

Contents

<b>1</b>	<b>Setting</b>	
1.1	Default code	1
1.2	SIMD	2
<b>2</b>	<b>Math</b>	
2.1	Linear Sieve	2
2.2	Primality Test	3
2.3	Integer Factorization (Pollard's rho)	3
2.4	Chinese Remainder Theorem	3
2.5	Query of nCr mod M in $O(Q + M)$	3
2.6	Kirchoff's Theorem	3
2.7	Lucas Theorem	3
2.8	FFT(Fast Fourier Transform)	4
2.9	NTT(Number Theoretic Transform)	4
2.10	FWHT(Fast Walsh-Hadamard Transform) and Convolution	4
2.11	Matrix Operations	5
2.12	Gaussian Elimination	5
2.13	Simplex Algorithm	5
2.14	Discrete Mathematics	6
2.15	DLAS Heuristic	7
2.16	Nim Game	7
2.17	Lifting The Exponent	7
<b>3</b>	<b>Data Structure</b>	
3.1	Order statistic tree(Policy Based Data Structure)	7
3.2	Hash Table	7
3.3	Rope	8
3.4	Persistent Segment Tree	8
3.5	Splay Tree	8
3.6	Bitset to Set	9
3.7	Li-Chao Tree	10
3.8	Wavelet Tree	10
<b>4</b>	<b>DP</b>	
4.1	Convex Hull Optimization	11
4.2	Divide & Conquer Optimization	11
4.3	Knuth Optimization	11
4.4	Bitset Optimization	11
4.5	Kitamasa & Berlekamp-Massey	11
4.6	SOS(Subset of Sum) DP	11
<b>5</b>	<b>Graph</b>	
5.1	SCC	11
5.2	2-SAT	11
5.3	BCC, Cut vertex, Bridge	11
5.4	Block-cut Tree	11
5.5	Shortest Path Faster Algorithm	11

5.6	Centroid Decomposition	13
5.7	Lowest Common Ancestor	14
5.8	Heavy-Light Decomposition	14
5.9	Hall's Theorem	14
5.10	Stable Marriage	14
5.11	Bipartite Matching (Kuhn)	15
5.12	Maximum Flow (Dinic)	15
5.13	Maximum Flow with Edge Demands	15
5.14	Min-cost Maximum Flow	16
5.15	General Min-cut (Stoer-Wagner)	16
5.16	Hungarian Algorithm	17
5.17	General Unweighted Maximum Matching(Tutte)	17
5.18	General Weighted Maximum Matching(Blossom)	17
5.19	Offline Dynamic Connectivity	19
<b>6</b>	<b>Geometry</b>	<b>19</b>
6.1	Basic Operations	19
6.2	Convex Hull & Rotating Calipers	20
6.3	Half Plane Intersection	21
6.4	Minimum Perimeter Triangle	21
6.5	Minimum Enclosing Circle	21
6.6	Point in Polygon Test	21
6.7	Polygon Cut	22
6.8	Number of Point in Triangle	22
6.9	Voronoi Diagram	23
6.10	KD-Tree	24
6.11	Pick's theorem	24
<b>7</b>	<b>String</b>	<b>24</b>
7.1	KMP	24
7.2	Z Algorithm	25
7.3	Aho-Corasick	25
7.4	Suffix Array with LCP	25
7.5	Manacher's Algorithm	25
7.6	EERTREE	25
<b>11</b>	<b>1 Setting</b>	
<b>11</b>	<b>1.1 Default code</b>	
11		
11	<code>#pragma GCC optimize ("O3,unroll-loops")</code>	
11	<code>#pragma GCC target ("avx,avx2,fma")</code>	
11	<code>#define debug(...) __dbg(#__VA_ARGS__, __VA_ARGS__)</code>	
12	<code>template&lt;typename T&gt;</code>	
12	<code>ostream&amp; operator&lt;&lt;(ostream&amp; out, vector&lt;T&gt; v) {</code>	
	<code>string _;</code>	
	<code>out &lt;&lt; '(';</code>	
12	<code>for (T x : v) out &lt;&lt; _ &lt;&lt; x, _ = " ";</code>	
12	<code>out &lt;&lt; ')';</code>	
	<code>return out;</code>	
13	<code>}</code>	
13	<code>void __dbg(string s, auto... x) {</code>	
	<code>string _;</code>	
13	<code>cout &lt;&lt; '(' &lt;&lt; s &lt;&lt; ") : ";</code>	
13	<code>(..., (cout &lt;&lt; _ &lt;&lt; x, _ = " ", "));</code>	

```
    cout << '\n';
}
auto gen_tree = [](int n) {
    auto prufer_decode = [](const vector<int>& v) {
        const int n = v.size() + 2;
        vector deg(n + 1, 1);
        for (int i : v) deg[i]++;
        int p = 1, leaf = 1;
        while (deg[p] != 1) p++, leaf++;
        vector res(0, pair(0, 0));
        for (int i : v) {
            res.push_back({ leaf, i });
            if (--deg[i] == 1 && i < p) leaf = i;
            else { do p++; while (deg[p] != 1); leaf = p; }
        }
        res.push_back({ leaf, n });
        return res;
    };
    vector v(n - 2, 0);
    for (int& i : v) i = gen_rand(1, n);
    return prufer_decode(v);
};
auto vectors(const int n, auto&& val) {
    return vector(n, val);
}
auto vectors(const int n, auto&&... args) {
    return vector(n, vectors(args...));
}
struct query { // mo's algorithm
    int l, r, i;
    bool operator< (const query& x) {
        if ((l ^ x.l) >> 9) return l < x.l;
        return l >> 9 & 1 ^ r < x.r;
    }
};
uint32_t xorshift32(uint32_t x) {
    x ^= x << 13; x ^= x >> 17; x ^= x << 5;
    return x;
}
uint64_t xorshift64(uint64_t x) {
    x ^= x << 13; x ^= x >> 7; x ^= x << 17;
    return x;
}
uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049b133111eb;
    return x ^ (x >> 31);
}
vector e(m, tuple(0, 0, 0));
for (auto& [a, b, c] : e) cin >> a >> b >> c;
vector cnt(n + 2, 0); vector csr(m, pair(0, 0));
for (auto [a, b, c] : e) cnt[a + 1]++;
for (int i = 1; i < cnt.size(); i++) cnt[i] += cnt[i - 1];
for (auto [a, b, c] : e) csr[cnt[a]++] = pair(b, c);
int cur = /* ... */;
for (int i = cnt[cur - 1]; i < cnt[cur]; i++) {
    auto [nxt, cost] = csr[i]; /* ... */
}
arr.reserve(n) // reserve n elements + O(1) push_back
```

1.2 SIMD

```
#include <immintrin.h>
alignas(32) int A[8]{ 1, 2, 3, 1, 2, 3, 1, 2 }, B[8]{ 1, 2, 3, 4, 5, 6, 7, 8 };
alignas(32) int C[8]; // alignas(bit size of <type>) <type> var[256/(bit size)]
```

```
// Must compute "index is multiply of 256bit"(ex> short->16k, int->8k, ...)
__m256i a = _mm256_load_si256((__m256i*)A);
__m256i b = _mm256_load_si256((__m256i*)B);
__m256i c = _mm256_add_epi32(a, b);
__mm256_store_si256((__m256i*)C, c);

__m256i _mm256_abs_epi32 (__m256i a)
__mm256_set1_epi32(__m256i a, __m256i b)
__m256i _mm256_and_si256 (__m256i a, __m256i b)
__m256i _mm256_setzero_si256 (void)
__mm256_add_pd(__m256d a, __m256d b) // double precision(64-bit)
__mm256_sub_pd(__m256 a, __m256 b) // double precision(64-bit)
__m256d _mm256_andnot_pd (__m256d a, __m256d b) // (~a)&b
__m256i _mm256_avg_epu16 (__m256i a, __m256i b) // unsigned, (a+b+1)>>1
__m256d _mm256_ceil_pd (__m256d a)
__m256d _mm256_floor_pd (__m256d a)
__m256i _mm256_cmpeq_epi64 (__m256i a, __m256i b)
__m256i _mm256_cmpgt_epi16 (__m256i a, __m256i b)
__m256d _mm256_div_pd (__m256d a, __m256d b)
__m256i _mm256_max_epi32 (__m256i a, __m256i b)
__m256i _mm256_mul_epi32 (__m256i a, __m256i b)
__m256 _mm256_rcp_ps (__m256 a) // 1/a
__m256 _mm256_rsqrt_ps (__m256 a) // 1/sqrt(a)
__m256i _mm256_set1_epi64x (long long a)
__m256i _mm256_sign_epi16 (__m256i a, __m256i b) // a*(sign(b))
__m256i _mm256_sll_epi32 (__m256i a, __m128i count) // a << count
__m256d _mm256_sqrt_pd (__m256d a)
__m256i _mm256_sra_epi16 (__m256i a, __m128i count)
__m256i _mm256_xor_si256 (__m256i a, __m256i b)
void _mm256_zeroall (void)
void _mm256_zeroupper (void)
```

2 Math

2.1 Linear Sieve

```
struct sieve {
    const ll MAXN = 101010;
    vector<ll> sp, e, phi, mu, tau, sigma, primes;
    // sp : smallest prime factor, e : exponent, phi : euler phi, mu : mobius
    // tau : num of divisors, sigma : sum of divisors
    sieve(ll sz) {
        sp.resize(sz + 1), e.resize(sz + 1), phi.resize(sz + 1), mu.resize(sz + 1),
        tau.resize(sz + 1), sigma.resize(sz + 1);
        phi[1] = mu[1] = tau[1] = sigma[1] = 1;
        for (ll i = 2; i <= sz; i++) {
            if (!sp[i]) {
                primes.push_back(i), e[i] = 1, phi[i] = i - 1, mu[i] = -1, tau[i] = 2;
                sigma[i] = i + 1;
            }
            for (auto j : primes) {
                if (i * j > sz) break;
                sp[i * j] = j;
                if (i % j == 0) {
                    e[i * j] = e[i] + 1, phi[i * j] = phi[i] * j, mu[i * j] = 0,
                    tau[i * j] = tau[i] / e[i * j] * (e[i * j] + 1),
                    sigma[i * j] = sigma[i] * (j - 1) / (powm(j, e[i * j]) - 1) *
                    (powm(j, e[i * j] + 1) - 1) / (j - 1);
                    break;
                }
                e[i * j] = 1, phi[i * j] = phi[i] * phi[j], mu[i * j] = mu[i] * mu[j],
                tau[i * j] = tau[i] * tau[j], sigma[i * j] = sigma[i] * sigma[j];
            }
        }
    }
    sieve() : sieve(MAXN) {}
};
```

## 2.2 Primality Test

```
// test whether n is prime based on miller-rabin test
// O(logn*logn)
bool is_prime(ll n) {
    if (n < 2 || n % 2 == 0 || n % 3 == 0) return n == 2 || n == 3;
    ll k = __builtin_ctzll(n - 1), d = n - 1 >> k;
    for (ll a : { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 }) {
        ll p = modpow(a % n, d, n), i = k;
        while (p != 1 && p != n - 1 && a % n && i--) p = modmul(p, p, n);
        if (p != n - 1 && i != k) return 0;
    }
    return 1;
}
```

## 2.3 Integer Factorization (Pollard's rho)

```
ll pollard(ll n) {
    auto f = [n](ll x) { return modadd(modmul(x, x, n), 3, n); };
    ll x = 0, y = 0, t = 30, p = 2, i = 1, q;
    while (t++ % 40 || gcd(p, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if (q = modmul(p, abs(x - y), n)) p = q;
        x = f(x), y = f(f(y));
    }
    return gcd(p, n);
}
// integer factorization
// O(n^0.25 * logn)
vector<ll> factor(ll n) {
    if (n == 1) return {};
    if (is_prime(n)) return { n };
    ll x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    sort(l.begin(), l.end());
    return l;
}
```

## 2.4 Chinese Remainder Theorem

```
// x = r_i mod m_i
// (y, m) 'x = y mod m' 'm = lcm(m_i)', if not exists return (0, 0)
auto crt = [](auto r, auto m) {
    const int n = r.size();
    i64 r0 = 0, m0 = 1;
    for (int i = 0; i < n; i++) {
        i64 r1 = r[i], m1 = m[i];
        if (m0 < m1) swap(r0, r1), swap(m0, m1);
        if (m0 % m1 == 0 && r0 % m1 != r1) return pair(0LL, 0LL);
        if (m0 % m1 == 0) continue;
        i64 g = gcd(m0, m1);
        if ((r1 - r0) % g) return pair(0LL, 0LL);
        i64 u0 = m0 / g, u1 = m1 / g;
        i64 x = (r1 - r0) / g % u1 * modinv(u0, u1) % u1;
        r0 += x * m0, m0 *= u1;
        if (r0 < 0) r0 += m0;
    }
    return pair(r0, m0);
};
```

## 2.5 Query of nCr mod M in $O(Q + M)$

```
auto sol_p_e = [](int q, const auto& qs, const int p, const int e, const int mod) {
    // qs[i] = {n, r}, nCr mod p^e in O(p^e)
    vector dp(mod, 1);
```

```
for (int i = 0; i < mod; i++) {
    if (i) dp[i] = dp[i - 1];
    if (i % p == 0) continue;
    dp[i] = mul(dp[i], i);
}
auto f = [](i64 n) {
    i64 res = 0;
    while (n /= p) res += n;
    return res;
};
auto g = [](i64 n) {
    auto rec = [&](const auto& self, i64 n) -> int {
        if (n == 0) return 1;
        int q = n / mod, r = n % mod;
        int ret = mul(self(self, n / p), dp[r]);
        if (q & 1) ret = mul(ret, dp[mod - 1]);
        return ret;
    };
    return rec(rec, n);
};
auto bino = [](i64 n, i64 r) {
    if (n < r) return 0;
    if (r == 0 || r == n) return 1;
    i64 a = f(n) - f(r) - f(n - r);
    if (a >= e) return 0;
    int b = mul(g(n), modinv(mul(g(r), g(n - r)), mod));
    return mul(pow(p, a), b);
};
vector res(q, 0);
for (int i = 0; i < q; i++) {
    auto [n, r] = qs[i];
    res[i] = bino(n, r);
}
return res;
};
auto sol = [](int q, const auto& qs, const int mod) {
    vector fac = factor(mod);
    vector r(q, vector(fac.size(), 0));
    vector m(fac.size(), 1);
    for (int i = 0; i < fac.size(); i++) {
        auto [p, e] = fac[i];
        for (int j = 0; j < e; j++) m[i] *= p;
        auto res = sol_p_e(q, qs, p, e, m[i]);
        for (int j = 0; j < q; j++) r[j][i] = res[j];
    }
    vector res(q, 0);
    for (int i = 0; i < q; i++) res[i] = crt(r[i], m).first;
    return res;
};
```

## 2.6 Kirchoff's Theorem

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

## 2.7 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;
}
```

2.8 FFT(Fast Fourier Transform)

```
void fft(int sign, int n, double *real, double *imag) {
    double theta = sign * 2 * pi / n;
    for (int m = n; m >= 2; m >>= 1, theta *= 2) {
        double wr = 1, wi = 0, c = cos(theta), s = sin(theta);
        for (int i = 0, mh = m >> 1; i < mh; ++i) {
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                double xr = real[j] - real[k], xi = imag[j] - imag[k];
                real[j] += real[k], imag[j] += imag[k];
                real[k] = wr * xr - wi * xi, imag[k] = wr * xi + wi * xr;
            }
            double _wr = wr * c - wi * s, _wi = wr * s + wi * c;
            wr = _wr, wi = _wi;
        }
    }
    for (int i = 1, j = 0; i < n; ++i) {
        for (int k = n >> 1; k > (j ^= k); k >>= 1)
            ;
        if (j < i) swap(real[i], real[j]), swap(imag[i], imag[j]);
    }
}

// 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 = 100;
    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(1, fn, ra, ia);
    fft(1, fn, rb, ib);
    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(-1, fn, ra, ia);
    for (int i = 0; i < fn; ++i) r[i] = (int)floor(ra[i] / fn + 0.5);
    return fn;
}
```

2.9 NTT(Number Theoretic Transform)

```
void ntt(poly& f, bool inv = 0) {
    int n = f.size(), j = 0;
    vector<ll> root(n >> 1);
    for (int i = 1; i < n; i++) {
        int bit = (n >> 1);
        while (j >= bit) {
            j -= bit;
            bit >>= 1;
        }
        j += bit;
        if (i < j) swap(f[i], f[j]);
    }
    ll ang = pw(w, (mod - 1) / n);
    if (inv) ang = pw(ang, mod - 2);
    root[0] = 1;
    for (int i = 1; i < (n >> 1); i++) root[i] = root[i - 1] * ang % mod;
    for (int i = 2; i <= n; i <= 1) {
        int step = n / i;
        for (int j = 0; j < n; j += i) {
            for (int k = 0; k < (i >> 1); k++) {
                ll u = f[j | k], v = f[j | k | i >> 1] * root[step * k] % mod;
                f[j | k] = (u + v) % mod;
                f[j | k | i >> 1] = (u - v) % mod;
                if (f[j | k | i >> 1] < 0) f[j | k | i >> 1] += mod;
            }
        }
    }
    ll t = pw(n, mod - 2);
    if (inv)
        for (int i = 0; i < n; i++) f[i] = f[i] * t % mod;
}

vector<ll> multiply(poly& _a, poly& _b) {
    vector<ll> a(all(_a)), b(all(_b));
    int n = 2;
    while (n < a.size() + b.size()) n <= 1;
    a.resize(n);
    b.resize(n);
    ntt(a);
    ntt(b);
    for (int i = 0; i < n; i++) a[i] = a[i] * b[i] % mod;
    ntt(a, 1);
    return a;
}
```

998244353 = 119 × 2<sup>23</sup> + 1. Primitive root: 3.  
985661441 = 235 × 2<sup>22</sup> + 1. Primitive root: 3.  
1012924417 = 483 × 2<sup>21</sup> + 1. Primitive root: 5.

2.10 FWHT(Fast Walsh-Hadamard Transform) and Convolution

```
// (fwht_or(a))_i = sum of a_j for all j s.t. i | j = j
// (fwht_and(a))_i = sum of a_j for all j s.t. i & j = i
// x @ y = popcount(x & y) mod 2
// (fwht_xor(a))_i = (sum of a_j for all j s.t. i @ j = 0)
```

```
// - (sum of a_j for all j s.t. i @ j = 1)
// inv = 0 for fwht, 1 for ifwht(inverse fwht)
// {convolution(a,b)}_i = sum of a_j * b_k for all j,k s.t. j op k = i
// = ifwht(fwht(a) * fwht(b))
vector<ll> fwht_or(vector<ll> &x, bool inv) {
    vector<ll> a = x;
    ll n = a.size();
    int dir = inv ? -1 : 1;
    for(int s = 2, h = 1; s <= n; s <= 1, h <= 1) {
        for(int l = 0; l < n; l += s) {
            for(int i = 0; i < h; i++) a[l + h + i] += dir * a[l + i];
        }
    }
    return a;
}
vector<ll> fwht_and(vector<ll> &x, bool inv) {
    vector<ll> a = x;
    ll n = a.size();
    int dir = inv ? -1 : 1;
    for(int s = 2, h = 1; s <= n; s <= 1, h <= 1) {
        for(int l = 0; l < n; l += s) {
            for(int i = 0; i < h; i++) a[l + h] += dir * a[l + h + i];
        }
    }
    return a;
}
vector<ll> fwht_xor(vector<ll> &x, bool inv) {
    vector<ll> a = x;
    ll n = a.size();
    for(int s = 2, h = 1; s <= n; s <= 1, h <= 1) {
        for(int l = 0; l < n; l += s) {
            for(int i = 0; i < h; i++) {
                int t = a[l + h + i];
                a[l + h + i] = a[l + i] - t;
                a[l + i] += t;
                if(inv) a[l + h + i] /= 2, a[l + i] /= 2;
            }
        }
    }
    return a;
}
}
```

## 2.11 Matrix Operations

```
inline bool is_zero(ld a) { return abs(a) < eps; }
// returns {det(A), A^-1, rank(A), tr(A)}
// A becomes invalid after call this O(n^3)
tuple<ld, vector<vector<ld>>, ll, ll> inv_det_rnk(auto A) {
    ld n = A.size(); ld det = 1; vector out(n, vector<ld>(n)); ld tr = 0;
    for (int i = 0; i < n; i++) {
        out[i][i] = 1; tr += A[i][i];
    }
    for (int i = 0; i < n; i++) {
        if (is_zero(A[i][i])) {
            ld maxv = 0;
            int maxid = -1;
            for (int j = i + 1; j < n; j++) {
                auto cur = abs(A[j][i]);
                if (maxv < cur) {
                    maxv = cur;
                    maxid = j;
                }
            }
            if (maxid == -1 || is_zero(A[maxid][i])) return {0, out, i, tr};
            for (int k = 0; k < n; k++) {
                A[i][k] += A[maxid][k]; out[i][k] += out[maxid][k];
            }
        }
    }
}
```

```
}
}
det *= A[i][i];
ld coeff = 1.0 / A[i][i];
for (int j = 0; j < n; j++) A[i][j] *= coeff, out[i][j] *= coeff;
for (int j = 0; j < n; j++) if (j != i) {
    ld mp = A[j][i];
    for (int k = 0; k < n; k++) A[j][k] -= A[i][k] * mp;
    for (int k = 0; k < n; k++) out[j][k] -= out[i][k] * mp;
}
}
return {det, out, n, tr};
}
```

## 2.12 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
//         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)
bool 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);

    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 false; // matrix is singular
        ipiv[pj]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        irow[i] = pj;
        icol[i] = pk;

        double c = 1.0 / 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 true;
}
```

## 2.13 Simplex Algorithm

```
// Two-phase simplex algorithm for solving linear programs of the form
// maximize c^T x s.t. Ax <= b; x >= 0
// A -- m x n mat, b -- m-dimensional vec, c -- n-dimensional vec
// return {value of the optimal solution, solution vector}
```

```

struct LPSolver {
    ll m, n;
    vector<ll> B, N;
    vector<vector<ld>>> D;
    LPSolver(const vector<vector<ld>>& A, const vector<ld>& b, const vector<ld>& c):
        m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, vector<ld>(n + 2)) {
        for (ll i = 0; i < m; i++) for (ll j = 0; j < n; j++) D[i][j] = A[i][j];
        for (ll i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; }
        for (ll j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m + 1][n] = 1;
    }
    void pivot(ll r, ll s) {
        ld inv = 1.0 / D[r][s];
        for (ll i = 0; i < m + 2; i++) if (i != r)
            for (ll j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] * inv;
        for (ll j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
        for (ll i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv; swap(B[r], N[s]);
    }
    bool simplex(ll phase) {
        ll x = phase == 1 ? m + 1 : m;
        while (true) {
            ll s = -1;
            for (ll 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;
            ll r = -1;
            for (ll 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);
        }
    }
    pair<ld, vector<ld>> solve() {
        ll r = 0; vector<ld> x(n);
        for (ll 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<ld>::infinity();
            for (ll i = 0; i < m; i++) if (B[i] == -1) {
                ll s = -1;
                for (ll 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<ld>::infinity();
        for (ll i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};

```

## 2.14 Discrete Mathematics

```

/* Solve x for x^P = A mod Q
 * O((LgQ)^2 + Q^0.25 (LgQ)^3)
 * (P, Q-1) = 1 -> P^-1 mod (Q-1) exists
 * x has solution iff A^((Q-1) / P) = 1 mod Q
 * PP | (Q-1) -> P < sqrt(Q), solve LgQ rounds of discrete log
 * else -> find a s.t. s | (Pa - 1) -> ans = A^a */

```

```

const int X = 1e5;
ll base, ae[X], aXe[X], iaXe[X];
unordered_map<ll, ll> ht;
#define FOR(i, c) for (int i = 0; i < (c); ++i)
#define REP(i, l, r) for (int i = (l); i <= (r); ++i)
// discrete log : O(sqrt(Q))
void build(ll a) { // ord(a) = P < sqrt(Q)
    base = a;
    ht.clear();
    ae[0] = 1; ae[1] = a; aXe[0] = 1; aXe[1] = pw(a, X, Q);
    iaXe[0] = 1; iaXe[1] = pw(aXe[1], Q-2, Q);
    REP(i, 2, X-1) {
        ae[i] = mul(ae[i-1], ae[1], Q);
        aXe[i] = mul(aXe[i-1], aXe[1], Q);
        iaXe[i] = mul(iaXe[i-1], iaXe[1], Q);
    }
    FOR(i, X) ht[ae[i]] = i;
}
ll dis_log(ll x) {
    FOR(i, X) {
        ll iaXi = iaXe[i];
        ll rst = mul(x, iaXi, Q);
        if (ht.count(rst)) return i*X + ht[rst];
    }
}
ll main2() {
    ll g;
    ll t = 0, s = Q-1;
    while (s % P == 0) {
        ++t;
        s /= P;
    }
    if (A == 0) return 0;
    if (t == 0) {
        // a^{P^-1 mod phi(Q)}
        auto [x, y, _] = extended_gcd(P, Q-1);
        if (x < 0) {
            x = (x % (Q-1) + Q-1) % (Q-1);
        }
        ll ans = pw(A, x, Q);
        if (pw(ans, P, Q) != A) while(1);
        return ans;
    }
    // A is not P-residue
    if (pw(A, (Q-1) / P, Q) != 1) return -1;
    for (g = 2; g < Q; ++g) {
        if (pw(g, (Q-1) / P, Q) != 1) break;
    }
    ll alpha = 0;
    {
        ll y, _;
        gcd(P, s, alpha, y, _);
        if (alpha < 0) alpha = (alpha % (Q-1) + Q-1) % (Q-1);
    }
    if (t == 1) {
        ll ans = pw(A, alpha, Q);
        return ans;
    }
    ll a = pw(g, (Q-1) / P, Q);
    build(a);
    ll b = pw(A, add(mul(P%(Q-1), alpha, Q-1), Q-2, Q-1), Q);
    ll c = pw(g, s, Q);
    ll h = 1;
    ll e = (Q-1) / s / P; // r^{t-1}
    REP(i, 1, t-1) {
        e /= P;
    }
}

```

```

11 d = pw(b, e, Q);
11 j = 0;
if (d != 1) {
    j = -dis_log(d);
    if (j < 0) j = (j % (Q-1) + Q-1) % (Q-1);
}
b = mul(b, pw(c, mul(P%(Q-1), j, Q-1), Q), Q);
h = mul(h, pw(c, j, Q), Q);
c = pw(c, P, Q);
}
return mul(pw(A, alpha, Q), h, Q);
}
// only for sqrt
void calcH(int &t, int &h, const int p) {
    int tmp=p-1; for(t=0;(tmp&1)==0;tmp/=2) t++; h=tmp;
}
// solve equation x^2 mod p = a
bool solve(int a, int p, int &x, int &y) {
    if(p == 2) { x = y = 1; return true; }
    int p2 = p / 2, tmp = pw(a, p2, p);
    if (tmp == p - 1) return false;
    if ((p + 1) % 4 == 0) {
        x=pw(a,(p+1)/4,p); y=p-x; return true;
    } else {
        int t, h, b, pb; calcH(t, h, p);
        if (t >= 2) {
            do {b = rand() % (p - 2) + 2;
                } while (pw(b, p / 2, p) != p - 1);
            pb = pw(b, h, p);
        } int s = pw(a, h / 2, p);
        for (int step = 2; step <= t; step++) {
            int ss = (((11)(s * s) % p) * a) % p;
            for(int i=0;i<t-step;i++) ss=mul(ss,ss,p);
            if (ss + 1 == p) s = (s * pb) % p;
        }
        pb = ((11)pb * pb) % p;
        } x = ((11)s * a) % p; y = p - x;
        } return true;
    }
}

```

## 2.15 DLAS Heuristic

```

auto dlas = [](const auto& state, int iter) {
    vector s(3, state);
    vector buc(5, s[0].score());
    auto cur_score = buc[0], min_score = cur_score;
    int cur_pos = 0, min_pos = 0, k = 0;
    for (int i = 0; i < iter; i++) {
        auto prv_score = cur_score;
        int nxt_pos = cur_pos + 1 < 3 ? cur_pos + 1 : 0;
        if (nxt_pos == min_pos) nxt_pos = nxt_pos + 1 < 3 ? nxt_pos + 1 : 0;
        auto& cur_state = s[cur_pos];
        auto& nxt_state = s[nxt_pos];
        nxt_state = cur_state;
        nxt_state.mutate();
        auto nxt_score = nxt_state.score();
        if (min_score > nxt_score) {
            i = 0;
            min_pos = nxt_pos;
            min_score = nxt_score;
        }
        if (nxt_score == cur_score || nxt_score < ranges::max(buc)) {
            cur_pos = nxt_pos;
            cur_score = nxt_score;
        }
        auto& fit = buc[k];
        if (cur_score > fit || cur_score < min(fit, prv_score)) {

```

```

        fit = cur_score;
    }
    k = k + 1 < 5 ? k + 1 : 0;
}
return pair(s[min_pos], min_score);
};

```

## 2.16 Nim Game

Nim Game의 해법 : 모두 XOR했을 때 0이 아니면 첫번째, 0이면 두번째 플레이어가 승리.  
 Grundy Number : XOR(MEX(next state grundy))  
 Subtraction Game : 한 번에  $k$ 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를  $k + 1$ 로 나눈 나머지를 XOR 합하여 판단한다.  
 Index-k Nim : 한 번에 최대  $k$ 개의 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.

## 2.17 Lifting The Exponent

For any integers  $x, y$  a positive integer  $n$ , and a prime number  $p$  such that  $p \nmid x$  and  $p \nmid y$ , the following statements hold:

- When  $p$  is odd:
  - If  $p \mid x - y$ , then  $\nu_p(x^n - y^n) = \nu_p(x - y) + \nu_p(n)$ .
  - If  $n$  is odd and  $p \mid x + y$ , then  $\nu_p(x^n + y^n) = \nu_p(x + y) + \nu_p(n)$ .
- When  $p = 2$ :
  - If  $2 \mid x - y$  and  $n$  is even, then  $\nu_2(x^n - y^n) = \nu_2(x - y) + \nu_2(x + y) + \nu_2(n) - 1$ .
  - If  $2 \mid x - y$  and  $n$  is odd, then  $\nu_2(x^n - y^n) = \nu_2(x - y)$ .
  - Corollary:
    - \* If  $4 \mid x - y$ , then  $\nu_2(x + y) = 1$  and thus  $\nu_2(x^n - y^n) = \nu_2(x - y) + \nu_2(n)$ .
- For all  $p$ :
  - If  $\gcd(n, p) = 1$  and  $p \mid x - y$ , then  $\nu_p(x^n - y^n) = \nu_p(x - y)$ .
  - If  $\gcd(n, p) = 1, p \mid x + y$  and  $n$  odd, then  $\nu_p(x^n + y^n) = \nu_p(x + y)$ .

## 3 Data Structure

### 3.1 Order statistic tree(Policy Based Data Structure)

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <functional>
#include <iostream>
using namespace __gnu_pbds;
using namespace std;
// order_of_key(k) : Number of items strictly smaller than k
// find_by_order(k) : -Kth element in a set (counting from zero)
// O(Lgn)
using ordered_set =
    tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>;
using ordered_multi_set = tree<int, null_type, less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
void m_erase(ordered_multi_set &OS, int val) {
    int index = OS.order_of_key(val);
    ordered_multi_set::iterator it = OS.find_by_order(index);
    if (*it == val) OS.erase(it);
}

```

### 3.2 Hash Table

```

// gp_hash_table, cc_hash_table, hash for pair
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

```



```
const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;
struct pair_hash {
    template <class T1, class T2>
    size_t operator () (const pair<T1,T2> &p) const {
        auto h1 = hash<T1>{}(p.first);
        auto h2 = hash<T2>{}(p.second);
        return h1 ^ h2;
    }
};
gp_hash_table<int, int, chash> table;
unordered_set<pll, pair_hash> st;
```

### 3.3 Rope

```
#include<ext/rope>
using namespace __gnu_cxx;
crope arr; string str; // or rope<T> arr; vector<T> str;
arr.insert(i, str); // Insert at position i with O(log n)
arr.erase(i, n); // Delete n characters from position i with O(log n)
arr.replace(i, n, str); // Replace n characters from position i with str with O(log n)
crope sub = arr.substr(i, n); // Get substring of length n starting from position i with O(log n)
```

### 3.4 Persistent Segment Tree

```
// persistent segment tree impl: sum tree
// initial tree index is 0
struct 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]);
}
};
```

### 3.5 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);
}

```

### 3.6 Bitset to Set

```

typedef unsigned long long ull;
const int sz = 100001 / 64 + 1;
struct bset {
    ull x[sz];
    bset(){
        memset(x, 0, sizeof x);
    }
    bset operator|(const bset &o) const {
        bset a;
        for (int i = 0; i < sz; i++) a.x[i] = x[i] | o.x[i];
        return a;
    }
    bset &operator|=(const bset &o) {
        for (int i = 0; i < sz; i++) x[i] |= o.x[i];
        return *this;
    }
    inline void add(int val){
        x[val >> 6] |= (1ull << (val & 63));
    }
    inline void del(int val){
        x[val >> 6] &= ~(1ull << (val & 63));
    }
    int kth(int k){
        int i, cnt = 0;
        for (i = 0; i < sz; i++){
            int c = __builtin_popcountll(x[i]);
            if (cnt + c >= k){
                ull y = x[i];
                int z = 0;
                for (int j = 0; j < 64; j++){
                    z += ((x[i] & (1ull << j)) != 0);
                    if (cnt + z == k) return i * 64 + j;
                }
            }
            cnt += c;
        }
        return -1;
    }
    int lower(int z){
        int i = (z >> 6), j = (z & 63);
        if (x[i]){
            for (int k = j - 1; k >= 0; k--) if (x[i] & (1ull << k)) return (i << 6) | k;
        }
        while (i > 0)
            if (x[--i])

```

```

for (j = 63;; j--)
  if (x[i] & (1ull << j))return (i << 6) | j;
return -1;
}
int upper(int z){
  int i = (z >> 6), j = (z & 63);
  if (x[i]){
    for (int k = j + 1; k <= 63; k++)if (x[i] & (1ull << k))return (i << 6) | k;
  }
  while (i < sz - 1)if (x[++i])for (j = 0;; j++)if (x[i] & (1ull << j))return (i << 6) | j;
  return -1;
}
};

```

### 3.7 Li-Chao Tree

```

struct Line {
  ll a, b;
  ll get(ll x) { return a * x + b; }
};
struct Node {
  int l, r; // child
  ll s, e; // range
  Line line;
};
struct Li_Chao {
  vector<Node> tree;
  void init(ll s, ll e) { tree.push_back({-1, -1, s, e, {0, -INF}}); }
  void update(int node, Line v) {
    ll s = tree[node].s, e = tree[node].e, m;
    m = (s + e) >> 1;
    Line low = tree[node].line, high = v;
    if (low.get(s) > high.get(s)) swap(low, high);
    if (low.get(e) <= high.get(e)) {
      tree[node].line = high;
      return;
    }
    if (low.get(m) < high.get(m)) {
      tree[node].line = high;
      if (tree[node].r == -1) {
        tree[node].r = tree.size();
        tree.push_back({-1, -1, m + 1, e, {0, -INF}});
      }
      update(tree[node].r, low);
    } else {
      tree[node].line = low;
      if (tree[node].l == -1) {
        tree[node].l = tree.size();
        tree.push_back({-1, -1, s, m, {0, -INF}});
      }
      update(tree[node].l, high);
    }
  }
  ll query(int node, ll x) {
    if (node == -1) return -INF;
    ll s = tree[node].s, e = tree[node].e, m;
    m = (s + e) >> 1;
    if (x <= m)
      return max(tree[node].line.get(x), query(tree[node].l, x));
    else
      return max(tree[node].line.get(x), query(tree[node].r, x));
  }
  // usage : seg.init(-2e8, 2e8); seg.update(0, {-c[i], c[i] * a[i - 1]});
  // seg.query(0, a[n - 1]);
};

```

### 3.8 Wavelet Tree

```

struct bit_array { // 0-indexed
  using u64 = unsigned long long;
  explicit bit_array(int sz) : n(sz + 64 >> 6), data(n), psum(n) {}
  void set(int i) { data[i >> 6] |= u64(1) << (i & 63); }
  int rank(int i, bool x) const {
    auto res = rank(i);
    return x ? res : i - res;
  }
  int rank(int l, int r, bool x) const {
    auto res = rank(r) - rank(l);
    return x ? res : r - l - res;
  }
  bool operator[](int i) const {
    return data[i >> 6] >> (i & 63) & 1;
  }
  void init() {
    for (int i = 1; i < n; i++)
      psum[i] = psum[i - 1] + __builtin_popcountll(data[i - 1]);
  }
private:
  int n;
  vector<u64> data;
  vector<int> psum;
  int rank(int i) const {
    return psum[i >> 6] + __builtin_popcountll(data[i >> 6] & (u64(1) << (i & 63)) - 1);
  }
};
// 전처리 O(nlgn) 각쿼리별 O(lgn)
template<typename T, enable_if_t<is_integral_v<T>, int> = 0>
struct wavelet_matrix { // 0-indexed
  explicit wavelet_matrix(vector<T> v) :
    n(v.size()),
    lg(__lg(*max_element(v.begin(), v.end())) + 1),
    data(lg, bit_array(n)),
    zero(lg, 0) {
    for (int i = lg - 1; i >= 0; i--) {
      for (int j = 0; j < n; j++) if (v[j] >> i & 1) data[i].set(j);
      data[i].init();
      auto it = stable_partition(v.begin(), v.end(), [&](T x) { return ~x >> i & 1; });
      zero[i] = it - v.begin();
    }
  }
  int rank(int l, int r, T x) const { // count i s.t. (l <= i < r) && (v[i] == x)
    if (x >> lg) return 0;
    for (int i = lg - 1; i >= 0; i--) {
      bool f = x >> i & 1;
      adjust(i, l, r, f);
    }
    return r - l;
  }
  int count(int l, int r, T x) const { // count i s.t. (l <= i < r) && (v[i] < x)
    if (x >> lg) return r - l + 1;
    int res = 0;
    for (int i = lg - 1; i >= 0; i--) {
      bool f = x >> i & 1;
      if (f) res += data[i].rank(l, r, 0);
      adjust(i, l, r, f);
    }
    return res;
  }
  T quantile(int l, int r, int k) const { // kth (0-indexed) smallest number in v[l, r)
    T res = 0;
    for (int i = lg - 1; i >= 0; i--) {
      int c = data[i].rank(l, r, 0);

```

```

    bool f = c <= k;
    if (f) res |= T(1) << i, k -= c;
    adjust(i, l, r, f);
}
return res;
}
private:
int n, lg;
vector<bit_array> data;
vector<int> zero;
void adjust(int i, int& l, int& r, bool f) const {
    if (!f) {
        l = data[i].rank(1, 0);
        r = data[i].rank(r, 0);
    }
    else {
        l = zero[i] + data[i].rank(1, 1);
        r = zero[i] + data[i].rank(r, 1);
    }
}
};

```

## 4 DP

### 4.1 Convex Hull Optimization

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

DP 점화식 풀

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

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

특수조건)  $a[i] \leq a[i+1]$  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 - l1.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);
    }
};

```

### 4.2 Divide & Conquer Optimization

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

조건 1) DP 점화식 풀

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

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

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

조건 2-1) 비용  $C$ 가 다음의 사각부등식을 만족하는 경우도 조건 2)를 만족하게 됨

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

```

//To get D[t][s...e] and range of j is [l, r]
void f(int t, int s, int e, int l, int r){
    if(s > e) return;
    int m = s + e >> 1;
    int opt = l;
    for(int i=l; i<=r; i++){
        if(D[t-1][opt] + C[opt][m] > D[t-1][i] + C[i][m]) opt = i;
    }
    D[t][m] = D[t-1][opt] + C[opt][m];
    f(t, s, m-1, l, opt);
    f(t, m+1, e, opt, r);
}

```

### 4.3 Knuth Optimization

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

조건 1) DP 점화식 풀

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

조건 2) 사각 부등식

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

조건 3) 단조성

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

결론) 조건 2, 3을 만족한다면  $A[i][j]$ 를  $D[i][j]$ 의 답이 되는 최소의  $k$ 라 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

```

for (i = 1; i <= n; i++) {
    cin >> a[i];
    s[i] = s[i - 1] + a[i];
    dp[i - 1][i] = 0;
    assist[i - 1][i] = i;
}
for (i = 2; i <= n; i++) {
    for (j = 0; j <= n - i; j++) {
        dp[j][i + j] = 1e9 + 7;
        for (k = assist[j][i + j - 1]; k <= assist[j + 1][i + j]; k++) {
            if (dp[j][i + j] > dp[j][k] + dp[k][i + j] + s[i + j] - s[j]) {
                dp[j][i + j] = dp[j][k] + dp[k][i + j] + s[i + j] - s[j];
                assist[j][i + j] = k;
            }
        }
    }
}
}

```

### 4.4 Bitset Optimization

```

#define private public
#include <bitset>
#undef private
#include <x86intrin.h>
template <size_t _Nw>

```

```
void _M_do_sub(_Base_bitset<_Nw> &A, const _Base_bitset<_Nw> &B) {
    for (int i = 0, c = 0; i < _Nw; i++)
        c = _subborrow_u64(c, A._M_w[i], B._M_w[i], (unsigned long long *)&A._M_w[i]);
}
template <>
void _M_do_sub(_Base_bitset<1> &A, const _Base_bitset<1> &B) {
    A._M_w -= B._M_w;
}
template <size_t _Nb>
bitset<_Nb> &operator--=(bitset<_Nb> &A, const bitset<_Nb> &B) {
    _M_do_sub(A, B);
    return A;
}
template <size_t _Nb>
inline bitset<_Nb> operator-(const bitset<_Nb> &A, const bitset<_Nb> &B) {
    bitset<_Nb> C(A);
    return C -= B;
}
template <size_t _Nw>
void _M_do_add(_Base_bitset<_Nw> &A, const _Base_bitset<_Nw> &B) {
    for (int i = 0, c = 0; i < _Nw; i++)
        c = _addcarry_u64(c, A._M_w[i], B._M_w[i], (unsigned long long *)&A._M_w[i]);
}
template <>
void _M_do_add(_Base_bitset<1> &A, const _Base_bitset<1> &B) {
    A._M_w += B._M_w;
}
template <size_t _Nb>
bitset<_Nb> &operator+=(bitset<_Nb> &A, const bitset<_Nb> &B) {
    _M_do_add(A, B);
    return A;
}
template <size_t _Nb>
inline bitset<_Nb> operator+(const bitset<_Nb> &A, const bitset<_Nb> &B) {
    bitset<_Nb> C(A);
    return C += B;
}
```

4.5 Kitamasa & Berlekamp-Massey

```
// Linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$
// Time: $O(n^2 \log k)$
ll get_nth(Poly S, Poly tr, ll k) { // get kth term of recurrence
    int n = sz(tr);
    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        rep(i, 0, n + 1) rep(j, 0, n + 1) res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i)
            rep(j, 0, n) res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
        return res;
    };
    Poly pol(n + 1), e(pol);
    pol[0] = e[1] = 1;
    for (++k; k; k /= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    }
    ll res = 0;
    rep(i, 0, n) res = (res + pol[i + 1] * S[i]) % mod;
    return res;
}

// Usage: berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}
// Time: $O(N^2)$
vector<ll> berlekampMassey(vector<ll> s) {
```

```
ll n = s.size(), L = 0, m = 0, d, coef;
vector<ll> C(n), B(n), T;
C[0] = B[0] = 1;
ll b = 1;
for (ll i = 0; i < n; i++) {
    ++m, d = s[i] % mod;
    for (ll j = 1; j <= L; j++) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C, coef = d * modpow(b, mod - 2) % mod;
    for (j = m; j < n; j++) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
    L = i + 1 - L, B = T, b = d, m = 0;
}
C.resize(L + 1), C.erase(C.begin());
for (ll& x : C) x = (mod - x) % mod;
return C;
}
ll guess_nth_term(vector<ll> x, lint n) {
    if (n < x.size()) return x[n];
    vector<ll> v = berlekamp_massey(x);
    if (v.empty()) return 0;
    return get_nth(v, x, n);
}
```

4.6 SOS(Subset of Sum) DP

```
//iterative version $O(N*2^N)$ with TC, MC
for(int mask = 0; mask < (1<<N); ++mask){
    dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
    for(int i = 0; i < N; ++i){
        if(mask & (1<<i)) dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
        else dp[mask][i] = dp[mask][i-1];
    }
    F[mask] = dp[mask][N-1];
}
// toggling, $O(N*2^N)$ with TC, $O(2^N)$ with MC
for(int i = 0; i < (1<<N); ++i) F[i] = A[i];
for(int i = 0; i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
    if(mask & (1<<i)) F[mask] += F[mask^(1<<i)];
}
```

5 Graph

5.1 SCC

```
// find SCCs in given directed graph
// $O(V+E)$
// the order of scc_idx constitutes a reverse topological sort
auto get_scc = [&](const auto& adj) { // 1-indexed
    const int n = adj.size() - 1;
    int dfs_cnt = 0, scc_cnt = 0;
    vector scc(n + 1, 0), dfn(n + 1, 0), s(0, 0);
    auto dfs = [&](const auto& self, int cur) -> int {
        int ret = dfn[cur] = ++dfs_cnt;
        s.push_back(cur);
        for (int nxt : adj[cur]) {
            if (!dfn[nxt]) ret = min(ret, self(self, nxt));
            else if (!scc[nxt]) ret = min(ret, dfn[nxt]);
        }
        if (ret == dfn[cur]) {
            scc_cnt++;
            while (s.size()) {
                int x = s.back(); s.pop_back();
                scc[x] = scc_cnt;
                if (x == cur) break;
            }
        }
    };
    for (int i = 1; i <= n; ++i) if (!dfn[i]) dfs(self, i);
    return scc;
}
```

```

    }
    return ret;
};
for (int i = 1; i <= n; i++) if (!dfn[i]) dfs(dfs, i);
return pair(scc_cnt, scc);
};

```

## 5.2 2-SAT

boolean variable  $b_i$  마다  $b_i$ 를 나타내는 정점,  $\neg b_i$ 를 나타내는 정점 2개를 만들. 각 clause  $b_i \vee b_j$  마다  $\neg b_i \rightarrow b_j$ ,  $\neg b_j \rightarrow b_i$  이렇게 edge를 이어줌. 그렇게 만든 그래프에서 SCC를 다 구함. 어떤 SCC 안에  $b_i$  와  $\neg b_i$ 가 같이 포함되어있다면 해가 존재하지 않음. 아니라면 해가 존재함. 해가 존재할 때 구체적인 해를 구하는 방법. 위에서 SCC를 구하면서 SCC DAG를 만들어준다. 거기서 위상정렬을 한 후, 앞에서부터 SCC를 하나씩 봐준다. 현재 보고있는 SCC에  $b_i$ 가 속해있는데 애가  $\neg b_i$  보다 먼저 등장했다면  $b_i = \text{false}$ , 반대의 경우라면  $b_i = \text{true}$ , 이미 값이 assign되었다면 pass.

## 5.3 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<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_edges[MAXN]; // list of edge ids in a bcc
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, eid = e.second;
        if (eid == par_edge) continue;
        if (visit[next] == 0) {
            stk.push_back(eid);
            ++child;
            dfs(next, eid);
            if (up[next] == visit[next]) bridge.push_back(eid);
            if (up[next] >= visit[nod]) {
                ++bcc_cnt;
                do {
                    auto lasteid = stk.back();
                    stk.pop_back();
                    bcc_edges[bcc_cnt].push_back(lasteid);
                    if (lasteid == eid) break;
                } while (!stk.empty());
                is_cut[nod]++;
            }
        }
        up[nod] = min(up[nod], up[next]);
    }
    else if (visit[next] < visit[nod]) {
        stk.push_back(eid);
        up[nod] = min(up[nod], visit[next]);
    }
}
if (par_edge == -1 && is_cut[nod] == 1)
    is_cut[nod] = 0;
}

```

```

// find BCCs & cut vertexes & 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_edges[i].clear();
bcc_cnt = 0;
for (int i = 0; i < n; ++i) {
    if (visit[i] == 0)
        dfs(i, -1);
}
}

```

## 5.4 Block-cut Tree

각 BCC 및 cut vertex가 block-cut tree의 vertex가 되며, BCC와 그 BCC에 속한 cut vertex 사이에 edge를 이어주면 된다.

## 5.5 Shortest Path Faster Algorithm

```

// shortest path faster algorithm
// average for random graph : O(E) , worst : O(VE)
const int MAXN = 20001;
const int INF = 100000000;
int n, m;
vector<pair<int, int>> graph[MAXN];
bool inqueue[MAXN];
int dist[MAXN];

void spfa(int st) {
    for (int i = 0; i < n; ++i) {
        dist[i] = INF;
    }
    dist[st] = 0;

    queue<int> q;
    q.push(st);
    inqueue[st] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inqueue[u] = false;
        for (auto& e : graph[u]) {
            if (dist[u] + e.second < dist[e.first]) {
                dist[e.first] = dist[u] + e.second;
                if (!inqueue[e.first]) {
                    q.push(e.first);
                    inqueue[e.first] = true;
                }
            }
        }
    }
}

```

## 5.6 Centroid Decomposition

```

// O(n lg n) for centroid decomposition
auto cent_decom = [](const auto& adj) {
    const int n = adj.size() - 1;
    vector sz(n + 1, 1), dep(n + 1, 0), par(n + 1, 0);
    auto dfs = [&](const auto& self, int cur, int prv) -> void {
        for (auto [nxt, cost] : adj[cur]) {
            if (nxt == prv) continue;
            self(self, nxt, cur);
            sz[cur] += sz[nxt];
        }
    };
    auto adjust = [&](int cur) {
        while (1) {
            int f = 0;
            for (auto [nxt, cost] : adj[cur]) {

```

```

    if (dep[nxt] || sz[cur] >= 2 * sz[nxt]) continue;
    sz[cur] -= sz[nxt], sz[nxt] += sz[cur];
    cur = nxt, f = 1;
    break;
}
if (!f) return cur;
}
};
auto rec = [&](const auto& self, int cur, int prv) -> void {
    cur = adjust(cur);
    par[cur] = prv;
    dep[cur] = dep[prv] + 1;
    for (auto [nxt, cost] : adj[cur]) {
        if (dep[nxt]) continue;
        self(self, nxt, cur);
    }
};
dfs(dfs, 1, 0);
rec(rec, 1, 0);
return pair(dep, par);
};

```

## 5.7 Lowest Common Ancestor

```

const int MAXN = 100;
const int MAXLN = 9;
vector<int> tree[MAXN];
int depth[MAXN];
int par[MAXLN][MAXN];

void dfs(int nod, int parent) {
    for (int next : tree[nod]) {
        if (next == parent) continue;
        depth[next] = depth[nod] + 1;
        par[0][next] = nod;
        dfs(next, nod);
    }
}

void prepare_lca() {
    const int root = 0;
    dfs(root, -1);
    par[0][root] = root;
    for (int i = 1; i < MAXLN; ++i)
        for (int j = 0; j < n; ++j)
            par[i][j] = par[i - 1][par[i - 1][j]];
}

// find Lowest common ancestor in tree between u & v
// assumption : must call 'prepare_lca' once before call this
// O(logV)
int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    if (depth[u] > depth[v]) {
        for (int i = MAXLN - 1; i >= 0; --i)
            if (depth[u] - (1 << i) >= depth[v])
                u = par[i][u];
    }
    if (u == v) return u;
    for (int i = MAXLN - 1; i >= 0; --i) {
        if (par[i][u] != par[i][v]) {
            u = par[i][u];
            v = par[i][v];
        }
    }
    return par[0][u];
}

```

```

}

```

## 5.8 Heavy-Light Decomposition

```

// heavy-light decomposition in O(n)
auto get_hld = [](auto adj) {
    const int n = adj.size() - 1;
    int ord = 0;
    vector sz(n + 1, 1), dep(n + 1, 0), par(n + 1, 0);
    vector in(n + 1, 0), out(n + 1, 0), top(n + 1, 0);
    auto dfs1 = [&](const auto& self, int cur, int prv) -> void {
        if (prv) adj[cur].erase(ranges::find(adj[cur], prv));
        for (int& nxt : adj[cur]) {
            dep[nxt] = dep[cur] + 1;
            par[nxt] = cur;
            self(self, nxt, cur);
            sz[cur] += sz[nxt];
            if (sz[adj[cur][0]] < sz[nxt]) swap(adj[cur][0], nxt);
        }
    };
    auto dfs2 = [&](const auto& self, int cur) -> void {
        in[cur] = ++ord;
        for (int nxt : adj[cur]) {
            top[nxt] = adj[cur][0] == nxt ? top[cur] : nxt;
            self(self, nxt);
        }
        out[cur] = ord;
    };
    dfs1(dfs1, 1, 0);
    dfs2(dfs2, top[1] = 1);
    return tuple(sz, dep, par, in, out, top);
};

```

## 5.9 Hall's Theorem

- Let  $G = (L \cup R, E)$  be a bipartite graph. For  $S \subseteq L$ , let  $N(S) \subseteq R$  be the set of vertices adjacent to some vertex in  $S$ . Then,  $\exists M$  matching in  $G$  that covers all vertex of  $L \Leftrightarrow \forall S \subseteq L, |S| \leq |N(S)|$
- Hall's Theorem is equivalent to the following statement: Let  $S = \{S_1, S_2, \dots, S_n\}$  be a set of sets. Then, we can choose  $x_i \in S_i$  for all  $i$  such that  $x_i \neq x_j$  for all  $i \neq j$  iff.  $\forall T \subseteq \{1, 2, \dots, n\}, |\bigcup_{i \in T} S_i| \geq |T|$ .

## 5.10 Stable Marriage

```

// man : 1~n, woman : n+1~2n, O(n^2) stable marriage
struct StableMarriage{
    int n; vector<vector<int>> g;
    StableMarriage(int n) : n(n), g(2*n+1) { for(int i=1; i<=n+n; i++) g[i].reserve(n); }
    void addEdge(int u, int v){ g[u].push_back(v); } // insert in decreasing order of preference.
    vector<int> run(){
        queue<int> q; vector<int> match(2*n+1), ptr(2*n+1);
        for(int i=1; i<=n; i++) q.push(i);
        while(q.size()){
            int i = q.front(); q.pop();
            for(int &p=ptr[i]; p<g[i].size(); p++){
                int j = g[i][p];
                if(!match[j]){ match[i] = j; match[j] = i; break; }
                int m = match[j], u = -1, v = -1;
                for(int k=0; k<g[j].size(); k++){
                    if(g[j][k] == i) u = k; if(g[j][k] == m) v = k;
                }
                if(u < v){
                    match[m] = 0; q.push(m); match[i] = j; match[j] = i; break;
                }
            }
        }
        return match; } //vector<int> run*/
};

```

### 5.11 Bipartite Matching (Kuhn)

```
auto bipartite_matching = [](const auto& adj) { // O(VE)
    const int n = adj.size() - 1;
    vector par(n + 1, 0), c(n + 1, 0);
    auto dfs = [&](const auto& self, int cur) -> bool {
        if (c[cur]++) return 0;
        for (int nxt : adj[cur])
            if (!par[nxt] || self(self, par[nxt]))
                return par[nxt] = cur, 1;
        return 0;
    };
    int res = 0;
    for (int i = 1; i <= n; i++) {
        fill(c.begin(), c.end(), 0);
        if (dfs(dfs, i)) res++;
    }
    return res;
};
```

### 5.12 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 unreachable.
//
// 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;
            }
        }
        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;
    }
};
```

```
        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;
};
```

### 5.13 Maximum Flow with Edge Demands

그래프  $G = (V, E)$  가 있고 source  $s$  와 sink  $t$  가 있다. 각 간선마다  $d(e) \leq f(e) \leq c(e)$  를 만족하도록 flow  $f(e)$  를 흘려야 한다. 이 때의 maximum flow 를 구하는 문제다. 먼저 모든 demand 를 합한 값  $D$  를 아래와 같이 정의한다.

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

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

이렇게 만든 새로운 그래프  $G'$  에서 maximum flow 를 구했을 때 그 값이  $D$  라면 원래 문제의 해가 존재하고, 그 값이  $D$  가 아니라면 원래 문제의 해는 존재하지 않는다.

위에서 maximum flow 를 구하고 난 상태의 residual graph 에서  $s'$  과  $t'$  을 떼버리고  $s$  에서  $t$  사이의 augment 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) };
}
};
```

5.14 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);
}
};
```

5.15 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;
    }
};
```

```
};

5.16 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];
    }
}
```

### 5.17 General Unweighted Maximum Matching(Tutte)

그래프  $G = (V, E)$ 에 대해 랜덤한 소수  $p$ 를 골라 다음과 같은  $|V| \times |V|$  행렬  $T$ 를 만들자. 이 때  $r_{i,j}$ 는  $[1, p-1]$  사이의 랜덤한 정수이다. 최대 매칭의 크기는 높은 확률로  $rank(T)/2$ 이다.

$$T_{i,j} = \begin{cases} r_{i,j} & \text{if } (i,j) \in E \wedge i < j \\ r_{j,i} & \text{if } (i,j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

### 5.18 General Weighted Maximum Matching(Blossom)

```
// O(N^3) (but fast in practice)
static const int INF = INT_MAX;
static const int N = 514;
struct edge{
    int u,v,w; edge(){ }
    edge(int ui,int vi,int wi)
        :u(ui),v(vi),w(wi){ }
};
```

```

};
int n,n_x;
edge g[N*2][N*2];
int lab[N*2];
int match[N*2],slack[N*2],st[N*2],pa[N*2];
int flo_from[N*2][N+1],S[N*2],vis[N*2];
vector<int> flo[N*2];
queue<int> q;
int e_delta(const edge &e){
    return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
}
void update_slack(int u,int x){
    if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]][x]))slack[x]=u;
}
void set_slack(int x){
    slack[x]=0;
    for(int u=1;u<=n;++u)
        if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
            update_slack(u,x);
}
void q_push(int x){
    if(x<=n)q.push(x);
    else for(size_t i=0;i<flo[x].size();i++)
        q_push(flo[x][i]);
}
void set_st(int x,int b){
    st[x]=b;
    if(x>n)for(size_t i=0;i<flo[x].size();++i)
        set_st(flo[x][i],b);
}
int get_pr(int b,int xr){
    int pr=find(flo[b].begin(),flo[b].end(),xr)-flo[b].begin();
    if(pr%2==1){
        reverse(flo[b].begin()+1,flo[b].end());
        return (int)flo[b].size()-pr;
    }else return pr;
}
void set_match(int u,int v){
    match[u]=g[u][v].v;
    if(u<=n) return;
    edge e=g[u][v];
    int xr=flo_from[u][e.u],pr=get_pr(u,xr);
    for(int i=0;i<pr;++i)set_match(flo[u][i],flo[u][i^1]);
    set_match(xr,v);
    rotate(flo[u].begin(),flo[u].begin()+pr,flo[u].end());
}
void augment(int u,int v){
    for(;;){
        int xnv=st[match[u]];
        set_match(u,v);
        if(!xnv)return;
        set_match(xnv,st[pa[xnv]]);
        u=st[pa[xnv]],v=xnv;
    }
}
int get_lca(int u,int v){
    static int t=0;
    for(++t;u||v;swap(u,v)){
        if(u==0)continue;
        if(vis[u]==t)return u;
        vis[u]=t;
        u=st[match[u]];
        if(u)u=st[pa[u]];
    }
    return 0;
}

```

```

void add_blossom(int u,int lca,int v){
    int b=n+1;
    while(b<=n_x&&st[b])++b;
    if(b>n_x)+n_x;
    lab[b]=0,S[b]=0;
    match[b]=match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for(int x=u,y,x!=lca;x=st[pa[y]])
        flo[b].push_back(x),flo[b].push_back(y=st[match[x]]),q_push(y);
    reverse(flo[b].begin()+1,flo[b].end());
    for(int x=v,y,x!=lca;x=st[pa[y]])
        flo[b].push_back(x),flo[b].push_back(y=st[match[x]]),q_push(y);
    set_st(b,b);
    for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
    for(int x=1;x<=n;++x)flo_from[b][x]=0;
    for(size_t i=0;i<flo[b].size();++i){
        int xs=flo[b][i];
        for(int x=1;x<=n_x;++x)
            if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b][x]))
                g[b][x]=g[xs][x],g[x][b]=g[x][xs];
        for(int x=1;x<=n;++x)
            if(flo_from[xs][x])flo_from[b][x]=xs;
    }
    set_slack(b);
}
void expand_blossom(int b){
    for(size_t i=0;i<flo[b].size();++i)
        set_st(flo[b][i],flo[b][i]);
    int xr=flo_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
    for(int i=0;i<pr;i+=2){
        int xs=flo[b][i],xns=flo[b][i+1];
        pa[xs]=g[xns][xs].u;
        S[xs]=1,S[xns]=0;
        slack[xs]=0,set_slack(xns);
        q_push(xns);
    }
    S[xr]=1,pa[xr]=pa[b];
    for(size_t i=pr+1;i<flo[b].size();++i){
        int xs=flo[b][i];
        S[xs]=-1,set_slack(xs);
    }
    st[b]=0;
}
bool on_found_edge(const edge &e){
    int u=st[e.u],v=st[e.v];
    if(S[v]==-1){
        pa[v]=e.u,S[v]=1;
        int nu=st[match[v]];
        slack[v]=slack[nu]=0;
        S[nu]=0,q_push(nu);
    }else if(S[v]==0){
        int lca=get_lca(u,v);
        if(!lca)return augment(u,v),augment(v,u),true;
        else add_blossom(u,lca,v);
    }
    return false;
}
bool matching(){
    memset(S+1,-1,sizeof(int)*n_x);
    memset(slack+1,0,sizeof(int)*n_x);
    q=queue<int>();
    for(int x=1;x<=n_x;++x)
        if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
    if(q.empty())return false;
    for(;;){

```

```

while(q.size()){
    int u=q.front();q.pop();
    if(S[st[u]]==1)continue;
    for(int v=1;v<=n;v++){
        if(g[u][v].w>0&&st[u]!=st[v]){
            if(e_delta(g[u][v])==0){
                if(on_found_edge(g[u][v]))return true;
            }else update_slack(u,st[v]);
        }
    }
    int d=INF;
    for(int b=n+1;b<=n_x;v++){
        if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
    }
    for(int x=1;x<=n_x;v++){
        if(st[x]==x&&slack[x]){
            if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
            else if(S[x]==0)d=min(d,e_delta(g[slack[x]][x])/2);
        }
    }
    for(int u=1;u<=n;v++){
        if(S[st[u]]==0){
            if(lab[u]<=d)return 0;
            lab[u]-=d;
        }else if(S[st[u]]==1)lab[u]+=d;
    }
    for(int b=n+1;b<=n_x;v++){
        if(st[b]==b){
            if(S[st[b]]==0)lab[b]+=d*2;
            else if(S[st[b]]==1)lab[b]-=d*2;
        }
    }
    q=queue<int>();
    for(int x=1;x<=n_x;v++){
        if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&e_delta(g[slack[x]][x])==0)
            if(on_found_edge(g[slack[x]][x]))return true;
    }
    for(int b=n+1;b<=n_x;v++){
        if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(b);
    }
    return false;
}
pair<long long,int> solve(){
    memset(match+1,0,sizeof(int)*n);
    n_x=n;
    int n_matches=0;
    long long tot_weight=0;
    for(int u=0;u<=n;v++){
        st[u]=u,flo[u].clear();
    }
    int w_max=0;
    for(int u=1;u<=n;v++){
        for(int v=1;v<=n;v++){
            flo_from[u][v]=(u==v?u:0);
            w_max=max(w_max,g[u][v].w);
        }
    }
    for(int u=1;u<=n;v++){
        lab[u]=w_max;
    }
    while(matching())n_matches++;
    for(int u=1;u<=n;v++){
        if(match[u]&&match[u]<u)
            tot_weight+=g[u][match[u]].w;
    }
    return make_pair(tot_weight,n_matches);
}
void add_edge( int ui , int vi , int wi ){
    g[ui][vi].w = g[vi][ui].w = wi;
}
void init( int _n ){
    n = _n;
    for(int u=1;u<=n;v++){
        for(int v=1;v<=n;v++){
            g[u][v]=edge(u,v,0);
        }
    }
}

```

## 5.19 Offline Dynamic Connectivity

```

struct OFDC{
    // offline dynamic connectivity in O(q lg^2 q)
    ll n,q; vector<ll>par, sz; vector<pll>query; stack<ll>st;
    vector<vector<pll>>tree; map<pll,ll>at;
    void update(ll node, ll tl, ll tr, ll l, ll r, pll v){
        if(r<tl||tr<l)return;
        if(l<=tl&&tr<=r) { tree[node].push_back(v); return; }
        ll tm=(tl+tr)>>1;
        update(node<<1,tl,tm,l,r,v); update(node<<1|1,tm+1,tr,l,r,v);
    }
    ll _find(ll x){ return x==par[x]?x:_find(par[x]); }
    bool _same(pll a){return _find(a.first)==_find(a.second);}
    bool _union(ll x, ll y){
        x=_find(x),y=_find(y); if(x==y)return false;
        if(sz[x]<sz[y])swap(x,y); par[y]=x; sz[x]+=sz[y]; st.push(y);
        return true;
    }
    void _delete(){
        if(st.empty())return;
        ll x=st.top(); st.pop(); sz[par[x]]-=sz[x]; par[x]=x;
    }
    void dfs(ll node, ll tl, ll tr){
        ll cnt=0;
        for(auto [x,y]:tree[node]) if(!_union(x,y)) cnt++;
        if(tl==tr){if(query[tl].first)cout<<(_same(query[tl])?1:0)<<'\n';}
        else{ll tm=(tl+tr)>>1;dfs(node<<1,tl,tm); dfs(node<<1|1,tm+1,tr);}
        while(cnt--)_delete();
    }
    void run(ll _n, ll _q, vector<tl1>_query){
        // 1 : add, 2 : del, 3 : query
        n=_n, q=_q; query.resize(q); tree.resize(q<<2|1),par.resize(n+1);
        sz.assign(n+1,1),iota(par.begin(),par.end(),0);
        for(ll i=0;i<q;i++){
            auto [op,a,b]=_query[i]; if(a>b)swap(a,b);
            if(op==1) at[{a,b}]=i;
            else if(op==2) update(1,0,q-1,at[{a,b}],i,{a,b}), at.erase({a,b});
            else query[i]={a,b};
        }
        for(auto [x,y]:at) update(1,0,q-1,y,q-1,x);
        dfs(1,0,q-1);
    }
};

```

## 6 Geometry

### 6.1 Basic Operations

```

const ld eps = 1e-12;
inline ll diff(ld lhs, ld rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}
inline bool is_between(ld check, ld a, ld b) {
    return (a < b) ? (a - eps < check && check < b + eps)
        : (b - eps < check && check < a + eps);
}
struct Point {
    ld 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*(ld t) const { return Point{x * t, y * t}; }
    int pos() const {

```

```

    if (y < 0) return -1;
    if (y == 0 && 0 <= x) return 0;
    return 1;
}
bool operator<(Point r) const { // sort by angle, ccw order from half line ≤x0,y=0
    if (pos() != r.pos()) return pos() < r.pos();
    return 0 < (x * r.y - y * r.x);
}
Point rotate(ld theta) const { // rotate ccw by theta
    return Point{x * cos(theta) - y * sin(theta), x * sin(theta) + y * cos(theta)};
};
struct Circle {
    Point center;
    ld r;
};
struct Line {
    Point pos, dir;
};
inline ld inner(const Point& a, const Point& b) { return a.x * b.x + a.y * b.y; }
inline ld outer(const Point& a, const Point& b) { return a.x * b.y - a.y * b.x; }
inline ll ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}
inline ll ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}
inline ld dist(const Point& a, const Point& b) { return sqrt(inner(a - b, a - b)); }
inline ld dist2(const Point& a, const Point& b) { return inner(a - b, a - b); }
inline ld dist(const Line& line, const Point& point, bool segment = false) {
    ld c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    ld 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) {
    ld mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    ld 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) {
    ld mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    ld t1 = -outer(b.pos - a.pos, b.dir) / mdet;
    ld 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) {
    ld wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
    ld 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};
}
Point outer_center(const Point& a, const Point& b, const Point& c) {
    Point d1 = b - a, d2 = c - a;
    ld area = outer(d1, d2);
    ld dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y + d1.y * d2.y * (d1.y - d2.y);
    ld 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;
    ld a = 2 * inner(line.dir, line.dir);
    ld b = 2 * (line.dir.x * (line.pos.x - circle.center.x) +
                line.dir.y * (line.pos.y - circle.center.y));
    ld c = inner(line.pos - circle.center, line.pos - circle.center) - circle.r * circle.r;
    ld det = b * b - 2 * a * c;
    ll 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;
    ll 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;
    }
    ld aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
    ld bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
    ld 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;
        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, ld r) {
    ld det = r * r / dist2(a, b) - 0.25;
    Circle circle;
    if (det < 0)
        circle.r = -1;
    else {
        ld 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;
}
Circle circle_from_2pts(const Point& a, const Point& b) {
    Circle circle;
    circle.center = (a + b) * 0.5;
    circle.r = dist(a, b) / 2;
    return circle;
}

```

## 6.2 Convex Hull & Rotating Calipers

```
// get all antipodal pairs with O(n)
// calculate convex hull with O(nlgn)
void antipodal_pairs(vector<Point>& pt, vector<Point>& 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.push_back(p);
        lo.push_back(p);
    }
    for (int i = 0, j = (int)lo.size() - 1; i + 1 < up.size() || j > 0; i++) {
        get_pair(up[i], lo[j]); // DO WHAT YOU WANT
        if (i + 1 == up.size()) --j;
        else if (j == 0) ++i;
        else if ((up[i + 1].y - up[i].y) * (lo[j].x - lo[j - 1].x) >
            (up[i + 1].x - up[i].x) * (lo[j].y - lo[j - 1].y)) ++i;
        else --j;
    }
    upper.insert(upper.end(), ++lower.rbegin(), --lower.rend());
    swap(upper, convex_hull);
}
```

### 6.3 Half Plane Intersection

```
typedef pair<long double, long double> pi;
bool z(long double x) { return fabs(x) < eps; }
struct line {
    long double a, b, c;
    bool operator<(const line &l) const {
        bool flag1 = pi(a, b) > pi(0, 0);
        bool flag2 = pi(l.a, l.b) > pi(0, 0);
        if (flag1 != flag2) return flag1 > flag2;
        long double t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b));
        return z(t) ? c * hypot(l.a, l.b) < l.c * hypot(a, b) : t > 0;
    }
    pi slope() { return pi(a, b); }
};
pi cross(line a, line b) {
    long double det = a.a * b.b - b.a * a.b;
    return pi((a.c * b.b - a.b * b.c) / det, (a.a * b.c - a.c * b.a) / det);
}
bool bad(line a, line b, line c) {
    if (ccw(pi(0, 0), a.slope(), b.slope()) <= 0) return false;
    pi crs = cross(a, b);
    return crs.first * c.a + crs.second * c.b >= c.c;
}
bool solve(vector<line> v, vector<pi> &solution) { // ax + by <= c;
    sort(v.begin(), v.end());
    deque<line> dq;
    for (auto &i : v) {
        if (!dq.empty() && z(ccw(pi(0, 0), dq.back().slope(), i.slope()))) continue;
        while (dq.size() >= 2 && bad(dq[dq.size() - 2], dq.back(), i)) dq.pop_back();
        while (dq.size() >= 2 && bad(i, dq[0], dq[1])) dq.pop_front();
        dq.push_back(i);
    }
    while (dq.size() > 2 && bad(dq[dq.size() - 2], dq.back(), dq[0])) dq.pop_back();
    while (dq.size() > 2 && bad(dq.back(), dq[0], dq[1])) dq.pop_front();
    vector<pi> tmp;
    for (int i = 0; i < dq.size(); i++) {
        line cur = dq[i], nxt = dq[(i + 1) % dq.size()];
        if (ccw(pi(0, 0), cur.slope(), nxt.slope()) <= eps) return false;
        tmp.push_back(cross(cur, nxt));
    }
}
```

```
solution = tmp;
return true;
}
```

### 6.4 Minimum Perimeter Triangle

```
bool cmp_x(pt a, pt b) {return a.x < b.x;}
bool cmp_y(pt a, pt b) {return a.y < b.y;}
double dist(pt a, pt b) {return hypot(abs(a.x - b.x), abs(a.y - b.y));}
double perimeter(pt a, pt b, pt c) {return dist(a, b) + dist(b, c) + dist(c, a);}
double dac3(int l, int r) {
    // get the smallest triangle perimeter in pts[l, r]
    if (r - l <= 1) return INF;
    if (r - l == 2) return perimeter(pts[l], pts[l + 1], pts[l + 2]);
    int mid = (l + r) / 2;
    double d1 = dac3(l, mid), d2 = dac3(mid + 1, r);
    double ans = min(d1, d2);
    vector<pt> strip;
    for (int i = l; i <= r; i++) {
        if (abs(pts[i].x - pts[mid].x) < ans) strip.push_back(pts[i]);
    }
    sort(strip.begin(), strip.end(), cmp_y);
    for (int i = 0; i < strip.size(); i++) {
        for (int j = i + 1; j < strip.size() && (strip[j].y - strip[i].y) < ans; j++) {
            for (int k = j + 1; k < strip.size() && (strip[k].y - strip[j].y) < ans; k++) {
                ans = min(ans, perimeter(strip[i], strip[j], strip[k]));
            }
        }
    }
    return ans;
}
double closest_triple(vector<pt> &pts) {
    sort(pts.begin(), pts.end(), cmp_x);
    return dac3(0, pts.size() - 1);
}
```

### 6.5 Minimum Enclosing Circle

```
Circle minimumEnclosingCost(vector<Point> v){
    // O(n^3) but if random_shuffle is used, it is amortized O(n)
    random_shuffle(v.begin(), v.end());
    Point p = {0, 0};
    ld r = 0; int n = v.size();
    for(int i=0; i<n; i++){
        if(dist(p, v[i]) > r){
            p = v[i], r = 0;
            for(int j=0; j<i; j++){
                if(dist(p, v[j]) > r){
                    auto tmp=circle_from_2pts(v[i], v[j]);
                    p = tmp.center, r = tmp.r;
                    for(int k=0; k<j; k++){
                        if(dist(p, v[k]) > r){
                            auto tmp=circle_from_3pts(v[i], v[j], v[k]);
                            p = tmp.center, r = tmp.r;
                        }
                    }
                }
            }
        }
    }
    return {p, r};
}
```

### 6.6 Point in Polygon Test

```
inline ld 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
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;
}
```

## 6.7 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;
}
```

## 6.8 Number of Point in Triangle

```
// N arr , M brr points, O(NMLg(NM)+Q) solution
// query : 3 points a,b,c : arr index
// find brr points in triangle arr_abc(line excluded)
```

```
template<class Int = long long, class Int2 = long long>
struct VecI2 {
    Int x, y;
    VecI2() : x(0), y(0) {}
    VecI2(Int _x, Int _y) : x(_x), y(_y) {}
    VecI2 operator+(VecI2 r) const { return VecI2(x+r.x, y+r.y); }
    VecI2 operator-(VecI2 r) const { return VecI2(x-r.x, y-r.y); }
    VecI2 operator-() const { return VecI2(-x, -y); }
    Int2 operator*(VecI2 r) const { return Int2(x) * Int2(r.x) + Int2(y) * Int2(r.y); }
    Int2 operator^(VecI2 r) const { return Int2(x) * Int2(r.y) - Int2(y) * Int2(r.x); }
    static bool compareYX(VecI2 a, VecI2 b){ return a.y < b.y || (!(b.y < a.y) && a.x < b.x); }
    static bool compareXY(VecI2 a, VecI2 b){ return a.x < b.x || (!(b.x < a.x) && a.y < b.y); }
};
using namespace std;
using Vec = VecI2<ll>;

void func(vector<Vec>& A, vector<Vec>& B){
    auto pointL = vector<int>(N); // bx < Ax
    auto pointM = vector<int>(N); // bx = Ax
    rep(i,N) rep(j,M) if(A[i].y == B[j].y){
        if(B[j].x < A[i].x) pointL[i]++;
        if(B[j].x == A[i].x) pointM[i]++;
    }
    auto edgeL = vector<vector<int>>(N, vector<int>(N)); // bx < Lerp(Ax, Bx)
    auto edgeM = vector<vector<int>>(N, vector<int>(N)); // bx = Lerp(Ax, Bx)
    rep(a,N){
        struct PointId { int i; int c; Vec v; };
        vector<PointId> points;
        rep(b,N) if(A[a].y < A[b].y) points.push_back({ b, 0, A[b] - A[a] });
        rep(b,M) if(A[a].y < B[b].y) points.push_back({ b, 1, B[b] - A[a] });
        rep(b,N) if(A[a].y < A[b].y) points.push_back({ b, 2, A[b] - A[a] });
        sort(points.begin(), points.end(), [&](const PointId& l, const PointId& r){
            ll det = l.v ^ r.v;
            if(det != 0) return det < 0;
            return l.c < r.c;
        });
        int qN = points.size();
        vector<int> queryOrd(qN); rep(i,qN) queryOrd[i] = i;
        sort(queryOrd.begin(), queryOrd.end(), [&](int l, int r){
            return pll{points[l].v.y, points[l].c%2} < pll{points[r].v.y, points[r].c%2};
        });
        vector<int> BIT(qN);
        for(int qi=0; qi<qN; qi++){
            int q = queryOrd[qi];
            if(points[q].c == 0){
                int buf = 0;
                int p = q+1;
                while(p > 0){ buf += BIT[p-1]; p -= p & -p; }
                edgeL[a][points[q].i] = buf;
            } else if(points[q].c == 1) {
                int p = q+1;
                while(p <= qN){ BIT[p-1]++; p += p & -p; }
            } else {
                int buf = 0;
                int p = q+1;
                while(p > 0){ buf += BIT[p-1]; p -= p & -p; }
                edgeM[a][points[q].i] = buf;
            }
        }
    }
    rep(b,N) edgeM[a][b] -= edgeL[a][b];
}

int Q; cin >> Q;
rep(qi, Q){
    int a,b,c; cin >> a >> b >> c;
    if(Vec::compareYX(A[b], A[a])) swap(a, b);
```



```

if(Vec::compareYX(A[c], A[b])) swap(b, c);
if(Vec::compareYX(A[b], A[a])) swap(a, b);
auto det = (A[a] - A[c]) ^ (A[b] - A[c]);
int ans = 0;
if(det != 0){
    if(A[a].y == A[b].y){ // A[a].x < A[b].x
        ans = edgeL[b][c] - (edgeL[a][c] + edgeM[a][c]);
    } else if(A[b].y == A[c].y){ // A[b].x < A[c].x
        ans = edgeL[a][c] - (edgeL[a][b] + edgeM[a][b]);
    } else if(det < 0){
        ans += edgeL[a][c];
        ans -= edgeL[b][c] + edgeM[b][c];
        ans -= edgeL[a][b] + edgeM[a][b];
        ans -= pointL[b] + pointM[b];
    } else {
        ans += edgeL[a][b];
        ans += edgeL[b][c];
        ans += pointL[b];
        ans -= edgeL[a][c] + edgeM[a][c];
    }
}
cout << ans << '\n';
}
}

```

## 6.9 Voronoi Diagram

```

typedef pair<ld, ld> pdd;
const ld EPS = 1e-12;
ll dcmp(ld x){ return x < -EPS ? -1 : x > EPS ? 1 : 0; }
ld operator / (pdd a, pdd b){ return a.first * b.second - a.second * b.first; }
pdd operator * (ld b, pdd a){ return pdd(b * a.first, b * a.second); }
pdd operator + (pdd a, pdd b){ return pdd(a.first + b.first, a.second + b.second); }
pdd operator - (pdd a, pdd b){ return pdd(a.first - b.first, a.second - b.second); }
ld sq(ld x){ return x*x; }
ld size(pdd p){ return hypot(p.first, p.second); }
ld sz2(pdd p){ return sq(p.first) + sq(p.second); }
pdd r90(pdd p){ return pdd(-p.second, p.first); }
pdd inter(pdd a, pdd b, pdd u, pdd v){ return u + (((a-u)/b)/(v/b))*v; }
pdd get_circumcenter(pdd p0, pdd p1, pdd p2){
    return inter(0.5*(p0+p1), r90(p0-p1), 0.5*(p1+p2), r90(p1-p2)); }
ld pb_int(pdd left, pdd right, ld sweepline){
    if(dcmp(left.second-right.second) == 0) return (left.first + right.first) / 2.0;
    ll sign = left.second < right.second ? -1 : 1;
    pdd v = inter(left, right-left, pdd(0, sweepline), pdd(1, 0));
    ld d1 = sz2(0.5 * (left+right) - v), d2 = sz2(0.5 * (left-right));
    return v.first + sign * sqrt(max(0.0, d1 - d2)); }
class Beachline{
public:
    struct node{
        node(){ }
        node(pdd point, ll idx):point(point), idx(idx), end(0),
            link[0, 0], par(0), prv(0), nxt(0) {}
        pdd point; ll idx; ll end;
        node *link[2], *par, *prv, *nxt;
    };
    node *root;
    ld sweepline;
    Beachline(): sweepline(-1e20), root(NULL){ }
    inline ll dir(node *x){ return x->par->link[0] != x; }
    void rotate(node *n){
        node *p = n->par; ll d = dir(n); p->link[d] = n->link[!d];
        if(n->link[!d]) n->link[!d]->par = p; n->par = p->par;
        if(p->par) p->par->link[dir(p)] = n; n->link[!d] = p; p->par = n;
    } void splay(node *x, node *f = NULL){
        while(x->par != f){

```

```

            if(x->par->par == f);
            else if(dir(x) == dir(x->par)) rotate(x->par);
            else rotate(x);
            rotate(x);
        }
        if(f == NULL) root = x;
    } void insert(node *n, node *p, ll d){
        splay(p); node* c = p->link[d];
        n->link[d] = c; if(c) c->par = n; p->link[d] = n; n->par = p;
        node *prv = !d?p->prv:p, *nxt = !d?p->nxt:
        n->prv = prv; if(prv) prv->nxt = n; n->nxt = nxt; if(nxt) nxt->prv = n;
    } void erase(node* n){
        node *prv = n->prv, *nxt = n->nxt;
        if(!prv && !nxt){ if(n == root) root = NULL; return; }
        n->prv = NULL; if(prv) prv->nxt = nxt;
        n->nxt = NULL; if(nxt) nxt->prv = prv;
        splay(n);
        if(!nxt){
            root->par = NULL; n->link[0] = NULL;
            root = prv;
        }
        else{
            splay(nxt, n); node* c = n->link[0];
            nxt->link[0] = c; c->par = nxt; n->link[0] = NULL;
            n->link[1] = NULL; nxt->par = NULL; root = nxt;
        }
    } bool get_event(node* cur, ld &next_sweep){
        if(!cur->prv || !cur->nxt) return false;
        pdd u = r90(cur->point - cur->prv->point);
        pdd v = r90(cur->nxt->point - cur->point);
        if(dcmp(u/v) != 1) return false;
        pdd p = get_circumcenter(cur->point, cur->prv->point, cur->nxt->point);
        next_sweep = p.second + size(p - cur->point); return true;
    } node* find_bl(ld x){
        node* cur = root;
        while(cur){
            ld left = cur->prv ? pb_int(cur->prv->point, cur->point, sweepline) : -1e30;
            ld right = cur->nxt ? pb_int(cur->point, cur->nxt->point, sweepline) : 1e30;
            if(left <= x && x <= right){ splay(cur); return cur; }
            cur = cur->link[x > right];
        }
    }
};
using BNode = Beachline::node; static BNode* arr; static ll sz;
static BNode* new_node(pdd point, ll idx){
    arr[sz] = BNode(point, idx); return arr + (sz++); }
struct event{
    event(ld sweep, ll idx):type(0), sweep(sweep), idx(idx){ }
    event(ld sweep, BNode* cur):type(1), sweep(sweep), prv(cur->prv->idx), cur(cur), nxt(cur->nxt->idx){ }
    ll type, idx, prv, nxt;
    BNode* cur;
    ld sweep;
    bool operator>(const event &l)const{ return sweep > l.sweep; }
};
void Voronoi(vector<pdd> &input, vector<pdd> &vertex, vector<p11> &edge, vector<p11> &area){
    Beachline bl = Beachline();
    priority_queue<event, vector<event>, greater<event>> events;
    auto add_edge = [&](ll u, ll v, ll a, ll b, BNode* c1, BNode* c2){
        if(c1) c1->end = edge.size()*2;
        if(c2) c2->end = edge.size()*2 + 1;
        edge.emplace_back(u, v);
        area.emplace_back(a, b);
    };
    auto write_edge = [&](ll idx, ll v){ idx%2 == 0 ? edge[idx/2].first = v : edge[idx/2].second = v; };
}

```

```
auto add_event = [&](BNode* cur){ ld nxt; if(bl.get_event(cur, nxt)) events.emplace(nxt, cur);
};
ll n = input.size(), cnt = 0;
arr = new BNode[n*4]; sz = 0;
sort(input.begin(), input.end(), [](const pdd &l, const pdd &r){
    return l.second != r.second ? l.second < r.second : l.first < r.first; });
BNode* tmp = bl.root = new_node(input[0], 0), *t2;
for(ll i = 1; i < n; i++){
    if(dcmp(input[i].second - input[0].second) == 0){
        add_edge(-1, -1, i-1, i, 0, tmp);
        bl.insert(t2 = new_node(input[i], i), tmp, 1);
        tmp = t2;
    }
    else events.emplace(input[i].second, i);
}
while(events.size()){
    event q = events.top(); events.pop();
    BNode *prv, *cur, *nxt, *site;
    ll v = vertex.size(), idx = q.idx;
    bl.sweepline = q.sweep;
    if(q.type == 0){
        pdd point = input[idx];
        cur = bl.find_bl(point.first);
        bl.insert(site = new_node(point, idx), cur, 0);
        bl.insert(prv = new_node(cur->point, cur->idx), site, 0);
        add_edge(-1, -1, cur->idx, idx, site, prv);
        add_event(prv); add_event(cur);
    }
    else{
        cur = q.cur, prv = cur->prv, nxt = cur->nxt;
        if(!prv || !nxt || prv->idx != q.prv || nxt->idx != q.nxt) continue;
        vertex.push_back(get_circumcenter(prv->point, nxt->point, cur->point));
        write_edge(prv->end, v); write_edge(cur->end, v);
        add_edge(v, -1, prv->idx, nxt->idx, 0, prv);
        bl.erase(cur);
        add_event(prv); add_event(nxt);
    }
}
delete arr;
}
```

6.10 KD-Tree

```
// k-d tree : find closest point from arbitrary point
// Time Complexity : average O(log N), worst O(N)

struct KNode{
    pll v; bool dir;
    ll sx, ex, sy, ey;
    KNode(){ sx = sy = inf; ex = ey = -inf; }
};

const auto xcmp = [](pll a, pll b){ return tie(a.x, a.y) < tie(b.x, b.y); };
const auto ycmp = [](pll a, pll b){ return tie(a.y, a.x) < tie(b.y, b.x); };
struct KDTree{
    // Segment Tree Size
    static const int S = 1 << 18;
    KNode nd[S]; int chk[S];
    vector<pll> v;
    KDTree(){ init(); }
    void init(){ memset(chk, 0, sizeof chk); }
    void _build(int node, int s, int e){
        chk[node] = 1;
        nd[node].sx = min_element(v.begin()+s, v.begin()+e+1, xcmp)->x;
        nd[node].ex = max_element(v.begin()+s, v.begin()+e+1, xcmp)->x;
        nd[node].sy = min_element(v.begin()+s, v.begin()+e+1, ycmp)->y;
        nd[node].ey = max_element(v.begin()+s, v.begin()+e+1, ycmp)->y;
    }
};
```

```
nd[node].dir = !nd[node/2].dir;

if(nd[node].dir) sort(v.begin()+s, v.begin()+e+1, ycmp);
else sort(v.begin()+s, v.begin()+e+1, xcmp);

int m = s + e >> 1; nd[node].v = v[m];
if(s <= m-1) _build(node << 1, s, m-1);
if(m+1 <= e) _build(node << 1 | 1, m+1, e);
}

void build(const vector<pll> &_v){
    v = _v; sort(all(v));
    _build(1, 0, v.size()-1);
}

ll query(pll t, int node = 1){
    ll tmp, ret = inf;
    if(t != nd[node].v) ret = min(ret, dst(t, nd[node].v));
    bool x_chk = (!nd[node].dir && xcmp(t, nd[node].v));
    bool y_chk = (nd[node].dir && ycmp(t, nd[node].v));
    if(x_chk || y_chk){
        if(chk[node << 1]) ret = min(ret, query(t, node << 1));
        if(chk[node << 1 | 1]){
            if(nd[node].dir) tmp = nd[node << 1 | 1].sy - t.y;
            else tmp = nd[node << 1 | 1].sx - t.x;
            if(tmp*tmp < ret) ret = min(ret, query(t, node << 1 | 1));
        }
    }
    else{
        if(chk[node << 1 | 1]) ret = min(ret, query(t, node << 1 | 1));
        if(chk[node << 1]){
            if(nd[node].dir) tmp = nd[node << 1].ey - t.y;
            else tmp = nd[node << 1].ex - t.x;
            if(tmp*tmp < ret) ret = min(ret, query(t, node << 1));
        }
    }
    return ret;
}
};
```

6.11 Pick’s theorem

격자점으로 구성된 simple polygon에 대해  $i$ 는 polygon 내부의 격자수,  $b$ 는 polygon 선분 위 격자수,  $A$ 는 polygon 넓이라고 할 때  $A = i + \frac{b}{2} - 1$ .

7 String

7.1 KMP

```
void calculate_pi(vector<int>& pi, const string& 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 string& text, const string& 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;
}
```

7.2 Z Algorithm

```
// Z[i] : maximum common prefix length of &s[0] and &s[i] with O(|s|)
auto get_z = [](const string& s) {
    const int n = s.size(); vector z(n, 0); z[0] = n;
    for (int i = 1, l = -1, r = -1; i < n; i++) {
        if (i <= r) z[i] = min(r - i + 1, z[i - 1]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
        if (r < i + z[i] - 1) l = i, r = i + z[i] - 1;
    }
    return z;
};
```

7.3 Aho-Corasick

```
struct aho_corasick_with_trie {
    const ll MAXN = 100005, MAXC = 26;
    ll trie[MAXN][MAXC], fail[MAXN], term[MAXN], piv = 0;
    void init(vector<string> &v) {
        memset(trie, 0, sizeof(trie));memset(fail, 0, sizeof(fail));
        memset(term, 0, sizeof(term));piv = 0;
        for (auto &i : v) {
            ll p = 0;
            for (auto &j : i) {
                if (!trie[p][j]) trie[p][j] = ++piv;
                p = trie[p][j];
            }
            term[p] = 1;
        }
        queue<ll> que;
        for (ll i = 0; i < MAXC; i++) if (trie[0][i]) que.push(trie[0][i]);
        while (!que.empty()) {
            ll x = que.front(); que.pop();
            for (ll i = 0; i < MAXC; i++) if (trie[x][i]) {
                ll p = fail[x];
                while (p && !trie[p][i]) p = fail[p];
                p = trie[p][i];
                fail[trie[x][i]] = p;
                if (term[p]) term[trie[x][i]] = 1;
                que.push(trie[x][i]);
            }
        }
    }
    bool query(string &s) {
        ll p = 0;
        for (auto &i : s) {
            while (p && !trie[p][i]) p = fail[p];
            p = trie[p][i]; if (term[p]) return 1;
        }
        return 0;
    }
};
```

7.4 Suffix Array with LCP

```
// calculates suffix array with O(n*logn)
auto get_sa(const string& s) {
    const int n = s.size(), m = max(256, n) + 1;
    vector<int> sa(n), r(n << 1), nr(n << 1), cnt(m), idx(n);
    for (int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];
    for (int d = 1; d < n; d <= 1) {
        auto cmp = [&](int a, int b) { return r[a] < r[b] || r[a] == r[b] && r[a + d] < r[b + d];};
        for (int i = 0; i < m; ++i) cnt[i] = 0;
        for (int i = 0; i < n; ++i) cnt[r[i + d]]++;
        for (int i = n - 1; ~i; --i) idx[--cnt[r[i + d]]] = i;
        nr[sa[0]] = 1;
        for (int i = 1; i < n; ++i) nr[sa[i]] = nr[sa[i - 1]] + cmp(sa[i - 1], sa[i]);
        for (int i = 0; i < n; ++i) r[i] = nr[i];
        if (r[sa[n - 1]] == n) break;
    }
    return sa;
}
```

```
for (int i = 0; i < m; ++i) cnt[i] = 0;
for (int i = 0; i < n; ++i) cnt[r[i + d]]++;
for (int i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
for (int i = n - 1; ~i; --i) idx[--cnt[r[i + d]]] = i;
for (int i = 0; i < m; ++i) cnt[i] = 0;
for (int i = 0; i < n; ++i) cnt[r[i]]++;
for (int i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
for (int i = n - 1; ~i; --i) sa[--cnt[r[idx[i]]]] = idx[i];
nr[sa[0]] = 1;
for (int i = 1; i < n; ++i) nr[sa[i]] = nr[sa[i - 1]] + cmp(sa[i - 1], sa[i]);
for (int i = 0; i < n; ++i) r[i] = nr[i];
if (r[sa[n - 1]] == n) break;
}
return sa;
}
// calculates lcp array. it needs suffix array & original sequence with O(n)
auto get_lcp(const string& s, const auto& sa) {
    const int n = s.size(); vector lcp(n - 1, 0), isa(n, 0);
    for (int i = 0; i < n; i++) isa[sa[i]] = i;
    for (int i = 0, k = 0; i < n; i++) if (isa[i]) {
        for (int j = sa[isa[i] - 1]; s[i + k] == s[j + k]; k++);
        lcp[isa[i] - 1] = k ? k - 1 : 0;
    }
    return lcp;
}
```

7.5 Manacher's Algorithm

```
// find longest palindromic span for each element in str with O(|str|)
auto manacher = [](const string& s) {
    const int n = s.size(); vector d(n, 0);
    for (int i = 0, l = -1, r = -1; i < n; i++) {
        if (i < r) d[i] = min(r - i, d[l + r - i]);
        while (d[i] < min(i + 1, n - i) && s[i - d[i]] == s[i + d[i]]) d[i]++;
        if (i + d[i] > r) l = i - d[i], r = i + d[i];
    }
    return d;
};
```

7.6 EERTREE

```
template<class S = string , class T = typename S::value_type>
struct eertree {
    struct node { int len, link;map<T, int> child; };
    S s; vector<node> data; int max_suf;
    eertree() : max_suf(1) {
        data.push_back({ -1, 0 }); data.push_back({ 0, 0 });
    }
    void add(T c) {
        s.push_back(c); int i = max_suf;
        while (data[i].len + 2 > s.size() || s[s.size() - data[i].len - 2] != c) i = data[i].link;
        if (data[i].child.count(c) == 0) {
            if (i == 0) data[i].child[c] = data.size(), data.push_back({ data[i].len + 2, 1 });
            else {
                int j = data[i].link; while (s[s.size() - data[j].len - 2] != c) j = data[j].link;
                data[i].child[c] = data.size(); data.push_back({ data[i].len + 2, data[j].child[c] });
            }
        }
        i = data[i].child[c];
        max_suf = i;
    }
};
```