

friend bigint operator+(bigint a, const bigint& b) { return a += b; }

bigint& operator-=(const bigint& other) {
  if (sign == other.sign) {
    if (sign == 1 && *this >= other || sign == -1 && *this <= other) {
      for (int i = 0, carry = 0; i < other.z.size() || carry; ++i) {
        z[i] -= carry + (i < other.z.size() ? other.z[i] : 0);
        carry = z[i] < 0;
        if (carry) z[i] += base;
      }
      trim();
    } else {
      *this = other - *this;
      this->sign = -this->sign;
    }
  } else {
    *this += -other;
  }
  return *this;
}

friend bigint operator-(bigint a, const bigint& b) { return a -= b; }

bigint& operator*=(int v) {
  if (v < 0) sign = -sign, v = -v;
  for (int i = 0, carry = 0; i < z.size() || carry; ++i) {
    if (i == z.size()) z.push_back(0);
    long long cur = (long long)z[i] * v + carry;
    carry = (int)(cur / base);
    z[i] = (int)(cur % base);
  }
  trim();
  return *this;
}

bigint operator*(int v) const { return bigint(*this) *= v; }

friend pair<bigint, bigint> divmod(const bigint& a1, const bigint& b1) {
  int norm = base / (b1.z.back() + 1);
  bigint a = a1.abs() * norm;
  bigint b = b1.abs() * norm;
  bigint q, r;
  q.z.resize(a.z.size());

  for (int i = (int)a.z.size() - 1; i >= 0; i--) {
    r *= base;
    r += a.z[i];
    int s1 = b.z.size() < r.z.size() ? r.z[b.z.size()] : 0;
    int s2 = b.z.size() - 1 < r.z.size() ? r.z[b.z.size() - 1] : 0;
    int d = (int)(((long long)s1 * base + s2) / b.z.back());
    r -= b * d;
    while (r < 0) r += b, --d;
    q.z[i] = d;
  }

  q.sign = a1.sign * b1.sign;
  r.sign = a1.sign;
  q.trim();
  r.trim();
  return {q, r / norm};
}

friend bigint sqrt(const bigint& a1) {
  bigint a = a1;
  while (a.z.empty() || a.z.size() % 2 == 1) a.z.push_back(0);

  int n = a.z.size();

  int firstDigit = (int)::sqrt((double)a.z[n - 1] * base + a.z[n - 2]);
  int norm = base / (firstDigit + 1);
  a *= norm;
  a *= norm;
  while (a.z.empty() || a.z.size() % 2 == 1) a.z.push_back(0);

  bigint r = (long long)a.z[n - 1] * base + a.z[n - 2];
  firstDigit = (int)::sqrt((double)a.z[n - 1] * base + a.z[n - 2]);
  int q = firstDigit;
  bigint res;

  for (int j = n / 2 - 1; j >= 0; j--) {
    for (;; --q) {
      bigint r1 =
          (r - (res * 2 * base + q) * q) * base * base +
          (j > 0 ? (long long)a.z[2 * j - 1] * base + a.z[2 * j - 2] : 0);
      if (r1 >= 0) {
        r = r1;
        break;
      }
    }
    res *= base;
    res += q;

    if (j > 0) {
      int d1 = res.z.size() + 2 < r.z.size() ? r.z[res.z.size() + 2] : 0;
      int d2 = res.z.size() + 1 < r.z.size() ? r.z[res.z.size() + 1] : 0;
      int d3 = res.z.size() < r.z.size() ? r.z[res.z.size()] : 0;
      q = (int)(((long long)d1 * base * base + (long long)d2 * base + d3) /
                (firstDigit * 2));
    }
  }

  res.trim();
  return res / norm;
}

bigint operator/(const bigint& v) const { return divmod(*this, v).first; }

bigint operator%(const bigint& v) const { return divmod(*this, v).second; }

bigint& operator/=(int v) {
  if (v < 0) sign = -sign, v = -v;
  for (int i = (int)z.size() - 1, rem = 0; i >= 0; --i) {
    long long cur = z[i] + rem * (long long)base;
    z[i] = (int)(cur / v);
    rem = (int)(cur % v);
  }
  trim();
  return *this;
}

bigint operator/(int v) const { return bigint(*this) /= v; }

int operator%(int v) const {
  if (v < 0) v = -v;
  int m = 0;
  for (int i = (int)z.size() - 1; i >= 0; --i)
    m = (int)((z[i] + m * (long long)base) % v);
  return m * sign;
}

bigint& operator*=(const bigint& v) {
  *this = *this * v;
  return *this;
}

bigint& operator/=(const bigint& v) {
  *this = *this / v;
  return *this;
}

bool operator<(const bigint& v) const {
  if (sign != v.sign) return sign < v.sign;
  if (z.size() != v.z.size()) return z.size() * sign < v.z.size() * v.sign;
  for (int i = (int)z.size() - 1; i >= 0; i--)
    if (z[i] != v.z[i]) return z[i] * sign < v.z[i] * sign;
  return false;
}

bool operator>(const bigint& v) const { return v < *this; }
```

```cpp
bool operator<=(const bigint& v) const { return !(v < *this); }

bool operator>=(const bigint& v) const { return !(*this < v); }

bool operator==(const bigint& v) const {
  return !(*this < v) && !(v < *this);
}

bool operator!=(const bigint& v) const { return *this < v || v < *this; }

void trim() {
  while (!z.empty() && z.back() == 0) z.pop_back();
  if (z.empty()) sign = 1;
}

bool isZero() const { return z.empty(); }

friend bigint operator-(bigint v) {
  if (!v.z.empty()) v.sign = -v.sign;
  return v;
}

bigint abs() const { return sign == 1 ? *this : -*this; }

long long longValue() const {
  long long res = 0;
  for (int i = (int)z.size() - 1; i >= 0; i--) res = res * base + z[i];
  return res * sign;
}

friend bigint gcd(const bigint& a, const bigint& b) {
  return b.isZero() ? a : gcd(b, a % b);
}

friend bigint lcm(const bigint& a, const bigint& b) {
  return a / gcd(a, b) * b;
}

void read(const string& s) {
  sign = 1;
  z.clear();
  int pos = 0;
  while (pos < s.size() && (s[pos] == '-' || s[pos] == '+')) {
    if (s[pos] == '-') sign = -sign;
    ++pos;
  }
  for (int i = (int)s.size() - 1; i >= pos; i -= base_digits) {
    int x = 0;
    for (int j = max(pos, i - base_digits + 1); j <= i; j++)
      x = x * 10 + s[j] - '0';
    z.push_back(x);
  }
  trim();
}

friend istream& operator>>(istream& stream, bigint& v) {
  string s;
  stream >> s;
  v.read(s);
  return stream;
}

friend ostream& operator<<(ostream& stream, const bigint& v) {
  if (v.sign == -1) stream << '-';
  stream << (v.z.empty() ? 0 : v.z.back());
  for (int i = (int)v.z.size() - 2; i >= 0; --i)
    stream << setw(base_digits) << setfill('0') << v.z[i];
  return stream;
}

static vector<int> convert_base(const vector<int>& a, int old_digits,
                                int new_digits) {
  vector<long long> p(max(old_digits, new_digits) + 1);
  p[0] = 1;
  for (int i = 1; i < p.size(); i++) p[i] = p[i - 1] * 10;
  vector<int> res;
  long long cur = 0;
  int cur_digits = 0;
  for (int v : a) {
    cur += v * p[cur_digits];
    cur_digits += old_digits;
    while (cur_digits >= new_digits) {
      res.push_back(int(cur % p[new_digits]));
      cur /= p[new_digits];
      cur_digits -= new_digits;
    }
  }
  res.push_back((int)cur);
  while (!res.empty() && res.back() == 0) res.pop_back();
  return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll& a, const vll& b) {
  int n = a.size();
  vll res(n + n);
  if (n <= 32) {
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++) res[i + j] += a[i] * b[j];
    return res;
  }

  int k = n >> 1;
  vll a1(a.begin(), a.begin() + k);
  vll a2(a.begin() + k, a.end());
  vll b1(b.begin(), b.begin() + k);
  vll b2(b.begin() + k, b.end());

  vll a1b1 = karatsubaMultiply(a1, b1);
  vll a2b2 = karatsubaMultiply(a2, b2);
  for (int i = 0; i < k; i++) a2[i] += a1[i];
  for (int i = 0; i < k; i++) b2[i] += b1[i];

  vll r = karatsubaMultiply(a2, b2);
  for (int i = 0; i < a1b1.size(); i++) r[i] -= a1b1[i];
  for (int i = 0; i < a2b2.size(); i++) r[i] -= a2b2[i];

  for (int i = 0; i < r.size(); i++) res[i + k] += r[i];
  for (int i = 0; i < a1b1.size(); i++) res[i] += a1b1[i];
  for (int i = 0; i < a2b2.size(); i++) res[i + n] += a2b2[i];
  return res;
}

bigint operator*(const bigint& v) const {
  vector<int> a6 = convert_base(this->z, base_digits, 6);
  vector<int> b6 = convert_base(v.z, base_digits, 6);
  vll a(a6.begin(), a6.end());
  vll b(b6.begin(), b6.end());
  while (a.size() < b.size()) a.push_back(0);
  while (b.size() < a.size()) b.push_back(0);
  while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);
  vll c = karatsubaMultiply(a, b);
  bigint res;
  res.sign = sign * v.sign;
  for (int i = 0, carry = 0; i < c.size(); i++) {
    long long cur = c[i] + carry;
    res.z.push_back((int)(cur % 1000000));
    carry = (int)(cur / 1000000);
  }
  res.z = convert_base(res.z, 6, base_digits);
  res.trim();
  return res;
}
};
```

高精度实数

```
#define rep(i,l,r) for(int i=l;i<=r;++i)
#define per(i,r,l) for(int i=r;i>=l;--i)
template <typename T> void chmin(T&x,const T &y) { if(x>y)x=y; }
template <typename T> void chmax(T&x,const T &y) { if(x<y)x=y; }

const int W=1e8,B=20,T=60;
struct db {
    int a[T],n;
    bool is_neg;
    int& operator [](int x) { return a[x]; }
    int operator [](int x) const { return a[x]; }
    void left_move(int l) {
        per(i,n-1,0)a[i+l]=a[i];
        rep(i,0,l-1)a[i]=0; n+=l;
    }
    db (int x=0) {
        memset(a,0,sizeof(a));n=B; is_neg=0;
        if(x<0){is_neg=1;x=-x;}
        while(x){a[n++]=x%W;x/=W;}
    }
    void print(char c='\n')const {
        if(is_neg)putchar('-');
        if(n>B) {
            printf("%d",a[n-1]);
            per(i,n-2,B)printf("%.8d",a[i]);
        }
        else printf("0");
        printf(".");
        per(i,B-1,B-2)printf("%.8d",a[i]);
        printf("%c",c);
    }
    long double evalu(int l)const {
        long double x=0;
        per(i,n-1,l)x=x*W+a[i];
        return x;
    }
};
bool operator <(const db &a,const db &b)
{
    if(a.is_neg) {
        if(!b.is_neg)return 1;
        per(i,T-1,0) if(a[i]!=b[i])return a[i]>b[i];
        return 0;
    }
    else {
        if(b.is_neg)return 0;
        per(i,T-1,0) if(a[i]!=b[i])return a[i]<b[i];
        return 0;
    }
}
bool operator >=(const db &a,const db &b) { return !(a<b); }
bool operator <=(const db &a,const db &b) { return b>=a; }
bool operator >(const db &a,const db &b) { return b<a; }
bool operator !(const db &a) { int n=a.n; while(n&&!a[n-1])--n; return !n;}
db operator -(db a) { a.is_neg^=1; return a; }
db operator +(db a,const db &b);
db operator -(db a,db b) {
    b=-b;
    if(a.is_neg) {
        if(!b.is_neg) { a=-a; swap(a,b); }
        else return a+b;
    }
    else {
        if(b.is_neg) b=-b;
        else return a+b;
    }
    if(a<b)return -(b-a);
    rep(i,0,a.n-1) if((a[i]-=b[i])<0) { a[i]+=W; --a[i+1]; }
    while(a.n&&!a[a.n-1])--a.n;
    return a;
}
db operator +(db a,const db &b)
{
    if(a.is_neg) { if(!b.is_neg) return b-(-a); }
    else { if(b.is_neg)return a-(-b); }
    chmax(a.n,b.n);
    rep(i,0,a.n-1) if((a[i]+=b[i])>=W) { a[i]-=W; ++a[i+1]; }
    if(a[a.n])++a.n;
    return a;
}
void operator +=(db &a,const db &b) { a=a+b; }
void operator -=(db &a,const db &b) { a=a-b; }
db eps1,eps2,eps3;
db operator *(const db &a,const db &b) {
    db ans;
    ans.is_neg=a.is_neg^b.is_neg;
    ans.n=max(0,a.n+b.n-B);
    rep(i,0,ans.n) {
        int jk=i+B;s64 sum=0;
        rep(j,max(0,jk-(b.n-1)),min(jk,a.n-1))sum+=(s64)a[j]*b[jk-j];
        int x=i;
        while(sum) { ans[x]+=sum%W; sum/=W; ++x; }
    }
    rep(i,0,ans.n) while(ans[i]>=W) { ans[i]-=W; ++ans[i+1]; }
    if(ans.n&&!ans[ans.n-1])--ans.n;
    return ans;
}
void operator *=(db &a,const db &b) { a=a*b; }
db operator *(db a,int k) {
    per(i,a.n-1,0) {
        s64 sum=(s64)a[i]*k;
        a[i]=sum%W;
        a[i+1]+=sum/W;
    }
    rep(i,0,a.n-1)
    while(a[i]>=W) { a[i]-=W; ++a[i+1]; }
    if(a[a.n])++a.n;
    return a;
}
db operator *(int k,const db &a) { return a*k; }
db operator /(db a,db b) {
    a.is_neg^=b.is_neg;
    b.is_neg=0;
    a.left_move(B);
    int l=max(0,b.n-20);
    long double b_e=b.evalu(l);
    db x; x.n=0;
    per(i,a.n-1,0) {
        x.left_move(1);
        x[0]=a[i];
        if(x>=b)
        {
            int k=x.evalu(l)/b_e;
            if(k)--k;
            a[i]=k;
            x-=k*b;
            while(x>=b) { x-=b; ++a[i]; }
        }
        else a[i]=0;
    }
    while(a.n && !a[a.n - 1]) --a.n;
    return a;
```

```
}
void operator /=(db &a,const db &b) { a = a / b; }
int cmp(const db &x,const db &eps) { return x < -eps ? -1 : x > eps; }
```