

Bangladesh University of Engineering and Technology

BUET Supernova

Rubai, Nafees, Tahjib

1 Contest

2 Mathematics

3 Number theory

4 Combinatorial

5 Numerical

6 Data structures

7 Strings

8 Geometry

9 Graph

10 Various

Contest (1)

template.cpp

14 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
}
```

.bashrc

4 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
-fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <
alias b='g++ -Wshadow -Wall -o "%e" -g -fsanitize=address -
    fsanitize=undefined -D_GLIBCXX_DEBUG'
```

hash.sh

3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum | cut -c-6
```

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

1 The extremum is given by $x = -b/2a$.

$$\begin{aligned} 1 & ax + by = e \quad x = \frac{ed - bf}{ad - bc} \\ 2 & cx + dy = f \quad y = \frac{af - ec}{ad - bc} \end{aligned}$$

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

$$9 \quad x_i = \frac{\det A'_i}{\det A}$$

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

2.2 Recurrences

13 If $a_n = c_1 a_{n-1} + \dots + c_k a_{n-k}$, and r_1, \dots, r_k are distinct roots of $x^k - c_1 x^{k-1} - \dots - c_k$, there are d_1, \dots, d_k s.t.

$$16 \quad a_n = d_1 r_1^n + \dots + d_k r_k^n.$$

22 Non-distinct roots r become polynomial factors, e.g.

$$a_n = (d_1 n + d_2) r^n.$$

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

2.4 Geometry

2.4.1 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):

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

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

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

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

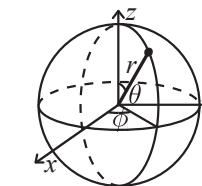
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}} \quad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x \quad \frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \quad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x) \quad \int x e^{ax} dx = \frac{e^{ax}}{a^2} (ax - 1)$$

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

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$

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

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x . It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y ,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

2.8.1 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 .

template .bashrc hash ModularArithmetic

First success distribution

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

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

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

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 $\text{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

2.8.2 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

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

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let X_1, X_2, \dots be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{P}^{(n)} = \mathbf{P}^n \mathbf{P}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{P}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi = \pi \mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i . π_j/π_i is the expected number of visits in state j between two visits in state i .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an *A-chain* if the states can be partitioned into two sets \mathbf{A} and \mathbf{G} , such that all states in \mathbf{A} are absorbing ($p_{ii} = 1$), and all states in \mathbf{G} leads to an absorbing state in \mathbf{A} . The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j , is $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$. The expected time until absorption, when the initial state is i , is $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$.

Number theory (3)

3.1 Modular arithmetic

ModularArithmetic.h

Description: Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

```
*euclid.h*                                         35bfea, 18 lines
const ll mod = 17; // change to something else
struct Mod {
    ll x;
    Mod(ll xx) : x(xx) {}
    Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
    Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
    Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
    Mod operator/(Mod b) { return *this * invert(b); }
    Mod invert(Mod a) {
        ll x, y, g = euclid(a.x, mod, x, y);
        assert(g == 1); return Mod((x + mod) % mod);
    }
    Mod operator^(ll e) {
        if (!e) return Mod(1);
        Mod r = *this ^ (e / 2); r = r * r;
        return e&1 ? *this * r : r;
    }
};
```

ModInverse.h

Description: Pre-computation of modular inverses. Assumes $LIM \leq mod$ and that mod is a prime.

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

}

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i, 2, LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

ModPow.h

b83e45, 8 lines

```
const ll mod = 1000000007; // faster if const
```

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

ModLog.h

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

Time: $\mathcal{O}(\sqrt{m})$

c040b8, 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.

$\text{modsum}(to, c, k, m) = \sum_{i=0}^{to-1} (ki + c) \% m$. 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. $x^2 \equiv a \pmod{p}$ ($-x$ 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;
    }
}
```

3.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM .

Time: $LIM=1e9 \approx 1.5s$

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 $a^b \pmod{c}$.

"ModMulLL.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-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. $2299 \rightarrow \{11, 19, 11\}$).

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

"ModMulLL.h", "MillerRabin.h"

d8d98d, 18 lines

```
ull pollard(ull n) {
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    auto f = [&](ull x) { return modmul(x, x, n) + i; };
    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;
}
```

3.3 Divisibility

euclid.h

Description: Finds two integers x and y , such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in $_gcd$ instead. If a and b are coprime, then x is the inverse of a (\pmod{b}).

33ba8f, 5 lines

```
ll euclid(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d;
}
```

CRT.h

Description: Chinese Remainder Theorem.

$\text{crt}(a, m, b, n)$ 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 $mn < 2^{62}$.

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;
}
```

3.3.1 Bézout's identity

For $a \neq b \neq 0$, then $d = \gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a, b)}, y - \frac{ka}{\gcd(a, b)} \right), \quad k \in \mathbb{Z}$$

phiFunction.h

Description: Euler's ϕ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with n . $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1}p_2^{k_2}\dots p_r^{k_r}$ then $\phi(n) = (p_1-1)p_1^{k_1-1}\dots(p_r-1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n} (1 - 1/p)$.

$$\sum_{d|n} \phi(d) = n, \sum_{1 \leq k \leq n, \gcd(k, n)=1} k = n\phi(n)/2, n > 1$$

Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.

Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p} \forall a$.

cf7d6d, 8 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
    rep(i, 0, LIM) phi[i] = i&1 ? i : i/2;
    for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

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

dd6c5e, 21 lines

```
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: O(log(N))
27ab3e, 25 lines

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;
}
```

3.5 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.

3.6 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

3.7 Estimates

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

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 $n < 1e19$.

3.8 Möbius Function

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

Möbius Inversion:

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

Other useful formulas/forms:

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

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

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

Combinatorial (4)

4.1 Permutations

4.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

IntPerm.h

Description: Permutation \rightarrow integer conversion. (Not order preserving.) Integer \rightarrow permutation can use a lookup table.

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

044568, 6 lines

```
int permToInt(vi v) {
    int use = 0, i = 0, r = 0;
    for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
        use |= 1 << x; // (note: minus, not ~!)
    return r;
}
```

4.1.2 Cycles

Let $gs(n)$ be the number of n -permutations whose cycle lengths all belong to the set S . Then

$$\sum_{n=0}^{\infty} gs(n) \frac{x^n}{n!} = \exp \left(\sum_{n \in S} \frac{x^n}{n} \right)$$

4.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left[\frac{n!}{e} \right]$$

4.1.4 Burnside's lemma

Given a group G of symmetries and a set X , the number of elements of X up to symmetry equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts “configurations” (of some sort) of length n , we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k).$$

4.2 Partitions and subsets

4.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$\begin{aligned} p(0) &= 1, \quad p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2) \\ p(n) &\sim 0.145/n \cdot \exp(2.56\sqrt{n}) \\ \begin{array}{c|cccccccccc} n & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 20 & 50 & 100 \\ \hline p(n) & 1 & 1 & 2 & 3 & 5 & 7 & 11 & 15 & 22 & 30 & 627 & \sim 2e5 & \sim 2e8 \end{array} \end{aligned}$$

4.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \dots + n_1 p + n_0$ and $m = m_k p^k + \dots + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$.

4.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$. a0a312, 5 lines

```
11 multinomial(vi& v) {
    11 c = 1, m = v.empty() ? 1 : v[0];
    rep(i, 1, sz(v)) rep(j, 0, v[i]) c = c * ++m / (j+1);
    return c;
}
```

4.3 General purpose numbers

4.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t-1}$ (FFT-able).

$$B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$$

Sums of powers:

$$\sum_{i=1}^n i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\begin{aligned} \sum_{i=m}^{\infty} f(i) &= \int_m^{\infty} f(x) dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m) \\ &\approx \int_m^{\infty} f(x) dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m)) \end{aligned}$$

4.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1$$

$$\sum_{k=0}^n c(n, k)x^k = x(x+1)\dots(x+n-1)$$

multinomial Polynomial PolyRoots

$$\begin{aligned} c(8, k) &= 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1 \\ c(n, 2) &= 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots \end{aligned}$$

4.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j :s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j :s s.t. $\pi(j) \geq j$, k j :s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

4.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

4.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

4.3.6 Labeled unrooted trees

on n vertices: n^{n-2}

on k existing trees of size n_i : $n_1 n_2 \dots n_k n^{k-2}$

with degrees d_i : $(n-2)!/((d_1-1)! \dots (d_n-1)!)$

4.3.7 Catalan numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

4.4 Added but no description yet

4.4.1 Tahjib

$$\frac{1}{(1-x)^{m+1}} = \sum_{k \geq 0} \binom{k+m}{m} x^k$$

$$\frac{x^m}{(1-x)^{m+1}} = \sum_{k \geq 0} \binom{k}{m} x^k$$

$$\frac{1}{\sqrt{1-4x}} = \sum_{k \geq 0} \binom{2k}{k} x^k$$

$$\frac{1 - \sqrt{1-4x}}{2x} = \sum_{k \geq 0} C_k x^k$$

$$\frac{x}{1-x-x^2} = \sum_{i \geq 0} F_i x^i (F_0 = 0, F_1 = 1)$$

Numerical (5)

5.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

```
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[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 $x^2-3x+2 = 0$

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

```
"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"                                96548b, 20 lines
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: $\mathcal{O}(n^2 \log k)$

```

f4e444, 26 lines
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;
}

```

5.2 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 and no local maximum. The maximum error in the result is eps . Works especially 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; }

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

```

31d45b, 14 lines
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;
}

```

HillClimbing.h

Description: Poor man's optimization for unimodal functions

```

seeeaf, 14 lines
typedef array<double, 2> P;

template<class F> pair<double, P> hillClimb(P start, F f) {
    pair<double, P> cur(f(start), start);
    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
        rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        }
    }
    return cur;
}

```

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.

4756fc, 7 lines

```

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; });});});

92dd79, 15 lines

```

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));
}

```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.

Usage: vvd A = {{1,-1}, {-1,1}, {-1,-2}};

vd b = {1,1,-4}, c = {-1,-1}, x;

T val = LPSolver(A, b, c).solve(x);

Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

aa8530, 68 lines

```

typedef double T; // long double, Rational, double + modP>...
typedef vector<T> vd;
typedef vector<vd> vvd;

```

```

const T eps = 1e-8, inf = 1./0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j], N[j]) < MP(X[s], N[s])) s=j

```

```

struct LPSolver {

```

```

    int m, n;
    vi N, B;
    vvd D;

```

```

LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
    rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
    rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
    rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;
}

```

```

}

void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
        T *b = D[i].data(), inv2 = b[s] * inv;
        rep(j,0,n+2) b[j] -= a[j] * inv2;
        b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
}

bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
        int s = -1;
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        if (D[x][s] >= -eps) return true;
        int r = -1;
        rep(i,0,m) {
            if (D[i][s] <= eps) continue;
            if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                < MP(D[r][n+1] / D[r][s], B[r])) r = i;
        }
        if (r == -1) return false;
        pivot(r, s);
    }
}

T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
        pivot(r, n);
        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
        rep(i,0,m) if (B[i] == -1) {
            int s = 0;
            rep(j,1,n+1) ltj(D[i]);
            pivot(i, s);
        }
        bool ok = simplex(1); x = vd(n);
        rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
        return ok ? D[m][n+1] : inf;
    }
}

```

5.3 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;
}

```

```

}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

Time: $\mathcal{O}(N^3)$

3313dc, 18 lines

```

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;
    }
}

```

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

44c9ab, 38 lines

```

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;
        }
        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:

80e495, 7 lines

```

" 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:; }

```

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

fa2d7a, 34 lines

```

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;
        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 mod p, and k is doubled in each step.

Time: $\mathcal{O}(n^3)$

ebffff, 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]) < le-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;
}

```

Tridiagonal.h

Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, 1 \leq i \leq n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, \dots, -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique.

If $|d_i| > |p_i| + |q_{i-1}|$ for all i , or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither tr nor the check for $\text{diag}[i] == 0$ is needed.

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

8f9fa8, 26 lines

```

typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
    int n = sz(b); vi tr(n);
    rep(i,0,n-1) {
        if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
            b[i+1] -= b[i] * diag[i+1] / super[i];
            if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
            diag[i+1] = sub[i]; tr[i+1] = 1;
        } else {
            diag[i+1] -= super[i]*sub[i]/diag[i];
            b[i+1] -= b[i]*sub[i]/diag[i];
        }
    }
    for (int i = n; i--;) {
        if (tr[i]) {
            swap(b[i], b[i-1]);
            diag[i-1] = diag[i];
            b[i] /= super[i-1];
        } else {
    }

```

```

        b[i] /= diag[i];
        if (i) b[i-1] -= b[i]*super[i-1];
    }
    return b;
}

```

5.4 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 pointwise, 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.

Usage: $\text{vl a, b; vl c} = \text{conv}(a, b)$

Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (~1s for $N = 2^{22}$)

```

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) / li;
}
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: $\text{ntt}(a)$ computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k , where $g = \text{root}^{(mod-1)/N}$. 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. $\text{conv}(a, b) = c$, 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, mod].

Usage: $\text{vl a, b; vl c} = \text{conv}(a, b)$

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

```

"../number-theory/ModPower.h"
ced03d, 39 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.
// 7340033 = 7 * 2^20, G = 3
// 645922817 = 77 * 2^23, G = 3
// 897581057 = 107 * 2^23, G = 3
// 998244353 = 119 * 2^23, G = 3
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) {
            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, mod].

Usage: $\text{vl a, b; vl c} = \text{convMod}<\text{MOD}>(a, b)$

Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

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;
    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) / li;
    }
    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;
    }
}
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;
}
```

Data structures (6)**Rng.h**

Description: random num generator. rnd is in integer range. rnd.range is in long long range.

Time: $\mathcal{O}(1)$

89247f, 7 lines

```
mt19937 rng(random_device{}());
ll rnd(ll r) {
    return rng() % r;
}
ll rng_range(ll l, ll r) {
    return uniform_int_distribution<ll>(l, r)(rng);
}
```

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 nulltype.

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

782797, 16 lines

```
#include <bits/extc++.h>
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
}
```

SegTreeBeats.h

Description: range chmin and range add operations and range sum queries.

```
struct SegtreeBeats {
    vector<long long> sum, mx, smx, mx_c, lz_add;
    int n;
    void merge(int tv) {
        int l = tv*2, r = tv*2 + 1;
```

```
if(mx[l] < mx[r]) {
    mx[tv] = mx[r], mx_c[tv] = mx_c[r];
    smx[tv] = max(mx[l], max(smx[l], smx[r]));
}
else if(mx[l] > mx[r]) {
    mx[tv] = mx[l], mx_c[tv] = mx_c[l];
    smx[tv] = max(mx[r], max(smx[l], smx[r]));
}
else {
    mx[tv] = mx[l], mx_c[tv] = mx_c[l] + mx_c[r];
    smx[tv] = max(smx[l], smx[r]);
}
sum[tv] = sum[tv*2] + sum[tv*2 + 1];

void build(int tv, int tl, int tr, vector<long long> &v) {
    if(tl == tr) {
        sum[tv] = mx[tv] = v[tl-1];
        smx[tv] = -inf;
        mx_c[tv] = 1;
    }
    else {
        int tm = (tl + tr) >> 1;
        build(tv*2, tl, tm, v);
        build(tv*2 + 1, tm+1, tr, v);
        merge(tv);
    }
}

void add_to_node(int tv, int tl, int tr, long long val) {
    mx[tv] += val;
    smx[tv] += val;
    sum[tv] += val * (tr-tl+1);
    lz_add[tv] += val;
}

void update_node(int tv, long long val) {
    if(val < mx[tv]) {
        sum[tv] -= mx_c[tv] * (mx[tv] - val);
        mx[tv] = val;
    }
}

void push(int tv, int tl, int tr) {
    int tm = (tl + tr) / 2;
    int l = tv*2, r = tv*2 + 1;
    add_to_node(l, tl, tm, lz_add[tv]);
    add_to_node(r, tm+1, tr, lz_add[tv]);
    lz_add[tv] = 0;
    update_node(l, mx[tv]);
    update_node(r, mx[tv]);
}

void update_min(int tv, int tl, int tr, int l, int r, long long val) {
    if(l > r || val >= mx[tv]) return; // break condition: val >= mx
    if(tl == l && tr == r && smx[tv] < val) { // tag condition : smx < val < mx
        update_node(tv, val);
        return;
    }
    else {
        push(tv, tl, tr);
        int tm = (tl + tr) >> 1;
        update_min(tv*2, tl, tm, l, min(tm, r), val);
        update_min(tv*2+1, tm+1, tr, max(tm+1, l), r, val);
        merge(tv);
    }
}

void update_add(int tv, int tl, int tr, int l, int r, long long val) {
    if(l > r) return;
    if(l == r) return new Node(a[l]);
```

```
if(tl == l && tr == r) {
    add_to_node(tv, tl, tr, val);
}
else {
    push(tv, tl, tr);
    int tm = (tl + tr) >> 1;
    update_add(tv*2, tl, tm, l, min(tm, r), val);
    update_add(tv*2+1, tm+1, tr, max(tm+1, l), r, val);
    merge(tv);
}

long long get_sum(int tv, int tl, int tr, int l, int r) {
    if(l > r) return 0;
    if(tl == l && tr == r) {
        return sum[tv];
    }
    else {
        push(tv, tl, tr);
        int tm = (tl + tr) >> 1;
        return get_sum(tv*2, tl, tm, l, min(tm, r)) + get_sum(tv*2+1, tm+1, tr, max(tm+1, l), r);
    }
}

SegtreeBeats(vector<long long> v) {
    n = v.size();
    sum = vector<long long>(4*n + 10);
    mx = vector<long long>(4*n + 10);
    smx = vector<long long>(4*n + 10);
    mx_c = vector<long long>(4*n + 10);
    lz_add = vector<long long>(4*n + 10);
    build(1, 1, n, v);
}

void upd_min(int l, int r, long long val) {
    update_min(1, 1, n, l, r, val);
}
long long get(int l, int r){
    return get_sum(1, 1, n, l, r);
}
void upd_add(int l, int r, long long val) {
    update_add(1, 1, n, l, r, val);
}
```

PersistentSegtree.h

Description: as it says.

ffaf19, 67 lines

```
//q(1) -> a[k][i] = x
//q(2) -> a[k][l] + ... + a[k][r]
//a(3) -> append(a[k]), n += 1
struct Node {
    ll val;
    Node *l, *r;
    Node(ll x) : val(x), l(nullptr), r(nullptr) {}
    Node(Node *ll, Node *rr) {
        l = ll, r = rr;
        val = 0;
        if (l) val += l->val;
        if (r) val += r->val;
    }
    Node(Node *cp) : val(cp->val), l(cp->l), r(cp->r) {}
};

int n, cnt = 1;
ll a[200001];
Node *roots[200001];

Node *build(int l = 1, int r = n) {
    if (l == r) return new Node(a[l]);
```

```

int mid = (l + r) / 2;
return new Node(build(l, mid), build(mid + 1, r));
}

Node *update(Node *node, int val, int pos, int l = 1, int r = n) {
    if (l == r) return new Node(val);
    int mid = (l + r) / 2;
    if (pos > mid) return new Node(node->l, update(node->r, val,
        pos, mid + 1, r));
    else return new Node(update(node->l, val, pos, l, mid), node-
        >r);
}

ll query(Node *node, int a, int b, int l = 1, int r = n) {
    if (l > b || r < a) return 0;
    if (l >= a && r <= b) return node->val;
    int mid = (l + r) / 2;
    return query(node->l, a, b, l, mid) + query(node->r, a, b,
        mid + 1, r);
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++) cin >> a[i];
    roots[cnt++] = build();

    while (q--) {
        int t;
        cin >> t;
        if (t == 1) {
            int k, i, x;
            cin >> k >> i >> x;
            roots[k] = update(roots[k], x, i);
        } else if (t == 2) {
            int k, l, r;
            cin >> k >> l >> r;
            cout << query(roots[k], l, r) << '\n';
        } else {
            int k;
            cin >> k;
            roots[cnt++] = new Node(roots[k]);
        }
    }
    return 0;
}

```

Matrix.h**Description:** Basic operations on square matrices.**Usage:** Matrix<int, 3> A;
A.d = {{1,2,3}, {4,5,6}, {7,8,9}};array<int, 3> vec = {1,2,3};
vec = (A^N) * vec;

6ab5db, 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;
    }
    array<T, N> operator*(const array<T, N>& vec) const {
        array<T, N> ret();
        rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    }
}

```

Matrix LineContainer Treap FenwickTree

```

    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;
}
};

LineContainer.h
Description: Container where you can add lines of the form  $kx+m$ , and query maximum values at points  $x$ . Useful for dynamic programming ("convex hull trick").
Usage: Max queries: lc.insert(m, c); result = lc.query(x)
Min queries: lc.insert(-m, -c); result = -lc.query(x)
Time:  $\mathcal{O}(\log N)$  sec1c7, 30 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;
    }
};

```

Treap.h**Description:** A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.**Time:** $\mathcal{O}(\log N)$ 1754b4, 53 lines

```

struct Node {
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int val) : val(val), y(rand()) {}
    void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

```

```

template<class F> void each(Node* n, F f) {
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};
    if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
        auto [L, R] = split(n->l, k);
        n->l = R;
        n->recalc();
        return {L, n};
    } else {
        auto [L, R] = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = L;
        n->recalc();
        return {n, R};
    }
}

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        return l->recalc(), l;
    } else {
        r->l = merge(l, r->l);
        return r->recalc(), r;
    }
}

Node* ins(Node* t, Node* n, int pos) {
    auto [l,r] = split(t, pos);
    return merge(merge(l, n), r);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
    Node *a, *b, *c;
    tie(a,b) = split(t, l); tie(b,c) = split(b, r - 1);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}

FenwickTree.h
Description: Computes partial sums  $a[0] + a[1] + \dots + a[pos - 1]$ , and updates single elements  $a[i]$ , taking the difference between the old and new value.
Time: Both operations are  $\mathcal{O}(\log N)$ . e62fac, 22 lines
struct FT {
    vector<ll> s;
    FT(int n) : s(n) {}
    void update(int pos, ll dif) { // a[pos] += dif
        for ( ; pos < sz(s); pos |= pos + 1) s[pos] += dif;
    }
    ll query(int pos) { // sum of values in [0, pos)
        ll res = 0;
        for ( ; pos > 0; pos &= pos - 1) res += s[pos-1];
        return res;
    }
    int lower_bound(ll sum) { // min pos st sum of [0, pos] >= sum
        // Returns n if no sum is >= sum, or -1 if empty sum is.
        if (sum <= 0) return -1;
        int pos = 0;
        for (int pw = 1 << 25; pw; pw >>= 1) {
            if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
                pos += pw, sum -= s[pos-1];
        }
    }
}

```

```
    return pos;
}
};
```

FenwickTree2d.h

Description: Computes sums $a[i,j]$ for all $i < I$, $j < J$, and increases single elements $a[i,j]$. Requires that the elements to be updated are known in advance (call `fakeUpdate()` before `init()`).

Time: $\mathcal{O}(\log^2 N)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

`"FenwickTree.h"` 157f07, 22 lines

```
struct FT2 {
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
    }
    void init() {
        for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
    }
    int ind(int x, int y) {
        return (int)(lower_bound(all(ys[x])), y) - ys[x].begin();
    }
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) {
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum;
    }
};
```

FenwickTree2dDense.h

Description: Same convention as above, just a bit better constant factor. Suitable for dense Grid. For the sparse one, n is around $1e9$

`"FenwickTree2dDense.h"` 1d1f25, 21 lines

```
struct FT2D {
    vector<vector<ll>> s;
    int n, m;
    FT2D(int r, int c) : n(r), m(c), s(r, vector<ll>(c, 0)) {}
    void update(int r, int c, ll dif) {
        for (int i = r; i < n; i |= i + 1) {
            for (int j = c; j < m; j |= j + 1) {
                s[i][j] += dif;
            }
        }
    }
    ll query(int r, int c) {
        ll res = 0;
        for (int i = r; i > 0; i &= i - 1) {
            for (int j = c; j > 0; j &= j - 1) {
                res += s[i-1][j-1];
            }
        }
        return res;
    }
};
```

SparseTable.h

Description: Gives range minimum in constant time

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

`"SparseTable.h"` 5d596c, 19 lines

```
struct SparseTable {
    vector<vector<int>> st;
    int n, K;
    SparseTable() {}
    SparseTable(vector<int>a) {
        n = a.size(); K = __lg(n) + 1;
```

```
        st = vector<vector<int>>(K, vector<int>(n));
        for(int i = 0; i < n; i++) st[0][i] = a[i];
        for (int i = 1; i < K; i++) {
            for (int j = 0; j + (1 << i) <= n; j++) {
                st[i][j] = min(st[i-1][j], st[i-1][j + (1 << (i-1))]);
            }
        }
        int get(int L, int R) {
            int i = __lg(R - L + 1);
            return min(st[i][L], st[i][R - (1 << i) + 1]);
        }
    };
}
```

MoSAlgoNew.h

Description: Just Mo's Algo. The current template considers 0-based indexing.

Time: $\mathcal{O}(N\sqrt{Q})$

`"MoSAlgoNew.h"` 9de03d, 24 lines

```
void remove(idx); void add(idx); int get_answer();
int block_size;
struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};
vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    int cur_l = 0, cur_r = -1;
    // invariant: data structure will always reflect the range
    [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {cur_l--; add(cur_l);}
        while (cur_r < q.r) {cur_r++; add(cur_r);}
        while (cur_l < q.l) {remove(cur_l); cur_l++;}
        while (cur_r > q.r) {remove(cur_r); cur_r--;}
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

Strings (7)

KMP.h

Description: $\text{pi}[x]$ computes the length of the longest prefix of s that ends at x , other than $s[0..x]$ itself (abacaba \rightarrow 0010123). Can be used to find all occurrences of a string.

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

`"KMP.h"` d4375c, 16 lines

```
vi pi(const string& s) {
    vi p(sz(s));
    rep(i, 1, sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}

vi match(const string& s, const string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p)-sz(s), sz(p))
        if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
```

```
    return res;
}
```

PrefixFuncAutomaton.h

Description: If the current prefix function value is i (i.e., you've matched the first i characters of string s), and you now append the character ' a ' + c , then: $\text{aut}[i][c]$ gives the new value of the prefix function for the updated string.

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

`"PrefixFuncAutomaton.h"` 23475a, 14 lines

```
void compute_automaton(string s, vector<vector<int>>& aut) {
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c == s[i])
                aut[i][c] = aut[pi[i-1]][c];
            else
                aut[i][c] = i + ('a' + c == s[i]);
        }
    }
}
```

Zfunc.h

Description: $z[i]$ computes the length of the longest common prefix of $s[i:]$ and s , except $z[0] = 0$. (abacaba \rightarrow 0010301)

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

`"Zfunc.h"` ee09e2, 12 lines

```
vi Z(const string& S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i, 1, sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - 1]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}
```

Manacher.h

Description: For each position in a string, computes $p[0][i] =$ half length of longest even palindrome around pos i , $p[1][i] =$ longest odd (half rounded down).

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

`"Manacher.h"` e7ad79, 13 lines

```
array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi, 2> p = {vi(n+1), vi(n)};
    rep(z, 0, 2) for (int i=0, l=0, r=0; i < n; i++) {
        int t = r-i+1;
        if (i < r) p[z][i] = min(t, p[z][1+t]);
        int L = i-p[z][i], R = i+p[z][i]-t;
        while (L>=1 && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin() + minRotation(v), v.end());`

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

`"MinRotation.h"` d07a42, 8 lines

```
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;
}

```

SuffixArray.h

Description: Builds suffix array for a string. $sa[i]$ is the starting index of the suffix which is i 'th in the sorted suffix array. The returned vector is of size $n + 1$, and $sa[0] = n$. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: $lcp[i] = lcp(sa[i], sa[i-1])$, $lcp[0] = 0$. The input string must not contain any nul chars. After an iteration, $x[i]$ contains the rank of the suffix $[i..n-1]$ $\text{Comp}()$ compares two ranges, returns -1 if a is lexicographically smaller than b , 0 if equal, 1 otherwise

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

44ddf3, 49 lines

```

struct SuffixArray {
    vi sa, lcp, rank;
    SparseTable stable;
    string _s;
    SuffixArray() {}
    SuffixArray(string s, int lim=256) { // or vector<int>
        _s = s;
        s.push_back(0); int n = sz(s), k = 0, a, b;
        vi x(all(s)), y(n), ws(max(n, lim));
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim =
            p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--; ) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p
                ++;
        }
        for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
        for (k && k--, j = sa[x[i] - 1];
            s[i + k] == s[j + k]; k++);
        stable = SparseTable(lcp);
        rank = x;
    }
    int comp(pair<int,int>a, pair<int,int>b) {
        int pos1 = rank[a.first], pos2 = rank[b.first];
        if(pos1 == pos2) {
            if(a.second == b.second) return 0;
            else if(a.second < b.second) return -1;
            return 1;
        }
        int len = stable.get(min(pos1, pos2)+1, max(pos1, pos2));
        len = min(len, min(a.second-a.first+1, b.second-b.first
            +1));
        if(len == a.second-a.first + 1) {
            if(b.second-b.first == a.second-a.first) return 0;
            return -1;
        }
        if(len == b.second-b.first + 1) {
            if(b.second-b.first == a.second-a.first) return 0;
            return 1;
        }
        if(s[a.first + len] < s[b.first + len]) return -1;
    }
}

```

```

        return 1;
    }
}

```

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+1]=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]-1[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 il, int i2, int olen) {
        if (l[node] <= il && il < 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], il, 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;
    }
}

```

HashingNew.h

Description: as it says.

7bca18, 91 lines

```

long long binpow(long long a, long long b, long long m) {
    a %= m;
}

```

```

long long res = 1;
while (b > 0) {
    if (b & 1)
        res = res * a % m;
    a = a * a % m;
    b >>= 1;
}
return res;
}

```

```

ll inv(ll x, ll m) {
    return binpow(x, m-2, m);
}

```

```

typedef pair<ll, ll> pll;
ostream& operator<<(ostream& os, pll hash) {
    return os<<"(" <<hash.first<<, " <<hash.second<<")";
}
pll operator + (const pll &a, const pll &b) {
    return {a.first + b.first, a.second + b.second};
}
pll operator - (const pll &a, const pll &b) {
    return {a.first - b.first, a.second - b.second};
}
pll operator * (const pll &a, const pll &b) {
    return {a.first * b.first, a.second * b.second};
}
pll operator % (const pll &a, const pll &b) {
    return {a.first % b.first, a.second % b.second};
}
pll operator * (const pll &a, const ll b) {
    return {a.first * b, a.second * b};
}
const pll _primes={773,709};
const pll _mods = {281559881,398805713};
const int MAXLEN = 1e6 + 10;
pll p_pow[MAXLEN], pinv_pow[MAXLEN];
pll inv(pll x) {
    return {inv(x.first, _mods.first), inv(x.second, _mods.second)};
}
void calc_pow() {
    p_pow[0] = {1, 1};
    for (int j = 1; j < MAXLEN; j++) {
        p_pow[j] = (p_pow[j-1] * _primes) % _mods;
    }
    pinv_pow[0] = {1, 1};
    pinv_pow[1] = inv(_primes);
    for (int j = 2; j < MAXLEN; j++) {
        pinv_pow[j] = (pinv_pow[j-1] * pinv_pow[1]) % _mods;
    }
}

```

```

class hashing {
    public:
        int n, limit;
        string s;
        vector<pll>pref, suff;
        hashing() {}
        hashing(int _n, string _s) {
            this->s = _s; //1 indexed
            this->n = _n;
            pref.resize(n+2);
            suff.resize(n+2);
            precompute();
        }
        void precompute() {
            pll hash_value = {0, 0};
            pref[0] = {0, 0};
            suff[0] = {0, 0};
            for (int i = 1; i <= n; i++) {

```

```

                a % m;
            }
        }
    }
}

```

```

        hash_value = (hash_value + p_pow[i-1] * (s[i]-a'+1)) % _mods;
        pref[i] = hash_value;
    }
    suff[n+1] = {0, 0};
    hash_value = {0, 0};
    for(int i = n; i >= 1; i--) {
        hash_value = (hash_value + p_pow[n-i] * (s[i]-a'+1)) % _mods;
        suff[i] = hash_value;
    }
}
pll get_pref(int l, int r) {
    return ((pref[r]-pref[l-1]+_mods) * pinv_pow[l-1]) % _mods;
}
pll get_suff(int l, int r) {
    return ((suff[l]-suff[r+1]+_mods) * pinv_pow[n-r]) % _mods;
}
pll get(int l, int r){
    return get_pref(l, r);
}
};

AhoCorasickNew.h
Description: as it says.
e878da, 47 lines

```

```

struct AC {
    int N, P;
    const int A = 26;
    vector<vector<int>> next;
    vector<int> link, out_link; // out_link[v] = nearest
        ancestor of v where an input pattern ended which is also
        a suffix link of v.
    vector<vector<int>> out;
    AC(): N(0), P(0) {node();}
    int node() {
        next.emplace_back(A, 0);
        link.emplace_back(0);
        out_link.emplace_back(0);
        out.emplace_back(0);
        return N++;
    }
    inline int get (char c) {
        return c - 'a';
    }
    int add_pattern_(const string T) {
        int u = 0;
        for (auto c : T) {
            if (!next[u][get(c)]) next[u][get(c)] = node();
            u = next[u][get(c)];
        }
        out[u].push_back(P);
        return P++;
    }
    void compute() {
        queue<int> q;
        for (q.push(0); !q.empty();) {
            int u = q.front(); q.pop();
            for (int c = 0; c < A; ++c) {
                int v = next[u][c];
                if (!v) next[u][c] = next[link[u]][c];
                else {
                    link[v] = u ? next[link[u]][c] : 0;
                    out_link[v] = out[link[v]].empty() ? out_link[link[v]] : link[v];
                    q.push(v);
                }
            }
        }
    }
};

AhoCorasickNew.h
Description: as it says.
e878da, 47 lines

```

```

        }
    }
    int advance (int u, char c) {
        while (u && !next[u][get(c)]) u = link[u];
        u = next[u][get(c)];
        return u;
    }
};

SegmentIntersection.h

```

Geometry (8)

8.1 Geometric primitives

Point.h

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

Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
"Point.h"

5c88f4, 6 lines

```

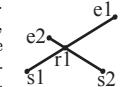
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
    return ((p-s)*d-(e-s)*t).dist()/d;
}

```

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

```

```

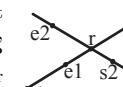
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b *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

```

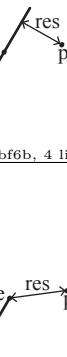
```

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

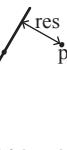
Description: Returns where p is as seen from s towards e. 1/0/-1 \leftrightarrow left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.



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"

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: bool left = sideOf(p1,p2,q)==1;
"Point.h"
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].t180() ...}; // 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
of0602, 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()));
}

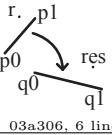
```
Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

8.2 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 (dr == 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: $\mathcal{O}(n)$

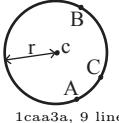
```
"/.../content/geometry/Point.h"
19add1, 19 lines
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
        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 = q + d * (t-1);
    };
    auto sum = 0.0;
    rep(i,0,ps.size())
        sum += tri(ps[i] - c, ps[(i + 1) % ps.size()] - c);
    return sum;
}
```

```
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};
}
```

8.3 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

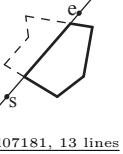
Description:

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" d07181, 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();
        auto a = s.cross(e, cur), b = s.cross(e, prev);
        if ((a < 0) != (b < 0))
            res.push_back(cur + (prev - cur) * (a / (a - b)));
        if (a < 0)
            res.push_back(cur);
    }
    return res;
}
```

ConvexHull.h

Description:

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

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

"Point.h" 310954, 13 lines



```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
    if (sz(pts) <= 1) return pts;
    sort(all(pts));
    vector<P> h(sz(pts)+1);
    int s = 0, t = 0;
    for (int it = 2; it-->0; s = --t, reverse(all(pts)))
        for (P p : pts) {
            while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
            h[t++] = p;
        }
    return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

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

```
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;
}
```

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

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;
}
```

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: • (-1, -1) if no collision, • (i, -1) if touching the corner i, • (i, i) if along side (i, i+1), • (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 the line.

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

"point.h" 7cf45b, 39 lines

```
#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;
}
#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;
}
```

MinkowskiSum.h

Description: as it says.

df6c11, 34 lines

```
typedef Point<ll> P;
P dir;
bool half(P p) {
    return dir.cross(p) < 0 || (dir.cross(p) == 0 && dir.dot(p) > 0);
}

bool polarComp(P p, P q) {
    return make_tuple(half(p), 0) < make_tuple(half(q), p.cross(q));
}

void process(vector<P> &P) {
    int mnid = 0;
    for (int i = 0; i < P.size(); i++)
        if (P[i] < P[mnid])
            mnid = i;
    rotate(P.begin(), P.begin() + mnid, P.end());
}

vector<P> MinkowskiSum(vector<P> A, vector<P> B) {
    process(A);
    process(B);

    int n = A.size(), m = B.size();
    vector<P> PP(n), QQ(m);
    for (int i = 0; i < n; i++) PP[i] = A[(i + 1) % n] - A[i];
    for (int i = 0; i < m; i++) QQ[i] = B[(i + 1) % m] - B[i];

    dir = P(0, -1);
    vector<P> C(n + m + 1);
    merge(PP.begin(), PP.end(), QQ.begin(), QQ.end(), C.begin() + 1, polarComp);
    C[0] = A[0] + B[0];

    for (int i = 1; i < C.size(); i++)
        C[i] = C[i] + C[i - 1];
    C.pop_back();
    return C;
}
```

8.4 Misc. Point Set Problems**ClosestPair.h**

Description: Finds the closest pair of points.

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

"Point.h" ac41a6, 17 lines

```

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;
}

```

8.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

```

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

```

Point3D.h

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

8058ae, 32 lines

```

template<class T> struct Point3D {
    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); }
    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*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

```

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; };

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);
        }
    }
}

```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) $f_1(\phi_1)$ and $f_2(\phi_2)$ from x axis and zenith angles (latitude) $t_1(\theta_1)$ and $t_2(\theta_2)$ 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. $dx \cdot \text{radius}$ is then the difference between the two points in the x direction and $d \cdot \text{radius}$ 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 (9)

9.1 Benq's Macros

Benq.h

Description: My custom header file with helper functions. 6b0ad7, 9 lines

```

#define vi vector<int>
#define f first
#define s second
#define pb push_back
#define all(v) (v).begin(), (v).end()
#define rsz(x) resize(x)
#define sz(x) (x).size()
template<class T> using V = vector<T>;
template<class T> bool ckmin(T& a, const T& b) { return a > b ? ((a = b), true) : false; }

```

9.2 Network flow

MinCostMaxFlow.h

Description: Minimum-cost maximum flow, assumes no negative cycles. It is possible to choose negative edge costs such that the first run of Dijkstra is slow, but this hasn't been an issue in the past. Edge weights ≥ 0 for every subsequent run. To get flow through original edges, assign ID's during ae. Ignoring first run of Dijkstra, $O(FM \log M)$ if caps are integers and F is max flow.

072c1b, 40 lines

```

struct MCMF {
    using F = ll; using C = ll; // flow type, cost type
    struct Edge { int to; F flo, cap; C cost; };
    int N; V<C> p, dist; vi pre; V<Edge> eds; V<vi> adj;
    void init(int _N) { N = _N;
        p.rsz(N), dist.rsz(N), adj.rsz(N); }
    void ae(int u, int v, F cap, C cost) { assert(cap >= 0);
        adj[u].pb(sz(eds)); eds.pb({v,0,cap,cost});
        adj[v].pb(sz(eds)); eds.pb({u,0,0,-cost}); }
    // use asserts, don't try smart dumb
    bool path(int s, int t) { // find lowest cost path to send
        flow through
        const C inf = numeric_limits<C>::max(); FOR(i,N) dist[i] =
            inf;
        using T = pair<C,int>; priority_queue<T,vector<T>,greater<T>> todo;
        todo.push({dist[s] = 0,s});
        while (sz(todo)) { // Dijkstra
            T x = todo.top(); todo.pop(); if (x.f > dist[x.s])
                continue;
            each(e,adj[x.s]) { const Edge& E = eds[e]; // all weights
                should be non-negative
                if (E.flo < E.cap && ckmin(dist[E.to],x.f+E.cost+p[x.s]
                    -p[E.to]))
                    pre[E.to] = e, todo.push({dist[E.to],E.to});
            }
        } // if costs are doubles, add some EPS so you
        // don't traverse ~0-weight cycle repeatedly
        return dist[t] != inf; // return flow
    }
    pair<F,C> calc(int s, int t) { assert(s != t);
        FOR(_,_N) FOR(e,sz(eds)) { const Edge& E = eds[e]; // Bellman-Ford
            if (E.cap) ckmin(p[E.to],p[eds[e^1].to]+E.cost); }
        F totFlow = 0; C totCost = 0;
    }
}

```

```

while (path(s,t)) { // p -> potentials for Dijkstra
    FOR(i,N) p[i] += dist[i]; // don't matter for unreachable
    nodes
    F df = numeric_limits<F>::max();
    for (int x = t; x != s; x = eds[pre[x]^1].to) {
        const Edge& E = eds[pre[x]]; ckmin(df,E.cap-E.flo);
        totFlow += df; totCost += (p[t]-p[s])*df;
        for (int x = t; x != s; x = eds[pre[x]^1].to)
            eds[pre[x]].flo += df, eds[pre[x]^1].flo -= df;
    } // get max flow you can send along path
    return {totFlow,totCost};
}

```

Dinic.h

Description: Fast flow. After computing flow, edges $\{u, v\}$ such that $lev[u] \neq -1$, $lev[v] = -1$ are part of min cut. $O(N^2M)$ flow, $O(M\sqrt{N})$ bipartite matching

907d26, 38 lines

```

struct Dinic {
    using F = ll; // flow type
    struct Edge { int to; F flo, cap, id; };
    int N; V<Edge> eds; V<vi> adj;
    void init(int _N) { N = _N; adj.rsz(N), cur.rsz(N); }
    void ae(int u, int v, F cap, int id, F rcap = 0) { assert(min(
        (cap,rcap) >= 0);
        adj[u].pb(sz(eds)); eds.pb({v,0,cap,id});
        adj[v].pb(sz(eds)); eds.pb({u,0,rcap, -1});
    }
    vi lev; V<vi::iterator> cur;
    bool bfs(int s, int t) { // level = shortest distance from
        source
        lev = vi(N,-1); FOR(i,N) cur[i] = begin(adj[i]);
        queue<int> q({s}); lev[s] = 0;
        while (sz(q)) { int u = q.front(); q.pop();
            trav(e,adj[u]) { const Edge& E = eds[e];
                int v = E.to; if (lev[v] < 0 && E.flo < E.cap)
                    q.push(v), lev[v] = lev[u]+1;
            }
        }
        return lev[t] >= 0;
    }
    F dfs(int v, int t, F flo) {
        if (v == t) return flo;
        for (; cur[v] != end(adj[v]); cur[v]++) {
            Edge& E = eds[*cur[v]];
            if (lev[E.to]==lev[v]+1||E.flo==E.cap) continue;
            F df = dfs(E.to,t,min(flo,E.cap-E.flo));
            if (df) { E.flo += df; eds[*cur[v]^1].flo -= df;
                return df; } // saturated >=1 one edge
        }
        return 0;
    }
    F maxFlow(int s, int t) {
        F tot = 0; while (bfs(s,t)) while (F df =
            dfs(s,t,numeric_limits<F>::max())) tot += df;
        return tot;
    }
}

```

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) { // O(V^2) -> 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;
}

```

9.3 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.

Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);

Time: $\mathcal{O}(\sqrt{VE})$

```

bool dfs(int a, int L, vector<vi> g, vi& btoa, vi& A, vi& B) {
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}

```

```

int hopcroftKarp(vector<vi> g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (; ) {
        fill(all(A), 0);
        fill(all(B), 0);
        cur.clear();
        for (int a : btoa) if(a != -1) A[a] = -1;
        rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
        for (int lay = 1; lay++) {
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                    next.push_back(btoa[b]);
                }
            }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay;
            cur.swap(next);
        }
        rep(a,0,sz(g))
            res += dfs(a, 0, g, btoa, A, B);
    }
}

```

```

}
DFSMatching.h
Description: Simple bipartite matching algorithm. Graph  $g$  should be a list of neighbors of the left partition, and  $btoa$  should be a vector full of -1's of the same size as the right partition. Returns the size of the matching.  $btoa[i]$  will be the match for vertex  $i$  on the right side, or -1 if it's not matched.
Usage: vi btoa(m, -1); dfsMatching(g, btoa);
Time:  $\mathcal{O}(VE)$ 

```

522b98, 22 lines

```

bool find(int j, vector<vi> g, vi& btoa, vi& vis) {
    if (btoa[j] == -1) return 1;
    vis[j] = 1; int di = btoa[j];
    for (int e : g[di])
        if (!vis[e] && find(e, g, btoa, vis)) {
            btoa[e] = di;
            return 1;
        }
    return 0;
}
int dfsMatching(vector<vi> g, vi& btoa) {
    vi vis;
    rep(i,0,sz(g)) {
        vis.assign(sz(btoa), 0);
        for (int j : g[i])
            if (find(j, g, btoa, vis)) {
                btoa[j] = i;
                break;
            }
    }
    return sz(btoa) - (int)count(all(btoa), -1);
}

```

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.

```

"DFSMatching.h"
vi cover(vector<vi> g, int n, int m) {
    vi match(m, -1);
    int res = dfsMatching(g, match);
    vector<bool> lfound(n, true), seen(m);
    for (int it : match) if (it != -1) lfound[it] = false;
    vi q, cover;
    rep(i,0,n) if (lfound[i]) q.push_back(i);
    while (!q.empty()) {
        int i = q.back(); q.pop_back();
        lfound[i] = 1;
        for (int e : g[i]) if (!seen[e] && match[e] != -1) {
            seen[e] = true;
            q.push_back(match[e]);
        }
    }
    rep(i,0,n) if (!lfound[i]) cover.push_back(i);
    rep(i,0,m) if (seen[i]) cover.push_back(n+i);
    assert(sz(cover) == res);
    return cover;
}

```

Hungarian.h

Description: Given J jobs and W workers ($J \leq W$), computes the minimum cost to assign each prefix of jobs to distinct workers. T must be a type large enough to represent integers on the order of $J * \max(\text{abs}(C))$ C is a matrix of dimensions $J \times W$ such that $C[j][w] = \text{cost to assign j-th job to w-th worker (possibly negative)}$ vector<T> answer = a vector of length J , with the j -th entry equaling the minimum cost to assign the first $(j+1)$ jobs to distinct workers

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

2f0920, 37 lines

```

template<class T> using V = vector<T>;
template <class T> vector<T> hungarian(const vector<vector<T>>
    &C) {
const int J = (int)size(C), W = (int)size(C[0]);
assert(J <= W);
vector<T> job(W + 1, -1);
vector<T> ys(J), yt(W + 1);
vector<T> answers;
const T inf = numeric_limits<T>::max();
for (int j_cur = 0; j_cur < J; ++j_cur) {
    int w_cur = W;
    job[w_cur] = j_cur;
    vector<T> min_to(W + 1, inf);
    vector<int> prv(W + 1, -1);
    vector<bool> in_Z(W + 1);
    while (job[w_cur] != -1) {
        in_Z[w_cur] = true;
        const int j = job[w_cur];
        T delta = inf;
        int w_next;
        for (int w = 0; w < W; ++w) {
            if (!in_Z[w]) {
                if (ckmin(min_to[w], C[j][w] - ys[j] - yt[w]))
                    prv[w] = w_cur;
                if (ckmin(delta, min_to[w])) w_next = w;
            }
        }
        for (int w = 0; w <= W; ++w) {
            if (in_Z[w]) ys[job[w]] += delta, yt[w] -= delta;
            else min_to[w] -= delta;
        }
        w_cur = w_next;
    }
    for (int w; w_cur != -1; w_cur = w) job[w_cur] = job[w =
        prv[w_cur]];
    answers.push_back(-yt[W]);
}
return job;
}

```

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod .
Time: $\mathcal{O}(N^3)$

```

.../numerical/MatrixInverse-mod.h"                                cb1912, 40 lines
vector<pii> generalMatching(int N, vector<pii>& ed) {
    vector<vector<ll>> mat(N, vector<ll>(N)), A;
    for (pii pa : ed) {
        int a = pa.first, b = pa.second, r = rand() % mod;
        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
    }

    int r = matInv(A = mat), M = 2*N - r, fi, fj;
    assert(r % 2 == 0);

    if (M != N) do {
        mat.resize(M, vector<ll>(M));
        rep(i, 0, N) {
            mat[i].resize(M);
            rep(j, N, M) {
                int r = rand() % mod;
                mat[i][j] = r, mat[j][i] = (mod - r) % mod;
            }
        }
    } while (matInv(A = mat) != M);

    vi has(M, 1); vector<pii> ret;
    rep(it, 0, M/2) {
        rep(i, 0, M) if (has[i])

```

GeneralMatching SCC 2-edge-cc 2-bcc

```

        rep(j, i+1, M) if (A[i][j] && mat[i][j]) {
            fi = i; fj = j; goto done;
        } assert(0); done:
        if (fj < N) ret.emplace_back(fi, fj);
        has[fi] = has[fj] = 0;
        rep(sw, 0, 2) {
            ll a = modpow(A[fi][fj], mod-2);
            rep(i, 0, M) if (has[i] && A[i][fj]) {
                ll b = A[i][fj] * a % mod;
                rep(j, 0, M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
            }
            swap(fi, fj);
        }
    }
    return ret;
}

```

9.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.

Time: $\mathcal{O}(E + V)$ 76b5c9, 24 lines

```

vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for (auto e : g[j]) if (comp[e] < 0)
        low = min(low, val[e] ?: dfs(e, g, f));

    if (low == val[j]) {
        do {
            x = z.back(); z.pop_back();
            comp[x] = ncomps;
            cont.push_back(x);
        } while (x != j);
        f(cont); cont.clear();
        ncomps++;
    }
    return val[j] = low;
}

template<class G, class F> void scc(G& g, F f) {
    int n = sz(g);
    val.assign(n, 0); comp.assign(n, -1);
    Time = ncomps = 0;
    rep(i, 0, n) if (comp[i] < 0) dfs(i, g, f);
}

```

2-edge-cc.h

Description: Finds 2-Edge-CC and bridges

```

int timer, scc; // Number of strongly connected components
int id[MAX_N];
int low[MAX_N]; // Lowest ID in node's subtree in DFS tree
vector<int> neighbors[MAX_N];
vector<int> two_edge_components[MAX_N];
stack<int> st; // Keeps track of the path in our DFS

void dfs(int node, int parent = -1) {
    id[node] = low[node] = ++timer;
    st.push(node);
    bool multiple_edges = false;
    for (int child : neighbors[node]) {
        if (child == parent && !multiple_edges) {

```

```

            multiple_edges = true;
            continue;
        }
        if (!id[child]) {
            dfs(child, node);
            low[node] = min(low[node], low[child]);
        } else {
            low[node] = min(low[node], id[child]);
        }
    }
    if (low[node] == id[node]) {
        // if(parent != -1) { child, parent} is a bridge
        while (st.top() != node) {
            two_edge_components[scc].push_back(st.top());
            st.pop();
        }
        two_edge_components[scc++].push_back(st.top());
        st.pop();
    }
}

```

2-bcc.h

Description: Finds 2-BCC and Articulation Points

b4a090, 68 lines

```

/*no mult edge*/
vector<vector<int>> block_cut_tree(
    vector<vector<int>> &g,
    vector<bool> &is_cutpoint, vector<int> &id) {
    int n = (int)g.size();

    vector<vector<int>> comps; // stores bccs
    vector<int> stk;
    vector<int> num(n);
    vector<int> low(n);

    is_cutpoint.resize(n);

    // Finds the biconnected components
    function<void(int, int, int &)> dfs =
        [&](int node, int parent, int &timer) {
        num[node] = low[node] = ++timer;
        stk.push_back(node);
        for (int son : g[node]) {
            if (son == parent) { continue; }
            if (num[son]) {
                low[node] = min(low[node], num[son]);
            } else {
                dfs(son, node, timer);
                low[node] = min(low[node], low[son]);
            }
            if (low[son] >= num[node]) {
                is_cutpoint[node] = (num[node] > 1 || num[son] > 2);
                comps.push_back({node});
            }
            while (comps.back().back() != son) {
                comps.back().push_back(stk.back());
                stk.pop_back();
            }
        }
    };
    int timer = 0;
    dfs(0, -1, timer);
    id.resize(n);

    // Build the block-cut tree
    function<vector<vector<int>>()> build_tree = [&]() {
        vector<vector<int>> t(1);
        int node_id = 0;

```

```

for (int node = 0; node < n; node++) {
    if (is_cutpoint[node]) {
        id[node] = node_id++;
        t.push_back({});
    }
}

for (auto &comp : comps) {
    int node = node_id++;
    t.push_back({});
    for (int u : comp)
        if (!is_cutpoint[u])
            id[u] = node;
        else {
            t[node].push_back(id[u]);
            t[id[u]].push_back(node);
        }
}
return t;
};

return build_tree();
}

```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c, \dots to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!\!a||c)\&\&(d||!\!b)\&\&\dots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ($\sim x$).

Usage: TwoSat ts(number of boolean variables);

ts.either(0, ~3); // Var 0 is true or var 3 is false

ts.setValue(2); // Var 2 is true

ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true

ts.solve(); // Returns true iff it is solvable

ts.values[0..N-1] holds the assigned values to the vars

Time: $\mathcal{O}(N + E)$, where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

```

struct TwoSat {
    int N;
    vector<vi> gr;
    vi values; // 0 = false, 1 = true

```

TwoSat(int n = 0) : N(n), gr(2*n) {}

```

int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
}

```

```

void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
}

```

void setValue(int x) { either(x, x); }

```

void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i, 2, sz(li)) {

```

```

        int next = addVar();
        either(cur, ~li[i]);
        either(cur, next);
        either(~li[i], next);
        cur = ~next;
    }
}

```

2sat EulerWalk Offline-deletion

```

        either(cur, ~li[1]);
    }

    vi val, comp, z; int time = 0;
    int dfs(int i) {
        int low = val[i] = ++time, x; z.push_back(i);
        for (int e : gr[i]) if (!comp[e])
            low = min(low, val[e] ?: dfs(e));
        if (low == val[i]) do {
            x = z.back(); z.pop_back();
            comp[x] = low;
            if (values[x>>1] == -1)
                values[x>>1] = x&1;
        } while (x != i);
        return val[i] = low;
    }

    bool solve() {
        values.assign(N, -1);
        val.assign(2*N, 0); comp = val;
        rep(i, 0, 2*N) if (!comp[i]) dfs(i);
        rep(i, 0, N) if (comp[2*i] == comp[2*i+1]) return 0;
        return 1;
    }
};

```

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: $\mathcal{O}(V + E)$

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()};
    }
}

```

Offline-deletion.h

Description: Solution to CF813-F

19ca5f, 111 lines

```

struct DSU {
    std::vector<std::pair<int &, int>> his;

    int n;
    std::vector<int> f, g, bip;

    DSU(int n_) : n(n_), f(n, -1), g(n), bip(n, 1) {}

    std::pair<int, int> find(int x) {
        if (f[x] < 0) {
            return {x, 0};
        }
        auto [u, v] = find(f[x]);
        return {u, v ^ g[x]};
    }
};

```

```

void set(int &a, int b) {
    his.emplace_back(a, a);
    a = b;
}

void merge(int a, int b, int &ans, int &comp) {
    auto [u, xa] = find(a);
    auto [v, xb] = find(b);
    int w = xa ^ xb ^ 1;
    if (u == v) {
        if (bip[u] && w) {
            set(bip[u], 0);
            ans--;
        }
        return;
    }
    if (f[u] > f[v]) {
        std::swap(u, v);
    }
    comp--;
    ans -= bip[u];
    ans -= bip[v];
    set(bip[u], bip[u] && bip[v]);
    set(f[u], f[u] + f[v]);
    set(f[v], u);
    set(g[v], w);
    ans += bip[u];
}

int timeStamp() {
    return his.size();
}

void rollback(int t) {
    while (his.size() > t) {
        auto [x, y] = his.back();
        x = y;
        his.pop_back();
    }
}

int main()
{
    int n, m; cin >> n >> m;
    map<array<int, 2>, int> mp;
    const int N = (1<<(_lg(m)+2));
    vector<vector<array<int, 2>>> ed(N);
    auto add = [&] (auto add, int p, int l, int r, int x, int y,
                   array<int, 2>e) -> void {
        if (l >= y || r <= x) return;
        if (l >= x && r <= y) {
            ed[p].emplace_back(e);
            return;
        }
        int m = (l+r)>>1;
        add(add, (p<<1), l, m, x, y, e);
        add(add, (p<<1)^1, m, r, x, y, e);
    };
    FOR(i, m) {
        int u, v; cin >> u >> v;
        u--; v--;
        if (mp.find({u, v}) == mp.end()) {
            mp[{u, v}] = i;
        }
        else {
            add(add, 1, 0, m, mp[{u, v}], i, {u, v});
            mp.erase({u, v});
        }
    }
}

```



```

tie(a, b) = minmax(time[a], time[b]);
return path[rmq.query(a, b)];
}
//dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};

```

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

"LCA.h" 9775a0, 21 lines

```

typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i, 0, m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i, 0, sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i, 0, sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
}

```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most $\log(n)$ light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_IN_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.

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

"../data-structures/LazySegmentTree.h" 0629cf, 61 lines

```

template<bool VALS_IN_EDGES> struct HLD {
    int N; vector<vi> adj;
    vi par, root, depth, sz, pos;
    int ti; vi rpos;
    lazy_segtree<S, op, e, F,
    mapping, composition, id> tree;
    HLD(int N) : N(N), adj(N), par(N), root(N),
    depth(N), sz(N), pos(N), tree(N) {}
    void ae(int x, int y) { adj[x].pb(y), adj[y].pb(x); }
    void dfsSz(int x) {
        sz[x] = 1;
        trav(y, adj[x]) {
            par[y] = x; depth[y] = depth[x]+1;
            adj[y].erase(find(all(adj[y]), x));
            dfsSz(y); sz[x] += sz[y];
            if (sz[y] > sz[adj[x][0]]) swap(y, adj[x][0]);
        }
    }
    void dfsHld(int x) {
        pos[x] = ti++; rpos.pb(x);
        trav(y, adj[x]) {
            root[y] = (y == adj[x][0] ? root[x] : y);
            dfsHld(y);
        }
    }
}

```

CompressTree HLD LinkCutTree DsuOnTrees

```

}
void init(vector<tuple<int, int, int>> ed, int R = 0) {
    // ed is empty if the edge has
    // no initial weight
    // for node: tree.set(pos[node], val)
    par[R] = depth[R] = ti = 0; dfsSz(R);
    root[R] = R; dfsHld(R);
    for(auto [u, v, c] : ed) {
        int idx=-1;
        if(u==par[v]) idx=pos[v];
        else idx=pos[u];
        tree.set(idx, {c, 1});
    }
}
int lca(int x, int y) {
    for (; root[x] != root[y]; y = par[root[y]])
        if (depth[root[x]] > depth[root[y]]) swap(x,y);
    return depth[x] < depth[y] ? x : y;
}
template <class BinaryOp>
void processPath(int x, int y, BinaryOp op) {
    for (; root[x] != root[y]; y = par[root[y]]) {
        if (depth[root[x]] > depth[root[y]]) swap(x,y);
        op(pos[root[y]], pos[y]);
        if (depth[x] > depth[y]) swap(x,y);
        op(pos[x]+VALS_IN_EDGES, pos[y]);
    }
}
void modifyPath(int x, int y, int v) {
    processPath(x, y, [this, &v](int l, int r) {
        tree.apply(l, r+1, F(v)); });
}
mint queryPath(int x, int y) {
    mint res = e() .x;
    processPath(x, y, [this, &res](int l, int r) {
        res += tree.prod(l, r+1) .x; });
    return res;
}
void modifySubtree(int x, int v) {
    tree.apply(pos[x]+VALS_IN_EDGES, pos[x]+sz[x]-1, F(v));
}

```

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 $\mathcal{O}(\log N)$.

```

0fb462, 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 == 1 ? 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];
            y->c[h ^ 1] = x;
        }
    }
}

```

```

}
z->c[i ^ 1] = this;
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;
    }
};

```

DsOnTrees.h

Description: as it says.

26dcba, 22 lines

```
//Let st[v] dfs starting time of vertex v, ft[v] be it's
finishing time and ver[time] is the vertex for which it's
starting time is equal to time.
int cnt[maxn];
void dfs(int v, int p, bool keep) {
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
        if(u != p && sz[u] > mx)
            mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
            dfs(u, v, 0); // run a dfs on small childs and
                           // clear them from cnt
    if(bigChild != -1)
        dfs(bigChild, v, 1); // bigChild marked as big and not
                           // cleared from cnt
    for(auto u : g[v])
        if(u != p && u != bigChild)
            for(int p = st[u]; p < ft[u]; p++)
                cnt[col[ver[p]]]++;
    cnt[col[v]]++;
    //now cnt[c] is the number of vertices in subtree of vertex
    //v that has color c. You can answer the queries easily
}
if(keep == 0)
    for(int p = st[v]; p < ft[v]; p++)
        cnt[col[ver[p]]]--;
}
```

9.8 Divide and Conquer

CD.h

Description: O(NlogN) decomposition of a tree

b0ca4b, 46 lines

```
vector<bool> used(n);
vector<int> sz(n);
auto cd = [&] (auto cd, int v) -> void {
    auto getCentroid = [&] () {
        auto dfs = [&] (auto f, int v, int p=-1) -> int {
            sz[v] = 1;
            for (int u : adj[v]) {
                if (u == p || used[u]) continue;
                sz[v] += f(f, u, v);
            }
            return sz[v];
        };
        int tot = dfs(dfs, v), c = -1;
        auto dfs2 = [&] (auto f, int v, int p=-1) -> void {
            bool ok = (tot-sz[v])*2 <= tot;
            for (int u : adj[v]) {
                if (u == p || used[u]) continue;
                if (sz[u]*2 > tot) ok = false;
            }
            if (ok) c = v;
        };
        dfs2(dfs2, v);
        return c;
    };
    int c = getCentroid();
    // take care of centroid here
    for (int u : adj[c]) {
        if (used[u]) continue;
        auto dfs = [&] (auto f, int v, int p=-1, int dep=1) ->
            void {

```

CD DNC-DSU AuxiliaryTree IntervalContainer

```
        for (int u : adj[v]) {
            if (u == p || used[u]) continue;
            f(f, u, v, dep+1);
        }
        dfs(dfs, u);
    }
    for (int u : adj[c]) {
        if (used[u]) continue;
        cd(cd, u);
    }
};
```

DNC-DSU.h

Description: As name suggests

9de039, 43 lines

```
vector<set<int>> ap;
// (id, l, r)
auto dnc = [&] (auto dnc, int x, int l, int r) -> void {
    if(r-l<1) return;
    ap.push_back({}); // m is the partition point
    auto [val, m] = ST.prod(l, r);
    int c1 = ap.size();
    dnc(dnc, ap.size(), l, m);
    int c2 = ap.size();
    dnc(dnc, ap.size(), m+1, r);
    int c3 = ap.size();
    if(c1==c2 && c2==c3) {
        ap[x].insert(p[l]);
    }
    else if(c1==c2) {
        swap(ap[x], ap[c2]);
        ap[x].insert(p[m]);
    }
    else if(c2==c3) {
        swap(ap[x], ap[c1]);
        ap[x].insert(p[m]);
    }
    else {
        swap(ap[x], ap[c2]);
        pair<int, int> to_src = {l, m};
        if(m-1 > r-(m+1)) {
            swap(ap[x], ap[c1]);
            to_src = {m+1, r};
        }
        FOR(i, to_src.f, to_src.s-1) {
            // do processing first
            ans += ap[x].count(p[m]-p[i]);
        }
        FOR(i, to_src.f, to_src.s-1) {
            // merging of two segments
            ap[x].insert(p[i]);
        }
        // merging of two segments and midpoint
        ap[x].insert(p[m]);
    }
    dnc(dnc, 0, 0, n);
};
```

AuxiliaryTree.h

Description: Get the virtual tree for each color id

86fb51, 29 lines

```
vector<vector<int>> to2(n);
rep(ci, n) { // ci = color id
    vector<int>& vs = cvs[ci];
    // cvs = nodes of that col
```

```
if (vs.size() == 0) continue;
sort(vs.begin(), vs.end(), [&](int a, int b) {
    return in[a] < in[b];
});
int m = vs.size();
rep(i, m-1) {
    vs.push_back(lca(vs[i], vs[i+1]));
}
sort(vs.begin(), vs.end(), [&](int a, int b) {
    return in[a] < in[b];
});
vs.erase(unique(vs.begin(), vs.end()), vs.end());
{
    vector<int> st;
    for (int v : vs) {
        while (st.size()) {
            int p = st.back();
            if (in[p] < in[v] && in[v] < out[p]) break;
            st.pop_back();
        }
        if (st.size()) to2[st.back()].push_back(v);
        st.push_back(v);
    }
}
// process aux tree
for (int v : vs) to2[v] = vector<int>();
```

9.9 Math

9.9.1 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).

9.9.2 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 $k = 1 \dots n$,

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

Various (10)

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

9e9d8d, 19 lines

```

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: $\mathcal{O}(k \log \frac{n}{k})$

753a4c, 19 lines

```

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);
}

```

10.2 Misc. algorithms

TernarySearchCpAlgo.h

Description: f is a unimodal function in the range [l, r]. returns the max in the range.

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

373e7a, 14 lines

```

double ternary_search(double l, double r) {
    double eps = 1e-9; //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);
}

```

LIS.h

Description: Compute indices for the longest increasing subsequence.

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

2932a0, 17 lines

```

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 $\leq t$ such that S is the sum of some subset of the weights.

Time: $\mathcal{O}(N \max(w_i))$

b20ccc, 16 lines

```

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;
}

```

10.3 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 $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

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

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)} (f(i, k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.

Time: $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 18 lines

```

struct DP { // Modify at will:
    int lo(int ind) { return 0; }
    int hi(int ind) { return ind; }
    ll f(int ind, int k) { return dp[ind][k]; }
    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

    void rec(int L, int R, int LO, int HI) {
        if (L >= R) return;
        int mid = (L + R) >> 1;
        pair<ll, int> best(LLONG_MAX, LO);
        rep(k, max(LO, lo(mid)), min(HI, hi(mid)))
            best = min(best, make_pair(f(mid, k), k));
        store(mid, best.second, best.first);
        rec(L, mid, LO, best.second+1);
        rec(mid+1, R, best.second, HI);
    }
    void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
}

```

10.4 Debugging tricks

- signal(SIGSEGV, [](int) { _Exit(0); }); converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). _GLIBCXX_DEBUG failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- feenableexcept(29); kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

10.5 Optimization tricks

_builtin_ia32_ldmxcsr(40896); disables denormals (which make floats 20x slower near their minimum value).

10.5.1 Bit hacks

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

FastMod

FastInput

BumpAllocator

SmallPtr

BumpAllocatorSTL

SIMD

10.5.2 Pragmas

- **#pragma GCC optimize ("Ofast")** will make GCC auto-vectorize loops and optimizes floating points better.
- **#pragma GCC target ("avx2")** can double performance of vectorized code, but causes crashes on old machines.
- **#pragma GCC optimize ("trapv")** kills the program on integer overflows (but is really slow).

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(-ULL / b) {}
    ull reduce(ull a) { // a % b + (0 or b)
        return a - (ull)((__uint128_t(m) * a) >> 64) * b;
    }
};
```

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.

Usage: ./a.out < input.txt**Time:** About 5x as fast as cin/scanf.

7b3c70, 17 lines

```
inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin);
    }
    return buf[bc++]; // returns 0 on EOF
}

int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 48;
    return a - 48;
}
```

BumpAllocator.h

Description: When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

745db2, 8 lines

```
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
    static size_t i = sizeof(buf);
    assert(s < i);
    return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory.

BumpAllocator.h 2dd6c9, 10 lines

```
template<class T> struct ptr {
    unsigned ind;
```

```
ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
    assert(ind < sizeof(buf));
}
T& operator*() const { return *(T*)(buf + ind); }
T* operator->() const { return &*this; }
T& operator[](int a) const { return (&*this)[a]; }
explicit operator bool() const { return ind; }
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.

Usage: `vector<vector<int, small<int>> ed(N);`

bb66d4, 14 lines

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;

template<class T> struct small {
    typedef T value_type;
    small() {}
    template<class U> small(const U&) {}
    T* allocate(size_t n) {
        buf_ind -= n * sizeof(T);
        buf_ind &= 0 - alignof(T);
        return (T*)(buf + buf_ind);
    }
    void deallocate(T*, size_t) {}
};
```

SIMD.h

Description: Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "`_mm(256)?_name_(si(128|256)|epi(8|16|32|64)|pd|ps)`". Not all are described here; grep for `_mm` in `/usr/lib/gcc/*4.9/include/` for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h" and `#define __SSE__` and `__MMX__` before including it. For aligned memory use `_mm_malloc(size, 32)` or `int buf[N] alignas(32)`, but prefer loadu/storeu.

551b82, 43 lines

```
#pragma GCC target ("avx2") // or sse4.1
#include "immintrin.h"

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))

// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd) (z:y:x), movemask_epi8 (hibits of bytes)
// i32gather-epi32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_ep16: dot product of unsigned i16's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(i, 256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x, x, 1) swaps 128-bit lanes
// shuffle_ep132(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_ep18(x, y) takes a vector instead of an imm

// Methods that work with most data types (append e.g. _epi32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)

int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
    int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }

ll example_filteredDotProduct(int n, short* a, short* b) {
```

```
    int i = 0; ll r = 0;
    mi zero = _mm256_setzero_si256(), acc = zero;
    while (i + 16 <= n) {
        mi va = L(a[i]), vb = L(b[i]); i += 16;
        va = _mm256_and_si256(_mm256_cmplt_epi16(va, vb), va);
        mi vp = _mm256_madd_epi16(va, vb);
        acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
            _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
    }
    union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
    for (i;n;++i) if (a[i] < b[i]) r += a[i]*b[i]; //<- equiv
    return r;
}
```