



Nanyang Technological University

NTRLover

Pu Fanyi, Shao Siyang, Shi Zhengyu

2023-11-30

1 Combinatorial
2 Mathematics
3 Data structures
4 Geometry
5 Graph
6 Numerical
7 Number theory
8 Strings
9 Various

Combinatorial (1)

FactMod.h
Description: Compute n! % p in O(log_p n).
Also Lucas's Theorem: (n choose k) % p = product of (ni choose ki) % p
Corollary. C(m, n) % p = 0 iff at least one digit of n > m.
int factmod(int n, int p) {
 vector<int> f(p);
 f[0] = 1;
 for (int i = 1; i < p; i++)
 f[i] = f[i-1] * i % p;
 int res = 1;
 while (n > 1) {
 if ((n/p) % 2)
 res = p - res;
 res = res * f[n%p] % p;
 n /= p;
 }
 return res;
}

1.1 Notes
1.1.1 Twelvethfold Way

Table with 5 columns: Ball, Bag, Any, Iwj. <= 1, (x >= n) Surj. >= 1. Rows show combinations of balls and bags with their respective counts.

1.1.2 Identities on C
Vandermonde's Id. (m+n choose k) = sum_{r in [0,k]} (m choose r) (n choose k-r)
Pascal's Rule. (n choose k) = (n-1 choose k-1) + (n-1 choose k)
Hockey Stick Id. sum n over same k
Binomial Theorem. (x+y)^n = sum_{k in [0,n]} C_n^k * x^{n-k} * y^k
Stirling's Approximation. n! approx sqrt(2*pi*n) * (n/e)^n with error 1/(n+1)
Lucas Theorem. (n choose k) mod p = product (ni choose ki) mod p
Convention is C(n, k) = 0 if k > n. Corollary. mod = 0 iff there is a digit of n greater than k.

1.1.3 Other Basic Comb Functions
Stirling number of first kind. Count number of permutations on n item with k cycles.

Equations for Stirling numbers of the first kind: c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), c(0, 0) = 1, sum_{k=0}^n c(n, k)x^k = x(x+1)...(x+n-1), c(8,k)=8,0,5040,13068, c(n,2)=0,0,1,3,11

Eulerian Numbers. Number of permutation s.t. exactly k elems are greater than prev elem.

Equations for Eulerian numbers: E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k), E(n, 0) = E(n, n-1) = 1, E(n, k) = sum_{j=0}^k (-1)^j * (n+1 choose j) * (k+1-j)^n

Stirling Number of second kind. Partition of n distinct numbers into exact k group.

Equations for Stirling numbers of the second kind: S(n, k) = S(n-1, k-1) + kS(n-1, k), S(n, 1) = S(n, n) = 1, S(n, k) = 1/k! * sum_{i in [0,k]} (-1)^{k-i} * (k choose i) * i^n = sum_{i=0}^k (-1)^{k-i} * i^n / (i! * (k-i)!), 11 S2(11 n, 11 k) { \{ 11 ans = 0; FOR(i, 0, k) { 11 sgn = ((k - i) % 2 ? -1 : 1); 11 in = fpm(i, n, p); 11 bottom = invfac[i] * invfac[k - i] % p; ans = (ans + sgn * in * bottom % p) % p; \} return (ans % p + p) % p; \}

Bell number. Number of partition of n distinct nums. B(n) = 1, 1, 2, 5, 15, 52... For prime p, B(p^m + n) congruent mB(n) + B(n+1) mod p. Partition Function. Number of ways of writing n as sum of positive integers, disregarding order of summands. p(0) = 1, p(n) = sum_{k in [-inf, 0) union (0, inf]} (-1)^{k+1} p(n - k(3k-1)/2). p(n) ~ 0.145/n * exp(2.56*sqrt(n)). p = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, p(20) = 627, p(50) approx 2e5, p(100) approx 2e8.

1.1.4 Misc Functions
Derangements. D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = floor(n!/e)

Catalan Numbers. Various C_n = 1/(n+1) * (2n choose n) = (2n choose n) / (n+1) = (2n)! / ((n+1)!n!). C_0 = 1, C_{n+1} = sum_{i=0}^n C_i C_{n-i}, C_n = 1, 1, 2, 5, 14, 42, 132, 429...

- Applications.
• number of RBS of length 2n, or ways n+1 object can be completely parenthesized
• number of full binary tree with n+1 leaves (n internal operations)
• number of way to cut n+2 polygon into triangles.

Burnside's Lemma. Number of elements in X up to symmetry action group G is 1/|G| * sum_{g in G} |X^g|, where X^g are elements fixed by g.

1.1.5 Applications
Number of Trees.
n vertice: n^{n-2}
k existing tree of size n_i: n_1 n_2 ... n_k n^{k-2}
with degree d_i: (n-2)! / ((d_1-1)! ... (d_n-1)!)

Interleave k sorted incomparable sequence.
n=2: same as finding m in n+m
n=k: multinomial: (n_1 + ... + n_k) / (n_1! n_2! ... n_k!)

Chromatic Polynomial
P(G, t) describes number of ways to t-color G. e.g. when G has 1 point. P(G, t) = t.
t(t-1)(t-2)...(t-(n-1)) for complete graph K_n

$$t(t-1)^{n-1} \text{ for tree } C_n$$

$$(t-1)^n + (-1)^n(t-1) \text{ for ring } C_n$$

假设给出图为 G ，定义一个 $n \times n$ 的矩阵 $D(G)$ 表示 G 个点的度数，当 $i \neq j$ 时， $d_{i,j} = 0$ ，当 $i = j$ 时， $d_{i,j}$ 等于节点 i 的度数。再定义一个 $n \times n$ 的矩阵 A_G 表示 G 的邻接矩阵， $A_{i,j}$ 表示 i 到 j 的边数。然后我们定义基尔霍夫矩阵 $C(G) = D(G) - A(G)$ 。则 G 中生成树个数等于 $C(G)$ 中任意一个 $n-1$ 阶主子式的行列式的绝对值。所谓一个矩阵 M 的 $n-1$ 阶主子式就是对于两个整数 $r(1 \leq r \leq n)$ ，将 M 去掉第 r 行和第 r 列后形成的 $n-1$ 阶的矩阵，记作 $M_{r,}$ 。

5.1.2 最小生成树计数

发现每个最小生成树每种边权的边数应该是一样的，且将这些边去掉后所得的连通块相同。
于是我们考虑建出一棵最小生成树，枚举边权然后把原来最小生成树上该边权的边删掉，然后跑矩阵树。

复杂度？假设离散之后边权 i 共有 a_i 条边，那么显然 $\sum a_i = m$ 。如果图没有重边，则 Kruscal 复杂度 $\mathcal{O}(m \log m)$ ，矩阵树复杂度为 $\mathcal{O}(\sum (n+m+\min(n,a_i)^3))$ ，由于没有重边，前面的 $n+m$ 那一项卡满不过 $\mathcal{O}(m \times (n+m)) = \mathcal{O}(m^2) = \mathcal{O}(n^2m)$ ，而后面那一项当每个 a_i 取到 n 时最大，即 $\mathcal{O}(\frac{m}{n} \times n^3) = \mathcal{O}(n^2m)$ ，所以总复杂度 $\mathcal{O}(n^2m)$ 。

Mathematics (2)

2.1 OEIS

Fibonacci Numbers. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418

$$f_n = f_{n-1} + f_{n-2}$$

Catalan Numbers. 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440

$$C_n = \sum_{i=1}^n H_{i-1}H_{n-i} = \frac{\binom{2n}{n}}{n+1} = \binom{2n}{n} - \binom{2n}{n-1}$$

Bell or Exponential Numbers. Number of ways to partition a set of n labeled elements.

1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597

Lucas Numbers. Lucas numbers beginning at 2:
 $L(n) = L(n-1) + L(n-2)$, $L(0) = 2$, $L(1) = 1$.

2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349, 15127, 24476, 39603, 64079, 103682, 167761, 271443, 439204

Derangement. Subfactorial or rencontres numbers, or derangements: number of permutations of n elements with no fixed points.

1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961

$$D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$$

Prufer. Number of labeled rooted trees with n nodes: n^{n-1} .

1, 2, 9, 64, 625, 7776, 117649, 2097152, 43046721

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

2.2 Equations

Given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A'_i}{\det A}$$

where A'_i is A with the i 'th column replaced by b .

2.3 Recurrences

If $a_n = c_1a_{n-1} + \cdots + c_ka_{n-k}$, and r_1, \dots, r_k are distinct roots of $x^k - c_1x^{k-1} - \cdots - c_k$, there are d_1, \dots, d_k s.t.

$$a_n = d_1r_1^n + \cdots + d_kr_k^n.$$

Non-distinct roots r become polynomial factors, e.g.
 $a_n = (d_1n + d_2)r^n$.

2.4 Trigonometry

$$\sin(v+w) = \sin v \cos w + \cos v \sin w$$

$$\cos(v+w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$(V+W) \tan(v-w)/2 = (V-W) \tan(v+w)/2$$

where V, W are lengths of sides opposite angles v, w .

$$a \cos x + b \sin x = r \cos(x - \phi)$$

$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}$, $\phi = \text{atan2}(b, a)$.

2.5 Geometry

2.5.1 Triangles

Side lengths: a, b, c

$$\text{Semiperimeter: } p = \frac{a+b+c}{2}$$

$$\text{Area: } A = \sqrt{p(p-a)(p-b)(p-c)}$$

$$\text{Circumradius: } R = \frac{abc}{4A}$$

$$\text{Inradius: } r = \frac{A}{p}$$

Length of median (divides triangle into two equal-area triangles):

$$m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b+c} \right)^2 \right]}$$

$$\text{Law of sines: } \frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$$

$$\text{Law of cosines: } a^2 = b^2 + c^2 - 2bc \cos \alpha$$

$$\text{Law of tangents: } \frac{a+b}{a-b} = \frac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$$

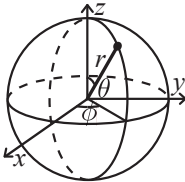
2.5.2 Quadrilaterals

With side lengths a, b, c, d , diagonals e, f , diagonals angle θ , area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.5.3 Spherical coordinates



$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \text{acos}(z/\sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \text{atan2}(y, x) \end{aligned}$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n + 1)(n + 1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n + 1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n + 1)(2n + 1)(3n^2 + 3n - 1)}{30}$$

2.7 Series

$$\sqrt{1 + x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \cdots, \quad (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \quad (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, \quad (-\infty < x < \infty)$$

2.7.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\text{Bin}(n, p)$, $n = 1, 2, \dots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1 - p)^{n - k}$$

$$\mu = np, \sigma^2 = np(1 - p)$$

$\text{Bin}(n, p)$ is approximately $\text{Po}(np)$ for small p .

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\text{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1 - p)^{k - 1}, \quad k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1 - p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\text{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, \quad k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

2.7.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\text{U}(a, b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b - a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a + b}{2}, \sigma^2 = \frac{(b - a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\text{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x - \mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

Data structures (3)

Mo.h

Description: Mo's Algorithm.	300136, 11 lines
1. put queries into sqrt(n) blocks by their left end in O(q log q) / O(q) time.	
2. for every block:	
a. sort all query by right end	
b. brute force from the first to last.	

The complexity is O((N+Q) sqrt(N) * F), where O(F) is the complexity of add **and** remove function.

Complexity Analysis when push is **not** O(1):
Left Push be done at most sqrt(n) *n times, worst **case** can be in n times per block;
Right Push be done sqrt(n) * n, but worst can only sqrt(n) per block, visiting everything sqrt(n) times.
if the right push is amortized O(1), it is also fine.(see Segment problems Query number of segments fully intersect with given segment)

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null.type.
Time: O(log N)

	ec84da, 19 lines
#include <bits/extc++.h> #include <ext/pb_ds/detail/standard_policies.hpp> #include <ext/pb_ds/assoc_container.hpp> #include <ext/pb_ds/tree_policy.hpp> using namespace __gnu_pbds; template<class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>; void example() { Tree<int> t, t2; t.insert(8); auto it = t.insert(10).first; assert(it == t.lower_bound(9)); assert(t.order_of_key(10) == 1); assert(t.order_of_key(11) == 2); assert(*t.find_by_order(0) == 8); t.join(t2); // assuming T < T2 or T > T2, merge t2 into t }	

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

	d77092, 7 lines
#include <bits/extc++.h> // To use most bits rather than just the lowest ones: struct chash { // large odd number for C const uint64_t C = 1l(4e18 * acos(0)) 71; ll operator ()(ll x) const { return __builtin_bswap64(x*C); } }; __gnu_pbds::gp_hash_table<ll, int , chash> h({}, {}, {}, {}, {1<16});	

zkw.h

Description: zkw	bb6869, 23 lines
struct ZKWTree { int no[maxn << 2]; int t; void build_tree(int n) { for (t = 1; t <= n; t <= 1); for (int i = t + 1; i <= t + n; ++i) no[i] = aa[i - t]; for (int i = t; i; --i) push_up(i); } inline void add(int pla, int x) { for (pla += t; pla; pla >>= 1) no[pla] += x; } inline int query(int l, int r) { int ans = 0; for (l += t - 1, r += t + 1; l ^ r ^ 1; l >>= 1, r >>= 1) { if (~l & 1) ans += no[l ^ 1]; if (r & 1) ans += no[r ^ 1]; } return ans; } };	

SegmentTree.h

Description: see top for advice	e20f0b, 48 lines
struct SegmentTree { struct segt{	

```

    ll l, r;
    ll ans;
};
vector<segt> T;

SegmentTree(ll n) { // don't forget to call build.
    T.resize(n << 2);
}

segt combine(segt l, segt r) {
    segt res; res.l = l.l, res.r = r.r;
    res.ans = max(l.ans, r.ans); // TODO
    return res;
}

void build(ll l, ll r, ll o = 1) {
    T[o].l = l, T[o].r = r;
    if (l == r) T[o].ans = a[l]; // TODO
    else {
        ll m = (l + r) >> 1;
        build(l, m, o << 1), build(m+1, r, o << 1 | 1);
        T[o] = combine(T[o << 1], T[o << 1 | 1]);
    }
}

segt query(ll l, ll r, ll o = 1) {
    assert(l <= r);
    if (T[o].l == l && T[o].r == r) return T[o];
    else {
        ll m = (T[o].l + T[o].r) >> 1;
        if (r <= m) return query(l, r, o << 1);
        else if (l >= m+1) return query(l, r, o << 1 | 1);
        return combine(query(l, m, o << 1), query(m+1, r,
            o << 1 | 1));
    }
}

void update(ll tx, ll nx, ll o = 1) { // singly update
    if (T[o].l == T[o].r) // TODO
    else {
        ll m = (T[o].l + T[o].r) >> 1;
        if (tx <= m) update(tx, nx, o << 1);
        else update(tx, nx, o << 1 | 1);
        T[o] = combine(T[o << 1], T[o << 1 | 1]);
    }
}

}

}

```

SegmentTreeTricks.h

Description: Dynamic Segtree Ig

adafe0, 32 lines

```

struct segt {
    ll l, r;
    ll ans;
    ll lp, rp; // physical, for dynamic
} T[sz << 5];
ll nxtFreeNode = 2;

void expand(ll o) { // guarantee empty subchild exist for o. Do
    not do anything on ans; rely on zero-init for correctness
    if (T[o].l == T[o].r) return; // no need expansion
    else if (T[o].lp != 0) return; // already expanded
    else {
        ll m = T[o].l + T[o].r >> 1;

        T[o].lp = nxtFreeNode;
        T[nxtFreeNode].l = T[o].l, T[nxtFreeNode].r = m;
        nxtFreeNode++;
    }
}

```

```

    T[o].rp = nxtFreeNode;
    T[nxtFreeNode].l = m + 1, T[nxtFreeNode].r = T[o].r;
    nxtFreeNode++;
}
}

void update(ll o, ll pos, ll x) {
    expand(o);
    if (T[o].l == T[o].r) T[o].ans += x;
    else {
        ll m = T[o].l + T[o].r >> 1;
        if (pos <= m) update(T[o].lp, pos, x);
        else update(T[o].rp, pos, x);
    }
}
}

```

SegTreeLazy.h

Description: lazy segtree

fc8352, 91 lines

```

template <class S, S (*op)(S, S), S (*e)(), class F, S (*
    mapping)(F, S),
    F (*composition)(F, F), F (*id)()>
struct lazy_segtree {
public:
    lazy_segtree() : lazy_segtree(0){};
    explicit lazy_segtree(int n) : lazy_segtree(vector<S>(n, e())
        ) {};
    explicit lazy_segtree(const vector<S> &v) : _n(int(v.size()))
        {
            log = ceil_pow2(_n), size = 1 << log, d = vector<S>(2 *
                size, e()),
            lz = vector<F>(size, id());
            for (int i = 0; i < _n; i++) d[size + i] = v[i];
            for (int i = size - 1; i >= 1; i--) update(i);
        }
    void set(int p, S x) {
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        d[p] = x;
        for (int i = 1; i <= log; i++) update(p >> i);
    }
    S get(int p) {
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        return d[p];
    }
    S prod(int l, int r) {
        if (l == r) return e();
        l += size, r += size;
        for (int i = log; i >= 1; i--) {
            if (((l >> i) << i) != 1) push(l >> i);
            if (((r >> i) << i) != r) push((r - 1) >> i);
        }
        S sml = e(), smr = e();
        while (l < r) {
            if (l & 1) sml = op(sml, d[l++]);
            if (r & 1) smr = op(d[--r], smr);
            l >>= 1, r >>= 1;
        }
        return op(sml, smr);
    }
    void apply(int p, F f) {
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        d[p] = mapping(f, d[p]);
        for (int i = 1; i <= log; i++) update(p >> i);
    }
    void apply(int l, int r, F f) {
        if (l == r) return;
    }
}

```

```

    l += size, r += size;
    for (int i = log; i >= 1; i--) {
        if (((l >> i) << i) != 1) push(l >> i);
        if (((r >> i) << i) != r) push((r - 1) >> i);
    }
    int l2 = l, r2 = r;
    while (l < r) {
        if (l & 1) all_apply(l++, f);
        if (r & 1) all_apply(--r, f);
        l >>= 1, r >>= 1;
    }
    l = l2, r = r2;
    for (int i = 1; i <= log; i++) {
        if (((l >> i) << i) != 1) update(l >> i);
        if (((r >> i) << i) != r) update((r - 1) >> i);
    }
}
private:
    int _n{}, size{}, log{};
    std::vector<S> d;
    std::vector<F> lz;
    void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
    void all_apply(int k, F f) {
        d[k] = mapping(f, d[k]);
        if (k < size) lz[k] = composition(f, lz[k]);
    }
    void push(int k) {
        all_apply(2 * k, lz[k]), all_apply(2 * k + 1, lz[k]);
        lz[k] = id();
    }
    int ceil_pow2(int n) {
        int x = 0;
        while ((1U << x) < (unsigned int)(n)) x++;
        return x;
    }
};
struct node {
    long long sum, len;
};
node op(node l, node r) { return node{l.sum + r.sum, l.len + r.
    len}; }
node e() { return node{0, 0}; }
using F = long long;
node mapping(F f, node x) { return node{x.sum + f * x.len, x.
    len}; }
F composition(F f, F g) { return F{f + g}; } // f(g())
F id() { return F{0}; }
}

```

2DPrefixSum.h

Description: one indexed. for zero idx write special case for init(). Tested in NERC 2021 J.

bd480e, 12 lines

```

struct prefix2D {
    const static ll M = 200, N = 200; // TODO;
    ll a[M + 2][N + 2], pf[M + 2][N + 2];
    void init() {
        FOR(i, 1, M) FOR(j, 1, N) pf[i][j] = pf[i - 1][j] + pf[i][j
            - 1] - pf[i - 1][j - 1] + a[i][j];
    }
    ll sum(ll u, ll v, ll w, ll h) {
        // u, v must at topleft, w, h must at bottomright
        if (u > w || v > h) return 0;
        return pf[w][h] - pf[w][v - 1] - pf[u - 1][h] + pf[u - 1][v
            - 1];
    }
} p2d;

```

Matrix.h

Description: Basic operations on square matrices.

Usage: Matrix<int, 3> A;

A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};

vector<int> vec = {1,2,3};

vec = (A^N) * vec;

c43c7d, 26 lines

```
template<class T, int N> struct Matrix {
    typedef Matrix M;
    array<array<T, N>, N> d{};
    M operator*(const M& m) const {
        M a;
        rep(i,0,N) rep(j,0,N)
            rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
        return a;
    }
    vector<T> operator*(const vector<T>& vec) const {
        vector<T> ret(N);
        rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
        return ret;
    }
    M operator^(ll p) const {
        assert(p >= 0);
        M a, b(*this);
        rep(i,0,N) a.d[i][i] = 1;
        while (p) {
            if (p&1) a = a*b;
            b = b*b;
            p >>= 1;
        }
        return a;
    }
};
```

SparseTable.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.

Usage: RMQ rmq(values);

rmq.query(inclusive, exclusive);

Time: $\mathcal{O}(|V|\log|V| + Q)$

510c32, 16 lines

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 + 1);
            rep(j,0,sz(jmp[k]))
                jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
        }
    }
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
};
```

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().

Usage: int t = uf.time(); ...; uf.rollback(t);

Time: $\mathcal{O}(\log(N))$

de4ad0, 21 lines

```
struct RollbackUF {
    vi e; vector<pii> st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return sz(st); }
```

```
void rollback(int t) {
    for (int i = time(); i --> t;)
        e[st[i].first] = st[i].second;
    st.resize(t);
}
bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
}
};
```

Treap.h

Description: Treap

909c6b, 126 lines

```
class Treap {
private:
    static const int inf = 0x3f3f3f3f;

    struct Node {
        int x, siz;
        unsigned rd;
        Node *s[2];
        void maintain() {
            siz = 1;
            if (s[0] != nullptr) siz += s[0]->siz;
            if (s[1] != nullptr) siz += s[1]->siz;
        }
        Node(int x) {
            this->x = x;
            siz = 1;
            s[0] = s[1] = nullptr;
            rd = rnd();
        }
    };

    Node *root;

    int getsiz(Node *x) { return x == nullptr ? 0 : x->siz; }

    void split(Node *now, int x, Node *&p1, Node *&p2) {
        if (now == nullptr) {
            p1 = p2 = nullptr;
            return;
        }
        if (now->x <= x) {
            p1 = now;
            split(now->s[1], x, now->s[1], p2);
        } else {
            p2 = now;
            split(now->s[0], x, p1, now->s[0]);
        }
        if (now != nullptr) now->maintain();
    }

    Node *merge(Node *p1, Node *p2) {
        if (p1 == nullptr) return p2;
        if (p2 == nullptr) return p1;
        if (p1->rd > p2->rd) {
            p1->s[1] = merge(p1->s[1], p2);
            p1->maintain();
            return p1;
        } else {
            p2->s[0] = merge(p1, p2->s[0]);
            p2->maintain();
            return p2;
        }
    }
};
```

```
        return p2;
    }
}

public:
    Treap() { root = nullptr; }

    void insert(int x) {
        Node *ll, *rr;
        split(root, x, ll, rr);
        root = merge(merge(ll, new Node(x)), rr);
    }

    void erase(int x) {
        Node *ll, *midd, *rr;
        split(root, x, ll, rr);
        split(ll, x - 1, ll, midd);
        Node *mid = nullptr;
        if (midd != nullptr) {
            mid = merge(midd->s[0], midd->s[1]);
            delete midd;
        }
        root = merge(merge(ll, mid), rr);
    }

    int kth(int k) {
        Node *now = root;
        while (true) {
            int tmp = getsiz(now->s[0]) + 1;
            if (tmp == k) return now->x;
            if (tmp < k) k -= tmp, now = now->s[1];
            else now = now->s[0];
        }
    }

    int rank(int x) {
        int ans = 0;
        int tmp = 0;
        Node *now = root;
        while (now != nullptr) {
            if (now->x == x)
                ans = tmp + getsiz(now->s[0]) + 1;
            if (x <= now->x) now = now->s[0];
            else {
                tmp += getsiz(now->s[0]) + 1;
                now = now->s[1];
            }
        }
        return ans;
    }

    int pred(int x) {
        int ans = -inf;
        Node *now = root;
        while (now != nullptr) {
            if (now->x < x) {
                ans = max(ans, now->x);
                now = now->s[1];
            } else {
                now = now->s[0];
            }
        }
        return ans;
    }

    int succ(int x) {
        int ans = inf;
        Node *now = root;
        while (now != nullptr) {
            if (now->x > x) {
                ans = min(ans, now->x);
                now = now->s[0];
            } else {
                now = now->s[1];
            }
        }
        return ans;
    }
};
```

```

        if (now->x > x) ans = min(ans, now->x);
        if (x < now->x) now = now->s[0];
        else now = now->s[1];
    }
    return ans;
}
};

```

LineContainer.h

Description: This queries maximum. For query minimum, reverse sign of all input line, as well as output. [Jakarta22]

8ec1c7, 31 lines

```

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};

```

LiChao.h

Description: Li chao segment tree. not rigorously tested. Only works in 64-bit as max. use min by default

bb7056, 49 lines

```

template <class Tp>
struct LiChaoSegtree {
    using pll = pair<Tp, Tp>;
    Tp f(pll& l, ll x) { // kx+b
        return l.first * x + l.second;
    }

    ll L, R; // support [L, R]
    vector<pll> T;

    LiChaoSegtree(ll l, ll r): L(l), R(r) { // [L, R] !!!!
        T.assign((r - l + 1) < 2, {0, 1e16}); // TODO change
        value; you can also abstract it out if have
        multiple tests
    }

    void insert(pll line, ll o = 1, ll l = LLONG_MAX, ll r =
        LLONG_MAX) {
        if (l == LLONG_MAX) l = L, r = R;
        ll m = (l + r) / 2;

```

```

        bool left = f(line, l) < f(T[o], l);
        bool mid = f(line, m) < f(T[o], m);

        if (mid) swap(T[o], line); // best in center is kept.
        // TODO think about it
        if (l == r - 1) return; // single leaf
        else if (left != mid) insert(line, o << 1, l, m); //
            since strongness switched, intersect at left
        else insert(line, o << 1 | 1, m, r);
    }

    Tp query(ll x, ll o = 1, ll l = LLONG_MAX, ll r = LLONG_MAX)
    {
        if (l == LLONG_MAX) l = L, r = R;
        ll m = (l + r) / 2;
        ll tval = f(T[o], x);
        if (l == r - 1) return tval;
        else if (x < m) return min(tval, query(x, o << 1, l, m));
        else return min(tval, query(x, o << 1 | 1, m, r));
    }
};

void solve() {
    LiChaoSegtree<ll> st(-1e5 - 1, 1e5 + 1);
    while (true) {
        ll mode; cin >> mode;
        if (mode == 1) {
            ll u, v; cin >> u >> v;
            st.insert({u, v});
        } else {
            ll x; cin >> x;
            cout << st.query(x) << endl;
        }
    }
}

```

Geometry (4)

4.1 Geometric primitives

Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```

template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P unit() const { return *this/dist(); } // makes dist()==1
    P perp() const { return P(-y, x); } // rotates +90 degrees
    P normal() const { return perp().unit(); }
    // returns point rotated 'a' radians ccw around the origin

```

```

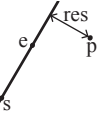
    P rotate(double a) const {
        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << "," << p.y << ")"; }
};

```

lineDistance.h

Description:

Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.



"Point.h"

f6bf6b, 4 lines

```

template<class P>
double lineDist(const P& a, const P& b, const P& p) {
    return (double)(b-a).cross(p-a)/(b-a).dist();
}

```

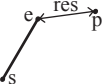
SegmentDistance.h

Description:

Returns the shortest distance between point p and the line segment from point s to e.

Usage: Point<double> a, b(2,2), p(1,1);

bool onSegment = segDist(a,b,p) < 1e-10;



"Point.h"

5c88f4, 6 lines

```

typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
    return ((p-s)*d-(e-s)*t).dist()/d;
}

```

SegmentIntersection.h

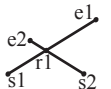
Description:

If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

Usage: vector<P> inter = segInter(s1,e1,s2,e2);

if (sz(inter)==1)

cout << "segments intersect at " << inter[0] << endl;



"Point.h", "OnSegment.h"

9d57f2, 13 lines

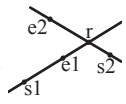
```

template<class P> vector<P> segInter(P a, P b, P c, P d) {
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b * oa) / (ob - oa)};
    set<P> s;
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d);
    return {all(s)};
}

```

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.
Usage: auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;

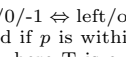


"Point.h"

a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // if parallel
        return {-(s1.cross(e1, s2) == 0), P(0, 0)};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
    return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h
Description: Returns where p is as seen from s towards e . $1/0/-1 \Leftrightarrow$ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.
Usage: bool left = sideOf(p1,p2,q)==1;



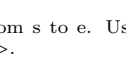
"Point.h"

3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
    auto a = (e-s).cross(p-s);
    double l = (e-s).dist()*eps;
    return (a > l) - (a < -l);
}
```

OnSegment.h
Description: Returns true iff p lies on the line segment from s to e . Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

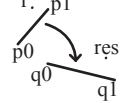


"Point.h"

c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
    return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h
Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

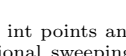


"Point.h"

03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
    P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
    return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

Angle.h
Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.



Usage: vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i

"Point.h"

0f0602, 35 lines

```
struct Angle {
    int x, y;
    int t;
    Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
    Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
    int half() const {
        assert(x || y);
        return y < 0 || (y == 0 && x < 0);
    }
    Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
    Angle t180() const { return {-x, -y, t + half()}; }
    Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
        make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

4.2 Circles

CircleIntersection.h
Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h"

84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
    if (a == b) { assert(r1 != r2); return false; }
    P vec = b - a;
    double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false;
    P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per};
    return true;
}
```

CircleTangents.h
Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h"

b0153d, 13 lines

```
template<class P>
```

```
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
    P d = c2 - c1;
    double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) return {};
    vector<pair<P, P>> out;
    for (double sign : {-1, 1}) {
        P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    if (h2 == 0) out.pop_back();
    return out;
}
```

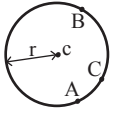
CirclePolygonIntersection.h
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../content/geometry/Point.h"

alee63, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        P u = p + d * s, v = p + d * t;
        return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
    };
    auto sum = 0.0;
    rep(i,0,sz(ps))
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

circumcircle.h
Description:
The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



"Point.h"

1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist()/
        abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h
Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h"

09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0;
        rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;
        }
    }
```



```

    r = (o - ps[i]).dist();
    rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
    }
}
}
return {o, r};
}

```

4.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};

bool in = inPolygon(v, P{3, 3}, false);

Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
    int cnt = 0, n = sz(p);
    rep(i,0,n) {
        P q = p[(i + 1) % n];
        if (onSegment(p[i], q, a)) return !strict;
        //or: if (segDist(p[i], q, a) <= eps) return !strict;
        cnt ^= ((a.y < p[i].y) - (a.y < q.y)) * a.cross(p[i], q) > 0;
    }
    return cnt;
}

```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

```

template<class T>
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
}

```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```

typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    }
    return res / A / 3;
}

```

PolygonCut.h

Description:

Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

Usage: vector<P> p = ...;

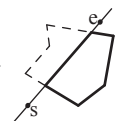
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```

typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
    vector<P> res;

```



```

rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
        res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
        res.push_back(cur);
}
return res;
}

```

mygeo.h

Description: Caution: cannot use x, y, use cout for debug, beware of the expected bugs: norm(); complex<ll>; always use ll / ld for T not integers. Before Use: modify EPS

Notice: angle is in $[-\pi, \pi]$, remainder(x, 2 M_PI) for remainder in <cmath> to $[-\pi, \pi]$

Conversion to other types: <ll> works fine against local and CF gcc; tested in nwcrc21f

Input: pt(a, b); cin >> pt only works for a+bi afaik

cannot use map unordered_map consider convert to pair for that

Calculate precision and adjust EPS accordingly; sometimes this is nontrivial; there is no unsgned long double, so if EPs is very small, consider use integer arithmetic, or change the variable you are storing.

7e9472, 92 lines

```

namespace Geometry {
    using T = ld;
    using pt = complex<T>;
    #define x real()
    #define y imag()

```

// basic operations

```

#define dot(a, b) (conj(a) * (b)).x
#define cross(a, b) (conj(a) * (b)).y
#define dist_2(a, b) norm((a) - (b)) // beware precision
    loss bug in C++
#define dist(a, b) abs((a) - (b))
#define sign_area(a, b, c) (cross((b) - (a), (c) - (b))) //
    negative if c is rhs of AB (clockwise), zero if
    collinear; do not use thumb rule
#define a_norm(a) (remainder(a, 2.0 * M_PI))

```

// extra functions

```

#define slope(a, b) arg((b) - (a))
#define project_vec(p, a) ((v) * dot(p, v) / norm(v))
#define project(p, a, b) ((a) + ((b) - (a)) * dot((p) - (a),
    (b) - (a)) / norm((b) - (a)))
#define reflect(p, a, b) ((a) + conj(((p) - (a)) / ((b) - (a))) * ((b) - (a)))
#define rotate(x, base, theta) (((x) - (base)) * polar(1.0,
    theta) + (base))
#define angDiff(a, b, base) (a_norm(arg(a - base) - arg(b -
    base))) // positive angle yield by letting a "counter
    clockwise" from b. i.e. a is at left if y > 0. think
    a bout it.

```

// floating point

#define EPS 1e-12

#define EQ(a, b) (fabs((a) - (b)) < EPS)

```

bool clockwise(pt a, pt b, pt c, bool include_collinear) {
    T o = sign_area(a, b, c);
    return ((EQ(o, 0) && include_collinear) || (!EQ(o, 0)
        && o < 0));
}

```

// noncolinear tested on Kattis, Codeforces and Luogu;

Luogu C++17 O2 wa/re for some reason.

// colinear not rig. tested

```

vector<pt> convexHull(vector<pt>& v, bool include_collinear
    = false) {
    assert(!v.empty());
    pt p0 = *min_element(v.begin(), v.end(), [](pt a, pt b)
        {
            return make_pair(a.y, a.x) < make_pair(b.y, b.x);
        });
    sort(v.begin(), v.end(), [&p0](const pt& a, const pt& b)
        ) {
        T o = sign_area(p0, a, b);
        if (a == b) return false;
        if (!EQ(o, 0)) return o < 0; // clockwise
        else return dist(p0, a) < dist(p0, b);
    });
    if (include_collinear) {
        ll i = (ll)v.size() - 1;
        while (i >= 0 && EQ(sign_area(p0, v[i], v.back()),
            0)) i--;
        reverse(v.begin() + i + 1, v.end());
    }
}

```

```

vector<pt> ret;
ll n = v.size();
FOR(i, 0, n - 1) {
    while (ret.size() >= 2 && !clockwise(ret[ret.size()
        - 2], ret[ret.size() - 1], v[i],
            include_collinear)) ret.pop_back();
    ret.push_back(v[i]);
}
return ret;
}

```

// tested in Kattis

```

T circum(vector<pt>& v) {
    T ans = 0;
    for (int i = 0; i <= (signed)v.size() - 2; ++i) ans +=
        dist(v[i], v[i + 1]);
    ans += dist(v[0], v.back());
    return ans;
}

```

```

vector<pt> minkowski(vector<pt>& v1, vector<pt>& v2) {
    assert(v1.size() >= 2 && v2.size() >= 2);
    auto reorderPolygon = [](vector<pt>& P) {
        ll pos = 0;
        for (ll i = 1; i < P.size(); ++i) if (P[i].y <= P[
            pos].y || (P[i].y == P[pos].y && P[i].x < P[
                pos].x)) pos = i;
        rotate(P.begin(), P.begin() + pos, P.end());
    };
    reorderPolygon(v1), reorderPolygon(v2);
}

```

// cyclic

```

v1.push_back(v1[0]), v1.push_back(v1[1]);
v2.push_back(v2[0]), v2.push_back(v2[1]);

```

```

vector<pt> ans;
ll i = 0, j = 0;
while (i < v1.size() - 2 || j <= v2.size() - 2) {
    ans.push_back(v1[i] + v2[j]);
    auto cross = cross(v1[i + 1] - v1[i], v2[i + 1] -
        v2[j]);
    if (EQ(cross, 0)) ++i, ++j;
    else if (cross > 0) ++i;
    else ++j;
}
return ans;
}
}

```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h"	c571b8, 12 lines
<pre>typedef Point<ll> P; array<P, 2> hullDiameter(vector<P> S) { int n = sz(S), j = n < 2 ? 0 : 1; pair<ll, array<P, 2>> res({0, {S[0], S[0]}}); rep(i,0,j) for (; j = (j + 1) % n) { res = max(res, {S[i] - S[j]].dist2(), {S[i], S[j]}); if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0) break; } return res.second; }</pre>	

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h"	71446b, 14 lines
<pre>typedef Point<ll> P; bool inHull(const vector<P>& l, P p, bool strict = true) { int a = 1, b = sz(l) - 1, r = !strict; if (sz(l) < 3) return r && onSegment(l[0], l.back(), p); if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b); if (sideOf(l[0], l[a], p) >= r sideOf(l[0], l[b], p)<= -r) return false; while (abs(a - b) > 1) { int c = (a + b) / 2; (sideOf(l[0], l[c], p) > 0 ? b : a) = c; } return sgn(l[a].cross(l[b], p)) < r; }</pre>	

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: $\bullet(-1, -1)$ if no collision, $\bullet(i, -1)$ if touching the corner i , $\bullet(i, i)$ if along side $(i, i + 1)$, $\bullet(i, j)$ if crossing sides $(i, i + 1)$ and $(j, j + 1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i, i + 1)$. The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(\log n)$

"Point.h"	7cf45b, 39 lines
<pre>#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n])) #define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0 template <class P> int extrVertex(vector<P>& poly, P dir) { int n = sz(poly), lo = 0, hi = n; if (extr(0)) return 0; while (lo + 1 < hi) { int m = (lo + hi) / 2; if (extr(m)) return m; int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m); (ls < ms (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m; } return lo; }</pre>	

```
#define cmpl(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
```

```
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpl(endA) < 0 || cmpl(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i,0,2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpl(m) == cmpl(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpl(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpl(res[0]) && !cmpl(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
```

4.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h"	ac41a6, 17 lines
<pre>typedef Point<ll> P; pair<P, P> closest(vector<P> v) { assert(sz(v) > 1); set<P> S; sort(all(v), [](P a, P b) { return a.y < b.y; }); pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}}; int j = 0; for (P p : v) { P d{1 + (ll)sqrt(ret.first), 0}; while (v[j].y <= p.y - d.x) S.erase(v[j++]); auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d); for (; lo != hi; ++lo) ret = min(ret, {(lo - p).dist2(), {lo, p}}); S.insert(p); } return ret.second; }</pre>	

4.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

template<class V, class L>	3058c3, 6 lines
<pre>double signedPolyVolume(const V& p, const L& trilst) { double v = 0; for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]); return v / 6; }</pre>	

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

template<class T> struct Point3D {	8058ae, 32 lines
<pre> typedef Point3D P; typedef const P& R; T x, y, z; explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {} bool operator<(R p) const { return tie(x, y, z) < tie(p.x, p.y, p.z); }</pre>	

```
    bool operator==(R p) const {
        return tie(x, y, z) == tie(p.x, p.y, p.z);
    }
    P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
    P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
    P operator*(T d) const { return P(x*d, y*d, z*d); }
    P operator/(T d) const { return P(x/d, y/d, z/d); }
    T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
    P cross(R p) const {
        return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
    }
    T dist2() const { return x*x + y*y + z*z; }
    double dist() const { return sqrt((double)dist2()); }
    //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
    double phi() const { return atan2(y, x); }
    //Zenith angle (latitude) to the z-axis in interval [0, pi]
    double theta() const { return atan2(sqrt(x*x+y*y),z); }
    P unit() const { return *this/(T)dist(); } //makes dist()==1
    //returns unit vector normal to *this and p
    P normal(P p) const { return cross(p).unit(); }
    //returns point rotated 'angle' radians ccw around axis
    P rotate(double angle, P axis) const {
        double s = sin(angle), c = cos(angle); P u = axis.unit();
        return u.dot(u)*(1-c) + (*this)*c - cross(u)*s;
    }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
Time: $\mathcal{O}(n^2)$

"Point3D.h"	5b45fc, 49 lines
<pre>typedef Point3D<double> P3; struct PR { void ins(int x) { (a == -1 ? a : b) = x; } void rem(int x) { (a == x ? a : b) = -1; } int cnt() { return (a != -1) + (b != -1); } int a, b; }; struct F { P3 q; int a, b, c; };</pre>	

```
vector<F> hull3d(const vector<P3>& A) {
    assert(sz(A) >= 4);
    vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
    #define E(x,y) E[f.x][f.y]
    vector<F> FS;
    auto mf = [&](int i, int j, int k, int l) {
        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i]))
            q = q * -1;
        F f{q, i, j, k};
        E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
        FS.push_back(f);
    };
    rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
        mf(i, j, k, 6 - i - j - k);

    rep(i,4,sz(A)) {
        rep(j,0,sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                E(a,b).rem(f.c);
                E(a,c).rem(f.b);
                E(b,c).rem(f.a);
                swap(FS[j--], FS.back());
                FS.pop_back();
            }
        }
    }
```

```
    }
    int nw = sz(FS);
    rep(j,0,nw) {
        F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
        C(a, b, c); C(a, c, b); C(b, c, a);
    }
}
for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
return FS;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius r between the points with azimuthal angles (longitude) f_1 (ϕ_1) and f_2 (ϕ_2) from x axis and zenith angles (latitude) t_1 (θ_1) and t_2 (θ_2) from z axis ($0 =$ north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. $dx \cdot radius$ is then the difference between the two points in the x direction and $d \cdot radius$ is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
    double dz = cos(t2) - cos(t1);
    double d = sqrt(dx*dx + dy*dy + dz*dz);
    return radius*2*asin(d/2);
}
```

Graph (5)

5.1 Fundamentals

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m , where $m[i][j] = \text{inf}$ if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j , inf if no path, or $-\text{inf}$ if the path goes through a negative-weight cycle.

Time: $O(N^3)$

531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
    int n = sz(m);
    rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
    rep(k,0,n) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) {
            auto newDist = max(m[i][k] + m[k][j], -inf);
            m[i][j] = min(m[i][j], newDist);
        }
    rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

DifferenceConstraints.h

Description: Solve problem for $x_u - x_v \leq y$

27a086, 43 lines

```
template <class length> struct Difference_Constraints {
    int n;
    vector<vector<pair<int, length>>> E;
    vector<length> dis;
    vector<int> cnt, vis;
    Difference_Constraints(int _n)
        : n(_n), E(_n + 1), dis(_n + 1, 1e16), cnt(_n + 1), vis(
            _n + 1){};
};
```

```
void add_condition(int u, int v, length y) {
    //  $x_u - x_v \leq y$ 
    E[v].emplace_back(u, y);
}

auto spfa(int s = 0) {
    queue<int> q;
    q.push(s);
    vis[s] = 1, dis[s] = 0;
    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        vis[u] = 0;
        for (auto [v, w] : E[u])
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                cnt[v] = cnt[u] + 1;
                if (cnt[v] > n)
                    return false;
                if (!vis[v]) {
                    q.push(v);
                    vis[v] = 1;
                }
            }
        }
    return true;
}

auto solve() {
    for (int i = 1; i <= n; ++i)
        E[0].emplace_back(i, 0);
    if (spfa())
        for (int i = 1; i <= n; ++i)
            cout << dis[i] << " ";
    else
        cout << "NO\n";
}

};
```

5.2 Network flow

MinCostMaxFlow.h

Description: Primal Dual
Time: Approximately $O(V \log VF)$

596aa3, 87 lines

```
template <class Flow, class Cost> struct Primal_Dual { // based
    on EK
    struct edge {
        int u, v;
        Flow f;
        Cost c;
        edge(int _u, int _v, Flow _f, Cost _c) : u(_u), v(_v), f(_f), c(_c){};
    };
    int n;
    Cost inf_cost;
    Flow inf_flow;
    vector<vector<int>> E;
    vector<edge> edg;
    vector<pair<int, int>> p;
    vector<Cost> dis, h;
    vector<int> vis;
    Primal_Dual(int _n) : n(_n), E(_n), p(_n), dis(_n), h(_n), vis(_n) {
        inf_flow = 1e16, inf_cost = 1e16;
    };
    void addEdge(int u, int v, Flow f, Cost c) {
        int id = (int)edg.size();
        edg.emplace_back(u, v, f, c);
        edg.emplace_back(v, u, 0, -c);
        E[u].emplace_back(id), E[v].emplace_back(id ^ 1);
    }
};
```

```
bool dijkstra(int s, int t) {
    using pii = pair<Cost, int>;
    priority_queue<pii, vector<pii>, greater<>> Q;
    fill(dis.begin(), dis.end(), inf_cost);
    fill(vis.begin(), vis.end(), 0);
    dis[s] = 0;
    Q.emplace(dis[s], s);
    while (!Q.empty()) {
        int u = Q.top().second;
        Q.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        for (auto id : E[u]) {
            auto [_, v, f, c] = edg[id];
            c = c + h[u] - h[v];
            if (f > 0 && dis[v] > dis[u] + c) {
                dis[v] = dis[u] + c;
                p[v].first = u;
                p[v].second = id;
                if (!vis[v])
                    Q.emplace(dis[v], v);
            }
        }
    }
    return dis[t] != inf_cost;
}

void spfa(int s) {
    queue<int> Q;
    fill(h.begin(), h.end(), inf_cost);
    h[s] = 0, vis[s] = 1;
    Q.emplace(s);
    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        vis[u] = 0;
        for (auto id : E[u]) {
            auto [_, v, f, c] = edg[id];
            if (f > 0 && h[v] > h[u] + c) {
                h[v] = h[u] + c;
                if (!vis[v])
                    vis[v] = 1, Q.push(v);
            }
        }
    }
}

pair<Flow, Cost> solve(int s, int t) {
    spfa(s);
    Flow maxf = 0;
    Cost minc = 0;
    while (dijkstra(s, t)) {
        auto minf = inf_flow;
        for (int i = 0; i < n; ++i)
            h[i] += dis[i];
        for (int i = t; i != s; i = p[i].first)
            minf = min(minf, edg[p[i].second].f);
        for (int i = t; i != s; i = p[i].first)
            edg[p[i].second].f -= minf, edg[p[i].second ^ 1].f += minf;
        maxf += minf, minc += minf * h[t];
    }
    return {maxf, minc};
}

};
```

NegativeLoop.h

Description: Forced Full Flow for negative edges

e47205, 20 lines

```
long long ans = 0;
```

```
vector<long long> d(n);
for (int i = 0; i < m; ++i) {
    if (v >= 0) {
        G.addEdge(x, y, f, v);
    } else {
        G.addEdge(y, x, f, -v);
        ans += f * v, d[x] -= f, d[y] += f;
    }
}
for (int i = 0; i < n; ++i) {
    if (d[i] > 0)
        G.addEdge(ss, i, d[i], 0);
    else if (d[i] < 0)
        G.addEdge(i, tt, -d[i], 0);
}
auto [_, bs] = G.solve(ss, tt); // fake ss and tt
G.n -= 2;
auto [mx, mi] = G.solve(s, t); // real s and t
cout << mx << " " << mi + bs + ans << "\n";
```

Dinic.h
Description: dinic algo
Time: $\mathcal{O}(V^2E)$

928805, 54 lines

```
template <class Flow> struct dinic {
    static constexpr Flow inf = 1e16;
    struct edge {
        int u, v;
        Flow f;
        edge(int _u, int _v, Flow _f) : u(_u), v(_v), f(_f){};
    };
    vector<vector<int>> E;
    vector<edge> edg;
    vector<int> d, cur;
    int n;
    dinic(int _n) : E(_n), d(_n), cur(_n), n(_n) {}
    void addEdge(int u, int v, Flow w) {
        int id = (int)edg.size();
        edg.emplace_back(u, v, w), edg.emplace_back(v, u, 0);
        E[u].emplace_back(id), E[v].emplace_back(id ^ 1);
    }
    auto dfs(int u, int t, Flow flow) {
        if (u == t) return flow;
        Flow sum = 0;
        while (cur[u] < (int)E[u].size()) {
            auto id = E[u][cur[u]];
            auto [_, v, f] = edg[id];
            auto c = min(flow, f);
            if (d[u] + 1 == d[v] && c > 0) {
                auto add = dfs(v, t, c);
                sum += add, flow -= add, edg[id].f -= add, edg[id ^ 1].f += add;
            }
            if (!flow) break;
            cur[u]++;
        }
        if (!sum) d[u] = -1;
        return sum;
    }
    bool level(int s, int t) {
        fill(d.begin(), d.end(), -1), fill(cur.begin(), cur.end(), 0);
        queue<int> Q;
        Q.emplace(s), d[s] = 0;
        while (!Q.empty()) {
            int u = Q.front();
            Q.pop();
            for (auto id : E[u]) {
                auto v = edg[id].v;
```

```
                if (d[v] == -1 && edg[id].f != 0) Q.emplace(v), d[v] = d[u] + 1;
            }
        }
        return d[t] != -1;
    }
    auto solve(int s, int t) {
        Flow ans = 0;
        while (level(s, t)) ans += dfs(s, t, inf);
        return ans;
    }
};
```

BoundedFlow.h
Description: for edge u, v , the flow should satisfy: $b(u, v) \leq f(u, v) \leq c(u, v)$, in which $b(u, v)$ is the lower bound and $c(u, v)$ is the upper bound. For feasible flow without sink and source, if $ans = \sum_{d[i]>0} d[i] = \sum_{d[i]<0} -d[i]$, there exist a feasible flow. The exact flow in each edge is the sum of the flow in the graph and the lower bound the the edge. Otherwise, there's no solution. If we want to calculate maximum flow, we should note that for any feasible flow, running maximum flow algorithm, we can always get the correct answer. The base is the flow of edge t, s . Then, we delete all edges added (including edge t, s) and run maximum flow algorithm on the residual network. The answer is the maximum flow add the base (feasible flow). For Minimum Flow with Sink and Source, we should notice that $MinF(s, t) = -MaxF(t, s)$. Same as Maximum Flow, but the answer is the base (feasible fow) minus the maximum flow of t, s .

7669a6, 17 lines

```
// Feasible Flow without Sink and Source
G.addEdge(u, v, R - L);
d[u] -= L, d[v] += L;
if (d[i] > 0) {
    G.addEdge(SS, i, d[i]);
} else if (d[i] < 0) {
    G.addEdge(i, TT, -d[i]);
}
auto ans = G.solve(SS, TT);
// Feasible Flow with Sink and Source only need to add edge t, s, inf
G.addEdge(t, s, inf)
// Maximum Flow with Sink and Source
auto base = G.edg[id ^ 1].f;
// delete all edges (set f to 0)
auto ans = G.solve(s, t) + base;
// Minimum Flow with Sink and Source
auto ans = base - G.solve(t, s);
```

MinCut.h
Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s , only traversing edges with positive residual capacity. If want to Minimize Cutting Edge under Minimum-Cut first. Then change full edge's capacity into 1, other edge's capacity into inf, run minimum cut again. This time the answer is the minimum cut edge number. If no need under Minimum-Cut, just simply Change all the edge's capacity into 1

481659, 8 lines

```
function<void(int)>> dfs = [&](int u) {
    vis[u] = 1;
    for (auto id : E[u]) {
        auto [_, v, f] = edg[id];
        if (f > 0 && !vis[v]) dfs(v);
    }
};
dfs(S);
```

MaximumWeightClosedSubgraph.h

Description: A directed graph, each node has a value. Choose a subgraph with maixmum sum value, in which each node's conected point is also in the subgraph. Use minimum cut. Connect S with every point with positive value (weight is val_i), and every negative value with T (weight is $-val_i$). All edges in the graph's weight is inf . The subgraph is the point connected with S after minimum cut, and the sum is sum of positive points - minimum cut.

GlobalMinCut.h
Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}(V^3)$

8b0e19, 21 lines

```
pair<int, vi> globalMinCut(vector<vi> mat) {
    pair<int, vi> best = {INT_MAX, {}};
    int n = sz(mat);
    vector<vi> co(n);
    rep(i, 0, n) co[i] = {i};
    rep(ph, 1, n) {
        vi w = mat[0];
        size_t s = 0, t = 0;
        rep(it, 0, n-ph) { //  $\mathcal{O}(V^2) \rightarrow \mathcal{O}(E \log V)$  with prio. queue
            w[t] = INT_MIN;
            s = t, t = max_element(all(w)) - w.begin();
            rep(i, 0, n) w[i] += mat[t][i];
        }
        best = min(best, {w[t] - mat[t][t], co[t]});
        co[s].insert(co[s].end(), all(co[t]));
        rep(i, 0, n) mat[s][i] += mat[t][i];
        rep(i, 0, n) mat[i][s] = mat[s][i];
        mat[0][t] = INT_MIN;
    }
    return best;
}
```

GomoryHu.h
Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 0418b3, 13 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i, 1, N) {
        PushRelabel D(N); // Dinic also works
        for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
        tree.push_back({i, par[i], D.calc(i, par[i])});
        rep(j, i+1, N)
            if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
    }
    return tree;
}
```

5.3 Matching

DFSMatching.h
Description: Maximum Matching
Time: $\mathcal{O}(VE)$

118082, 23 lines

```
bool dfs(int u) {
    for (auto v : E[u]) {
        if (vis[v])
            continue;
        vis[v] = 1;
        if (link[v] == -1 || dfs(link[v])) {
            link[v] = u;
```

```
        return true;
    }
}
return false;
};

void solve() {
    fill(link.begin(), link.end(), -1);
    int cnt = 0;
    for (int i = 0; i < n; ++i) {
        // dfs(i) x times means add x points same as point i in the
        // map
        if (dfs(i)) {
            fill(vis.begin(), vis.end(), 0);
            cnt++;
        }
    }
}
```

MinimumVertexCover.h
Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

5595b4, 18 lines

```
// Do maximum matching
for (auto i : lef) vis[i] = match[i] == -1;
stack<int> Q;
for (auto i : lef)
    if (vis[i])
        Q.emplace(i);
while (!Q.empty()) {
    auto u = Q.top();
    Q.pop();
    vis[u] = 1;
    for (auto v : E[u])
        if (match[v] != -1 && vis[v] == 0)
            vis[v] = 1, Q.emplace(match[v]);
}
for (auto i : lef)
    if (!vis[i]) ans.emplace_back(i);
for (auto i : rig)
    if (vis[i]) ans.emplace_back(i);
```

WeightedMatching.h
Description: Key of Hungarain is to find lx and ly which satisfy $lx_i + ly_j \geq w_{i,j}$ for all edge. For the subgraph with all the edges satisfy $lx_i + ly_i = w_{i,j}$ exists perfect matching, it is the maximum weight perfect matching. (As for all other perfect matching, $w_{i,j} \leq lx_i + ly_j$, the sum is no larger than this). Maximum possible sum of potentials is equal to the optimal solution of the assignment problem.
Thus, to find two array p and q satisfying $p_i + q_j \leq w_{i,j}$, we want to maximize $\sum p + \sum q$, one way is to add edge i, j with $-w_{i,j}$. And the answer is $p = -lx, q = -ly$.
Time: $\mathcal{O}(N^2M)$

d335ec, 92 lines

```
template <class Cap> struct Hungarain {
    int n;
    vector<int> match_x, match_y, pre;
    vector<bool> vis_x, vis_y;
    vector<vector<Cap>> g;
    vector<Cap> slack;
    vector<Cap> lx, ly;
    Cap inf, res;
    queue<int> q;
    int org_n, org_m;
    Hungarain(int _n, int _m) {
        org_n = _n, org_m = _m;
        n = max(_n, _m);
        inf = 1e16, res = 0;
        // if negative, change here to -inf
    }
```

```
    g = vector<vector<Cap>>(n, vector<Cap>(n));
    match_x = match_y = vector<int>(n, -1);
    pre = vector<int>(n);
    vis_x = vis_y = vector<bool>(n);
    lx = vector<Cap>(n, -inf), ly = vector<Cap>(n), slack =
        vector<Cap>(n);
};

void addEdge(int u, int v, Cap w) { g[u][v] = w; }
bool check(int v) {
    vis_y[v] = true;
    if (match_y[v] != -1) {
        q.push(match_y[v]);
        vis_x[match_y[v]] = true;
        return false;
    }
    while (v != -1) {
        match_y[v] = pre[v];
        swap(v, match_x[pre[v]]);
    }
    return true;
}

void bfs(int i) {
    while (!q.empty()) q.pop();
    q.push(i);
    vis_x[i] = true;
    while (true) {
        while (!q.empty()) {
            auto u = q.front();
            q.pop();
            for (int v = 0; v < n; ++v)
                if (!vis_y[v]) {
                    auto delta = lx[u] + ly[v] - g[u][v];
                    if (slack[v] >= delta) {
                        pre[v] = u;
                        if (delta)
                            slack[v] = delta;
                        else if (check(v))
                            return;
                    }
                }
            auto a = inf;
            for (int j = 0; j < n; ++j) {
                if (!vis_y[j]) a = min(a, slack[j]);
            }
            for (int j = 0; j < n; ++j) {
                if (vis_x[j]) lx[j] -= a;
                if (vis_y[j])
                    ly[j] += a;
                else
                    slack[j] -= a;
            }
            for (int j = 0; j < n; ++j)
                if (!vis_y[j] && slack[j] == 0 && check(j)) return;
        }
    }
}

void solve() {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) lx[i] = max(lx[i], g[i][j]);
    }
    for (int i = 0; i < n; ++i) {
        fill(slack.begin(), slack.end(), inf);
        fill(vis_x.begin(), vis_x.end(), false);
        fill(vis_y.begin(), vis_y.end(), false);
        bfs(i);
    }
    // output
    for (int i = 0; i < n; ++i) {
        if (g[i][match_x[i]] > 0) {
```

```
            // have edge, if negative change >0 to >-inf
            res += g[i][match_x[i]];
        } else {
            // no match
            match_x[i] = -1;
        }
    }
}
};
```

GeneralMatching.h
Description: Blossom tree

282286, 92 lines

```
struct blossom {
    int n, vis_t;
    vector<vector<int>> E;
    vector<int> match, label, org, vis, parent;
    queue<int> Q;
    blossom(int _n)
        : n(_n), E(_n), match(_n, -1), label(_n), org(_n), vis(_n),
        parent(_n, -1) {
        iota(org.begin(), org.end(), 0);
        vis_t = 0;
    }
    void addEdge(int u, int v) {
        E[u].emplace_back(v);
        E[v].emplace_back(u);
    }
    auto lca(int v, int u) {
        vis_t++;
        while (true) {
            if (v != -1) {
                if (vis[v] == vis_t) {
                    return v;
                }
                vis[v] = vis_t;
                if (match[v] == -1) {
                    v = -1;
                } else {
                    v = org[parent[match[v]]];
                }
            }
            swap(v, u);
        }
    }
    void agument(int v) {
        while (v != -1) {
            auto pv = parent[v];
            auto nxt = match[pv];
            match[v] = pv;
            match[pv] = v;
            v = nxt;
        }
    }
    void flower(int v, int u, int a) {
        while (org[v] != a) {
            parent[v] = u;
            u = match[v];
            if (label[u] == 1) {
                label[u] = 0;
                Q.emplace(u);
            }
            org[v] = org[u] = a;
            v = parent[u];
        }
    }
    auto bfs(int root) {
        fill(label.begin(), label.end(), -1);
```

```

iota(org.begin(), org.end(), 0);
while (!Q.empty()) {
    Q.pop();
}
Q.emplace(root);
label[root] = 0;
while (!Q.empty()) {
    auto u = Q.front();
    Q.pop();
    for (auto v : E[u]) {
        if (label[v] == -1) {
            label[v] = 1;
            parent[v] = u;
            if (match[v] == -1) {
                agument(v);
                return true;
            }
            label[match[v]] = 0;
            Q.push(match[v]);
            continue;
        } else if (label[v] == 0 && org[v] != org[u]) {
            auto a = lca(org[u], org[v]);
            flower(v, u, a);
            flower(u, v, a);
        }
    }
}
return false;
}
void solve() {
    for (int i = 0; i < n; ++i) {
        if (match[i] == -1) { // if match[i] == -1, no matching
            bfs(i);
        }
    }
}
};

```

5.4 DFS algorithms

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.

Time: $O(V + E)$

780b64, 15 lines

```

vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        if (it == end){ ret.push_back(x); s.pop_back(); continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.push_back(y);
        }
    }
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
    return {ret.rbegin(), ret.rend()};
}

```

SCC.h

Description: Tarjan SCC

c684c4b, 43 lines

```

struct Tarjan_SCC {
    int n, edg_id, _t, sc_id;
    vector<vector<int>> E, scc;

```

```

    vector<pair<int, int>> edges;
    vector<int> dfn, low, in_scc;
    stack<int> st;
    vector<bool> in_st;
    Tarjan_SCC(int _n)
        : n(_n), E(_n), dfn(_n, -1), low(_n), in_scc(_n, -1),
          in_st(_n, false) {
        sc_id = edg_id = _t = 0;
    }
    void addEdge(int u, int v) {
        edges.emplace_back(u, v), E[u].emplace_back(edg_id++);
    }
    void dfs(int u) {
        low[u] = dfn[u] = _t++;
        st.emplace(u), in_st[u] = true;
        for (auto id : E[u]) {
            auto v = edges[id].second;
            if (dfn[v] == -1)
                dfs(v), low[u] = min(low[u], low[v]);
            else if (in_st[v])
                low[u] = min(low[u], dfn[v]);
        }
        if (dfn[u] == low[u]) {
            vector<int> _scc;
            _scc.clear();
            while (st.top() != u) {
                auto v = st.top();
                in_scc[v] = sc_id, _scc.emplace_back(v), in_st[v] = false;
                st.pop();
            }
            in_scc[u] = sc_id, _scc.emplace_back(u);
            scc.emplace_back(_scc);
            sc_id++;
            st.pop(), in_st[u] = false;
        }
    }
    void solve() {
        for (int i = 0; i < n; ++i)
            if (dfn[i] == -1) dfs(i);
    }
};

```

BCC.h

Description: Tarjan BCC

7f3f57, 94 lines

```

struct Tarjan_BCC {
    int n, _t, edg_id;
    vector<vector<int>> E, child;
    vector<pair<int, int>> edges;
    vector<int> dfn, low, fa;
    vector<bool> cut;
    Tarjan_BCC(int _n)
        : n(_n), E(n), child(n), dfn(n, -1), low(n), fa(n, -1),
          cut(n, false) {
        edg_id = _t = 0;
    }
    void addEdge(int u, int v) {
        edges.emplace_back(u, v), E[u].emplace_back(edg_id),
            E[v].emplace_back(edg_id);
        edg_id++;
    }
    void dfs(int u, int fa_id) {
        low[u] = dfn[u] = _t++;
        for (auto id : E[u]) {
            auto v = edges[id].first ^ edges[id].second ^ u;
            if (id == fa_id) continue;
            if (dfn[v] != -1) {
                low[u] = min(low[u], dfn[v]);

```

```

            } else {
                fa[v] = u;
                dfs(v, id);
                child[u].emplace_back(v), low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) cut[u] = true;
            }
        }
        if (fa_id == -1) cut[u] = child[u].size() > 1;
    }
    void solve() {
        for (int i = 0; i < n; ++i) {
            if (dfn[i] == -1) dfs(i, -1);
        }
    }
    bool is_cut(int u) { return cut[u]; }
    vector<vector<int>> BCC_point() {
        vector<vector<int>> BCC;
        vector<int> tmp;
        vector<bool> vis(n, false);
        stack<int> st;
        function<void(int)> dfs = [&](int u) {
            st.emplace(u), vis[u] = true;
            for (auto v : child[u]) {
                dfs(v);
                if (low[v] >= dfn[u]) {
                    tmp.clear();
                    while (!st.empty() && st.top() != v) {
                        tmp.emplace_back(st.top());
                        st.pop();
                    }
                    tmp.emplace_back(st.top());
                    st.pop(), tmp.emplace_back(u);
                    BCC.emplace_back(tmp);
                }
            }
        };
        for (int i = 0; i < n; ++i) {
            if (!vis[i]) {
                if (child[i].empty()) BCC.emplace_back(vector{i});
                dfs(i);
            }
        }
        return BCC;
    }
    bool is_bridge(int id) {
        return is_bridge(edges[id].first, edges[id].second);
    }
    bool is_bridge(int u, int v) { // assert there's an edge u-v
        if (dfn[u] > dfn[v]) swap(u, v);
        return low[v] > dfn[u];
    }
    vector<vector<int>> BCC_edge() {
        vector<vector<int>> BCC;
        vector<int> tmp;
        vector<bool> vis(n, false);
        function<void(int)> dfs = [&](int u) {
            vis[u] = true, tmp.emplace_back(u);
            for (auto id : E[u]) {
                auto v = edges[id].first ^ edges[id].second ^ u;
                if (!is_bridge(u, v) && !vis[v]) dfs(v);
            }
        };
        for (int i = 0; i < n; ++i) {
            if (!vis[i]) {
                tmp.clear();
                dfs(i);
                BCC.emplace_back(tmp);
            }
        }
    }
};

```

```
    return BCC;
}
};
```

5.4.1 2 SAT

$$p_i : \neg p_i \rightarrow p_i \qquad \neg p_i : p_i \rightarrow \neg p_i$$
$$p_i \vee p_j : \neg p_i \rightarrow p_j, \neg p_j \rightarrow p_i \qquad p_i \vee \neg p_j : \neg p_i \rightarrow \neg p_j, p_j \rightarrow p_i$$
$$\neg p_i \vee \neg p_j : p_i \rightarrow \neg p_j, p_j \rightarrow \neg p_i$$

If p_i and $\neg p_i$ strongly connected, 2-SAT has no solution. The solution must be the one which has greater topological order (smaller SCC order) (as true point is p_i , and wrong point is $p_i + n$)

5.5 Coloring

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D , computes a $(D + 1)$ -coloring of the edges such that no neighboring edges share a color. (D -coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

Time: $\mathcal{O}(NM)$

e210e2, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    rep(i,0,sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
    return ret;
}
```

5.6 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.

Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

b0d5b1, 12 lines

```
typedef bitset<128> B;
```

```
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
    if (!P.any()) { if (!X.any()) f(R); return; }
    auto q = (P | X)._Find_first();
    auto cands = P & ~eds[q];
    rep(i,0,sz(eds)) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    }
}
```

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
        rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
    }
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (sz(R)) {
            if (sz(q) + R.back().d <= sz(qmax)) return;
            q.push_back(R.back().i);
            vv T;
            for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
            if (sz(T)) {
                if (S[lev]++ / ++pk < limit) init(T);
                int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
                C[1].clear(), C[2].clear();
                for (auto v : T) {
                    int k = 1;
                    auto f = [&](int i) { return e[v.i][i]; };
                    while (any_of(all(C[k]), f)) k++;
                    if (k > mxk) mxk = k, C[mxk + 1].clear();
                    if (k < mnk) T[j++].i = v.i;
                    C[k].push_back(v.i);
                }
                if (j > 0) T[j - 1].d = 0;
                rep(k,mnk,mxk + 1) for (int i : C[k])
                    T[j].i = i, T[j++].d = k;
                expand(T, lev + 1);
            } else if (sz(q) > sz(qmax)) qmax = q;
            q.pop_back(), R.pop_back();
        }
    }
    vi maxClique() { init(V), expand(V); return qmax; }
    Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
        rep(i,0,sz(e)) V.push_back({i});
    }
};
```

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertex-Cover.

Triangle.h

Description: Sort all node through degree from large to small. Only the edge from the higher focus to the lower focus is counted. Bruteforce count v, w , then $\mathcal{O}(1)$ check whether w is connected with u . Total time complexity is

Time: $\mathcal{O}\left(n\sqrt{n}\right)$

c3194f, 8 lines

```
for (int i = 0; i < n; ++i) {
    int u = rev[i];
    for (auto v : E[u]) if (id[v] < id[u]) f[v]++;
    for (auto v : E[u]) if (id[v] < id[u])
        for (auto w : E[v]) if (id[w] < id[v])
            ans += f[w];
    for (auto v : E[u]) if (id[v] < id[u]) f[v]--;
}
```

5.7 Trees

BinaryLifting.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.

Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

bfce85, 25 lines

```
vector<vi> treeJump(vi& P){
    int on = 1, d = 1;
    while(on < sz(P)) on *= 2, d++;
    vector<vi> jmp(d, P);
    rep(i,1,d) rep(j,0,sz(P))
        jmp[i][j] = jmp[i-1][jmp[i-1][j]];
    return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
    rep(i,0,sz(tbl))
        if(steps&(1<<i)) nod = tbl[i][nod];
    return nod;
}

int lca(vector<vi>& tbl, vi& depth, int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    a = jmp(tbl, a, depth[a] - depth[b]);
    if (a == b) return a;
    for (int i = sz(tbl); i--;) {
        int c = tbl[i][a], d = tbl[i][b];
        if (c != d) a = c, b = d;
    }
    return tbl[0][a];
}
```

HLD.h

Description: Heavy Light Decomposition

0b2875, 42 lines

```
struct heavy_light_decomposition {
    int n, m, s, tot;
    vector<int> fa, dep, siz, son, top, dfn, rnk;
    // E: edge, fa: father, dep: deep, siz: size of subtree, son:
    //      heavy son
    // top: top of path, dfn: dfs number, rnk: rank of dfs number
    vector<vector<int>> E;
    heavy_light_decomposition(int _n)
        : n(_n), fa(_n, -1), dep(_n), siz(_n), son(_n, -1), top(
            _n, -1), dfn(_n),
            rnk(_n), E(_n) {
        tot = -1;
    }
};
```

```

}
void add_edge(int u, int v) { E[u].emplace_back(v), E[v].
    emplace_back(u); }
int tree_build(int u) {
    siz[u] = 1;
    for (auto v : E[u]) {
        if (v == fa[u]) continue;
        fa[v] = u;
        dep[v] = dep[u] + 1;
        siz[u] += tree_build(v);
        if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;
    }
    return siz[u];
}
void tree_decomposition(int u, int tp) {
    top[u] = tp, dfn[u] = ++tot, rnk[tot] = u;
    if (son[u] != -1) {
        tree_decomposition(son[u], tp);
        for (auto v : E[u]) {
            if (v == son[u] || v == fa[u]) continue;
            tree_decomposition(v, v);
        }
    }
}
auto lca(int u, int v) {
    while (top[u] != top[v])
        if (dep[top[u]] > dep[top[v]])
            u = fa[top[u]];
        else
            v = fa[top[v]];
    return dep[u] > dep[v] ? v : u;
}
};

```

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S| - 1$) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.

Time: $\mathcal{O}(|S| \log |S|)$

"HLD.h" 8c6f9b, 42 lines

```

struct virtual_tree {
    heavy_light_decomposition &G;
    vector<vector<int>>> E;
    virtual_tree(heavy_light_decomposition &ptr) : G(ptr), E(ptr.
        n){};
    void build(vector<int> &node) {
        sort(node.begin(), node.end(),
            [&](int x, int y) { return G.dfn[x] < G.dfn[y]; });
        E[0].clear();
        stack<int> st;
        st.emplace(0);
        for (auto x : node) {
            E[x].clear();
            int top = G.lca(x, st.top());
            if (top == st.top()) {
                st.emplace(x);
                continue;
            }
            auto u = st.top();
            st.pop();
            while (!st.empty() && G.dep[st.top()] > G.dep[top]) {
                E[st.top()].emplace_back(u);
                u = st.top();
                st.pop();
            }
            if (!st.empty() && top == st.top()) {
                E[top].emplace_back(u);
            } else {

```

```

        E[top].clear();
        E[top].emplace_back(u);
        st.emplace(top);
    }
    st.emplace(x);
}
int u = st.top();
st.pop();
while (!st.empty()) {
    E[st.top()].emplace_back(u);
    u = st.top();
    st.pop();
}
};

```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

Time: All operations take amortized $\mathcal{O}(\log N)$.

5909e2, 90 lines

```

struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p == p)) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            z->c[h ^ 1] = b ? x : this;
        }
        y->c[i ^ 1] = b ? this : x;
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() {
        for (pushFlip(); p; ) {
            if (p->p) p->p->pushFlip();
            p->pushFlip(); pushFlip();
            int c1 = up(), c2 = p->up();
            if (c2 == -1) p->rot(c1, 2);
            else p->p->rot(c2, c1 != c2);
        }
    }
    Node* first() {
        pushFlip();
        return c[0] ? c[0]->first() : (splay(), this);
    }
};

struct LinkCut {
    vector<Node> node;

```

LinkCut(int N) : node(N) {}

```

void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
}
void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ? x->c[0]));
    if (x->pp) x->pp = 0;
    else {
        x->c[0] = top->p = 0;
        x->fix();
    }
}
bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
}
void makeRoot(Node* u) {
    access(u);
    u->splay();
    if (u->c[0]) {
        u->c[0]->p = 0;
        u->c[0]->flip ^= 1;
        u->c[0]->pp = u;
        u->c[0] = 0;
        u->fix();
    }
}
Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
        pp->splay(); u->pp = 0;
        if (pp->c[1]) {
            pp->c[1]->p = 0; pp->c[1]->pp = pp; }
        pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
}
};

```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

Time: $\mathcal{O}(E \log V)$

"../data-structures/UnionFindRollback.h" 39e620, 60 lines

```

struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b : a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}

```



```
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1,-1}), comp;
    deque<tuple<int, int, vector<Edge>>> cyps;
    rep(s,0,n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1,{};};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) {
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cyps.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
    }

    for (auto& [u,t,comp] : cyps) { // restore sol (optional)
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
    rep(i,0,n) par[i] = in[i].a;
    return {res, par};
}
```

TreeHashing.h
Description: $hash_{now} = 1 + \sum f(hash(son_i))$. In which f is a random function (but could not be polynomial). Hash Collision is $O(n^2/2^w)$. Magic number in h can be replaced.

644958, 2 lines

```
ll h(ll x) { return x * x * x * 156007 + 995669; }
ll f(ll x) { return h(x & ((1LL << 31) - 1)) + h(x >> 31); }
```

Kruskal.h
Description: When connect u, v , find there represent point U, V , create new node C which $w_c = w_{u,v}$ and connect $C \rightarrow U$ and $C \rightarrow V$. We will get a rooted tree T which has $2n - 1$ nodes. T is a binary tree. All the nodes in G is leaf node in T . For node u and it's ancestor v , $w_u \leq w_v$, which means point the points which point x can reach using edges with weight $\leq d$ is the subtree for the ancestor a ($w_a \leq d$) for x .

Dominator.h
Description: $v \text{ dom } u$ if and only if for all w in $\text{pre}(u)$, $v \text{ dom } w$ Dominator on DAG

9ecfc5, 49 lines

```
constexpr int N = 65536;
int n;
array<vector<int>, N> E, revE, Tree;
array<int, N> in, dep, idom;
vector<int> tpn;
array<array<int, 17>, N> fth;
stack<int> st;
```

```
void topo() {
    st.emplace(0);
    for (int i = 1; i <= n; ++i)
        if (!in[i])
            E[0].push_back(i), revE[i].push_back(0), in[i]++;
    while (!st.empty()) {
        auto u = st.top();
        st.pop();
        tpn.emplace_back(u);
        for (auto v : E[u]) {
            in[v]--;
            if (in[v] == 0)
                st.emplace(v);
        }
    }
}

int lca(int u, int v) {
    if (u == v) return u;
    if (dep[u] < dep[v]) swap(u, v);
    for (int i = 15; i >= 0; --i)
        if (dep[fth[u][i]] >= dep[v]) u = fth[u][i];
    if (u == v)
        return u;
    for (int i = 15; i >= 0; --i)
        if (fth[u][i] != fth[v][i])
            u = fth[u][i], v = fth[v][i];
    return fth[u][0];
}

void build() {
    topo();
    for (auto u : tpn) {
        if (u == 0)
            continue;
        int v = revE[u][0];
        for (auto nv : revE[u])
            v = lca(v, nv);
        idom[u] = v, Tree[v].emplace_back(u), fth[u][0] = v;
        dep[u] = dep[v] + 1;
        for (int i = 1; i <= 15; ++i)
            fth[u][i] = fth[fth[u][i - 1]][i - 1];
    }
}
```

5.8 PlannerGraph

A planar graph satisfy $|V| - |E| + |F| = 2$, in which $|F|$ is the number of faces. When $|V| \geq 3$, we have $|E| \leq 3|V| - 6$ and $|F| \leq 2|V| - 4$.

5.8.1 DualGraph

For each surface F , select one v inside. For all common edge e between F_0, F_x , there's an edge $e_{0,x}$ between v_0, v_x and intersect with e at one point. Iff e is only one boundary of the face F_0, v_0 has a self loop and e_0 intersect with e . The weight of e_0 is same as e . The Minimum Cut of a Planar graph equals to the shortest distance on its Dual graph. We can connect s and t to construct a new face as s' , and the inf face as t' . The shortest distance from s' to d' is the answer. Note: one edge in dual graph is a cut in origin graph.

5.9 Math

5.9.1 Number of Spanning Trees

Create an $N \times N$ matrix mat , and for each edge $a \rightarrow b \in G$, do $\text{mat}[a][b]--$, $\text{mat}[b][b]++$ (and $\text{mat}[b][a]--$, $\text{mat}[a][a]++$ if G is undirected). Remove the i th row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

5.9.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \dots \geq d_n$ exists iff $d_1 + \dots + d_n$ is even and for every $k = 1 \dots n$,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

Numerical (6)

6.1 High-Precision Arithmetic

AdditiveBitset.h
Description: High precision arithmetic over modules. Apparently better than 10-based, can be seen as a additive bitset.

e7bb27, 50 lines

```
// operations are in place.
//
using ull = unsigned long long;
template <ull n>
struct AdditiveBitset {
    vector<ull> v;
    AdditiveBitset() { v.resize(n / 32); }
    void ls1() {
        bool carry = false;
        for (ull i = 0; i < n / 32; ++i) {
            v[i] = (v[i] << 1) | carry;
            carry = (v[i] & (1ll << 32));
            v[i] %= (1ll << 32);
        }
    }
    void rs1() {
        ll carry = 0;
        for (ull i = n / 32 - 1; i >= 0; --i) {
            bool nc = v[i] & 1;
            v[i] = (v[i] >> 1) | (carry << 31);
            carry = nc;
            v[i] %= (1ll << 32);
        }
    }
    bool get(int x) { return v[x / 32] & (1 << (x % 32)); }
    void toggle(int x) { v[x / 32] ^= (1 << (x % 32)); }
    void add(AdditiveBitset& b) {
        bool carry = false;
        for (ull i = 0; i < n / 32; ++i) {
            v[i] = v[i] + b.v[i] + carry;
            carry = (v[i] & (1ll << 32));
            v[i] %= (1ll << 32);
        }
    }
    ll popcount() {
        ll ans = 0;
        for (ull i = 0; i < n / 32; ++i) {
            ans += __builtin_popcountll(v[i] % (1ll << 32));
        }
    }
}
```

```
        return ans;
    }
    void multInt(ll x) {
        ll carry = 0;
        for (ull i = 0; i < n / 32; ++i) {
            v[i] = v[i] * x + carry;
            carry = v[i] >> 32;
            v[i] %= (1ll << 32);
        }
    }
};
```

6.2 Polynomials and recurrences

Polynomial.h

```
struct Poly {
    vector<double> a;
    double operator()(double x) const {
        double val = 0;
        for (int i = sz(a); i--;) (val += x) += a[i];
        return val;
    }
    void diff() {
        rep(i,1,sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    }
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
        a.pop_back();
    }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.

Usage: polyRoots({{2,-3,1}},-1e9,1e9) // solve x²-3x+2 = 0

Time: $\mathcal{O}(n^2 \log(1/\epsilon))$

"Polynomial.h"

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    rep(i,0,sz(dr)-1) {
        double l = dr[i], h = dr[i+1];
        bool sign = p(l) > 0;
        if (sign ^ (p(h) > 0)) {
            rep(it,0,60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m;
                else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    }
    return ret;
}
```

PolyInterpolate.h

Description: Given n points $(x[i], y[i])$, computes an $n-1$ -degree polynomial p that passes through them: $p(x) = a[0] * x^0 + \dots + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$.

Time: $\mathcal{O}(n^2)$

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1;
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    }
    return res;
}
```

BerlekampMassey.h

Description: Recovers any n -order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.

Usage: berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}

Time: $\mathcal{O}(N^2)$

```
".../number-theory/ModPow.h"
vector<ll> berlekampMassey(vector<ll> s) {
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1;
    rep(i,0,n) { ++m;
        ll d = s[i] % mod;
        rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod;
        rep(j,m,n) 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;
}
```

LinearRecurrence.h

Description: Generates the k 'th term of an n -order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots \geq n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.

Usage: linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number

Time: $\mathcal{O}(n^2 \log k)$

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    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;
}
```

6.3 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is ϵ . Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

Usage: double func(double x) { return 4+x+.3*x*x; }

double xmin = gss(-1000,1000,func);

Time: $\mathcal{O}(\log((b-a)/\epsilon))$

```
double gss(double a, double b, double (*f)(double)) {
    double r = (sqrt(5)-1)/2, eps = 1e-7;
    double x1 = b - r*(b-a), x2 = a + r*(b-a);
    double f1 = f(x1), f2 = f(x2);
    while (b-a > eps)
        if (f1 < f2) { //change to > to find maximum
            b = x2; x2 = x1; f2 = f1;
            x1 = b - r*(b-a); f1 = f(x1);
        } else {
            a = x1; x1 = x2; f1 = f2;
            x2 = a + r*(b-a); f2 = f(x2);
        }
    return a;
}
```

Integrate.h

Description: Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4 , although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

```
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
    double h = (b - a) / 2 / n, v = f(a) + f(b);
    rep(i,1,n*2)
        v += f(a + i*h) * (i&1 ? 4 : 2);
    return v * h / 3;
}
```

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.

Usage: double sphereVolume = quad(-1, 1, [](double x) { return quad(-1, 1, [&](double y) { return quad(-1, 1, [&](double z) { return x*x + y*y + z*z < 1; }}});});

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
    d c = (a + b) / 2;
    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}

template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
```

```
    return rec(f, a, b, eps, S(a, b));
}
```

6.4 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$

```
double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1;
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
        }
    }
    return res;
}
```

bd5cec, 15 lines

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}(N^3)$

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}
```

3313dc, 18 lines

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}(n^2m)$

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
            rep(j,i,n) if (fabs(b[j]) > eps) return -1;
            break;
        }
    }
}
```

44c9ab, 38 lines

```
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
        double fac = A[j][i] * bv;
        b[j] -= fac * b[i];
        rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
}

x.assign(m, 0);
for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
}
return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

```
"SolveLinear.h"
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];
fail:; }
}
```

08e495, 7 lines

SolveLinearBinary.h

Description: Solves $Ax = b$ over \mathbb{F}_2 . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b .
Time: $\mathcal{O}(n^2m)$

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        }
        rank++;
    }

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
    }
}
```

fa2d7a, 34 lines

```
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
}
return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h

Description: Invert matrix A . Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of $A \bmod p$, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$

ebfff6, 35 lines

```
int matInv(vector<vector<double>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) {
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
            if (fabs(A[j][k]) > fabs(A[r][c]))
                r = j, c = k;
        if (fabs(A[r][c]) < 1e-12) return i;
        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
        rep(j,0,n)
            swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
        swap(col[i], col[c]);
        double v = A[i][i];
        rep(j,i+1,n) {
            double f = A[j][i] / v;
            A[j][i] = 0;
            rep(k,i+1,n) A[j][k] -= f*A[i][k];
            rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
        }
        rep(j,i+1,n) A[i][j] /= v;
        rep(j,0,n) tmp[i][j] /= v;
        A[i][i] = 1;
    }

    for (int i = n-1; i > 0; --i) rep(j,0,i) {
        double v = A[j][i];
        rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
    }

    rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
    return n;
}
```

6.5 Fourier transforms

FastFourierTransform.h

Description: $\text{fft}(a)$ computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k . N must be a power of 2. Useful for convolution: $\text{conv}(a, b) = c$, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pollwise, divide by n , reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16} ; higher for random inputs). Otherwise, use NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

00ced6, 38 lines

```
// 1. below use 32 bit int and double. check your data range
//    and precision. If you want long double, substitute
//    everywhere possible, and should also work.
// 2. ALWAYS round result by +−0.5 (+ 0.5 is pos; −0.5 if neg);
//    this is how I fail in Seoul one
```

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
```

```
static vector<complex<long double>> R(2, 1);
static vector<C> rt(2, 1); // (^ 10% faster if double)
for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
}
vi rev(n);
rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
        C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
    }
}

vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i,0,sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}
```

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.

Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)
"FastFourierTransform.h" b82773, 22 lines

```
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

NumberTheoreticTransform.h

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k , where $g = \text{root}^{(\text{mod}-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a . For arbitrary modulo, see FFTMod. conv(a, b) = c, where $c[x] = \sum a[i]b[x - i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in $[0, \text{mod})$.

```
Time:  $\mathcal{O}(N \log N)$ 
"/..number-theory/ModPow.h" ced03d, 33 lines

const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
    }
    vl conv(const vl &a, const vl &b) {
        if (a.empty() || b.empty()) return {};
        int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
        int inv = modpow(n, mod - 2);
        vl L(a), R(b), out(n);
        L.resize(n), R.resize(n);
        ntt(L), ntt(R);
        rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
        ntt(out);
        return {out.begin(), out.begin() + s};
    }
}
```

FastSubsetTransform.h

Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$, where \oplus is one of AND, OR, XOR. The size of a must be a power of two.

```
Time:  $\mathcal{O}(N \log N)$ 
464cf3, 16 lines

void FST(vi& a, bool inv) {
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? pii(v - u, u) : pii(v, u + v); // AND
                inv ? pii(v, u - v) : pii(u + v, u); // OR
                pii(u + v, u - v); // XOR
        }
    }
    if (inv) for (int& x : a) x /= sz(a); // XOR only
}

vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
    rep(i,0,sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
}
```

Number theory (7)

7.1 Modular arithmetic

ModLog.h

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,l,m) can be used to calculate the order of a .

```
Time:  $\mathcal{O}(\sqrt{m})$ 
d7e01b, 11 lines

ll modLog(ll a, ll b, ll m) {
    ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
    unordered_map<ll, ll> A;
    while (j <= n && (e = f = e * a % m) != b % m)
        A[e * b % m] = j++;
    if (e == b % m) return j;
    if (gcd(m, e) == gcd(m, b))
        rep(i,2,n+2) if (A.count(e = e * f % m))
            return n * i - A[e];
    return -1;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions. modsum(to, c, k, m) = $\sum_{i=0}^{\text{to}-1} (ki + c) \% m$. divsum is similar but for floored division.

Time: $\log(m)$, with a large constant. 5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
    ull res = k / m * sumsq(to) + c / m * to;
    k %= m; c %= m;
    if (!k) return res;
    ull to2 = (to * k + c) / m;
    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
    c = ((c % m) + m) % m;
    k = ((k % m) + m) % m;
    return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \pmod c$ (or $a^b \pmod c$) for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$. **Time:** $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow bbbd8f, 11 lines

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}

ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution). **Time:** $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p

```
"ModPow.h" 19a793, 24 lines

ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // else no solution
}
```

```
    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
    ll s = p - 1, n = 2;
    int r = 0, m;
    while (s % 2 == 0)
        ++r, s /= 2;
    while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
    ll x = modpow(a, (s + 1) / 2, p);
    ll b = modpow(a, s, p), g = modpow(n, s, p);
    for (; r = m) {
        ll t = b;
        for (m = 0; m < r && t != 1; ++m)
            t = t * t % p;
        if (m == 0) return x;
        ll gs = modpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
    }
}
```

7.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 ≈ 1.5s

6b2912, 20 lines

```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
    const int S = (int)round(sqrt(LIM)), R = LIM / 2;
    vi pr = {2}, sieve(S+1); pr.reserve((int)(LIM/log(LIM)*1.1));
    vector<pii> cp;
    for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
        cp.push_back({i, i * i / 2});
        for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
    }
    for (int L = 1; L <= R; L += S) {
        array<bool, S> block{};
        for (auto &[p, idx] : cp)
            for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
        rep(i,0,min(S, R - L))
            if (!block[i]) pr.push_back((L + i) * 2 + 1);
    }
    for (int i : pr) isPrime[i] = 1;
    return pr;
}
```

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \bmod c$. a is iteration count, b is the number

"ModMuLL.h" 60dcd1, 12 lines

```
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) { // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

Factor.h

Description: Pollard- randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}(n^{1/4})$, less for numbers with small factors.

"ModMuLL.h", "MillerRabin.h" c5522b, 18 lines

```
ull pollard(ull n) {
    auto f = [&n](ull x) { return modmul(x, x, n) + 1; };
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return gcd(prd, n);
}
vector<ull> factor(ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r));
    return l;
}
```

PrimeSieves.h

Description: Linear Prime Sieve. Use FastEratosthenes.h if need low constant. **For sieving range:** Each time store sqrt(n), Sieve 1..sqrt(n) with regular sieve, For each further segs, just have to iterate over all primes in first seg and sieve in that range this is because all composite number have factor <= sqrt(n). Roughly $\mathcal{O}(n \log \log n)$

8b5abf, 12 lines

```
ector<ll> primes;
void sieve(ll sz) {
    deque<bool> np(sz + 2);
    FOR(i, 2, sz) {
        if (!np[i]) primes.emplace_back(i);
        for (auto e: primes) {
            if (i * e > sz) break;
            np[i * e] = true;
            if (i % e == 0) break;
        }
    }
}
```

7.3 Divisibility

exgcd.h

Description: Recall euclidean’s correctness: $a, b \leftrightarrow a, b - a \leftrightarrow a, b \bmod a$

Also **ExGCD correctness:** $(x1, y1) \rightarrow (y1, x1 - y1 * (a/b))$
Also **Diophantine correctness:** If $g|c$ fail, no solution (Bezout: linear combination of 2 number is always divisible by common divisor)
Otherwise, solution is $a \cdot u_g + b \cdot v_g = g \rightarrow a \cdot (c/g * u_g) + (c/g * v_g) = c$
All solutions comes from $x = x_0 + k \cdot \frac{b}{g}$ and $y = y_0 - k \cdot \frac{a}{g}$

i.e. $u + v = u_0 + v_0 + k \cdot \frac{b-a}{g}$ ie if $a < b$ we need min k, $a > b$ need max k, $a = b$ k irrelevant.

Extend to negative GCD is meaningful for \mathbb{Z} by taking abs for its args. ExGCD does not work for neg args (since we are using primitive b so we just convert args to abs val
All Answers very complex as (1) intersection of segments (2) c++ round to 0 not floor (3) result have unobvious sign

4fb1cd, 75 lines

```
namespace Diophantine {
    ll floorDiv(ll x, ll y) {
        bool neg = (x < 0) ^ (y < 0); x = abs(x), y = abs(y);
        if (!neg) return x / y;
        else return - (x + y - 1) / y; // negation become ceil
    }
}
```

```
ll ceilDiv(ll x, ll y) {
    bool neg = (x < 0) ^ (y < 0); x = abs(x), y = abs(y);
    if (!neg) return (x + y - 1) / y;
    else return - x / y; // negation become ceil
}
```

```
bool intersection(pair<ll, ll>& ret, pair<ll, ll> p1, pair<ll, ll> p2) {
    if (p1.first > p1.second || p2.first > p2.second)
        return false; // TODO maybe even throw error?
    ret = {max(p1.first, p2.first), min(p1.second, p2.second)};
    return (ret.first <= ret.second);
}
```

```
#define sgn(x) ((x) > 0 ? 1 : ((x) < 0 ? -1 : 0))
pair<ll, ll> exgcd(ll a, ll b) {
    if (b == 0) return {1, 0};
    auto [x1, y1] = exgcd(b, a % b);
    return {y1, x1 - y1 * (a / b)};
}
```

```
pair<ll, ll> exgcd(ll a, ll b) { // wrapper for
    mathematically correct under negative numers
    ll k1 = sgn(a), k2 = sgn(b);
    auto [u, v] = exgcd(abs(a), abs(b));
    return {k1 * u, k2 * v};
}
```

```
bool diophantine(pair<ll, ll>& ret, ll a, ll b, ll c) {
    if (a == 0 && b == 0) {
        if (c == 0) {
            ret = {0, 0};
            return true;
        } else return false;
    }
```

if (c % gcd(a, b) != 0) return false; // this make solution nonexistent

```
ll k = c / gcd(a, b);
auto [u, v] = exgcd(a, b);
ret = {u * k, v * k};
return true;
}
```

// get the (continuous) solution parametrized by k, of diophantine equation with constraint on x and y.
// does not work when a == 0 and b == 0.

```
bool getDiophantineRanged(pair<ll, ll>& ret, ll a, ll b, ll c, pair<ll, ll> xConstraint, pair<ll, ll> yConstratint) {
    assert(a != 0 || b != 0); // for next line, actly these two lines should be done at input of data
    auto g = gcd(gcd(a, b), c); a /= g, b /= g, c /= g;
```

auto [Lx, Rx] = xConstraint; auto [Ly, Ry] = yConstratint;

```
pair<ll, ll> x0y0;
bool dio = diophantine(x0y0, a, b, c);
auto [x0, y0] = x0y0;
if (!dio) return false;
```

```
pair<ll, ll> k1 = {ceilDiv((b >= 0 ? Lx : Rx) - x0, b), floorDiv((b >= 0 ? Rx : Lx) - x0, b)};
pair<ll, ll> k2 = {ceilDiv(y0 - (a >= 0 ? Ry : Ly), a), floorDiv(y0 - (a >= 0 ? Ly : Ry), a)};
```

```
        return intersection(ret, k1, k2);
    }

    void debugSolutions(ll a, ll b, ll c, pair<ll, ll> range) {
        auto g = gcd(gcd(a, b), c); a /= g, b /= g, c /= g;
        pair<ll, ll> x0y0;
        assert(diophantine(x0y0, a, b, c));
        auto [x0, y0] = x0y0;
        FOR(k, range.first, range.second) cout << "k = " << k
            << ", x = " << x0 + k * b << ", y = " << y0 - k *
                a << endl;
    }
}
```

CRT.h

Description: Chinese Remainder Theorem.
crt(a, m, b, n) computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \leq x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
General Case: You have $a \equiv a_i \pmod{p_i}$, you can express ans as $a = x_1 + x_2p_1 + x_3p_1p_2 + \dots + x_kp_1 \dots p_{k-1}$.
Substitute into CRT, and solve for x yield e.g. $x_3 = ((a_3 - x_1)r_{13} - x_2)r_{23}$. This is $O(k^2)$.
Time: $\log(n)$

```
"euclid.h"                                04d93a, 7 lines

ll crt(ll a, ll m, ll b, ll n) {
    if (n > m) swap(a, b), swap(m, n);
    ll x, y, g = euclid(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m*n/g : x;
}
```

multSieve.h

Description: Multiplicative Sieve; default to euler phi

```
const ll N = ;
vl primes; bool isComposite[N + 2];
ll f[N + 2], cnt[N + 2];

ll fastpow(ll a, ll n) {
    if (n == 0) return 1;
    ll ans = fastpow(a, n/2); ans *= ans;
    if (n % 2) ans *= a;
    return ans;
}
// need to define it for all prime p, f(p ^ k) AND f(1).
// BUT f(1) = 1 ALWAYS since 1 coprime with any k, hence f(k)
// = f(1)f(k).
// hence need f(p ^ k) only.
ll fBase(ll p, ll k) {
    // TODO: return f(p ^ k) where p is a prime.
    // Mobius Func: [k = 0] - [k = 1].
    return fastpow(p, k) - fastpow(p, k - 1);
}

void sieve() {
    f[1] = 1;
    FOR(i, 2, N) {
        if (!isComposite[i]) {
            primes.eb(i);
            f[i] = fBase(i, 1), cnt[i] = 1;
        } // otherwise, f[i] and cnt[i] were already determined
    }

    for(auto p: primes) {
        if (p * i > N) break;
        isComposite[p * i] = true;
        if (i % p == 0) {
            cnt[i * p] = cnt[i] + 1;
        }
    }
}
```

```
// f(i * p) = f(i / p^cnt[i]) * f(p^cnt[i] + 1)
f[i * p] = fBase(p, cnt[i * p]) * f[i / fastpow
    (p, cnt[i])];
break;
}else {
    cnt[i * p] = 1;
    f[i * p] = f[i] * f[p];
}
}
}
```

GCDConvolution.h

Description: Compute $d_k = \sum_{gcd(i,j)=k} a_i \cdot b_j$ for array of size n in $O(n\log n)$. Status: Untested

```
template <typename T>
vector<T> gcdConvolution(vector<T> a, vector<T> b, T mod) {
    int n = a.size();
    vector<T> A(n), B(n), D(n);
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j += i + 1) {
            A[i] += a[j], B[i] += b[j];
            A[i] %= mod, B[i] %= mod;
        }
        D[i] = A[i] * B[i], D[i] %= mod;
    }
    vector<T> d(n);
    for (int i = n - 1; i >= 0; i--) {
        d[i] = D[i];
        for (int j = 2 * i + 1; j < n; j += i + 1) {
            d[i] -= d[j], d[i] = (d[i] + mod) % mod;
        }
    }
    return d;
}
```

7.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.
For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. (p_k/q_k alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞ ; if x is the root of a degree 2 polynomial the a 's eventually become cyclic.
Time: $\mathcal{O}(\log N)$

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
    for (;;) {
        ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
            a = (ll)floor(y), b = min(a, lim),
            NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If b > a/2, we have a semi-convergent that gives us a
            // better approximation; if b = a/2, we *may* have one.
            // Return {P, Q} here for a more canonical approximation.
            return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
                make_pair(NP, NQ) : make_pair(P, Q);
        }
        if (abs(y = 1/(y - (d)a)) > 3*N) {
            return {NP, NQ};
        }
        LP = P; P = NP;
        LQ = Q; Q = NQ;
    }
}
```

FracBinarySearch.h

Description: Given f and N , finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.
Usage: fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}
Time: $\mathcal{O}(\log(N))$

```
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >= si) {
            adv += step;
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                adv -= step; si = 2;
            }
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B; B = !adv;
    }
    return dir ? hi : lo;
}
```

7.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

7.6 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a , except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

7.7 Estimates

$$\sum_{d \mid n} d = O(n \log \log n).$$

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

7.8 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(n/d)$$

Other useful formulas/forms:

$$\sum_{d|n} \mu(d) = [n = 1] \text{ (very useful)}$$

$$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n)g(d)$$

$$g(n) = \sum_{1 \leq m \leq n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m)g(\lfloor \frac{n}{m} \rfloor)$$

Strings (8)

KMP-Zfunc.h
Description: NA

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}

vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

Manacher.h
Description: Useful in: Palindrome related problem to extend/remove characters to form palindrome The array d is a useful array in which d[i] stored the maximum size of odd length palindrome where middle is i. Be caution: The string have been extended with sharp symbol between each adjacent characters for general purpose of odd and even length palindrome

```
// Can return the vector d
ll manacher(string _s){
    // Add sharp symbol in between all the characters to
    // generalize even and odd length palindrome
    string s = "#";
    for(char c : _s) s += c, s += '#';
    int n = s.size();
    vector<int> d(n);
    // A little trick to ensure linearity
    for(int i = 0, l = 0, r = -1; i < n; i++){
        int k = (i > r)? l : min(d[l + r - i], r - i + 1);
        while(0 <= i - k && i + k < n && s[i - k] == s[i + k])
            k++;
    }
```

KMP-Zfunc Manacher SuffixAutomaton SuffixArray

```
        d[i] = k--;
        if(i + k > r) l = i - k, r = i + k;
    }
    // Calculate the number of palindrome
    ll ret = 0;
    for(int c : d) ret += c/2;
    return ret;
}
```

SuffixAutomaton.h
Description: o(n log k), Can achive O(n) by replacing map with fixed size array

```
struct state {
    int len, link;
    map<char, int> next;
};

const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last; // sz - size of the state, last - the state
               // corresponding whole string

// Init automaton with single state
void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

SuffixArray.h
Description: suffix array

```
typedef vector<ll> vl;

class SuffixArray{
private:
    ll max_range = 2e5+2; // TODO the maximum value (Ensure
                           // it won,t exceed the requirement)
```

```
ll n;

void countingSort(int k){
    vl freq(max_range);
    for (int i = 0; i < n; ++i) //
        count the frequency
        freq[ i + k < n ? rk[i+k] : 0]++; //
        of each integer rank
    for (int i = 0, sum = 0; i < max_range; ++i) {
        int t = freq[i];
        freq[i] = sum;
        sum += t;
    }
    vl tempsa(n);
    for (int i = 0; i < n; ++i) //
        sort SA
    tempsa[ freq[ sa[i]+k < n ? rk[sa[i]+k] : 0] ++ ] =
        sa[i];
    swap(sa, tempsa);
}

void constructLCP(){
    vl phi(n);
    vl plcp(n);
    phi[sa[0]] = -1;
    for(int i = 1; i < n; i++)
        phi[sa[i]] = sa[i-1];

    for(int i = 0, l = 0; i < n; i ++ ){
        if(phi[i] == -1){
            plcp[i] = 0;
            continue;
        }
        while((i+l < n) && (phi[i] + l < n) && s[i+l]
            == s[phi[i] + l]) l++;
        plcp[i] = l;
        l = max(l-1,0);
    }
    lcp.resize(n);
    for(int i = 0; i < n; i++)
        lcp[i] = plcp[sa[i]];
}

void constructSA() {
    // init
    rk.assign(n,0);
    sa.assign(n,0);
    iota(sa.begin(), sa.end(), 0);
    for (int i = 0; i < n; ++i) rk[i] = s[i];

    // repeat log_2 n times
    for (int k = 1; k < n; k <= 1) {
        countingSort(k); //
        radix sort
        countingSort(0); //
        stable-sort
        vl temprk(n);
        int r = 0;
        temprk[sa[0]] = r; //
        re-ranking process
        for (int i = 1; i < n; ++i) //
            compare adj suffixes, same pair => same
            rank r; otherwise, increase r
        temprk[sa[i]] =
            ((rk[sa[i]] == rk[sa[i-1]]) && (rk[sa[i]+k]
                == rk[sa[i-1]+k])) ?
                r : ++r;
        swap(rk, temprk); //
        update RA
```

```

        // constant optimization
        max_range = r + 1;
        if (rk[sa[n-1]] == n-1) break;
    }
}
// vl for s,a,q are changable to string
public:
    vl rk, sa, lcp, owner; // 0 - index lcp in sa form
        between me and previous
    vector<ll> s;
    void discrete(vector<ll> a){
        set<ll> st;
        for(int c:a) st.insert(c);
        ll idx = 0;
        unordered_map<ll,ll> mp;
        for(int c:st) mp[c] = idx++;
        s.assign(a.size(),0);
        for(int i = 0; i < n; i++) s[i] = mp[a[i]];
    }
    SuffixArray(vector<ll> a){
        n = a.size();
        discrete(a); // Comment if discretization is
            unnecessary;
        constructSA();
        constructLCP();
    }

    int lower_bound(vl q){
        int l = 1, r = n;
        while(l<r){
            int m = (l+r)/2;
            if(compare(sa[m],q)>=0) r = m;
            else l = m + 1;
        }
        return (l!=n)? sa[l]: -1;
    }

    int upper_bound(vl q){
        int l = 1, r = n;
        while(l < r){
            int m = (l + r)/2;
            if(compare(sa[m], q) > 0) r = m;
            else l = m + 1;
        }
        return (l!=n)? sa[l] : -1;
    }

    int compare(int idx, vl q){
        for(int i = 0; i < q.size(); i++){
            if(s[idx+i] < q[i]) return -1;
            else if(s[idx+i] > q[i]) return 1;
        }
        return 0;
    }
};
```

SuffixTree.h
Description: Ukkonen’s algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$

aae0b8, 50 lines

```

struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
```

```

    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m++]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q==-1 || c==toi(a[q])) q++; else {
            l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
            p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
            l[v]=q; p[v]=m; t[p[m]][toi(a[l[m])]]=m;
            v=s[p[m]]; q=l[m];
            while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }
            if (q==r[m]) s[m]=v; else s[m]=m+2;
            q=r[v]-(q-r[m]); m+=2; goto suff;
        }
    }

    SuffixTree(string a) : a(a) {
        fill(r,r+N,sz(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
        fill(t[1],t[1]+ALPHA,0);
        s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
        rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
    }

    // example: find longest common substring (uses ALPHA = 28)
    pii best;
    int lcs(int node, int i1, int i2, int olen) {
        if (l[node] <= i1 && i1 < r[node]) return 1;
        if (l[node] <= i2 && i2 < r[node]) return 2;
        int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
        rep(c,0,ALPHA) if (t[node][c] != -1)
            mask |= lcs(t[node][c], i1, i2, len);
        if (mask == 3)
            best = max(best, {len, r[node] - len});
        return mask;
    }
    static pii LCS(string s, string t) {
        SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
        st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
        return st.best;
    }
};
```

Hashing.h
Description: Self-explanatory methods for string hashing.

2d2a67, 44 lines

```

// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
typedef uint64_t ull;
struct H {
    ull x; H(ull x=0) : x(x) {}
    H operator+(H o) { return x + o.x + (x + o.x < x); }
    H operator-(H o) { return *this + ~o.x; }
    H operator*(H o) { auto m = (_uint128_t)x * o.x;
        return H((ull)m) + (ull)(m >> 64); }
    ull get() const { return x + !~x; }
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (1l)1e11+3; // (order ~ 3e9; random also ok)
```

```

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
        pw[0] = 1;
        rep(i,0,sz(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};

vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
    rep(i,0,length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    rep(i,length,sz(str)) {
        ret.push_back(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret;
}

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

AhoCorasick.h
Description: ans store the occurrence in dict. Build to be called after all insert. Dfs is dp that store total Number of occurrences, if want one occurrence, can set dp[u] = val as visited.

e596d9, 80 lines

```

class AC {
public:

    int tot = 0;
    struct Node{
        int next[26], ans, link, p;
        char pch;
        Node(int p):p(p){
            link = 0 , ans = 0;
            memset(next, 0,sizeof next);
        }
    };
    vector<Node> tr;

    vector<int> dp;

    AC(){
        tr.emplace_back(0);
    }
    // Trie
    void insert(string s) {
        int u = 0;
        for (int i = 0; i < s.size(); i++) {
            if (!tr[u].next[s[i] - 'a'])
                tr[u].next[s[i] - 'a'] = ++tot, tr.emplace_back(u);
            u = tr[u].next[s[i] - 'a'];
        }
        tr[u].ans++;
    }

    queue<int> q;
    void build() {
        for (int i = 0; i < 26; i++)
            if (tr[0].next[i]) q.push(tr[0].next[i]);
        while (!q.empty()) {
            int u = q.front();
```



```
q.pop();
for (int i = 0; i < 26; i++) {
    if (tr[u].next[i]) {
        tr[tr[u].next[i]].link = tr[tr[u].link].next[i];
        q.push(tr[u].next[i]);
    }
    else tr[u].next[i] = tr[tr[u].link].next[i];
}
dp.assign(tot+1,-1);

int dfs(int u){
    if(dp[u] != -1) return dp[u];
    if(u == 0) return dp[u] = 0;
    return dp[u] = dfs(tr[u].link) + tr[u].ans;
}

int query(string t) {
    int u = 0, res = 0;
    for (int i = 0; i < t.size(); i++) {
        u = tr[u].next[t[i] - 'a'];
        res += dfs(u);
    }
    return res;
}

};
int main(){
    string dict[] = {"i","in","tin", "sting"};
    AC ac;
    for(int i = 0; i < 4; i++)
        ac.insert(dict[i]);
    ac.build();
    cout << ac.query("stingin") << "\n";
    // Debug
    for(int i = 0; i <= ac.tot; i++){
        cout << i << " th Node - ";
        for(int k = 0; k < 26; k++)
            if(ac.tr[i].next[k] != 0)
                cout << (char)('a' + k) << ":" << ac.tr[i].next[k] <<
                    " ";
        cout << "Suffix: " << ac.tr[i].link;
        cout << "\n";
    }
}
```

MinRotation.h
Description: Finds the lexicographically smallest rotation of a string.
Usage: rotate(v.begin(), v.begin()+minRotation(v), v.end());
Time: $\mathcal{O}(N)$

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) { a = b; break; }
    }
    return a;
}
```

Various (9)

9.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

Time: $\mathcal{O}(\log N)$

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}), before = it;
    while (it != is.end() && it->first <= R) {
        R = max(R, it->second);
        before = it = is.erase(it);
    }
    if (it != is.begin() && (--it)->second >= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    }
    return is.insert(before, {L,R});
}
```

```
void removeInterval(set<pii>& is, int L, int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L;
    if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h
Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
    vi S(sz(I)), R;
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
    T cur = G.first;
    int at = 0;
    while (cur < G.second) { // (A)
        pair<T, int> mx = make_pair(cur, -1);
        while (at < sz(I) && I[S[at]].first <= cur) {
            mx = max(mx, make_pair(I[S[at]].second, S[at]));
            at++;
        }
        if (mx.second == -1) return {};
        cur = mx.first;
        R.push_back(mx.second);
    }
    return R;
}
```

ConstantIntervals.h
Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});
Time: $\mathcal{O}(k \log \frac{n}{k})$

```
template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
    if (p == q) return;
    if (from == to) {
        g(i, to, p);
        i = to; p = q;
    }
```

```
} else {
    int mid = (from + to) >> 1;
    rec(from, mid, f, g, i, p, f(mid));
    rec(mid+1, to, f, g, i, p, q);
}
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
    if (to <= from) return;
    int i = from; auto p = f(i), q = f(to-1);
    rec(from, to-1, f, g, i, p, q);
    g(i, to, q);
}
```

9.2 Misc. algorithms

TernarySearch.h
Description: Find the smallest i in [a,b] that maximizes f(i), assuming that f(a) < ... < f(i) ≥ ... ≥ f(b). To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: int ind = ternSearch(0,n-1,[&](int i){return a[i];});
Time: $\mathcal{O}(\log(b-a))$

```
template<class F>
int ternSearch(int a, int b, F f) {
    assert(a <= b);
    while (b - a >= 5) {
        int mid = (a + b) / 2;
        if (f(mid) < f(mid+1)) a = mid; // (A)
        else b = mid+1;
    }
    rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
}
```

LIS.h
Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$

```
template<class I> vi lis(const vector<I>& S) {
    if (S.empty()) return {};
    vi prev(sz(S));
    typedef pair<I, int> p;
    vector<p> res;
    rep(i,0,sz(S)) {
        // change 0 -> i for longest non-decreasing subsequence
        auto it = lower_bound(all(res), p{S[i], 0});
        if (it == res.end()) res.emplace_back(), it = res.end()-1;
        *it = {S[i], i};
        prev[i] = it == res.begin() ? 0 : (it-1)->second;
    }
    int L = sz(res), cur = res.back().second;
    vi ans(L);
    while (L--) ans[L] = cur, cur = prev[cur];
    return ans;
}
```

FastKnapsack.h
Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S ≤ t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$

```
int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1);
```

```
v[a+m-t] = b;
rep(i,b,sz(w)) {
    u = v;
    rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
    for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
        v[x-w[j]] = max(v[x-w[j]], j);
}
for (a = t; v[a+m-t] < 0; a--);
return a;
}
```

Discretization.h

Description: Discretization, two ways of writing it.

5860f0, 53 lines

```
//1-index, vectors
vi b = a;
sort(b.begin() + 1, b.end());
b.erase(unique(b.begin() + 1, b.end()), b.end()); // there is
// no need to do unique actually lol jiangly don't
// then you can iterate over b, having map<ll, ll> zip, unzip.

struct IntegerDisc {
    vector<ll> vecs;
    vector<ll> dedup;
    void init() {
        vecs.clear();
        dedup.clear();
    }
    void insert(ll v) { vecs.emplace_back(v); }
    void discretize() {
        sort(vecs.begin(), vecs.end());

        for (auto e: vecs) {
            while (!dedup.empty() && dedup.back() == e) dedup.
                pop_back();
            dedup.emplace_back(e);
        }
    }
    // assume x is in vecs; zero index.
    ll query(const ll v) {
        auto pos = lower_bound(dedup.begin(), dedup.end(), v);
        return pos - dedup.begin();
    }
};

struct FloatDisc {
    vector<ld> vecs;
    vector<ld> dedup;
    void init() {
        vecs.clear();
        dedup.clear();
    }
    void insert(ld v) { vecs.emplace_back(v); }
    void discretize() {
        sort(vecs.begin(), vecs.end());

        for (auto e: vecs) {
            while (!dedup.empty() && EQ(dedup.back(), e)) dedup
                .pop_back();
            dedup.emplace_back(e);
        }
    }
    // assume x is in vecs; zero index.
    ll query(const ld v) {
        auto pos = lower_bound(dedup.begin(), dedup.end(), v);
        if (pos != dedup.begin() && abs(*prev(pos) - v) < abs(*
            pos - v)) pos = prev(pos);
        return pos - dedup.begin();
    }
}
```

```
};

9.3 Dynamic programming
KnuthDP.h
Description: When doing DP on intervals: a[i][j] = min_{i < k < j} (a[i][k] +
a[k][j]) + f(i, j), where the (minimal) optimal k increases with both i
and j, one can solve intervals in increasing order of length, and search
k = p[i][j] for a[i][j] only between p[i][j - 1] and p[i + 1][j]. This is
known as Knuth DP. Sufficient criteria for this are if f(b, c) ≤ f(a, d) and
f(a, c) + f(b, d) ≤ f(a, d) + f(b, c) for all a ≤ b ≤ c ≤ d. Consider also:
LineContainer (ch. Data structures), monotone queues, ternary search.
Time: O(N^2)
```

DivideAndConquerDP2.h

Description: Alternative Divide and conquere DP

e52868, 19 lines

```
function<void(ll, ll, ll, ll, ll)> dfs = [&](ll l, ll r, ll L,
    ll R, ll x) {
    if (l > r || L > R) return;
    ll mid = (l + r) / 2;
    ll pos = L;
    for (ll i = L; i <= min(R, mid); ++i) {
        if (dp[x][mid] > dp[x - 1][i - 1] + cost[i][mid]) {
            dp[x][mid] = dp[x - 1][i - 1] + cost[i][mid];
            pos = i;
        }
    }
    dfs(l, mid - 1, L, pos, x);
    dfs(mid + 1, r, pos, R, x);
};

// init
for (ll x = 1; x <= k; ++x) {
    // init
    dfs(1, n, 1, n, x);
}
```

- 9.4 Optimization tricks
- 9.4.1 Bit hacks
- x & $\neg x$ is the least bit in x .
 - `for (int x = m; x;) { $\neg x$ &= m; ... }` loops over all subset masks of m (except m itself).
 - $c = x \& \neg x$, $r = x + c$; $((r \wedge x) \gg 2) / c$ | r is the next number after x with the same number of bits set.
 - `rep(b,0,K) rep(i,0,(1 << K))`
if $(i \& 1 \ll b)$ $D[i] += D[i \wedge (1 \ll b)]$; computes all sums of subsets.

FastMod.h

Description: Compute $a \% b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to a (mod b) in the range $[0, 2b)$.

751a02, 8 lines

```
typedef unsigned long long ull;
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m((-1ULL / b) {}
    ull reduce(ull a) { // a % b + (0 or b)
        return a - (ull)((__uint128_t(m) * a) >> 64) * b;
    }
};
```