Contents

1 Math

1.1 Basic Arithmetic

```cpp
// calculate a*b % m
// x86-64 only
ll large_mod_mul(ll a, ll b, ll m) {
    return ll((__int128)a*(__int128)b%m);
}

// calculate a*b % m
// |m| < 2^62, x86 available
// O(logb)
ll large_mod_mul(ll a, ll b, ll m) {
    a %= m; b %= m; ll r = 0, v = a;
    while (b) {
        if (b & 1) {
            r = r + v;
            if (r >= m) r -= m;
        }
```

```
        b >>= 1;
        v <<= 1; if (v >= m) v -= m;
    }
    return r;
}

// calculate n^k % m
ll modpow(ll n, ll k, ll m) {
    ll ret = 1;
    n %= m;
    while (k) {
        if (k & 1) ret = large_mod_mul(ret, n, m);
        n = large_mod_mul(n, n, m);
        k /= 2;
    }
    return ret;
}

// find a pair (c, d) s.t. ac + bd = gcd(a, b)
pair<ll, ll> extended_gcd(ll a, ll b) {
    if (b == 0) return { 1, 0 };
    auto t = extended_gcd(b, a % b);
    return { t.second, t.first - t.second * (a / b) };
}

// find x in [0,m) s.t. ax === gcd(a, m) (mod m)
ll modinverse(ll a, ll m) {
    return (extended_gcd(a, m).first % m + m) % m;
}

// calculate modular inverse for 1 ~ n
void calc_range_modinv(int n, int mod, int ret[]) {
    ret[1] = 1;
    for (int i = 2; i <= n; ++i)
        ret[i] = (ll)(mod - mod/i) * ret[mod%i] % mod;
}
```

1.2 Sieve Methods : Prime, Divisor, Euler phi

```
// find prime numbers in 1 ~ n
// ret[x] = false  ->  x is prime
// O(n*loglogn)
void sieve(int n, bool ret[]) {
    for (int i = 2; i * i <= n; ++i)
        if (!ret[i])
            for (int j = i * i; j <= n; j += i)
                ret[i] = true;
}

// calculate number of divisors for 1 ~ n
// when you need to calculate sum, change += 1 to += i
// O(n*logn)
void num_of_divisors(int n, int ret[]) {
    for (int i = 1; i <= n; ++i)
        for (int j = i; j <= n; j += i)
            ret[j] += 1;
}

// calculate euler totient function for 1 ~ n
// phi(n) = number of x s.t. 0 < x < n && gcd(n, x) = 1
// O(n*loglogn)
void euler_phi(int n, int ret[]) {
    for (int i = 1; i <= n; ++i) ret[i] = i;
    for (int i = 2; i <= n; ++i)
        if (ret[i] == i)
            for (int j = i; j <= n; j += i)
                ret[j] -= ret[j] / i;
}
```

1.3 Primality Test

```
bool test_witness(ull a, ull n, ull s) {
    if (a >= n) a %= n;
    if (a <= 1) return true;
    ull d = n >> s;
    ull x = modpow(a, d, n);
    if (x == 1 || x == n-1) return true;
    while (s-- > 1) {
        x = large_mod_mul(x, x, n);
        if (x == 1) return false;
        if (x == n-1) return true;
    }
    return false;
}

// test whether n is prime
// based on miller-rabin test
// O(logn*logn)
bool is_prime(ull n) {
    if (n == 2) return true;
    if (n < 2 || n % 2 == 0) return false;

    ull d = n >> 1, s = 1;
    for(; (d&1) == 0; s++) d >>= 1;

#define T(a) test_witness(a##ull, n, s)
    if (n < 4759123141ull) return T(2) && T(7) && T(61);
    return T(2) && T(325) && T(9375) && T(28178)
        && T(450775) && T(9780504) && T(1795265022);
#undef T
}
```

1.4 Integer Factorization (Pollard's rho)

```
ll pollard_rho(ll n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<ll> dis(1, n - 1);
    ll x = dis(gen);
```

```
    ll y = x;
    ll c = dis(gen);
    ll g = 1;
    while (g == 1) {
        x = (modmul(x, x, n) + c) % n;
        y = (modmul(y, y, n) + c) % n;
        y = (modmul(y, y, n) + c) % n;
        g = gcd(abs(x - y), n);
    }
    return g;
}

// integer factorization
// O(n^0.25 * logn)
void factorize(ll n, vector<ll>& fl) {
    if (n == 1) {
        return;
    }
    if (n % 2 == 0) {
        fl.push_back(2);
        factorize(n / 2, fl);
    }
    else if (is_prime(n)) {
        fl.push_back(n);
    }
    else {
        ll f = pollard_rho(n);
        factorize(f, fl);
        factorize(n / f, fl);
    }
}
```

1.5 Chinese Remainder Theorem

```
// find x s.t.  x === a[0] (mod n[0])
//                === a[1] (mod n[1])
//                ...
// assumption: gcd(n[i], n[j]) = 1
ll chinese_remainder(ll* a, ll* n, int size) {
    if (size == 1) return *a;
    ll tmp = modinverse(n[0], n[1]);
    ll tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
    ll ora = a[1];
    ll tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2;
    n[1] *= n[0] / tgcd;
    ll ret = chinese_remainder(a + 1, n + 1, size - 1);
    n[1] /= n[0] / tgcd;
    a[1] = ora;
    return ret;
}
```

1.6 Modular Equation

$x \equiv a \pmod m$, $x \equiv b \pmod n$을 만족시키는 x를 구하는 방법.

m과 n을 소인수분해한 후 소수의 제곱꼴의 합동식들로 각각 쪼갠다. 이 때 특정 소수에 대하여 모순이 생기면 불가능한 경우고, 모든 소수에 대해서 모순이 생기지 않으면 전체 식을 CRT로 합치면 된다. 이제 $x \equiv x_1 \pmod{p^{k_1}}$과 $x \equiv x_2 \pmod{p^{k_2}}$가 모순이 생길 조건은 $k_1 \le k_2$라고 했을 때, $x_1 \not\equiv x_2 \pmod{p^{k_1}}$인 경우이다. 모순이 생기지 않았을 때 답을 구하려면 CRT로 합칠 때 $x \equiv x_2 \pmod{p^{k_2}}$만을 남기고 합쳐주면 된다.

1.7 Catalan number

다양한 문제의 답이 되는 수열이다.

- 길이가 $2n$인 올바른 괄호 수식의 수
- $n+1$개의 리프를 가진 풀 바이너리 트리의 수
- $n+2$각형을 n개의 삼각형으로 나누는 방법의 수

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

1.8 Burnside's Lemma

경우의 수를 세는데, 특정 transform operation(회전, 반사, ..)해서 같은 경우들은 하나로 친다. 전체 경우의 수는?

- 각 operation마다 이 operation을 했을 때 변하지 않는 경우의 수를 센다 (단, "아무것도 하지 않는다"라는 operation도 있어야 함!)

- 전체 경우의 수를 더한 후, operation의 수로 나눈다. (답이 맞다면 항상 나누어 떨어져야 한다)

1.9 Kirchoff's Theorem

그래프의 스패닝 트리의 개수를 구하는 정리.

무향 그래프의 Laplacian matrix L를 만든다. 이것은 (정점의 차수 대각 행렬) - (인접행렬)이다. L에서 행과 열을 하나씩 제거한 것을 L'라 하자. 어느 행/열이든 관계 없다. 그래프의 스패닝 트리의 개수는 $det(L')$이다.

1.10 Lucas Theorem

```
// calculate nCm % p  when p is prime
int lucas_theorem(const char *n, const char *m, int p) {
    vector<int> np, mp;
    int i;
    for (i = 0; n[i]; i++) {
        if (n[i] == '0' && np.empty()) continue;
        np.push_back(n[i] - '0');
    }
    for (i = 0; m[i]; i++) {
        if (m[i] == '0' && mp.empty()) continue;
        mp.push_back(m[i] - '0');
    }

    int ret = 1;
    int ni = 0, mi = 0;
    while (ni < np.size() || mi < mp.size()) {
        int nmod = 0, mmod = 0;
        for (i = ni; i < np.size(); i++) {
            if (i + 1 < np.size())
                np[i + 1] += (np[i] % p) * 10;
            else
                nmod = np[i] % p;
            np[i] /= p;
        }
        for (i = mi; i < mp.size(); i++) {
            if (i + 1 < mp.size())
                mp[i + 1] += (mp[i] % p) * 10;
            else
                mmod = mp[i] % p;
            mp[i] /= p;
        }
        while (ni < np.size() && np[ni] == 0) ni++;
        while (mi < mp.size() && mp[mi] == 0) mi++;
        // implement binomial. binomial(m,n) = 0 if m < n
        ret = (ret * binomial(nmod, mmod)) % p;
    }
    return ret;
}
```

1.11 Fast Fourier Transform

```
const double PI = acos(-1);

void fft(double *r, double *im, int N, bool f) {
    for (int i = 1, j = 0; i < N; i++) {
        int k; for (k = N >> 1; j >= k; k >>= 1) j -= k;
        j += k; if (i < j) swap(r[i], r[j]), swap(im[i], im[j]);
    }
    for (int i = 1; i < N; i <<= 1) {
        double w = PI / i; if (f) w = -w;
        double c = cos(w), s = sin(w);
        for (int j = 0; j < N; j += i << 1) {
            double yr = 1, yi = 0;
            for (int k = 0; k < i; k++) {
                double zr = r[i + j + k] * yr - im[i + j + k] * yi;
                double zi = r[i + j + k] * yi + im[i + j + k] * yr;
                r[i + j + k] = r[j + k] - zr;
                im[i + j + k] = im[j + k] - zi;
                r[j + k] += zr; im[j + k] += zi;
                tie(yr, yi) = make_pair(yr * c - yi * s, yr * s + yi * c);
            }
        }
    }
}

// Compute Poly(a)*Poly(b), write to r; Indexed from 0
// O(n*logn)
int mult(int *a, int n, int *b, int m, int *r) {
    const int maxn = 1048576;
    static double ra[maxn], rb[maxn], ia[maxn], ib[maxn];
    int fn = 1;
    while (fn < n + m) fn <<= 1; // n + m: interested length
    for (int i = 0; i < n; ++i) ra[i] = a[i], ia[i] = 0;
    for (int i = n; i < fn; ++i) ra[i] = ia[i] = 0;
    for (int i = 0; i < m; ++i) rb[i] = b[i], ib[i] = 0;
    for (int i = m; i < fn; ++i) rb[i] = ib[i] = 0;
    fft(ra, ia, fn, false);
    fft(rb, ib, fn, false);
    for (int i = 0; i < fn; ++i) {
        double real = ra[i] * rb[i] - ia[i] * ib[i];
        double imag = ra[i] * ib[i] + rb[i] * ia[i];
        ra[i] = real, ia[i] = imag;
    }
    fft(ra, ia, fn, true);
    for (int i = 0; i < fn; ++i) r[i] = (int)floor(ra[i] / fn + 0.5);
    return fn;
}
```

1.12 Number Theoretic FFT

$p = a \cdot 2^b + 1$꼴의 소수 p와 p의 원시근 x에 대하여, $n \le b$를 만족하는 모든 2^n 크기의 배열에 대해 법 p로 FFT를 행할 수 있다. 다음은 위를 만족하는 충분히 큰 소수들 목록이다.

p	a	b	원시근	덧셈	곱셈
3221225473	3	30	5	64-bit signed	64-bit unsigned
2281701377	17	27	3	64-bit signed	64-bit signed
2013265921	15	27	31	32-bit unsigned	64-bit signed
998244353	119	23	3	32-bit signed	64-bit signed
469762049	7	26	3	32-bit signed	64-bit signed

NTT 사용 시에 자료형에 유의하여, 덧셈 혹은 곱셈에서 Integer overflow가 나지 않도록 하라.

```
const int A = 7, B = 26, P = A << B | 1, R = 3;
```

```
int Pow(int x, int y) {
    int r = 1;
    while (y) {
        if (y & 1) r = r * 1ll * x % P;
        x = x * 1ll * x % P;
        y >>= 1;
    }
    return r;
}

void fft(int *a, int N, bool f) {
    for (int i = 1, j = 0; i < N; i++) {
        int k; for (k = N >> 1; j >= k; k >>= 1) j -= k;
        j += k; if (i < j) swap(a[i], a[j]);
    }
    for (int i = 1; i < N; i <<= 1) {
        int x = Pow(f ? Pow(R, P - 2) : R, P / i >> 1);
        for (int j = 0; j < N; j += i << 1) {
            int y = 1;
            for (int k = 0; k < i; k++) {
                int z = a[i + j + k] * 1ll * y % P;
                a[i + j + k] = a[j + k] - z;
                if (a[i + j + k] < P) a[i + j + k] += P;
                a[j + k] += z;
                if (a[j + k] >= P) a[j + k] -= P;
                y = y * 1ll * x % P;
            }
        }
    }
}
```

1.13 Example for FFT

```
string S;
int ai, bi, ri;
int A[MAXL], B[MAXL], R[MAXL];
int main(){
  cin>>S;
  for(auto it = S.rbegin(); it != S.rend(); it++) A[ai++] = *it - '0';
  cin>>S;
  for(auto it = S.rbegin(); it != S.rend(); it++) B[bi++] = *it - '0';
  mult(A, ai, B, bi, R);
  for(ri = 0; ri < ai + bi; ri++) R[ri + 1] += R[ri] / 10;
  while(!R[ri] && ri) ri--;
  while(ri >= 0) cout<<R[ri--] % 10;
  cout<<'\n';
  return 0;
}
```

1.14 Polynomial Division

```
vll get_inv(const vll& v, int deg){
    if (deg == 1) return vll(1, fastpow(v[0], MOD - 2));

    if (deg & 1){
        vll a = get_inv(v, deg - 1);
        ll c = 0;
        for (int i = 1; i < deg - 1; i++) c = (c + a[i] * v[deg - 1 - i]) % MOD;
        ll h1 = v[deg - 1];

        ll b = MOD - (h1 * a[0] + c) % MOD * a[0] % MOD;
        if (b == MOD) b = 0; a.push_back(b);
        return a;
    }

    vll a = get_inv(v, deg >> 1);
    vll h0(v.begin(), v.begin() + (deg >> 1));
    vll h1(v.begin() + (deg >> 1), v.begin() + deg);
    vll ah0 = mult(a, h0); ah0.push_back(0);
    vll c(ah0.begin() + (deg >> 1), ah0.begin() + deg);
    vll h1a = mult(h1, a);
    vll b_ = mult(a, add(h1a, c));

    vll b(b_.begin(), b_.begin() + (deg >> 1));
    for (ll e : b) a.push_back(e ? MOD - e : 0);
    return a;
}

vll divide(const vll& F, const vll& G, bool newg = false){
    static vll G_INV;
    const int N = (int)F.size() - 1, M = (int)G.size() - 1; // deg of F, G
    if (N < M) return vll();
    if (N == M) return vll(1, F.back()*fastpow(G.back(), MOD - 2) % MOD);

    vll f = F;
    if (G_INV.empty() || newg)
        vll g = G; reverse(g.begin(), g.end());
        while (g.size() < N - M + 1) g.push_back(0);
        G_INV = get_inv(g, N - M + 1);
    }

    reverse(f.begin(), f.end());
    vll ret = mult(f, G_INV);
    ret.resize(N - M + 1);
    reverse(ret.begin(), ret.end());

    return ret;
}
```

1.15 Gaussian Elimination

```
const double EPS = 1e-10;
typedef vector<vector<double>> VVD;

// Gauss-Jordan elimination with full pivoting.
// solving systems of linear equations (AX=B)
// INPUT:    a[][] = an n*n matrix
```

```cpp
//           b[][] = an n*m matrix
// OUTPUT:   X      = an n*m matrix (stored in b[][])
//           A^{-1} = an n*n matrix (stored in a[][])
// O(n^3)
double gauss_jordan(VVD& a, VVD& b) {
    const int n = a.size();
    const int m = b[0].size();
    vector<int> irow(n), icol(n), ipiv(n);

    double det = 1;

    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk =
                  k; }
        if (fabs(a[pj][pk]) < EPS) return 0; // matrix is singular
        ipiv[pk]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        irow[i] = pj;
        icol[i] = pk;

        double c = 1.0 / a[pk][pk];
        det *= a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }
    }
    for (int p = n - 1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
    }
    return det;
}
```

1.16 Simplex Algorithm

```cpp
// Two-phase simplex algorithm for solving linear programs of the form
//     maximize     c^T x
//     subject to   Ax <= b
//                   x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
// OUTPUT: value of the optimal solution (infinity if unbounded
//          above, nan if infeasible)
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).
typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const double EPS = 1e-9;

struct LPSolver {
    int m, n;
    VI B, N;
    VVD D;

    LPSolver(const VVD& A, const VD& b, const VD& c) :
        m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
        for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j
          ];
        for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] =
          b[i]; }
        for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m + 1][n] = 1;
    }

    void pivot(int r, int s) {
        double inv = 1.0 / D[r][s];
        for (int i = 0; i < m + 2; i++) if (i != r)
            for (int j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] * inv;
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }

    bool simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
        while (true) {
            int s = -1;
            for (int j = 0; j <= n; j++) {
                if (phase == 2 && N[j] == -1) continue;
                if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] <
                   N[s]) s = j;
            }
            if (D[x][s] > -EPS) return true;
            int r = -1;
            for (int i = 0; i < m; i++) {
                if (D[i][s] < EPS) continue;
                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                    (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] <
                      B[r]) r = i;
            }
            if (r == -1) return false;
            pivot(r, s);
        }
    }

    double solve(VD& x) {
```

```
        int r = 0;
        for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] < -EPS) {
            pivot(r, n);
            if (!simplex(1) || D[m + 1][n + 1] < -EPS)
                return -numeric_limits<double>::infinity();
            for (int i = 0; i < m; i++) if (B[i] == -1) {
                int s = -1;
                for (int j = 0; j <= n; j++)
                    if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[
                      j] < N[s]) s = j;
                pivot(i, s);
            }
        }
        if (!simplex(2))
            return numeric_limits<double>::infinity();
        x = VD(n);
        for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};
```

1.17 きたまさ法

```
// Calculate n-th result of a = ¢²aw
// O(k^2 log n)
// Warning : 1 base index(NOT 0)
int kitamasa(long long n) {
    vector<int> c(2*k+1, 0); c[1] = 1;
    vector<int> d(2*k+1);
    int b = floor(log2(n) + 1e-15);
    while(b--) {
        // c(n) -> c(2n)
        fill(d.begin(), d.end(), 0);
        for (int i=1; i<=k; i++) for (int j=1; j<=k; j++) d[i+j] = add(d[i+j],
          mul(c[i], c[j]));
        for (int i=2*k; i>k; i--) for (int j=1; j<=k; j++) d[i-j] = add(d[i-j],
          mul(d[i], w[j]));
        swap(c, d);
        // c(n) -> c(n+1)
        if ((n>>b)&1) {
            fill(d.begin(), d.end(), 0);
            d[1] = mul(c[k], w[k]);
            for (int i=2; i<=k; i++) d[i] = c[i-1] + mul(c[k], w[k+1-i]);
            swap(c, d);
        }
    }
    int r = 0;
    for (int i=1; i<=k; i++) r = add(r, mul(a[i], c[i]));
    return r;
}
```

1.18 Nim Game

Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0이 아니면 첫번째, 0이면 두번째 플레이어가 승리.

Grundy Number : 가능한 다음 state의 Grundy Number를 모두 모은 다음, 그 set에 포함되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러 개의 state들로 나뉠 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.

Subtraction Game : 한 번에 k개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 $k+1$로 나눈 나머지를 XOR 합하여 판단한다.

Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 $k+1$로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.

2 Data Structure

2.1 Order statistic tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <functional>
#include <iostream>
using namespace __gnu_pbds;
using namespace std;

// tree<key_type, value_type(set if null), comparator, ...>
using ordered_set = tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;

int main()
{
    ordered_set X;
    for (int i = 1; i < 10; i += 2) X.insert(i); // 1 3 5 7 9
    cout << boolalpha;
    cout << *X.find_by_order(2) << endl; // 5
    cout << *X.find_by_order(4) << endl; // 9
    cout << (X.end() == X.find_by_order(5)) << endl; // true

    cout << X.order_of_key(-1) << endl; // 0
    cout << X.order_of_key(1) << endl; // 0
    cout << X.order_of_key(4) << endl; // 2
    X.erase(3);
    cout << X.order_of_key(4) << endl; // 1
    for (int t : X) printf("%d␣", t); // 1 5 7 9
}
```

2.2 Segment Tree with Lazy Propagation

```
// example implementation of sum tree
const int TSIZE = 131072; // always 2^k form && n <= TSIZE
int segtree[TSIZE * 2], prop[TSIZE * 2];
void seg_init(int nod, int l, int r) {
    if (l == r) segtree[nod] = dat[l];
    else {
        int m = (l + r) >> 1;
        seg_init(nod << 1, l, m);
        seg_init(nod << 1 | 1, m + 1, r);
        segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];
    }
}
void seg_relax(int nod, int l, int r) {
    if (prop[nod] == 0) return;
    if (l < r) {
        int m = (l + r) >> 1;
        segtree[nod << 1] += (m - l + 1) * prop[nod];
        prop[nod << 1] += prop[nod];
        segtree[nod << 1 | 1] += (r - m) * prop[nod];
        prop[nod << 1 | 1] += prop[nod];
    }
    prop[nod] = 0;
}
int seg_query(int nod, int l, int r, int s, int e) {
    if (r < s || e < l) return 0;
    if (s <= l && r <= e) return segtree[nod];
    seg_relax(nod, l, r);
    int m = (l + r) >> 1;
    return seg_query(nod << 1, l, m, s, e) + seg_query(nod << 1 | 1, m + 1, r, s
      , e);
}
void seg_update(int nod, int l, int r, int s, int e, int val) {
    if (r < s || e < l) return;
    if (s <= l && r <= e) {
        segtree[nod] += (r - l + 1) * val;
        prop[nod] += val;
        return;
    }
    seg_relax(nod, l, r);
    int m = (l + r) >> 1;
    seg_update(nod << 1, l, m, s, e, val);
    seg_update(nod << 1 | 1, m + 1, r, s, e, val);
    segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];
}
// usage:
// seg_update(1, 0, n - 1, qs, qe, val);
// seg_query(1, 0, n - 1, qs, qe);
```

2.3 Persistent Segment Tree

```
// persistent segment tree impl: sum tree
// initial tree index is 0
namespace pstree {
typedef int val_t;
const int DEPTH = 18;
const int TSIZE = 1 << 18;
const int MAX_QUERY = 262144;

struct node {
    val_t v;
    node *l, *r;
} npoll[TSIZE * 2 + MAX_QUERY * (DEPTH + 1)], *head[MAX_QUERY + 1];

int pptr, last_q;

void init() {
    // zero-initialize, can be changed freely
    memset(&npoll[TSIZE - 1], 0, sizeof(node) * TSIZE);

    for (int i = TSIZE - 2; i >= 0; i--) {
        npoll[i].v = 0;
        npoll[i].l = &npoll[i*2+1];
        npoll[i].r = &npoll[i*2+2];
    }

    head[0] = &npoll[0];
    last_q = 0;
    pptr = 2 * TSIZE - 1;
}

// update val to pos
// 0 <= pos < TSIZE
// returns updated tree index
int update(int pos, int val, int prev) {
    head[++last_q] = &npoll[pptr++];
    node *old = head[prev], *now = head[last_q];

    int flag = 1 << DEPTH;
    for (;;) {
        now->v = old->v + val;
        flag >>= 1;
        if (flag==0) {
            now->l = now->r = nullptr; break;
        }
        if (flag & pos) {
            now->l = old->l;
            now->r = &npoll[pptr++];
            now = now->r, old = old->r;
        } else {
            now->r = old->r;
            now->l = &npoll[pptr++];
            now = now->l, old = old->l;
        }
    }
    return last_q;
}

val_t query(int s, int e, int l, int r, node *n) {
```

```
        if (s == l && e == r) return n->v;
        int m = (l + r) / 2;
        if (m >= e) return query(s, e, l, m, n->l);
        else if (m < s) return query(s, e, m + 1, r, n->r);
        else return query(s, m, l, m, n->l) + query(m + 1, e, m + 1, r, n->r);
    }

    // query summation of [s, e] at time t
    val_t query(int s, int e, int t) {
        s = max(0, s); e = min(TSIZE - 1, e);
        if (s > e) return 0;
        return query(s, e, 0, TSIZE - 1, head[t]);
    }
}
```

2.4 Splay Tree

```
// example : https://www.acmicpc.net/problem/13159
struct node {
    node* l, * r, * p;
    int cnt, min, max, val;
    long long sum;
    bool inv;
    node(int _val) :
        cnt(1), sum(_val), min(_val), max(_val), val(_val), inv(false),
        l(nullptr), r(nullptr), p(nullptr) {
    }
};
node* root;

void update(node* x) {
    x->cnt = 1;
    x->sum = x->min = x->max = x->val;
    if (x->l) {
        x->cnt += x->l->cnt;
        x->sum += x->l->sum;
        x->min = min(x->min, x->l->min);
        x->max = max(x->max, x->l->max);
    }
    if (x->r) {
        x->cnt += x->r->cnt;
        x->sum += x->r->sum;
        x->min = min(x->min, x->r->min);
        x->max = max(x->max, x->r->max);
    }
}

void rotate(node* x) {
    node* p = x->p;
    node* b = nullptr;
    if (x == p->l) {
        p->l = b = x->r;
        x->r = p;
    }
    else {
        p->r = b = x->l;
        x->l = p;
    }
    x->p = p->p;
    p->p = x;
    if (b) b->p = p;
    x->p ? (p == x->p->l ? x->p->l : x->p->r) = x : (root = x);
    update(p);
    update(x);
}

// make x into root
void splay(node* x) {
    while (x->p) {
        node* p = x->p;
        node* g = p->p;
        if (g) rotate((x == p->l) == (p == g->l) ? p : x);
        rotate(x);
    }
}

void relax_lazy(node* x) {
    if (!x->inv) return;
    swap(x->l, x->r);
    x->inv = false;
    if (x->l) x->l->inv = !x->l->inv;
    if (x->r) x->r->inv = !x->r->inv;
}

// find kth node in splay tree
void find_kth(int k) {
    node* x = root;
    relax_lazy(x);
    while (true) {
        while (x->l && x->l->cnt > k) {
            x = x->l;
            relax_lazy(x);
        }
        if (x->l) k -= x->l->cnt;
        if (!k--) break;
        x = x->r;
        relax_lazy(x);
    }
    splay(x);
}

// collect [l, r] nodes into one subtree and return its root
node* interval(int l, int r) {
    find_kth(l - 1);
    node* x = root;
    root = x->r;
    root->p = nullptr;
    find_kth(r - l + 1);
    x->r = root;
    root->p = x;
```

```
    root = x;
    return root->r->l;
}

void traverse(node* x) {
    relax_lazy(x);
    if (x->l) {
        traverse(x->l);
    }
    // do something
    if (x->r) {
        traverse(x->r);
    }
}

void uptree(node* x) {
    if (x->p) {
        uptree(x->p);
    }
    relax_lazy(x);
}
```

2.5 Dynamic Connectivity with Example

```
#include <bits/stdc++.h>
using namespace std;

typedef long long lint;
typedef pair<int, int> pi;

vector<pi> tree[1050000];

void add(int s, int e, int ps, int pe, int p, pi v){
        if(e < ps || pe < s) return;
        if(s <= ps && pe <= e){
                tree[p].push_back(v);
                return;
        }
        int pm = (ps + pe) / 2;
        add(s, e, ps, pm, 2*p, v);
        add(s, e, pm+1, pe, 2*p+1, v);
}

vector<pi> tmp;

bool ok(pi a, pi b, pi c){
        return 1ll * (b.first - a.first) * (c.second - b.second) <= 1ll * (b.
          first - c.first) * (a.second - b.second);
}

void solve(int x){
        sort(tree[x].begin(), tree[x].end(), [&](const pi &a, const pi &b){
                return pi(a.first, -a.second) < pi(b.first, -b.second);
        });
        tmp.clear();
        int pv = -2e9;
        for(auto &i : tree[x]){
                if(i.first == pv) continue;
                pv = i.first;
                while(tmp.size() >= 2 && !ok(tmp[tmp.size()-2], tmp.back(), i)){
                        tmp.pop_back();
                }
                tmp.push_back(i);
        }
        tree[x] = tmp;
}

void dfs(int s, int e, int p){
        solve(p);
        if(s == e) return;
        int m = (s+e)/2;
        dfs(s, m, 2*p);
        dfs(m+1, e, 2*p+1);
}

lint nodequery(int p, int x){
        if(tree[p].empty()) return -5e18;
        auto func = [&](int q){
                return 1ll * tree[p][q].first * x + tree[p][q].second;
        };
        int s = 0, e = (int)tree[p].size() - 1;
        while(s != e){
                int m = (s+e)/2;
                if(func(m) < func(m+1)) s = m+1;
                else e = m;
        }
        return func(s);
}

lint query(int pos, int s, int e, int p, int x){
        lint ret = nodequery(p, x);
        if(s == e) return ret;
        int m = (s+e)/2;
        if(pos <= m) ret = max(ret, query(pos, s, m, 2*p, x));
        else ret = max(ret, query(pos, m+1, e, 2*p+1, x));
        return ret;
}

struct ins{
        int s, e, x, y;
};

int q;
vector<ins> inserts;
pi inslis[300005];
bool vis[300005];
int cnt[300005], qry[300005];
int N;
int main(){
        cin>>N;
```

```
        for(int i=1; i<=N; i++){
                int t;
        cin>>t;
                if(t == 1){
                        vis[i] = 1;
            cin>>inslis[i].first>>inslis[i].second;
                }
                if(t == 2){
                        int x;
            cin>>x;
                        inserts.push_back({cnt[x] + 1, cnt[i-1], inslis[x].first
                          , inslis[x].second});
                        vis[x] = 0;
                }
                if(t == 3){
            cin>>qry[i];
                        cnt[i]++;
                }
                cnt[i] += cnt[i-1];
        }
        if(cnt[N] == 0) return 0;
        for(int i=1; i<=N; i++){
                if(vis[i]){
                        inserts.push_back({cnt[i] + 1, cnt[N], inslis[i].first,
                          inslis[i].second});
                }
        }
        for(auto &i : inserts){
                add(i.s, i.e, 1, cnt[N], 1, pi(i.x, i.y));
        }
        dfs(1, cnt[N], 1);
        for(int i=1; i<=N; i++){
                if(cnt[i] != cnt[i-1]){
                        lint t = query(cnt[i], 1, cnt[N], 1, qry[i]);
                        if(t < -4e18) cout<<"EMPTY␣SET\n";
                        else cout<<t<<'\n';
                }
        }
}
```

3 DP

3.1 Convex Hull Optimization

$O(n^2) \to O(n \log n)$

DP 점화식 꼴

$D[i] = \max_{j<i}(D[j] + b[j] * a[i])$ $(b[k] \le b[k+1])$

$D[i] = \min_{j<i}(D[j] + b[j] * a[i])$ $(b[k] \ge b[k+1])$

특수조건) $a[i] \le a[i+1]$ 도 만족하는 경우, 마지막 쿼리의 위치를 저장해두면 이분검색이 필요없어지기 때문에 amortized $O(n)$ 에 해결할 수 있음

```
struct CHTLinear {
    struct Line {
        long long a, b;
        long long y(long long x) const { return a * x + b; }
    };
    vector<Line> stk;
    int qpt;
    CHTLinear() : qpt(0) { }
    // when you need maximum :  (previous l).a < (now l).a
    // when you need minimum :  (previous l).a > (now l).a
    void pushLine(const Line& l) {
        while (stk.size() > 1) {
            Line& l0 = stk[stk.size() - 1];
            Line& l1 = stk[stk.size() - 2];
            if ((l0.b - l.b) * (l0.a - l1.a) > (l1.b - l0.b) * (l.a - l0.a))
              break;
            stk.pop_back();
        }
        stk.push_back(l);
    }
    // (previous x) <= (current x)
    // it calculates max/min at x
    long long query(long long x) {
        while (qpt + 1 < stk.size()) {
            Line& l0 = stk[qpt];
            Line& l1 = stk[qpt + 1];
            if (l1.a - l0.a > 0 && (l0.b - l1.b) > x * (l1.a - l0.a)) break;
            if (l1.a - l0.a < 0 && (l0.b - l1.b) < x * (l1.a - l0.a)) break;
            ++qpt;
        }
        return stk[qpt].y(x);
    }
};
```

3.2 Divide & Conquer Optimization

$O(kn^2) \to O(kn \log n)$

조건 1) DP 점화식 꼴

$D[t][i] = \min_{j<i}(D[t-1][j] + C[j][i])$

조건 2) $A[t][i]$는 $D[t][i]$의 답이 되는 최소의 j라 할 때, 아래의 부등식을 만족해야 함

$A[t][i] \le A[t][i+1]$

조건 2-1) 비용C가 다음의 사각부등식을 만족하는 경우도 조건 2)를 만족하게 됨

$C[a][c] + C[b][d] \le C[a][d] + C[b][c]$ $(a \le b \le c \le d)$

3.3 Knuth Optimization

$O(n^3) \rightarrow O(n^2)$

조건 1) DP 점화식 꼴

$D[i][j] = \min_{i<k<j}(D[i][k] + D[k][j]) + C[i][j]$

조건 2) 사각 부등식

$C[a][c] + C[b][d] \leq C[a][d] + C[b][c] \;\; (a \leq b \leq c \leq d)$

조건 3) 단조성

$C[b][c] \leq C[a][d] \;\; (a \leq b \leq c \leq d)$

결론) 조건 2, 3을 만족한다면 $A[i][j]$를 $D[i][j]$의 답이 되는 최소의 k라 할 때, 아래의 부등식을 만족하게 됨

$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$

3중 루프를 돌릴 때 위 조건을 이용하면 최종적으로 시간복잡도가 $O(n^2)$ 이 됨

4 Graph

4.1 SCC

```
const int MAXN = 100;
vector<int> graph[MAXN];
int up[MAXN], visit[MAXN], vtime;
vector<int> stk;
int scc_idx[MAXN], scc_cnt;

void dfs(int nod) {
    up[nod] = visit[nod] = ++vtime;
    stk.push_back(nod);
    for (int next : graph[nod]) {
        if (visit[next] == 0) {
            dfs(next);
            up[nod] = min(up[nod], up[next]);
        }
        else if (scc_idx[next] == 0)
            up[nod] = min(up[nod], visit[next]);
    }
    if (up[nod] == visit[nod]) {
        ++scc_cnt;
        int t;
        do {
            t = stk.back();
            stk.pop_back();
            scc_idx[t] = scc_cnt;
        } while (!stk.empty() && t != nod);
    }
}
```

```
// find SCCs in given directed graph
// O(V+E)
// the order of scc_idx constitutes a reverse topological sort
void get_scc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    scc_cnt = 0;
    memset(scc_idx, 0, sizeof(scc_idx));
    for (int i = 0; i < n; ++i)
        if (visit[i] == 0) dfs(i);
}
```

4.2 BCC, Cut vertex, Bridge

```
const int MAXN = 100;
vector<pair<int, int>> graph[MAXN];  // { next vertex id, edge id }
int up[MAXN], visit[MAXN], vtime;
vector<pair<int, int>> stk;

int is_cut[MAXN];            // v is cut vertex if  is_cut[v] > 0
vector<int> bridge;          // list of edge ids
vector<int> bcc_idx[MAXN];   // list of bccids for vertex i
int bcc_cnt;

void dfs(int nod, int par_edge) {
    up[nod] = visit[nod] = ++vtime;
    int child = 0;
    for (const auto& e : graph[nod]) {
        int next = e.first, edge_id = e.second;
        if (edge_id == par_edge) continue;
        if (visit[next] == 0) {
            stk.push_back({ nod, next });
            ++child;
            dfs(next, edge_id);
            if (up[next] == visit[next]) bridge.push_back(edge_id);
            if (up[next] >= visit[nod]) {
                ++bcc_cnt;
                do {
                    auto last = stk.back();
                    stk.pop_back();
                    bcc_idx[last.second].push_back(bcc_cnt);
                    if (last == pair<int, int>{ nod, next }) break;
                } while (!stk.empty());
                bcc_idx[nod].push_back(bcc_cnt);
                is_cut[nod]++;
            }
            up[nod] = min(up[nod], up[next]);
        }
        else
            up[nod] = min(up[nod], visit[next]);
    }
    if (par_edge == -1 && is_cut[nod] == 1)
        is_cut[nod] = 0;
}
```

```cpp
// find BCCs & cut vertexs & bridges in undirected graph
// O(V+E)
void get_bcc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    memset(is_cut, 0, sizeof(is_cut));
    bridge.clear();
    for (int i = 0; i < n; ++i) bcc_idx[i].clear();
    bcc_cnt = 0;
    for (int i = 0; i < n; ++i) {
        if (visit[i] == 0)
            dfs(i, -1);
    }
}
```

4.3 Heavy-Light Decomposition

```cpp
// heavy-light decomposition
//
// hld h;
// insert edges to tree[0~n-1];
// h.init(n, root);
// h.decompose(root);
// h.hldquery(u, v); // edges from u to v
struct hld {
    static const int MAXLN = 18;
    static const int MAXN = 1 << (MAXLN - 1);
    vector<int> tree[MAXN];
    int subsize[MAXN], depth[MAXN], pa[MAXLN][MAXN];

    int chead[MAXN], cidx[MAXN];
    int lchain;
    int flatpos[MAXN + 1], fptr;

    void dfs(int u, int par) {
        pa[0][u] = par;
        subsize[u] = 1;
        for (int v : tree[u]) {
            if (v == pa[0][u]) continue;
            depth[v] = depth[u] + 1;
            dfs(v, u);
            subsize[u] += subsize[v];
        }
    }

    void init(int size, int root)
    {
        lchain = fptr = 0;
        dfs(root, -1);
        memset(chead, -1, sizeof(chead));

        for (int i = 1; i < MAXLN; i++) {
            for (int j = 0; j < size; j++) {
                if (pa[i - 1][j] != -1) {
                    pa[i][j] = pa[i - 1][pa[i - 1][j]];
                }
            }
        }
    }

    void decompose(int u) {
        if (chead[lchain] == -1) chead[lchain] = u;
        cidx[u] = lchain;
        flatpos[u] = ++fptr;

        int maxchd = -1;
        for (int v : tree[u]) {
            if (v == pa[0][u]) continue;
            if (maxchd == -1 || subsize[maxchd] < subsize[v]) maxchd = v;
        }
        if (maxchd != -1) decompose(maxchd);

        for (int v : tree[u]) {
            if (v == pa[0][u] || v == maxchd) continue;
            ++lchain; decompose(v);
        }
    }

    int lca(int u, int v) {
        if (depth[u] < depth[v]) swap(u, v);

        int diff = depth[u] - depth[v];
        int logu = floor(log2(depth[u]) + 1e-15);
        for (int i = logu; i >= 0; --i) {
            if (diff & (1 << i)) u = pa[i][u];
        }
        if (u == v) return u;

        for (int i = logu; i >= 0; --i) {
            if (pa[i][u] != pa[i][v]) {
                u = pa[i][u];
                v = pa[i][v];
            }
        }
        return pa[0][u];
    }

    // TODO: implement query functions
    inline int query(int s, int e) {
        return 0;
    }

    int subquery(int u, int v) {
        int uchain, vchain = cidx[v];
        int ret = 0;
        for (;;) {
            uchain = cidx[u];
            if (uchain == vchain) {
                ret += query(flatpos[v], flatpos[u]);
```

```
                break;
            }

            ret += query(flatpos[chead[uchain]], flatpos[u]);
            u = pa[0][chead[uchain]];
        }
        return ret;
    }

    inline int hldquery(int u, int v) {
        int p = lca(u, v);
        return subquery(u, p) + subquery(v, p) - query(flatpos[p], flatpos[p]);
    }
};
```

4.4 Bipartite Matching (Hopcroft-Karp)

```
// in: n, m, graph
// out: match, matched
// vertex cover: (reached[0][left_node] == 0) || (reached[1][right_node] == 1)
// O(E*sqrt(V))
struct BipartiteMatching {
    int n, m;
    vector<vector<int>> graph;
    vector<int> matched, match, edgeview, level;
    vector<int> reached[2];
    BipartiteMatching(int n, int m) : n(n), m(m), graph(n), matched(m, -1),
      match(n, -1) {}

    bool assignLevel() {
        bool reachable = false;
        level.assign(n, -1);
        reached[0].assign(n, 0);
        reached[1].assign(m, 0);
        queue<int> q;
        for (int i = 0; i < n; i++) {
            if (match[i] == -1) {
                level[i] = 0;
                reached[0][i] = 1;
                q.push(i);
            }
        }
        while (!q.empty()) {
            auto cur = q.front(); q.pop();
            for (auto adj : graph[cur]) {
                reached[1][adj] = 1;
                auto next = matched[adj];
                if (next == -1) {
                    reachable = true;
                }
                else if (level[next] == -1) {
                    level[next] = level[cur] + 1;
                    reached[0][next] = 1;
                    q.push(next);
                }
            }
        }
        return reachable;
    }

    int findpath(int nod) {
        for (int &i = edgeview[nod]; i < graph[nod].size(); i++) {
            int adj = graph[nod][i];
            int next = matched[adj];
            if (next >= 0 && level[next] != level[nod] + 1) continue;
            if (next == -1 || findpath(next)) {
                match[nod] = adj;
                matched[adj] = nod;
                return 1;
            }
        }
        return 0;
    }

    int solve() {
        int ans = 0;
        while (assignLevel()) {
            edgeview.assign(n, 0);
            for (int i = 0; i < n; i++)
                if (match[i] == -1)
                    ans += findpath(i);
        }
        return ans;
    }
};
```

4.5 Maximum Flow (Dinic)

```
// usage:
// MaxFlowDinic::init(n);
// MaxFlowDinic::add_edge(0, 1, 100, 100); // for bidirectional edge
// MaxFlowDinic::add_edge(1, 2, 100); // directional edge
// result = MaxFlowDinic::solve(0, 2); // source -> sink
// graph[i][edgeIndex].res -> residual
//
// in order to find out the minimum cut, use `l'.
// if l[i] == 0, i is unrechable.
//
// O(V*V*E)
// with unit capacities, O(min(V^(2/3), E^(1/2)) * E)
struct MaxFlowDinic {
    typedef int flow_t;
    struct Edge {
        int next;
        size_t inv; /* inverse edge index */
        flow_t res; /* residual */
    };
    int n;
    vector<vector<Edge>> graph;
    vector<int> q, l, start;
```

```
    void init(int _n) {
        n = _n;
        graph.resize(n);
        for (int i = 0; i < n; i++) graph[i].clear();
    }
    void add_edge(int s, int e, flow_t cap, flow_t caprev = 0) {
        Edge forward{ e, graph[e].size(), cap };
        Edge reverse{ s, graph[s].size(), caprev };
        graph[s].push_back(forward);
        graph[e].push_back(reverse);
    }
    bool assign_level(int source, int sink) {
        int t = 0;
        memset(&l[0], 0, sizeof(l[0]) * l.size());
        l[source] = 1;
        q[t++] = source;
        for (int h = 0; h < t && !l[sink]; h++) {
            int cur = q[h];
            for (const auto& e : graph[cur]) {
                if (l[e.next] || e.res == 0) continue;
                l[e.next] = l[cur] + 1;
                q[t++] = e.next;
            }
        }
        return l[sink] != 0;
    }
    flow_t block_flow(int cur, int sink, flow_t current) {
        if (cur == sink) return current;
        for (int& i = start[cur]; i < graph[cur].size(); i++) {
            auto& e = graph[cur][i];
            if (e.res == 0 || l[e.next] != l[cur] + 1) continue;
            if (flow_t res = block_flow(e.next, sink, min(e.res, current))) {
                e.res -= res;
                graph[e.next][e.inv].res += res;
                return res;
            }
        }
        return 0;
    }
    flow_t solve(int source, int sink) {
        q.resize(n);
        l.resize(n);
        start.resize(n);
        flow_t ans = 0;
        while (assign_level(source, sink)) {
            memset(&start[0], 0, sizeof(start[0]) * n);
            while (flow_t flow = block_flow(source, sink, numeric_limits<flow_t
              >::max()))
                ans += flow;
        }
        return ans;
    }
};
```

4.6 Maximum Flow with Edge Demands

그래프 $G = (V, E)$ 가 있고 source s와 sink t가 있다. 각 간선마다 $d(e) \le f(e) \le c(e)$ 를 만족하도록 flow $f(e)$를 흘려야 한다. 이 때의 maximum flow를 구하는 문제다.

먼저 모든 demand를 합한 값 D를 아래와 같이 정의한다.

$$D = \sum_{(u \to v) \in E} d(u \to v)$$

이제 G 에 몇개의 정점과 간선을 추가하여 새로운 그래프 $G' = (V', E')$ 을 만들 것이다. 먼저 새로운 source s' 과 새로운 sink t' 을 추가한다. 그리고 s' 에서 V 의 모든 점마다 간선을 이어주고, V 의 모든 점에서 t' 로 간선을 이어준다.

새로운 capacity function c' 을 아래와 같이 정의한다.

1. V 의 점 v에 대해 $c'(s' \to v) = \sum_{u \in V} d(u \to v)$, $c'(v \to t') = \sum_{w \in V} d(v \to w)$
2. E 의 간선 $u \to v$에 대해 $c'(u \to v) = c(u \to v) - d(u \to v)$
3. $c'(t \to s) = \infty$

이렇게 만든 새로운 그래프 G' 에서 maximum flow를 구했을 때 그 값이 D라면 원래 문제의 해가 존재하고, 그 값이 D가 아니라면 원래 문제의 해는 존재하지 않는다.

위에서 maximum flow를 구하고 난 상태의 residual graph 에서 s' 과 t' 을 떼버리고 s에서 t사이의 augument path 를 계속 찾으면 원래 문제의 해를 구할 수 있다.

```
struct MaxFlowEdgeDemands
{
    MaxFlowDinic mf;
    using flow_t = MaxFlowDinic::flow_t;

    vector<flow_t> ind, outd;
    flow_t D; int n;

    void init(int _n) {
        n = _n; D = 0; mf.init(n + 2);
        ind.clear(); outd.clear();
        ind.resize(n, 0); outd.resize(n, 0);
    }

    void add_edge(int s, int e, flow_t cap, flow_t demands = 0) {
        mf.add_edge(s, e, cap - demands);
        D += demands; ind[e] += demands; outd[s] += demands;
    }

    // returns { false, 0 } if infeasible
    // { true, maxflow } if feasible
    pair<bool, flow_t> solve(int source, int sink) {
        mf.add_edge(sink, source, numeric_limits<flow_t>::max());

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

```
            if (ind[i]) mf.add_edge(n, i, ind[i]);
            if (outd[i]) mf.add_edge(i, n + 1, outd[i]);
        }

        if (mf.solve(n, n + 1) != D) return{ false, 0 };

        for (int i = 0; i < n; i++) {
            if (ind[i]) mf.graph[i].pop_back();
            if (outd[i]) mf.graph[i].pop_back();
        }

        return{ true, mf.solve(source, sink) };
    }
};
```

4.7 Min-cost Maximum Flow

```
// precondition: there is no negative cycle.
// usage:
// MinCostFlow mcf(n);
// for(each edges) mcf.addEdge(from, to, cost, capacity);
// mcf.solve(source, sink); // min cost max flow
// mcf.solve(source, sink, 0); // min cost flow
// mcf.solve(source, sink, goal_flow); // min cost flow with total_flow >=
  goal_flow if possible
struct MinCostFlow {
    typedef int cap_t;
    typedef int cost_t;

    bool iszerocap(cap_t cap) { return cap == 0; }

    struct edge {
        int target;
        cost_t cost;
        cap_t residual_capacity;
        cap_t orig_capacity;
        size_t revid;
    };

    int n;
    vector<vector<edge>> graph;

    MinCostFlow(int n) : graph(n), n(n) {}

    void addEdge(int s, int e, cost_t cost, cap_t cap) {
        if (s == e) return;
        edge forward{ e, cost, cap, cap, graph[e].size() };
        edge backward{ s, -cost, 0, 0, graph[s].size() };
        graph[s].emplace_back(forward);
        graph[e].emplace_back(backward);
    }

    pair<cost_t, cap_t> augmentShortest(int s, int e, cap_t flow_limit) {
        auto infinite_cost = numeric_limits<cost_t>::max();
        auto infinite_flow = numeric_limits<cap_t>::max();
        vector<pair<cost_t, cap_t>> dist(n, make_pair(infinite_cost, 0));
        vector<int> from(n, -1), v(n);

        dist[s] = pair<cost_t, cap_t>(0, infinite_flow);
        queue<int> q;
        v[s] = 1; q.push(s);
        while(!q.empty()) {
            int cur = q.front();
            v[cur] = 0; q.pop();
            for (const auto& e : graph[cur]) {
                if (iszerocap(e.residual_capacity)) continue;
                auto next = e.target;
                auto ncost = dist[cur].first + e.cost;
                auto nflow = min(dist[cur].second, e.residual_capacity);
                if (dist[next].first > ncost) {
                    dist[next] = make_pair(ncost, nflow);
                    from[next] = e.revid;
                    if (v[next]) continue;
                    v[next] = 1; q.push(next);
                }
            }
        }

        auto p = e;
        auto pathcost = dist[p].first;
        auto flow = dist[p].second;
        if (iszerocap(flow)|| (flow_limit <= 0 && pathcost >= 0)) return pair<
          cost_t, cap_t>(0, 0);
        if (flow_limit > 0) flow = min(flow, flow_limit);

        while (from[p] != -1) {
            auto nedge = from[p];
            auto np = graph[p][nedge].target;
            auto fedge = graph[p][nedge].revid;
            graph[p][nedge].residual_capacity += flow;
            graph[np][fedge].residual_capacity -= flow;
            p = np;
        }
        return make_pair(pathcost * flow, flow);
    }

    pair<cost_t,cap_t> solve(int s, int e, cap_t flow_minimum = numeric_limits<
      cap_t>::max()) {
        cost_t total_cost = 0;
        cap_t total_flow = 0;
        for(;;) {
            auto res = augmentShortest(s, e, flow_minimum - total_flow);
            if (res.second <= 0) break;
            total_cost += res.first;
            total_flow += res.second;
        }
        return make_pair(total_cost, total_flow);
    }
};
```

4.8 General Min-cut (Stoer-Wagner)

```
// implementation of Stoer-Wagner algorithm
// O(V^3)
//usage
// MinCut mc;
// mc.init(n);
// for (each edge) mc.addEdge(a,b,weight);
// mincut = mc.solve();
// mc.cut = {0,1}^n describing which side the vertex belongs to.
struct MinCutMatrix
{
    typedef int cap_t;
    int n;
    vector<vector<cap_t>> graph;

    void init(int _n) {
        n = _n;
        graph = vector<vector<cap_t>>(n, vector<cap_t>(n, 0));
    }
    void addEdge(int a, int b, cap_t w) {
        if (a == b) return;
        graph[a][b] += w;
        graph[b][a] += w;
    }

    pair<cap_t, pair<int, int>> stMinCut(vector<int> &active) {
        vector<cap_t> key(n);
        vector<int> v(n);
        int s = -1, t = -1;
        for (int i = 0; i < active.size(); i++) {
            cap_t maxv = -1;
            int cur = -1;
            for (auto j : active) {
                if (v[j] == 0 && maxv < key[j]) {
                    maxv = key[j];
                    cur = j;
                }
            }
            t = s; s = cur;
            v[cur] = 1;
            for (auto j : active) key[j] += graph[cur][j];
        }
        return make_pair(key[s], make_pair(s, t));
    }

    vector<int> cut;

    cap_t solve() {
        cap_t res = numeric_limits<cap_t>::max();
        vector<vector<int>> grps;
        vector<int> active;
        cut.resize(n);
        for (int i = 0; i < n; i++) grps.emplace_back(1, i);
        for (int i = 0; i < n; i++) active.push_back(i);
        while (active.size() >= 2) {
            auto stcut = stMinCut(active);
            if (stcut.first < res) {
                res = stcut.first;
                fill(cut.begin(), cut.end(), 0);
                for (auto v : grps[stcut.second.first]) cut[v] = 1;
            }

            int s = stcut.second.first, t = stcut.second.second;
            if (grps[s].size() < grps[t].size()) swap(s, t);

            active.erase(find(active.begin(), active.end(), t));
            grps[s].insert(grps[s].end(), grps[t].begin(), grps[t].end());
            for (int i = 0; i < n; i++) { graph[i][s] += graph[i][t]; graph[i][t
              ] = 0; }
            for (int i = 0; i < n; i++) { graph[s][i] += graph[t][i]; graph[t][i
              ] = 0; }
            graph[s][s] = 0;
        }
        return res;
    }
};
```

4.9 General Max Matching

```
struct DisjointSet
{
    vector<int> parent, rnk;

    DisjointSet(int n = 0) : rnk(n) {
        parent.reserve(n);
        for (int i = 0; i < n; i++) parent.push_back(i);
    }

    void reset(int n) {
        parent.clear(); rnk.assign(n, 0);
        for (int i = 0; i < n; i++) parent.push_back(i);
    }

    void increase(int n) {
        int base = parent.size();
        for (int i = base; i < base + n; i++) {
            parent.push_back(i);
            rnk.push_back(0);
        }
    }

    int find(int p) {
        return parent[p] == p ? p : parent[p] = find(parent[p]);
    }

    void merge(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return;
        if (rnk[a] < rnk[b]) swap(a, b);
```

```
        else if (rnk[a] == rnk[b]) ++rnk[a];
        parent[b] = a;
    }
};

struct MaxMatching
{
    int n;
    vector<vector<int>> gnext;
    vector<int> matched;

    int vcnt;

    MaxMatching(int n) : n(n), gnext(n), matched(n, -1) {}

    void AddEdge(int a, int b) {
        gnext[a].push_back(b);
        gnext[b].push_back(a);
    }

    int Match() {
        int ans = 0;
        while (findAugment()) ++ans;
        return ans;
    }

    vector<int> parent; // shrunken -> real
    vector<int> forest;
    vector<int> level;
    vector<pair<int,int>> bridge;
    queue<int> q;
    DisjointSet blossomSet;
    vector<int> origin; // blossomSet number to -> origin vertex
    vector<int> ancestorChecker;
    int ancestorCheckerValue;


    vector<int> marker;
    void markBlossomPath(int vv, pair<int, int> vu, int ancestor){
        int p = vv;
        marker.clear();
        while (p != ancestor) {
            int np = origin[blossomSet.find(parent[p])];
            marker.push_back(p); p = np;

            np = origin[blossomSet.find(parent[p])];
            marker.push_back(p);
            bridge[p] = vu; // need original vertex number
            q.push(p); // odd level edges were not considered
            p = np;
        }
        for (auto x : marker) blossomSet.merge(ancestor, x);
        origin[blossomSet.find(ancestor)] = ancestor;
    }

    void mergeBlossom(int vv, int uu, int v, int u){
        if (uu == vv) return;
        ++ancestorCheckerValue;
        int p1 = uu, p2 = vv;
        int ancestor = -1;
        for (;;) {
            if (p1 >= 0) {
                if (ancestorChecker[p1] == ancestorCheckerValue) {
                    ancestor = p1;
                    break;
                }
                ancestorChecker[p1] = ancestorCheckerValue;
                if (parent[p1] >= 0) p1 = origin[blossomSet.find(parent[p1])];
                  else p1 = -1;
            }
            if (p2 >= 0) {
                if (ancestorChecker[p2] == ancestorCheckerValue) {
                    ancestor = p2;
                    break;
                }
                ancestorChecker[p2] = ancestorCheckerValue;
                if (parent[p2] >= 0) p2 = origin[blossomSet.find(parent[p2])];
                  else p2 = -1;
            }
        }
        markBlossomPath(uu, make_pair(u, v), ancestor);
        markBlossomPath(vv, make_pair(v, u), ancestor);
    }

    vector<int> augmentPathLink;

    void getRootPath(int v, int w, bool reversed){
        if (v == w) return;
        if (level[v] & 1) {
            // odd. use bridge
            int x, y, mate = matched[v];
            tie(x,y) = tie(bridge[v].first, bridge[v].second);
            getRootPath(x, mate, !reversed);
            getRootPath(y, w, reversed);
            if (reversed) {
                augmentPathLink[y] = x;
                augmentPathLink[mate] = v;
            } else {
                augmentPathLink[v] = mate;
                augmentPathLink[x] = y;
            }
        } else {
            // even
            int mate = matched[v];
            getRootPath(parent[mate], w, reversed);
            if (reversed) {
                augmentPathLink[parent[mate]] = mate;
                augmentPathLink[mate] = v;
            } else {
                augmentPathLink[v] = mate;
```

```
                augmentPathLink[mate] = parent[mate];
            }
        }
    }

    void augmentPath(int v, int w) {
        augmentPathLink = vector<int>(n,-1);
        int x = forest[v];
        int y = forest[w];
        getRootPath(v,x,true);
        getRootPath(w,y,false);
        augmentPathLink[v] = w;
        int p = x;
        for(;;) {
            int q = augmentPathLink[p];
            matched[p] = q;
            matched[q] = p;
            if (q == y) break;
            p = augmentPathLink[q];
        }
    }

    bool findAugment() {
        parent = vector<int>(n,-1);
        forest = vector<int>(n,-1);
        level = vector<int>(n);
        bridge = vector<pair<int,int>>(n,make_pair(-1,-1));
        q = queue<int>();
        blossomSet.reset(n);
        origin = vector<int>(n);
        ancestorChecker = vector<int>(n);
        ancestorCheckerValue = 0;

        for(int i = 0; i < n; i++) {
            origin[i] = i;
            if (matched[i] == -1) {
                forest[i] = i;
                q.push(i);
            }
        }
        bool foundPath = false;
        while(!q.empty() && !foundPath) {
            int v = q.front(); q.pop();
            for(auto u : gnext[v]) {
                int vv = origin[blossomSet.find(v)];
                int uu = origin[blossomSet.find(u)];
                if (forest[uu] == -1) {
                    // assert(u == uu)
                    parent[uu] = v;
                    forest[uu] = forest[vv];
                    level [uu] = level [vv] + 1;
                    parent[matched[uu]] = uu;
                    forest[matched[uu]] = forest[vv];
                    level [matched[uu]] = level [vv] + 2;
                    q.push(matched[uu]);
                } else if (level[uu]&1) {
                    // odd level
                } else if (forest[uu] != forest[vv]){
                    // found path. both are even level
                    foundPath = true;
                    augmentPath(v,u);
                    break;
                } else {
                    // blossom formed
                    mergeBlossom(vv, uu, v, u);
                }
            }
        }

        return foundPath;
    }
};
```

4.10 Hungarian Algorithm

```
int n, m;
int mat[MAX_N + 1][MAX_M + 1];

// hungarian method : bipartite min-weighted matching
// O(n^3) or O(m*n^2)
// http://e-maxx.ru/algo/assignment_hungary
// mat[1][1] ~ mat[n][m]
// matched[i] : matched column of row i
int hungarian(vector<int>& matched) {
    vector<int> u(n + 1), v(m + 1), p(m + 1), way(m + 1), minv(m + 1);
    vector<char> used(m + 1);
    for (int i = 1; i <= n; ++i) {
        p[0] = i;
        int j0 = 0;
        fill(minv.begin(), minv.end(), INF);
        fill(used.begin(), used.end(), false);
        do {
            used[j0] = true;
            int i0 = p[j0], delta = INF, j1;
            for (int j = 1; j <= m; ++j) {
                if (!used[j]) {
                    int cur = mat[i0][j] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta) delta = minv[j], j1 = j;
                }
            }
            for (int j = 0; j <= m; ++j) {
                if (used[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    minv[j] -= delta;
            }
            j0 = j1;
        } while (p[j0] != 0);
        do {
```

```
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }
    for (int j = 1; j <= m; ++j) matched[p[j]] = j;
    return -v[0];
}
```

5 Geometry

5.1 Basic Operations

```
const double eps = 1e-9;

inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}

inline bool is_between(double check, double a, double b) {
    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}

struct Point {
    double x, y;
    bool operator==(const Point& rhs) const {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    Point operator+(const Point& rhs) const {
        return Point{ x + rhs.x, y + rhs.y };
    }
    Point operator-(const Point& rhs) const {
        return Point{ x - rhs.x, y - rhs.y };
    }
    Point operator*(double t) const {
        return Point{ x * t, y * t };
    }
};

struct Circle {
    Point center;
    double r;
};

struct Line {
    Point pos, dir;
};

inline double inner(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}

inline double outer(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

inline int ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}

inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}

inline double dist(const Point& a, const Point& b) {
    return sqrt(inner(a - b, a - b));
}

inline double dist2(const Point &a, const Point &b) {
    return inner(a - b, a - b);
}

inline double dist(const Line& line, const Point& point, bool segment = false) {
    double c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    double c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1) <= 0) return dist(line.pos + line.dir, point);
    return dist(line.pos + line.dir * (c1 / c2), point);
}

bool get_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    ret = b.pos + b.dir * t2;
    return true;
}

bool get_segment_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t1 = -outer(b.pos - a.pos, b.dir) / mdet;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return false;
    ret = b.pos + b.dir * t2;
    return true;
}

Point inner_center(const Point &a, const Point &b, const Point &c) {
    double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
    double w = wa + wb + wc;
    return Point{ (wa * a.x + wb * b.x + wc * c.x) / w, (wa * a.y + wb * b.y +
      wc * c.y) / w };
}
```

```cpp
Point outer_center(const Point &a, const Point &b, const Point &c) {
    Point d1 = b - a, d2 = c - a;
    double area = outer(d1, d2);
    double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
        + d1.y * d2.y * (d1.y - d2.y);
    double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
        + d1.x * d2.x * (d1.x - d2.x);
    return Point{ a.x + dx / area / 2.0, a.y - dy / area / 2.0 };
}

vector<Point> circle_line(const Circle& circle, const Line& line) {
    vector<Point> result;
    double a = 2 * inner(line.dir, line.dir);
    double b = 2 * (line.dir.x * (line.pos.x - circle.center.x)
        + line.dir.y * (line.pos.y - circle.center.y));
    double c = inner(line.pos - circle.center, line.pos - circle.center)
        - circle.r * circle.r;
    double det = b * b - 2 * a * c;
    int pred = diff(det, 0);
    if (pred == 0)
        result.push_back(line.pos + line.dir * (-b / a));
    else if (pred > 0) {
        det = sqrt(det);
        result.push_back(line.pos + line.dir * ((-b + det) / a));
        result.push_back(line.pos + line.dir * ((-b - det) / a));
    }
    return result;
}

vector<Point> circle_circle(const Circle& a, const Circle& b) {
    vector<Point> result;
    int pred = diff(dist(a.center, b.center), a.r + b.r);
    if (pred > 0) return result;
    if (pred == 0) {
        result.push_back((a.center * b.r + b.center * a.r) * (1 / (a.r + b.r)));
        return result;
    }
    double aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
    double bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
    double tmp = (bb - aa) / 2.0;
    Point cdiff = b.center - a.center;
    if (diff(cdiff.x, 0) == 0) {
        if (diff(cdiff.y, 0) == 0)
            return result; // if (diff(a.r, b.r) == 0): same circle
        return circle_line(a, Line{ Point{ 0, tmp / cdiff.y }, Point{ 1, 0 } });
    }
    return circle_line(a,
        Line{ Point{ tmp / cdiff.x, 0 }, Point{ -cdiff.y, cdiff.x } });
}

Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
    Point ba = b - a, cb = c - b;
    Line p{ (a + b) * 0.5, Point{ ba.y, -ba.x } };
    Line q{ (b + c) * 0.5, Point{ cb.y, -cb.x } };
    Circle circle;
    if (!get_cross(p, q, circle.center))
        circle.r = -1;
    else
        circle.r = dist(circle.center, a);
    return circle;
}

Circle circle_from_2pts_rad(const Point& a, const Point& b, double r) {
    double det = r * r / dist2(a, b) - 0.25;
    Circle circle;
    if (det < 0)
        circle.r = -1;
    else {
        double h = sqrt(det);
        // center is to the left of a->b
        circle.center = (a + b) * 0.5 + Point{ a.y - b.y, b.x - a.x } * h;
        circle.r = r;
    }
    return circle;
}
```

5.2 Compare angles

```cpp
int ccw(pair<int, int> p1, pair<int, int> p2) {
    auto ret = p1.first * 1ll * p2.second - p2.first * 1ll * p1.second;
    return ret > 0 ? 1 : (ret < 0 ? -1 : 0);
}

bool upper(pair<int, int> p) {
    return tie(p.second, p.first) > tuple<int, int>();
}

// sorting criterion: [0 ~ 2 * pi)
sort(dat.begin(), dat.end(), [](pair<int, int> a, pair<int, int> b){
    if (upper(a) != upper(b)) return upper(a) > upper(b);
    if (ccw(a, b)) return ccw(a, b) > 0;

    // optional: closest to farthest
    return hypot(a.first, a.second) < hypot(b.first, b.second);
});
```

5.3 Convex Hull

```cpp
// find convex hull
// O(n*logn)
vector<Point> convex_hull(vector<Point>& dat) {
    if (dat.size() <= 3) return dat;
    vector<Point> upper, lower;
    sort(dat.begin(), dat.end(), [](const Point& a, const Point& b) {
        return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    });
    for (const auto& p : dat) {
        while (upper.size() >= 2 && ccw(*++upper.rbegin(), *upper.rbegin(), p)
          >= 0) upper.pop_back();
```

```
        while (lower.size() >= 2 && ccw(*++lower.rbegin(), *lower.rbegin(), p)
          <= 0) lower.pop_back();
        upper.emplace_back(p);
        lower.emplace_back(p);
    }
    upper.insert(upper.end(), ++lower.rbegin(), --lower.rend());
    return upper;
}
```

5.4 Rotating Calipers

```
// get all antipodal pairs
// O(n)
void antipodal_pairs(vector<Point>& pt) {
    // calculate convex hull
    sort(pt.begin(), pt.end(), [](const Point& a, const Point& b) {
        return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    });
    vector<Point> up, lo;
    for (const auto& p : pt) {
        while (up.size() >= 2 && ccw(*++up.rbegin(), *up.rbegin(), p) >= 0) up.
          pop_back();
        while (lo.size() >= 2 && ccw(*++lo.rbegin(), *lo.rbegin(), p) <= 0) lo.
          pop_back();
        up.emplace_back(p);
        lo.emplace_back(p);
    }

    for (int i = 0, j = (int)lo.size() - 1; i + 1 < up.size() || j > 0; ) {
        get_pair(up[i], lo[j]);  // DO WHAT YOU WANT
        if (i + 1 == up.size()) --j;
        else if (j == 0) ++i;
        else if ((long long)(up[i + 1].y - up[i].y) * (lo[j].x - lo[j - 1].x)
            > (long long)(up[i + 1].x - up[i].x) * (lo[j].y - lo[j - 1].y)) ++i;
        else --j;
    }
}
```

5.5 Point in Polygon Test

```
typedef double coord_t;

inline coord_t is_left(Point p0, Point p1, Point p2) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

// point in polygon test
// http://geomalgorithms.com/a03-_inclusion.html
bool is_in_polygon(Point p, vector<Point>& poly) {
    int wn = 0;
    for (int i = 0; i < poly.size(); ++i) {
        int ni = (i + 1 == poly.size()) ? 0 : i + 1;
        if (poly[i].y <= p.y) {
            if (poly[ni].y > p.y) {
                if (is_left(poly[i], poly[ni], p) > 0) {
                    ++wn;
                }
            }
        }
        else {
            if (poly[ni].y <= p.y) {
                if (is_left(poly[i], poly[ni], p) < 0) {
                    --wn;
                }
            }
        }
    }
    return wn != 0;
}
```

5.6 Polygon Cut

```
// left side of a->b
vector<Point> cut_polygon(const vector<Point>& polygon, Line line) {
    if (!polygon.size()) return polygon;
    typedef vector<Point>::const_iterator piter;
    piter la, lan, fi, fip, i, j;
    la = lan = fi = fip = polygon.end();
    i = polygon.end() - 1;
    bool lastin = diff(ccw_line(line, polygon[polygon.size() - 1]), 0) > 0;
    for (j = polygon.begin() ; j != polygon.end() ; j++) {
        bool thisin = diff(ccw_line(line, *j), 0) > 0;
        if (lastin && !thisin) {
            la = i;
            lan = j;
        }
        if (!lastin && thisin) {
            fi = j;
            fip = i;
        }
        i = j;
        lastin = thisin;
    }
    if (fi == polygon.end()) {
        if (!lastin) return vector<Point>();
        return polygon;
    }
    vector<Point> result;
    for (i = fi ; i != lan ; i++) {
        if (i == polygon.end()) {
            i = polygon.begin();
            if (i == lan) break;
        }
        result.push_back(*i);
    }
    Point lc, fc;
    get_cross(Line{ *la, *lan - *la }, line, lc);
    get_cross(Line{ *fip, *fi - *fip }, line, fc);
    result.push_back(lc);
```

```
    if (diff(dist2(lc, fc), 0) != 0) result.push_back(fc);
    return result;
}
```

5.7 Pick's theorem

격자점으로 구성된 simple polygon이 주어짐. i는 polygon 내부의 격자점 수, b는 polygon 선분 위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다.

$A = i + \frac{b}{2} - 1$

6 String

6.1 KMP

```
typedef vector<int> seq_t;

void calculate_pi(vector<int>& pi, const seq_t& str) {
    pi[0] = -1;
    for (int i = 1, j = -1; i < str.size(); i++) {
        while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
        if (str[i] == str[j + 1])
            pi[i] = ++j;
        else
            pi[i] = -1;
    }
}

// returns all positions matched
// O(|text|+|pattern|)
vector<int> kmp(const seq_t& text, const seq_t& pattern) {
    vector<int> pi(pattern.size()), ans;
    if (pattern.size() == 0) return ans;
    calculate_pi(pi, pattern);
    for (int i = 0, j = -1; i < text.size(); i++) {
        while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
        if (text[i] == pattern[j + 1]) {
            j++;
            if (j + 1 == pattern.size()) {
                ans.push_back(i - j);
                j = pi[j];
            }
        }
    }
    return ans;
}
```

6.2 Z Algorithm

```
// Z[i] : maximum common prefix length of &s[0] and &s[i]
// O(|s|)
using seq_t = string;
vector<int> z_func(const seq_t &s) {
    vector<int> z(s.size());
    z[0] = s.size();
    int l = 0, r = 0;

    for (int i = 1; i < s.size(); i++) {
        if (i > r) {
            int j;
            for (j = 0; i + j < s.size() && s[i + j] == s[j]; j++) ;
            z[i] = j; l = i; r = i + j - 1;
        } else if (z[i - l] < r - i + 1) {
            z[i] = z[i - l];
        } else {
            int j;
            for (j = 1; r + j < s.size() && s[r + j] == s[r - i + j]; j++) ;
            z[i] = r - i + j; l = i; r += j - 1;
        }
    }

    return z;
}
```

6.3 Aho-Corasick

```
struct AhoCorasick
{
    const int alphabet;
    struct node {
        node() {}
        explicit node(int alphabet) : next(alphabet) {}
        vector<int> next, report;
        int back = 0, output_link = 0;
    };
    int maxid = 0;
    vector<node> dfa;
    explicit AhoCorasick(int alphabet) : alphabet(alphabet), dfa(1, node(
      alphabet)) { }
    template<typename InIt, typename Fn> void add(int id, InIt first, InIt last,
       Fn func) {
        int cur = 0;
        for ( ; first != last; ++first) {
            auto s = func(*first);
            if (auto next = dfa[cur].next[s]) cur = next;
            else {
                cur = dfa[cur].next[s] = (int)dfa.size();
                dfa.emplace_back(alphabet);
            }
        }
        dfa[cur].report.push_back(id);
        maxid = max(maxid, id);
    }
    void build() {
        queue<int> q;
        vector<char> visit(dfa.size());
        visit[0] = 1;
```

```
        q.push(0);
        while(!q.empty()) {
            auto cur = q.front(); q.pop();
            dfa[cur].output_link = dfa[cur].back;
            if (dfa[dfa[cur].back].report.empty())
                dfa[cur].output_link = dfa[dfa[cur].back].output_link;
            for (int s = 0; s < alphabet; s++) {
                auto &next = dfa[cur].next[s];
                if (next == 0) next = dfa[dfa[cur].back].next[s];
                if (visit[next]) continue;
                if (cur) dfa[next].back = dfa[dfa[cur].back].next[s];
                visit[next] = 1;
                q.push(next);
            }
        }
    }
    template<typename InIt, typename Fn> vector<int> countMatch(InIt first, InIt
       last, Fn func) {
        int cur = 0;
        vector<int> ret(maxid+1);
        for (; first != last; ++first) {
            cur = dfa[cur].next[func(*first)];
            for (int p = cur; p; p = dfa[p].output_link)
                for (auto id : dfa[p].report) ret[id]++;
        }
        return ret;
    }
};
```

6.4 Suffix Array with LCP

```
typedef char T;

// calculates suffix array.
// O(n*logn)
vector<int> suffix_array(const vector<T>& in) {
    int n = (int)in.size(), c = 0;
    vector<int> temp(n), pos2bckt(n), bckt(n), bpos(n), out(n);
    for (int i = 0; i < n; i++) out[i] = i;
    sort(out.begin(), out.end(), [&](int a, int b) { return in[a] < in[b]; });
    for (int i = 0; i < n; i++) {
        bckt[i] = c;
        if (i + 1 == n || in[out[i]] != in[out[i + 1]]) c++;
    }
    for (int h = 1; h < n && c < n; h <<= 1) {
        for (int i = 0; i < n; i++) pos2bckt[out[i]] = bckt[i];
        for (int i = n - 1; i >= 0; i--) bpos[bckt[i]] = i;
        for (int i = 0; i < n; i++)
            if (out[i] >= n - h) temp[bpos[bckt[i]]++] = out[i];
        for (int i = 0; i < n; i++)
            if (out[i] >= h) temp[bpos[pos2bckt[out[i] - h]]++] = out[i] - h;
        c = 0;
        for (int i = 0; i + 1 < n; i++) {
            int a = (bckt[i] != bckt[i + 1]) || (temp[i] >= n - h)
                    || (pos2bckt[temp[i + 1] + h] != pos2bckt[temp[i] + h]);
            bckt[i] = c;
            c += a;
        }
        bckt[n - 1] = c++;
        temp.swap(out);
    }
    return out;
}

// calculates lcp array. it needs suffix array & original sequence.
// O(n)
vector<int> lcp(const vector<T>& in, const vector<int>& sa) {
    int n = (int)in.size();
    if (n == 0) return vector<int>();
    vector<int> rank(n), height(n - 1);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    for (int i = 0, h = 0; i < n; i++) {
        if (rank[i] == 0) continue;
        int j = sa[rank[i] - 1];
        while (i + h < n && j + h < n && in[i + h] == in[j + h]) h++;
        height[rank[i] - 1] = h;
        if (h > 0) h--;
    }
    return height;
}
```

6.5 Manacher's Algorithm

```
// find longest palindromic span for each element in str
// O(|str|)
void manacher(const string& str, int plen[]) {
    int r = -1, p = -1;
    for (int i = 0; i < str.length(); ++i) {
        if (i <= r)
            plen[i] = min((2 * p - i >= 0) ? plen[2 * p - i] : 0, r - i);
        else
            plen[i] = 0;
        while (i - plen[i] - 1 >= 0 && i + plen[i] + 1 < str.length()
                && str[i - plen[i] - 1] == str[i + plen[i] + 1]) {
            plen[i] += 1;
        }
        if (i + plen[i] > r) {
            r = i + plen[i];
            p = i;
        }
    }
}
```

7 Miscellaneous

7.1 account

```
ID : team242
PW : zx6utcpf
```

7.2 Fast I/O

```
namespace fio {
    const int BSIZE = 524288;
    char buffer[BSIZE];
    int p = BSIZE;
    inline char readChar() {
        if(p == BSIZE) {
            fread(buffer, 1, BSIZE, stdin);
            p = 0;
        }
        return buffer[p++];
    }
    int readInt() {
        char c = readChar();
        while ((c < '0' || c > '9') && c != '-') {
            c = readChar();
        }
        int ret = 0; bool neg = c == '-';
        if (neg) c = readChar();
        while (c >= '0' && c <= '9') {
            ret = ret * 10 + c - '0';
            c = readChar();
        }
        return neg ? -ret : ret;
    }
}
```

7.3 Header Hack

```
#pragma GCC optimize("-O3") // -Ofast
#pragma GCC target("arch=haswell")
#define private public
#define int long long // signed main
```

7.4 Bit Hack

```
// Returns the number of 1-bits in x.
int __builtin_popcount(unsigned int x)
int __builtin_popcountll(unsigned long long x)

// Returns the number of leading redundant sign bits in x.
int __builtin_clrsb(unsigned int x)
int __builtin_clrsbll(unsigned long long x)

// Hidden functions of bitset<> bs
bs._Find_next(int idx)
bs._Unchecked_set(int idx)
bs._Unchecked_reset(int idx)
bs._Unchecked_flip(int idx)

// Smallest bit
(x & -x)
```

7.5 Reversed DS

```
set<int, greater<>>st;
map<int, int, greater<>>mp;
priority_queue<int, vector<int>, greater<>>pq;
```

7.6 Facts

1. Maximum Number of Divisors

어느 범위까지의 약수의 최대 갯수는 대략 그 범위의 세제곱근이다.

10^9 까지 1 344 개이며 그 수는 735 134 400 이다.

10^{18} 까지 103 680 개이며 그 수는897 612 484 786 617 600 이다.

2. Property of Euler's Totient Funtion

만약 $e \geq log_2(m)$ 면 다음이 성립한다.

$$n^e \equiv n^{\phi(m)+e\%\phi(m)} \pmod{m}$$

또한 $\phi(\phi(...\phi(x)))$ 은 $log_2(x)$ 번 안에 0 이 된다.

3. Sum of Matrix

행렬 A 에 대해 아래의 식을 참고하여 분할 정복을 할 수 있다.

$$A + A^2 + ... + A^{2n} = (I + A^n)(A + A^2 + ... + A^n)$$

만약 A가 역행렬이 있을때는 특별히 아래의 식이 성립한다.

$$I + A + A^2 + ... + A^n = (I - A^n)(I - A)^{-1}$$

4. Wilson's Theorem

p 가 소수이면, $(p-1)! \equiv -1 \pmod{p}$ 이다.

7.7 Magic Numbers

소수 : 10 007 , 10 009 , 10 111 , 31 567 , 70 001 , 1 000 003 , 1 000 033 , 4 000 037 , 99 999 989 , 999 999 937 , 1 000 000 007 , 1 000 000 009 , 9 999 999 967 , 99 999 999 977