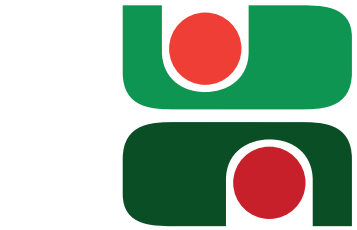


UDESC

Universidade do Estado de Santa Catarina

Não Treinamos o João

Enzo Rodrigues, Eric Grochowicz, João Oliveira

2024-09-12

Contest (1)

.vimrc

7 lines

```
set nu ai si cindent et ts=4 sw=4 so=10 nosm undofile

inoremap {} {}<left><return><up><end><return>
" remap de chaves

au BufReadPost * if line("'\"") > 0 && line("'\"") <= line("$")
      | exe "normal! g'\"" | endif
" volta pro lugar onde estava quando saiu do arquivo
```

run.sh

2 lines

```
#!/bin/bash
g++ -std=c++20 -DBRUTE -O2 -Wall -Wextra -Wconversion -Wfatal-
     errors -fsanitize=address,undefined $1 && ./a.out
```

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
```

Mint.cpp

d41d8c, 49 lines

```
template <auto MOD, typename T = decltype(MOD)>
struct Mint {
    using U = long long;
    // se o modulo for long long, usar U = __int128
    using m = Mint<MOD, T>;
    T v;
    Mint(T val = 0) : v(val) {
        assert(sizeof(T) * 2 <= sizeof(U));
        if (v < -MOD || v >= 2 * MOD) v %= MOD;
        if (v < 0) v += MOD;
        if (v >= MOD) v -= MOD;
    }
    Mint(U val) : v(T(val % MOD)) {
        assert(sizeof(T) * 2 <= sizeof(U));
        if (v < 0) v += MOD;
    }
    bool operator==(m o) const { return v == o.v; }
    bool operator<(m o) const { return v < o.v; }
    bool operator!=(m o) const { return v != o.v; }
    m pwr(m b, U e) {
        m res = 1;
        while (e > 0) {
            if (e & 1) res *= b;
            b *= b, e /= 2;
        }
        return res;
    }
    m &operator+=(m o) {
        v -= MOD - o.v;
        if (v < 0) v += MOD;
        return *this;
    }
    m &operator-=(m o) {
        v -= o.v;
        if (v < 0) v += MOD;
        return *this;
    }
    m &operator*=(m o) {
        v = (T)((U)v * o.v % MOD);
        return *this;
    }
    m &operator/=(m o) { return *this *= o.pwr(o, MOD - 2); }
    m &operator^=(U e) { return *this = pwr(*this, e); }
    friend m operator-(m a, m b) { return a -= b; }
    friend m operator+(m a, m b) { return a += b; }
    friend m operator*(m a, m b) { return a *= b; }
    friend m operator/(m a, m b) { return a /= b; }
    friend m operator^(m a, U e) { return a.pwr(a, e); }
};
```

troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Mathematics (2)

2.1 Equations

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

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

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \frac{ed - bf}{ad - bc} \\ y = \frac{af - ec}{ad - bc} \end{matrix}$$

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

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

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

2.2 Linear Operators

2.2.1 Rotate counter-clockwise by $\theta°$

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

2.2.2 Reflect about the line $y = mx$

$$\frac{1}{m^2 + 1}\begin{bmatrix} 1 - m^2 & 2m \\ 2m & m^2 - 1 \end{bmatrix}$$

2.2.3 Inverse of a 2x2 matrix A

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{det(A)}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

2.2.4 Horizontal shear by K

$$\begin{bmatrix} 1 & K \\ 0 & 1 \end{bmatrix}$$

2.2.5 Vertical shear by K

$$\begin{bmatrix} 1 & 0 \\ K & 1 \end{bmatrix}$$

2.2.6 Change of basis

$\vec{a}_\beta$ are the coordinates of vector $\vec{a}$ in basis β.
$\vec{a}$ are the coordinates of vector $\vec{a}$ in the canonical basis.
$\vec{b1}$ and $\vec{b2}$ are the basis vectors for β.
C is a matrix that changes from basis β to the canonical basis.

$$C\vec{a}_\beta = \vec{a}$$

$$C^{-1}\vec{a} = \vec{a}_\beta$$

$$C = \begin{bmatrix} b1_x & b2_x \\ b1_y & b2_y \end{bmatrix}$$

2.2.7 Properties of matrix operations

$$(AB)^{-1} = A^{-1}B^{-1}$$

$$(AB)^T = B^T A^T$$

$$(A^{-1})^T = (A^T)^{-1}$$

$$(A+B)^T = A^T + B^T$$

$$det(A) = det(A^T)$$

$$det(AB) = det(A)det(B)$$

Let A be an NxN matrix:

$$det(kA) = K^N det(A)$$

2.3 Recurrences

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

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.4 Trigonometry

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

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$

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

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

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

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

2.5 Geometry

2.5.1 Triangles

Side lengths: a, b, c

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

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

Circumradius: $R = \dfrac{abc}{4A}$

Inradius: $r = \dfrac{A}{p}$

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):

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

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

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

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

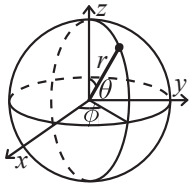
2.5.2 Quadrilaterals

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

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

2.5.3 Spherical coordinates

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



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

2.6 Derivatives/Integrals

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

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

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

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

Integration by parts:

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

2.7 Sums

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

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

2.8 Series

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

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

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

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

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

2.9 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,

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

2.9.1 Discrete distributions

Binomial distribution

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

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

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

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

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 $\mathrm{Fs}(p)$, $0 \le p \le 1$.

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

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

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

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

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

2.9.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

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

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

2.10 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ 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). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$.

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 $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki}t_k$.

Data structures (3)

OrderedSet.cpp

Description: Set with operations to find the element by index - find_by_order(i) and to find the index of an element
Usage: `ordered_set<int> seta;`
Time: All operations in $\mathcal{O}(\log N)$.

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> d41d8c, 8 lines

```
using namespace __gnu_pbds;

template <typename T>
using ordered_set =
    tree<T, null_type, less<T>, rb_tree_tag,
         tree_order_statistics_node_update>;

template <typename T, typename U>
using ordered_map = tree<T, U, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;
```

SegmentTreeBeats.cpp

Description: Updates of max, min and sum in range with queries of sum in range
Usage: `"seg" is already declared globally, just call build`
Time: If sum update is not used: $\mathcal{O}(\log N)$, otherwise $\mathcal{O}(\log^2 N)$.

d41d8c, 151 lines

```
const ll INF = 1e18;
struct node {
    ll mi, smi, mx, smx, sum, lazy;
    int fmi, fmx;
    node() {
        mi = smi = INF;
        mx = smx = -INF;
        fmi = 0, fmx = 0, sum = 0, lazy = 0;
    }
    node(ll val) {
        mi = mx = sum = val;
        smi = INF, smx = -INF;
        fmx = fmi = 1;
        lazy = 0;
    }
};

node operator+(node a, node b) {
    node ret;
    ret.sum = a.sum + b.sum;
    if (a.mi == b.mi) {
        ret.mi = a.mi;
        ret.fmi = a.fmi + b.fmi;
        ret.smi = min(a.smi, b.smi);
    } else if (a.mi < b.mi) {
        ret.mi = a.mi;
        ret.fmi = a.fmi;
        ret.smi = min(a.smi, b.mi);
    } else {
        ret.mi = b.mi;
        ret.fmi = b.fmi;
        ret.smi = min(b.smi, a.mi);
    }
```

```
    if (a.mx == b.mx) {
        ret.mx = a.mx;
        ret.fmx = a.fmx + b.fmx;
        ret.smx = max(a.smx, b.smx);
    } else if (a.mx > b.mx) {
        ret.mx = a.mx;
        ret.fmx = a.fmx;
        ret.smx = max(b.mx, a.smx);
    } else {
        ret.fmx = b.fmx;
        ret.mx = b.mx;
        ret.smx = max(a.mx, b.smx);
    }
    return ret;
}

struct SegBeats {
    vector<node> t;
    int n;
    void build(int _n) { // pra construir com tamanho, mas
         vazia
        n = _n;
        t.assign(n * 4, node());
    }
    void build(const vector<ll> &v) { // pra construir com
         vector
        n = (int)v.size();
        t.assign(n * 4, node());
        build(1, 0, n - 1, v);
    }
    void build(ll *bg, ll *en) { // pra construir com array de
         C
        build(vector<ll>(bg, en));
    }
    inline int lc(int p) { return 2 * p; }
    inline int rc(int p) { return 2 * p + 1; }
    node build(int p, int l, int r, const vector<ll> &a) {
        if (l == r) return t[p] = node(a[l]);
        int mid = (l + r) >> 1;
        return t[p] = build(lc(p), l, mid, a) + build(rc(p),
             mid + 1, r, a);
    }
    void pushsum(int p, int l, int r, ll x) {
        t[p].sum += (r - l + 1) * x;
        t[p].mi += x;
        t[p].mx += x;
        t[p].lazy += x;
        if (t[p].smi != INF) t[p].smi += x;
        if (t[p].smx != -INF) t[p].smx += x;
    }
    void pushmax(int p, ll x) {
        if (x <= t[p].mi) return;
        t[p].sum += t[p].fmi * (x - t[p].mi);
        if (t[p].mx == t[p].mi) t[p].mx = x;
        if (t[p].smx == t[p].mi) t[p].smx = x;
        t[p].mi = x;
    }
    void pushmin(int p, ll x) {
        if (x >= t[p].mx) return;
        t[p].sum += t[p].fmx * (x - t[p].mx);
        if (t[p].mi == t[p].mx) t[p].mi = x;
        if (t[p].smi == t[p].mx) t[p].smi = x;
        t[p].mx = x;
    }
    void pushdown(int p, int l, int r) {
        if (l == r) return;
        int mid = (l + r) >> 1;
        pushsum(lc(p), l, mid, t[p].lazy);
        pushsum(rc(p), mid + 1, r, t[p].lazy);
        t[p].lazy = 0;

        pushmax(lc(p), t[p].mi);
        pushmax(rc(p), t[p].mi);

        pushmin(lc(p), t[p].mx);
        pushmin(rc(p), t[p].mx);
    }
    node updatemin(int p, int l, int r, int L, int R, ll x) {
        if (l > R || r < L || x >= t[p].mx) return t[p];
        if (l >= L && r <= R && x > t[p].smx) {
            pushmin(p, x);
            return t[p];
        }
        pushdown(p, l, r);
        int mid = (l + r) >> 1;
        t[p] = updatemin(lc(p), l, mid, L, R, x) + updatemin(rc
             (p), mid + 1, r, L, R, x);
        return t[p];
    }
    node updatemax(int p, int l, int r, int L, int R, ll x) {
        if (l > R || r < L || x <= t[p].mi) return t[p];
        if (l >= L && r <= R && x < t[p].smi) {
            pushmax(p, x);
            return t[p];
        }
        pushdown(p, l, r);
        int mid = (l + r) >> 1;
        t[p] = updatemax(lc(p), l, mid, L, R, x) + updatemax(rc
             (p), mid + 1, r, L, R, x);
        return t[p];
    }
    node updatesum(int p, int l, int r, int L, int R, ll x) {
        if (l > R || r < L) return t[p];
        if (l >= L && r <= R) {
            pushsum(p, l, r, x);
            return t[p];
        }
        pushdown(p, l, r);
        int mid = (l + r) >> 1;
        return t[p] = updatesum(lc(p), l, mid, L, R, x) +
                      updatesum(rc(p), mid + 1, r, L, R, x);
    }
    node query(int p, int l, int r, int L, int R) {
        if (l > R || r < L) return node();
        if (l >= L && r <= R) return t[p];
        pushdown(p, l, r);
        int mid = (l + r) >> 1;
        return query(lc(p), l, mid, L, R) + query(rc(p), mid +
            1, r, L, R);
    }
    ll query(int l, int r) { return query(1, 0, n - 1, l, r).
         sum; }
    void updatemax(int l, int r, ll x) { updatemax(1, 0, n - 1,
          l, r, x); }
    void updatemin(int l, int r, ll x) { updatemin(1, 0, n - 1,
          l, r, x); }
    void updatesum(int l, int r, ll x) { updatesum(1, 0, n - 1,
          l, r, x); }
} seg;
```

SegmentTree2D.cpp

Description: Segment Tree with point update and rectangular range query
Usage: "seg" declared globally
Time: $\mathcal{O}(\log^2 N)$

d41d8c, 47 lines

```
struct SegTree2D {
    ll merge(ll a, ll b) { return a + b; }
    ll neutral = 0;
    int n, m;
    vector<vector<ll>> t;
    void build(int _n, int _m) {
        n = _n, m = _m;
        t.assign(2 * n, vector<ll>(2 * m, neutral));
        for (int i = 2 * n - 1; i >= n; i--)
            for (int j = m - 1; j > 0; j--)
                t[i][j] = merge(t[i][j << 1], t[i][j << 1 | 1])
                     ;
        for (int i = n - 1; i > 0; i--)
            for (int j = 2 * m - 1; j > 0; j--)
                t[i][j] = merge(t[i << 1][j], t[i << 1 | 1][j])
                     ;
    }
    ll inner_query(int idx, int l, int r) {
        ll res = neutral;
        for (l += m, r += m + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) res = merge(res, t[idx][l++]);
            if (r & 1) res = merge(res, t[idx][--r]);
        }
        return res;
    }
    // query do ponto (a, b) ate o ponto (c, d), retorna neutro
         se a > c ou b > d
    ll query(int a, int b, int c, int d) {
        ll res = neutral;
        for (a += n, c += n + 1; a < c; a >>= 1, c >>= 1) {
            if (a & 1) res = merge(res, inner_query(a++, b, d))
                 ;
            if (c & 1) res = merge(res, inner_query(--c, b, d))
                 ;
        }
        return res;
    }
    void inner_update(int idx, int i, ll x) {
        auto &c = t[idx];
        i += m;
        c[i] = x;
        for (i >>= 1; i > 0; i >>= 1) c[i] = merge(c[i << 1], c
             [i << 1 | 1]);
    }
    void update(int i, int j, ll x) {
        i += n;
        inner_update(i, j, x);
        for (i >>= 1; i > 0; i >>= 1) {
            ll val = merge(t[i << 1][j + m], t[i << 1 | 1][j +
                 m]);
            inner_update(i, j, val);
        }
    }
} seg;
```

ImplicitTreap.cpp

Description: Does everything
Time: Expected $\mathcal{O}(\log N)$

d41d8c, 124 lines

```
mt19937 rng((uint32_t)chrono::steady_clock::now().
    time_since_epoch().count());
namespace imp_treap {
    using T = ll; // mudar pra int se nao precisar pra melhorar
         a performance
    T merge(T a, T b) { return a + b; }
    T neutral = 0;
    struct node_info {
        node_info *l, *r;
        int y, size;
        T val, acc, add;
        bool rev;
        node_info() { }
```

```
    node_info(T _val)
        : l(0), r(0), y(rng()), size(0), val(_val), acc(0),
          add(0), rev(false) { }
};
using node = node_info *;
node root = 0;
inline int size(node t) { return t ? t->size : 0; }
inline T acc(node t) { return t ? t->acc : 0; }
inline bool rev(node t) { return t ? t->rev : false; }
inline void push(node t) {
    if (!t) return;
    if (rev(t)) {
        t->rev = false;
        swap(t->l, t->r);
        if (t->l) t->l->rev ^= 1;
        if (t->r) t->r->rev ^= 1;
    }
    t->acc += t->add * size(t);
    // t->acc += t->add se for RMQ
    t->val += t->add;
    if (t->l) t->l->add += t->add;
    if (t->r) t->r->add += t->add;
    t->add = 0;
}
inline void pull(node t) {
    if (t) {
        push(t->l), push(t->r);
        t->size = size(t->l) + size(t->r) + 1;
        t->acc = merge(t->val, merge(acc(t->l), acc(t->r)))
            ;
    }
}
void merge(node &t, node L, node R) {
    push(L), push(R);
    if (!L || !R) {
        t = L ? L : R;
    } else if (L->y > R->y) {
        merge(L->r, L->r, R);
        t = L;
    } else {
        merge(R->l, L, R->l);
        t = R;
    }
    pull(t);
}
void split(node t, int pos, node &L, node &R, int add = 0)
    {
    if (!t) {
        L = R = nullptr;
    } else {
        push(t);
        int imp_key = add + size(t->l);
        if (pos <= imp_key) {
            split(t->l, pos, L, t->l, add);
            R = t;
        } else {
            split(t->r, pos, t->r, R, imp_key + 1);
            L = t;
        }
    }
    pull(t);
}
inline void insert(node to, int pos) {
    node L, R;
    split(root, pos, L, R);
    merge(L, L, to);
    merge(root, L, R);
}
bool remove(node &t, int pos, int add = 0) {
    if (!t) return false;
    push(t);
    int imp_key = add + size(t->l);
    if (pos == imp_key) {
        node me = t;
        merge(t, t->l, t->r);
        delete me;
        return true;
    }
    bool ok;
    if (pos < imp_key) ok = remove(t->l, pos, add);
    else ok = remove(t->r, pos, imp_key + 1);
    pull(t);
    return ok;
}
inline T query(int l, int r) {
    if (l > r) return neutral;
    node L1, L2, R1, R2;
    split(root, r + 1, L1, R1);
    split(L1, l, L2, R2);
    T ans = acc(R2);
    merge(L1, L2, R2);
    merge(root, L1, R1);
    return ans;
}
inline void update_sum(int l, int r, T val) {
    if (l > r) return;
    node L1, L2, R1, R2;
    split(root, r + 1, L1, R1);
    split(L1, l, L2, R2);
    assert(R2);
    R2->add += val;
    merge(L1, L2, R2);
    merge(root, L1, R1);
}
inline void reverse(int l, int r) {
    if (l > r) return;
    node L1, L2, R1, R2;
    split(root, r + 1, L1, R1);
    split(L1, l, L2, R2);
    R2->rev ^= 1;
    merge(L1, L2, R2);
    merge(root, L1, R1);
}
inline void insert(int pos, int val) { insert(new node_info
    (val), pos); }
inline bool remove(int pos) { return remove(root, pos); }
}
```

LichaoLazy.cpp

Description: Sendo $N = MA - MI$: insert((a, b)) minimiza tudo com $ax+b$ $O(logN)$ insert((a, b), l, r) minimiza com ax+b no range $[l, r]$ $O(log^2 N)$ shift((a, b)) soma ax+b em tudo $O(1)$ shift((a, b), l, r) soma ax+b no range $[l, r]$ $O(log^2 N)$ query(x) retorna o valor da posicao x $O(logN)$ No inicio eh tudo LINF, se inserir (0, 0) fica tudo 0

Time: $\mathcal{O}(nlogN)$ de memoria, $\mathcal{O}(n)$ de memoria se nao usar as operacoes de range

d41d8c, 77 lines

```
template<int MI = int(-1e9), int MA = int(1e9)> struct lichao {
  struct line {
    ll a, b;
    ll la, lb; // lazy
    array<int, 2> ch;
    line(ll a_ = 0, ll b_ = LINF) :
      a(a_), b(b_), la(0), lb(0), ch({-1, -1}) {}
    ll operator ()(ll x) { return a*x + b; }
  };
  vector<line> ln;
  int ch(int p, int d) {
    if (ln[p].ch[d] == -1) {
      ln[p].ch[d] = ln.size();
      ln.emplace_back();
    }
    return ln[p].ch[d];
  }
  lichao() { ln.emplace_back(); }

  void prop(int p, int l, int r) {
    if (ln[p].la == 0 and ln[p].lb == 0) return;
    ln[p].a += ln[p].la, ln[p].b += ln[p].lb;
    if (l != r) {
      int pl = ch(p, 0), pr = ch(p, 1);
      ln[pl].la += ln[p].la, ln[pl].lb += ln[p].lb;
      ln[pr].la += ln[p].la, ln[pr].lb += ln[p].lb;
    }
    ln[p].la = ln[p].lb = 0;
  }

  ll query(int x, int p=0, int l=MI, int r=MA) {
    prop(p, l, r);
    ll ret = ln[p](x);
    if (ln[p].ch[0] == -1 and ln[p].ch[1] == -1) return ret;
    int m = l + (r-l)/2;
    if (x <= m) return min(ret, query(x, ch(p, 0), l, m));
    return min(ret, query(x, ch(p, 1), m+1, r));
  }

  void push(line s, int p, int l, int r) {
    prop(p, l, r);
    int m = l + (r-l)/2;
    bool L = s(l) < ln[p](l);
    bool M = s(m) < ln[p](m);
    bool R = s(r) < ln[p](r);
    if (M) swap(ln[p].a, s.a), swap(ln[p].b, s.b);
    if (s.b == LINF) return;
    if (L != M) push(s, ch(p, 0), l, m);
    else if (R != M) push(s, ch(p, 1), m+1, r);
  }
  void insert(line s, int a=MI, int b=MA, int p=0, int l=MI,
       int r=MA) {
    prop(p, l, r);
    if (a <= l and r <= b) return push(s, p, l, r);
    if (b < l or r < a) return;
    int m = l + (r-l)/2;
    insert(s, a, b, ch(p, 0), l, m);
    insert(s, a, b, ch(p, 1), m+1, r);
  }

  void shift(line s, int a=MI, int b=MA, int p=0, int l=MI, int
       r=MA) {
    prop(p, l, r);
    int m = l + (r-l)/2;
    if (a <= l and r <= b) {
      ln[p].la += s.a, ln[p].lb += s.b;
      return;
    }
    if (b < l or r < a) return;
    if (ln[p].b != LINF) {
      push(ln[p], ch(p, 0), l, m);
      push(ln[p], ch(p, 1), m+1, r);
      ln[p].a = 0, ln[p].b = LINF;
    }
    shift(s, a, b, ch(p, 0), l, m);
    shift(s, a, b, ch(p, 1), m+1, r);
  }
};
```

KDFenwickTree.cpp

Description: Fenwick Tree em k dimensões. Faz apenas queries de prefixo e updates pontuais em $O(k \log^k n)$ Para queries em range, deve-se fazer inclusão-exclusão, porém a complexidade fica exponencial, para k dimensões a query em range é $O(2^k k \log^k n)$.

d41d8c, 28 lines

```
const int MAX = 20;
long long tree[MAX][MAX][MAX][MAX]; // insira o numero de
    dimensoes aqui

long long query(vector<int> s, int pos = 0) { // s eh a
    coordenada
    long long sum = 0;
    while (s[pos] >= 0) {
        if (pos < (int)s.size() - 1) {
            sum += query(s, pos + 1);
        } else {
            sum += tree[s[0]][s[1]][s[2]][s[3]];
            // atualizar se mexer no numero de dimensoes
        }
        s[pos] = (s[pos] & (s[pos] + 1)) - 1;
    }
    return sum;
}

void update(vector<int> s, int v, int pos = 0) {
    while (s[pos] < MAX) {
        if (pos < (int)s.size() - 1) {
            update(s, v, pos + 1);
        } else {
            tree[s[0]][s[1]][s[2]][s[3]] += v;
            // atualizar se mexer no numero de dimensoes
        }
        s[pos] |= s[pos] + 1;
    }
}
```

Numerical (4)

4.1 Factorization

PollardRhoMillerRabin.cpp

Description: Pollard Rho and Miller Rabin

d41d8c, 87 lines

```
namespace MillerRabin {
    inline ll mul_mod(ll a, ll b, ll m) { return (ll)((__int128
        )a * b % m); }
    inline ll power(ll b, ll e, ll m) {
        ll r = 1;
        b = b % m;
        while (e > 0) {
            if (e & 1) r = mul_mod(r, b, m);
            b = mul_mod(b, b, m), e >>= 1;
        }
        return r;
    }
    inline bool composite(ll n, ll a, ll d, ll s) {
        ll x = power(a, d, n);
        if (x == 1 || x == n - 1 || a % n == 0) return false;
        for (int r = 1; r < s; r++) {
            x = mul_mod(x, x, n);
            if (x == n - 1) return false;
        }
        return true;
    }

    // com esses "primos", o teste funciona garantido para n <=
        2^64
    int primes[] = {2, 325, 9375, 28178, 450775, 9780504,
        1795265022};

    // funciona para n <= 3*10^24 com os primos ate 41, mas tem
        que cuidar com overflow
    // int primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
        37, 41};

    bool prime(ll n) {
        if (n <= 2 || (n % 2 == 0)) return n == 2;
        ll d = n - 1, r = 0;
        while (d % 2 == 0) d /= 2, r++;
        for (int a : primes)
            if (composite(n, a, d, r)) return false;
        return true;
    }
}

namespace PollardRho {
    mt19937 rng((uint32_t)chrono::steady_clock::now().
        time_since_epoch().count());
    const ll P = 1e6 + 1;
    ll seq[P];
    inline ll add_mod(ll x, ll y, ll m) { return (x += y) < m ?
         x : x - m; }
    inline ll mul_mod(ll a, ll b, ll m) { return (ll)((__int128
        )a * b % m); }
    ll rho(ll n) {
        if (n % 2 == 0) return 2;
        if (n % 3 == 0) return 3;
        ll x0 = rng() % n, c = rng() % n;
        while (1) {
            ll x = x0++, y = x, u = 1, v, t = 0;
            ll *px = seq, *py = seq;
            while (1) {
                *py++ = y = add_mod(mul_mod(y, y, n), c, n);
                *py++ = y = add_mod(mul_mod(y, y, n), c, n);
                if ((x = *px++) == y) break;
                v = u;
                u = mul_mod(u, abs(y - x), n);
                if (!u) return gcd(v, n);
                if (++t == 32) {
                    t = 0;
                    if ((u = gcd(u, n)) > 1 && u < n) return u;
                }
            }
            if (t && (u = gcd(u, n)) > 1 && u < n) return u;
        }
    }
}

vector<ll> factorize(ll x) {
    vector<ll> f;
    if (x == 1) return f;
    function<void(ll)> dfs = [&](ll x) {
        if (x == 1) return;
        if (x < Sieve::P) {
            auto fs = Sieve::factorize(x);
            f.insert(f.end(), fs.begin(), fs.end());
        } else if (MillerRabin::prime(x)) {
            f.push_back(x);
        } else {
            ll d = PollardRho::rho(x);
            dfs(d);
            dfs(x / d);
        }
    };
    dfs(x);
    sort(f.begin(), f.end());
    return f;
}
```

4.2 Polynomials and recurrences

TaylorShift.cpp

d41d8c, 18 lines

```
template <auto MOD, typename T = Mint<MOD>>
vector<T> shift(vector<T> a, int k) {
    int n = (int)a.size();
    vector<T> fat(n, 1), ifat(n), shifting(n);
    for (int i = 1; i < n; i++) fat[i] = fat[i - 1] * i;
    ifat[n - 1] = T(1) / fat[n - 1];
    for (int i = n - 1; i > 0; i--) ifat[i - 1] = ifat[i] * i;
    for (int i = 0; i < n; i++) a[i] *= fat[i];
    T pk = 1;
    for (int i = 0; i < n; i++) {
        shifting[n - i - 1] = pk * ifat[i];
        pk *= k;
    }
    auto ans = multiply<MOD>(a, shifting);
    ans.erase(ans.begin(), ans.begin() + n - 1);
    for (int i = 0; i < n; i++) ans[i] *= ifat[i];
    return ans;
}
```

EvaluateInterpolation.cpp

Description: Dado 'n' pontos (i, y[i]), $i \in [0, n)$, avalia o polinomio de grau n-1 que passa por esses pontos em 'x' Tudo modular, precisa do mint

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

d41d8c, 20 lines

```
mint evaluate_interpolation(int x, vector<mint> y) {
  int n = y.size();

  vector<mint> sulf(n+1, 1), fat(n, 1), ifat(n);
  for (int i = n-1; i >= 0; i--) sulf[i] = sulf[i+1] * (x - i);
  for (int i = 1; i < n; i++) fat[i] = fat[i-1] * i;
  ifat[n-1] = 1/fat[n-1];
  for (int i = n-2; i >= 0; i--) ifat[i] = ifat[i+1] * (i + 1);

  mint pref = 1, ans = 0;
  for (int i = 0; i < n; pref *= (x - i++)) {
    mint num = pref * sulf[i+1];

    mint den = ifat[i] * ifat[n-1 - i];
    if ((n-1 - i)%2) den *= -1;

    ans += y[i] * num * den;
  }
  return ans;
}
```

4.3 MIT Numerical

GoldenSectionSearch.h

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

Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`

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

d41d8c, 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;
}
```

Polynomial.h

d41d8c, 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.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `poly_roots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`
Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

"Polynomial.h" d41d8c, 23 lines

```
vector<double> poly_roots(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 = poly_roots(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: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

d41d8c, 13 lines

```
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}`

"../number-theory/ModPow.h" d41d8c, 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());
  trav(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 $S[0 \ldots n-1]$ and $tr[0 \ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2 \log k\right)$

d41d8c, 26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(S);

  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;
}
```

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.

d41d8c, 8 lines

```
double quad(double (*f)(double), double a, double b) {
  const int n = 1000;
  double h = (b - a) / 2 / n;
  double 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 z, y;`

```
double h(double x) { return x*x + y*y + z*z <= 1; }
double g(double y) { ::y = y; return quad(h, -1, 1); }
double f(double z) { ::z = z; return quad(g, -1, 1); }
double sphereVol = quad(f, -1, 1), pi = sphereVol*3/4;
```

d41d8c, 16 lines

```
typedef double d;
d simpson(d (*f)(d), d a, d b) {
  d c = (a+b) / 2;
  return (f(a) + 4*f(c) + f(b)) * (b-a) / 6;
}
d rec(d (*f)(d), d a, d b, d eps, d S) {
  d c = (a+b) / 2;
  d S1 = simpson(f, a, c);
  d S2 = simpson(f, 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);
}
d quad(d (*f)(d), d a, d b, d eps = 1e-8) {
  return rec(f, a, b, eps, simpson(f, a, b));
}
```

Determinant.h

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

d41d8c, 15 lines

```
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: $\mathcal{O}\left(N^3\right)$

d41d8c, 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;
}
```

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.

d41d8c, 68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
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;
  }
};
```

math-simplex.cpp

Description: Simplex algorithm. WARNING- segfaults on empty (size 0) max cx st Ax<=b, x>=0 do 2 phases; 1st check feasibility; 2nd check boundedness and ans

d41d8c, 40 lines

```
vector<double> simplex(vector<vector<double> > A, vector<double
    > b, vector<double> c) {
  int n = (int) A.size(), m = (int) A[0].size()+1, r = n, s = m
      -1;
  vector<vector<double> > D = vector<vector<double> > (n+2,
      vector<double>(m+1));
  vector<int> ix = vector<int> (n+m);
  for (int i=0; i<n+m; i++) ix[i] = i;
  for (int i=0; i<n; i++) {
    for (int j=0; j<m-1; j++)D[i][j]=-A[i][j];
    D[i][m-1] = 1;
    D[i][m] = b[i];
    if (D[r][m] > D[i][m]) r = i;
  }
  for (int j=0; j<m-1; j++) D[n][j]=c[j];
  D[n+1][m-1] = -1; int z = 0;
  for (double d;;) {
    if (r < n) {
      swap(ix[s], ix[r+m]);
      D[r][s] = 1.0/D[r][s];
      for (int j=0; j<=m; j++) if (j!=s) D[r][j] *= -D[r][s];
      for(int i=0; i<=n+1; i++)if(i!=r) {
        for (int j=0; j<=m; j++) if(j!=s) D[i][j] += D[r][j] *
            D[i][s];
        D[i][s] *= D[r][s];
      }
    }
    r = -1; s = -1;
    for (int j=0; j <m; j++) if (s<0 || ix[s]>ix[j]) {
      if (D[n+1][j]>eps || D[n+1][j]>-eps && D[n][j]>eps) s = j
          ;
    }
    if (s < 0) break;
    for (int i=0; i<n; i++) if(D[i][s]<-eps) {
      if (r < 0 || (d = D[r][m]/D[r][s]-D[i][m]/D[i][s]) < -eps
          || d < eps && ix[r+m] > ix[i+m]) r=i;
    }
    if (r < 0) return vector<double>(); // unbounded
  }
  if (D[n+1][m] < -eps) return vector<double>(); // infeasible
  vector<double> x(m-1);
  for (int i = m; i < n+m; i ++) if (ix[i] < m-1) x[ix[i]] = D[
      i-m][m];
  printf("%.21f\n", D[n][m]);
  return x; // ans: D[n][m]
}
```

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

d41d8c, 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:

"SolveLinear.h" d41d8c, 7 lines

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

d41d8c, 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: $\mathcal{O}(n^3)$

d41d8c, 35 lines

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

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

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

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

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

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

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

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

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 `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$

d41d8c, 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;
    } 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;
}
```

4.4 Convolutions

FFT.cpp

Description: Computa convolução (multiplicação) de polinômios em $O(N \log N)$, sendo N a soma dos graus dos polinômios. Testado e sem erros de precisão com polinômios de grau até 3.10^5 e constantes até 10^6.
Time: $\mathcal{O}(N \log N)$.

d41d8c, 67 lines

```
struct base {
    double a, b;
    base(double _a = 0, double _b = 0) : a(_a), b(_b) { }
    const base operator+(const base &c) const { return base(a +
        c.a, b + c.b); }
    const base operator-(const base &c) const { return base(a -
        c.a, b - c.b); }
    const base operator*(const base &c) const {
        return base(a * c.a - b * c.b, a * c.b + b * c.a);
    }
};

using poly = vector<base>;
const double PI = acos(-1);

void fft(poly &a, bool inv = 0) {
    int n = (int)a.size();

    for (int i = 0; i < n; i++) {
        int bit = n >> 1, j = 0, k = i;
        while (bit > 0) {
            if (k & 1) j += bit;
            k >>= 1, bit >>= 1;
        }
        if (i < j) swap(a[i], a[j]);
    }

    double angle = 2 * PI / n * (inv ? -1 : 1);
    poly wn(n / 2);
    for (int i = 0; i < n / 2; i++) wn[i] = {cos(angle * i),
         sin(angle * i)};

    for (int len = 2; len <= n; len <<= 1) {
        int aux = len / 2;
        int step = n / len;
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < aux; j++) {
                base v = a[i + j + aux] * wn[step * j];
                a[i + j + aux] = a[i + j] - v;
                a[i + j] = a[i + j] + v;
            }
        }
    }

    for (int i = 0; inv && i < n; i++) a[i].a /= n, a[i].b /= n
        ;
}

vector<ll> multiply(vector<ll> &ta, vector<ll> &tb) {
    int n = (int)ta.size(), m = (int)tb.size();
    int t = n + m - 1, sz = 1;
    while (sz < t) sz <<= 1;

    poly a(sz), b(sz), c(sz);

    for (int i = 0; i < sz; i++) {
        a[i] = i < n ? base((double)ta[i]) : base(0);
        b[i] = i < m ? base((double)tb[i]) : base(0);
    }

    fft(a, 0), fft(b, 0);
    for (int i = 0; i < sz; i++) c[i] = a[i] * b[i];
    fft(c, 1);

    vector<ll> res(sz);
    for (int i = 0; i < sz; i++) res[i] = ll(round(c[i].a));
```

```
    while ((int)res.size() > t && res.back() == 0) res.pop_back
        ();

    return res;
}
```

NTT.cpp

d41d8c, 37 lines

```
template <auto MOD, typename T = Mint<MOD>>
void ntt(vector<T> &a, bool inv = 0) {
    int n = (int)a.size();
    auto b = a;
    T g = 1;
    while ((g ^ (MOD / 2)) == 1) g += 1;
    if (inv) g = T(1) / g;
    for (int step = n / 2; step; step /= 2) {
        T w = g ^ (MOD / (n / step)), wn = 1;
        for (int i = 0; i < n / 2; i += step) {
            for (int j = 0; j < step; j++) {
                auto u = a[2 * i + j], v = wn * a[2 * i + j +
                    step];
                b[i + j] = u + v;
                b[i + n / 2 + j] = u - v;
            }
            wn = wn * w;
        }
        swap(a, b);
    }
    if (inv) {
        T invn = T(1) / n;
        for (int i = 0; i < n; i++) a[i] *= invn;
    }
}

template <auto MOD, typename T = Mint<MOD>>
vector<T> multiply(vector<T> a, vector<T> b) {
    int n = (int)a.size(), m = (int)b.size();
    int t = n + m - 1, sz = 1;
    while (sz < t) sz <<= 1;
    a.resize(sz), b.resize(sz);
    ntt<MOD>(a, 0), ntt<MOD>(b, 0);
    for (int i = 0; i < sz; i++) a[i] *= b[i];
    ntt<MOD>(a, 1);
    while ((int)a.size() > t) a.pop_back();
    return a;
}
```

NTTBigModulo.cpp

d41d8c, 78 lines

```
template <auto MOD, typename T = Mint<MOD>>
void ntt(vector<T> &a, bool inv = 0) {
    int n = (int)a.size();
    auto b = a;
    T g = 1;
    while ((g ^ (MOD / 2)) == 1) g += 1;
    if (inv) g = T(1) / g;
    for (int step = n / 2; step; step /= 2) {
        T w = g ^ (MOD / (n / step)), wn = 1;
        for (int i = 0; i < n / 2; i += step) {
            for (int j = 0; j < step; j++) {
                auto u = a[2 * i + j], v = wn * a[2 * i + j +
                    step];
                b[i + j] = u + v;
                b[i + n / 2 + j] = u - v;
            }
            wn = wn * w;
        }
        swap(a, b);
    }
    if (inv) {
        T invn = T(1) / n;
        for (int i = 0; i < n; i++) a[i] *= invn;
    }
}

template <auto MOD, typename T = Mint<MOD>>
vector<T> multiply(vector<T> a, vector<T> b) {
    int n = (int)a.size(), m = (int)b.size();
    int t = n + m - 1, sz = 1;
    while (sz < t) sz <<= 1;
    a.resize(sz), b.resize(sz);
    ntt<MOD>(a, 0), ntt<MOD>(b, 0);
    for (int i = 0; i < sz; i++) a[i] *= b[i];
    ntt<MOD>(a, 1);
    while ((int)a.size() > t) a.pop_back();
    return a;
}

ll extended_gcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    } else {
        ll g = extended_gcd(b, a % b, y, x);
        y -= a / b * x;
        return g;
    }
}

ll crt(array<int, 2> rem, array<int, 2> mod) {
    __int128 ans = rem[0], m = mod[0];
    ll x, y;
    ll g = extended_gcd(mod[1], (ll)m, x, y);
    if ((ans - rem[1]) % g != 0) return -1;
    ans = ans + (__int128)1 * (rem[1] - ans) * (m / g) * y;
    m = (__int128)(mod[1] / g) * (m / g) * g;
    ans = (ans % m + m) % m;
    return (ll)ans;
}

template <auto MOD1, auto MOD2, typename T = Mint<MOD1>,
     typename U = Mint<MOD2>>
vector<ll> big_multiply(vector<ll> ta, vector<ll> tb) {
    vector<T> a1(ta.size()), b1(tb.size());
    vector<U> a2(ta.size()), b2(tb.size());
    for (int i = 0; i < (int)ta.size(); i++) a1[i] = ta[i];
    for (int i = 0; i < (int)tb.size(); i++) b1[i] = tb[i];
    for (int i = 0; i < (int)ta.size(); i++) a2[i] = ta[i];
    for (int i = 0; i < (int)tb.size(); i++) b2[i] = tb[i];
    auto c1 = multiply<MOD1>(a1, b1);
    vector<ll> res(c1.size());
    for (int i = 0; i < (int)res.size(); i++)
        res[i] = crt({c1[i].v, c2[i].v}, {MOD1, MOD2});
    return res;
}

const int MOD1 = 1004535809;
const int MOD2 = 1092616193;
```

Convolutions.cpp

d41d8c, 165 lines

```
vector<mint> and_convolution(vector<mint> A, vector<mint> B) {
    int n = (int)max(A.size(), B.size());
    int N = 0;
    while ((1 << N) < n) N++;
    A.resize(1 << N);
    B.resize(1 << N);
    vector<mint> C(1 << N);
    for (int j = 0; j < N; j++) {
        for (int i = (1 << N) - 1; i >= 0; i--) {
            if (~i >> j & 1) {
                A[i] += A[i | (1 << j)];
                B[i] += B[i | (1 << j)];
            }
        }
    }
    for (int i = 0; i < 1 << N; i++) C[i] = A[i] * B[i];
    for (int j = 0; j < N; j++) {
        for (int i = 0; i < 1 << N; i++)
            if (~i >> j & 1) C[i] -= C[i | (1 << j)];
    }
    return C;
}

vector<mint> gcd_convolution(vector<mint> A, vector<mint> B) {
    int N = (int)max(A.size(), B.size());
    A.resize(N + 1);
    B.resize(N + 1);
    vector<mint> C(N + 1);
    for (int i = 1; i <= N; i++) {
        mint a = 0;
        mint b = 0;
        for (int j = i; j <= N; j += i) {
            a += A[j];
            b += B[j];
        }
        C[i] = a * b;
    }
    for (int i = N; i >= 1; i--)
        for (int j = 2 * i; j <= N; j += i) C[i] -= C[j];
    return C;
}

vector<mint> lcm_convolution(vector<mint> A, vector<mint> B) {
    int N = (int)max(A.size(), B.size());
    A.resize(N + 1);
    B.resize(N + 1);
    vector<mint> C(N + 1), a(N + 1), b(N + 1);
    for (int i = 1; i <= N; i++) {
        for (int j = i; j <= N; j += i) {
            a[j] += A[i];
            b[j] += B[i];
        }
        C[i] = a[i] * b[i];
    }
    for (int i = 1; i <= N; i++)
        for (int j = 2 * i; j <= N; j += i) C[j] -= C[i];
    return C;
}

vector<mint> or_convolution(vector<mint> A, vector<mint> B) {
    int n = (int)max(A.size(), B.size());
    int N = 0;
    while ((1 << N) < n) N++;
    A.resize(1 << N);
    B.resize(1 << N);
    vector<mint> C(1 << N);
    for (int j = 0; j < N; j++) {
        for (int i = 0; i < 1 << N; i++) {
            if (i >> j & 1) {
                A[i] += A[i ^ (1 << j)];
                B[i] += B[i ^ (1 << j)];
            }
        }
    }
    for (int i = 0; i < 1 << N; i++) C[i] = A[i] * B[i];
```

```
    for (int j = N - 1; j >= 0; j--) {
        for (int i = (1 << N) - 1; i >= 0; i--)
            if (i >> j & 1) C[i] -= C[i ^ (1 << j)];
    }
    return C;
}

vector<mint> subset_convolution(vector<mint> A, vector<mint> B)
     {
    int n = int(max(A.size(), B.size()));
    int N = 0;
    while ((1 << N) < n) N++;
    A.resize(1 << N), B.resize(1 << N);
    vector a(1 << N, vector<mint>(N + 1)), b(1 << N, vector<
         mint>(N + 1));
    for (int i = 0; i < 1 << N; i++) {
        int popcnt = __builtin_popcount(i);
        a[i][popcnt] = A[i];
        b[i][popcnt] = B[i];
    }
    for (int j = 0; j < N; j++) {
        for (int i = 0; i < 1 << N; i++) {
            if (~i >> j & 1) continue;
            for (int popcnt = 0; popcnt <= N; popcnt++) {
                a[i][popcnt] += a[i ^ (1 << j)][popcnt];
                b[i][popcnt] += b[i ^ (1 << j)][popcnt];
            }
        }
    }
    vector c(1 << N, vector<mint>(N + 1));
    for (int i = 0; i < 1 << N; i++) {
        for (int j = 0; j <= N; j++)
            for (int k = 0; k + j <= N; k++) c[i][j + k] += a[i
                 ][j] * b[i][k];
    }
    for (int j = N - 1; j >= 0; j--) {
        for (int i = (1 << N) - 1; i >= 0; i--) {
            if (~i >> j & 1) continue;
            for (int popcnt = 0; popcnt <= N; popcnt++)
                c[i][popcnt] -= c[i ^ (1 << j)][popcnt];
        }
    }
    vector<mint> ans(1 << N);
    for (int i = 0; i < 1 << N; i++) {
        int popcnt = __builtin_popcount(i);
        ans[i] = c[i][popcnt];
    }
    return ans;
}

vector<mint> xor_convolution(vector<mint> A, vector<mint> B) {
    int n = int(A.size());
    for (int rep = 0; rep < 2; rep++) {
        for (int len = n >> 1; len; len >>= 1) {
            for (int i = 0; i < n; i += len << 1) {
                for (int j = 0; j < len; j++) {
                    int id = i + j;
                    mint x = A[id];
                    mint y = A[id + len];
                    A[id] = x + y;
                    A[id + len] = x - y;
                }
            }
        }
        swap(A, B);
    }
    vector<mint> ans(n);
    for (int i = 0; i < n; i++) ans[i] = A[i] * B[i];
    for (int len = 1; len < n; len <<= 1) {
        for (int i = 0; i < n; i += len << 1) {
            for (int j = 0; j < len; j++) {
                int id = i + j;
                mint x = ans[id];
                mint y = ans[id + len];
                ans[id] = x + y;
                ans[id + len] = x - y;
            }
        }
    }
    return ans;
}

vector<mint> xor_multiply(vector<mint> A, vector<mint> B) {
    int N = 1;
    int n = int(max(A.size(), B.size()));
    while (N < n) N <<= 1;
    A.resize(N);
    B.resize(N);
    auto ans = xor_convolution(A, B);
    for (int i = 0; i < N; i++) ans[i] /= N;

    return ans;
}
```

Number theory (5)

5.1 Modular arithmetic

ModLog.h

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.
Time: $\mathcal{O}\left(\sqrt{m}\right)$ d41d8c, 11 lines

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

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) $= \sum_{i=0}^{to-1} (ki + c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant. d41d8c, 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 $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for `modmul`, $\mathcal{O}(\log b)$ for `modpow` d41d8c, 11 lines

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}(\log p)$ for most p
"ModPow.h" d41d8c, 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;
  }
}
```

5.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s d41d8c, 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 $a^b \mod c$.

"ModMulLL.h" d41d8c, 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 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h" d41d8c, 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;
}
```

5.3 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 $a \pmod b$.

d41d8c, 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.
`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 $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$

"euclid.h" d41d8c, 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;
}
```

5.3.1 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 $\le n$ 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 $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1} ... (p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.

d41d8c, 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;
}
```

5.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \ge 0$, finds the closest rational approximation p/q with $p, q \le N$. It will obey $|p/q - x| \le 1/qN$.
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: $\mathcal{O}(\log N)$

d41d8c, 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 $p, q \le N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.
Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`
Time: $\mathcal{O}(\log(N))$

d41d8c, 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;
}
```

5.5 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.

5.6 Primes

5.6.1 Left-truncatable primes

Prime numbers such that any suffix of them is also a prime:

33,333,31
357,686,312,646,216,567,629,137

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

5.7 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 $n < 1e19$.

5.8 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 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 -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.

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

d41d8c, 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;
}
```

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

6.2.2 Lucas' Theorem

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

6.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$.

d41d8c, 6 lines

```
ll multinomial(vi& v) {
  ll 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;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able).
$B[0, \ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

6.3.3 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. $\pi(j) \ge j$, k j:s s.t. $\pi(j) > j$.

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

on n vertices: n^{n-2}
on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
with degrees d_i: $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$

6.3.7 Catalan numbers

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

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

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

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with 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.

Graph (7)

7.1 Network flow

Dinitz.cpp

Description: Flow algorithm with complexity $O(V^2E)$. With $U = \max|\text{cap}|$: $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.

d41d8c, 68 lines

```
struct dinitz {
  const bool scaling = false; // com scaling -> O(nm log(U)),
  int lim;                    // com constante alta
  struct edge {
    int to, cap, rev, flow;
    bool res;
    edge(int to_, int cap_, int rev_, bool res_)
      : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
  };

  vector<vector<edge>> g;
  vector<int> lev, beg;
  ll F;
  dinitz(int n) : g(n), F(0) {}

  void add(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size(), false);
    g[b].emplace_back(a, 0, g[a].size()-1, true);
  }
  bool bfs(int s, int t) {
    lev = vector<int>(g.size(), -1); lev[s] = 0;
    beg = vector<int>(g.size(), 0);
    queue<int> q; q.push(s);
    while (q.size()) {
      int u = q.front(); q.pop();
      for (auto& i : g[u]) {
        if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
        if (scaling and i.cap - i.flow < lim) continue;
        lev[i.to] = lev[u] + 1;
        q.push(i.to);
      }
    }
    return lev[t] != -1;
  }
  int dfs(int v, int s, int f = INF) {
    if (!f or v == s) return f;
    for (int& i = beg[v]; i < g[v].size(); i++) {
      auto& e = g[v][i];
      if (lev[e.to] != lev[v] + 1) continue;
      int foi = dfs(e.to, s, min(f, e.cap - e.flow));
      if (!foi) continue;
      e.flow += foi, g[e.to][e.rev].flow -= foi;
      return foi;
    }
    return 0;
  }
  ll max_flow(int s, int t) {
    for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
      while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
    return F;
  }
};

// Recupera as arestas do corte s-t
vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {
  g.max_flow(s, t);
  vector<pair<int, int>> cut;
  vector<int> vis(g.g.size(), 0), st = {s};
  vis[s] = 1;
  while (st.size()) {
    int u = st.back(); st.pop_back();
    for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
      vis[e.to] = 1, st.push_back(e.to);
  }
  for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
    if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e
        .to);
  return cut;
}
```

MinCostMaxFlow.cpp

Description: $min_costflow(s, t, f)$ computa o par $(fluxo, custo)$ com $max(fluxo) <= f$ que tenha $min(custo)$ $min_costflow(s, t)$: Fluxo maximo de custo minimo de s pra t Se for um dag, da pra substituir o SPFA por uma DP pra naopagar $O(nm)$ no comeco Se nao tiver aresta com custo negativo, nao precisa do SPFA

Time: $\mathcal{O}(nm + f * mlogn)$

d41d8c, 115 lines

```
template<typename T> struct mcmf {
  struct edge {
    int to, rev, flow, cap; // para, id da reversa, fluxo,
        capacidade
    bool res; // se eh reversa
    T cost; // custo da unidade de fluxo
    edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false
        ) {}
    edge(int to_, int rev_, int flow_, int cap_, T cost_, bool
        res_)
      : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_),
          cost(cost_) {}
  };

  vector<vector<edge>> g;
  vector<int> par_idx, par;
  T inf;
  vector<T> dist;

  mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T
      >::max()/3) {}

  void add(int u, int v, int w, T cost) { // de u pra v com cap
      w e custo cost
    edge a = edge(v, g[v].size(), 0, w, cost, false);
    edge b = edge(u, g[u].size(), 0, 0, -cost, true);

    g[u].push_back(a);
    g[v].push_back(b);
  }

  vector<T> spfa(int s) { // nao precisa se nao tiver custo
      negativo
    deque<int> q;
    vector<bool> is_inside(g.size(), 0);
    dist = vector<T>(g.size(), inf);

    dist[s] = 0;
    q.push_back(s);
    is_inside[s] = true;

    while (!q.empty()) {
      int v = q.front();
      q.pop_front();
      is_inside[v] = false;

      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;

          if (is_inside[to]) continue;
          if (!q.empty() and dist[to] > dist[q.front()]) q.
              push_back(to);
          else q.push_front(to);
          is_inside[to] = true;
        }
      }
    }
    return dist;
  }
  bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>, greater
        <>> q;
    dist = vector<T>(g.size(), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (q.size()) {
      auto [d, v] = q.top();
      q.pop();
      if (dist[v] < d) continue;
      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        cost += pot[v] - pot[to];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;
          q.emplace(dist[to], to);
          par_idx[to] = i, par[to] = v;
        }
      }
    }
    return dist[t] < inf;
  }

  pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
    vector<T> pot(g.size(), 0);
    pot = spfa(s); // mudar algoritmo de caminho minimo aqui

    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
      for (int i = 0; i < g.size(); i++)
        if (dist[i] < inf) pot[i] += dist[i];

      int mn_flow = flow - f, u = t;
      while (u != s){
        mn_flow = min(mn_flow,
```

```
        g[par[u]][par_idx[u]].cap - g[par[u]][par_idx[u]].
            flow);
      u = par[u];
    }

    ret += pot[t] * mn_flow;

    u = t;
    while (u != s) {
      g[par[u]][par_idx[u]].flow += mn_flow;
      g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
      u = par[u];
    }

    f += mn_flow;
  }

  return make_pair(f, ret);
}

// Opcional: retorna as arestas originais por onde passa flow
    = cap
vector<pair<int,int>> recover() {
  vector<pair<int,int>> used;
  for (int i = 0; i < g.size(); i++) for (edge e : g[i])
    if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
  return used;
}
};
```

LowerBoundMaxFlow.cpp

Description: precisa do dinitz add(a, b, l, r) adiciona aresta de a pra b, onde precisa passar f de fluxo, $l <= f <= r$. add(a, b, c) adiciona aresta de a pra b com capacidade c.
Time: Mesma complexidade do dinitz

d41d8c, 36 lines

```
struct lb_max_flow : dinitz {
  vector<int> d;
  lb_max_flow(int n) : dinitz(n + 2), d(n, 0) {}
  void add(int a, int b, int l, int r) {
    d[a] -= l;
    d[b] += l;
    dinitz::add(a, b, r - l);
  }
  void add(int a, int b, int c) {
    dinitz::add(a, b, c);
  }
  bool has_circulation() {
    int n = d.size();

    ll cost = 0;
    for (int i = 0; i < n; i++) {
      if (d[i] > 0) {
        cost += d[i];
        dinitz::add(n, i, d[i]);
      } else if (d[i] < 0) {
        dinitz::add(i, n+1, -d[i]);
      }
    }

    return (dinitz::max_flow(n, n+1) == cost);
  }
  bool has_flow(int src, int snk) {
    dinitz::add(snk, src, INF);
    return has_circulation();
  }
  ll max_flow(int src, int snk) {
    if (!has_flow(src, snk)) return -1;
    dinitz::F = 0;
    return dinitz::max_flow(src, snk);
  }
};
```

PushRelabel.cpp

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

d41d8c, 51 lines

```
typedef ll Flow;
struct Edge {
  int dest, back;
  Flow f, c;
};

struct PushRelabel {
  vector<vector<Edge>> g;
  vector<Flow> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

  void add_edge(int s, int t, Flow cap, Flow rcap=0) {
    if (s == t) return;
    Edge a = {t, sz(g[t]), 0, cap};
    Edge b = {s, sz(g[s]), 0, rcap};
    g[s].push_back(a);
    g[t].push_back(b);
  }

  void add_flow(Edge& e, Flow f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  Flow maxflow(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    trav(e, g[s]) add_flow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          trav(e, g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          add_flow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
};
```

7.2 Matching

Hungarian.cpp

Description: Resolve o problema de assignment (matriz n por n). Colocar os valores da matriz em 'a' (pode < 0). assignment() retorna um par com o valor do assignment minimo, e a coluna escolhida por cada linha
Time: $\mathcal{O}\left(n^3\right)$.

d41d8c, 42 lines

```
template<typename T> struct hungarian {
  int n;
  vector<vector<T>> a;
  vector<T> u, v;
  vector<int> p, way;
  T inf;

  hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1), way(n+1) {
    a = vector<vector<T>>(n, vector<T>(n));
    inf = numeric_limits<T>::max();
  }
  pair<T, vector<int>> assignment() {
    for (int i = 1; i <= n; i++) {
      p[0] = i;
      int j0 = 0;
      vector<T> minv(n+1, inf);
      vector<int> used(n+1, 0);
      do {
        used[j0] = true;
        int i0 = p[j0], j1 = -1;
        T delta = inf;
        for (int j = 1; j <= n; j++) if (!used[j]) {
          T cur = a[i0-1][j-1] - u[i0] - v[j];
          if (cur < minv[j]) minv[j] = cur, way[j] = j0;
          if (minv[j] < delta) delta = minv[j], j1 = j;
        }
        for (int j = 0; j <= n; j++)
          if (used[j]) u[p[j]] += delta, v[j] -= delta;
          else minv[j] -= delta;
        j0 = j1;
      } while (p[j0] != 0);
      do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
      } while (j0);
    }
    vector<int> ans(n);
    for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
    return make_pair(-v[0], ans);
  }
};
```

Blossom.cpp

Description: Maximum matching in general graph.
Time: $\mathcal{O}\left(n^3\right)$. If bipartite, $\mathcal{O}\left(nm\right)$ and doesn't need 'contract'

d41d8c, 77 lines

```
vector<int> g[MAX];
int match[MAX]; // match[i] = com quem i esta matchzado ou -1
int n, pai[MAX], base[MAX], vis[MAX];
queue<int> q;

void contract(int u, int v, bool first = 1) {
  static vector<bool> bloss;
  static int l;
  if (first) {
    bloss = vector<bool>(n, 0);
    vector<bool> teve(n, 0);
    int k = u; l = v;
    while (1) {
      teve[k = base[k]] = 1;
      if (match[k] == -1) break;
      k = pai[match[k]];
    }
    while (!teve[l = base[l]]) l = pai[match[l]];
  }
  while (base[u] != l) {
    bloss[base[u]] = bloss[base[match[u]]] = 1;
```

```
    pai[u] = v;
    v = match[u];
    u = pai[match[u]];
  }
  if (!first) return;
  contract(v, u, 0);
  for (int i = 0; i < n; i++) if (bloss[base[i]]) {
    base[i] = l;
    if (!vis[i]) q.push(i);
    vis[i] = 1;
  }
}

int getpath(int s) {
  for (int i = 0; i < n; i++) base[i] = i, pai[i] = -1, vis[i]
      = 0;
  vis[s] = 1; q = queue<int>(); q.push(s);
  while (q.size()) {
    int u = q.front(); q.pop();
    for (int i : g[u]) {
      if (base[i] == base[u] or match[u] == i) continue;
      if (i == s or (match[i] != -1 and pai[match[i]] != -1))
        contract(u, i);
      else if (pai[i] == -1) {
        pai[i] = u;
        if (match[i] == -1) return i;
        i = match[i];
        vis[i] = 1; q.push(i);
      }
    }
  }
  return -1;
}

int blossom() {
  int ans = 0;
  memset(match, -1, sizeof(match));
  for (int i = 0; i < n; i++) if (match[i] == -1)
    for (int j : g[i]) if (match[j] == -1) {
      match[i] = j;
      match[j] = i;
      ans++;
      break;
    }
  for (int i = 0; i < n; i++) if (match[i] == -1) {
    int j = getpath(i);
    if (j == -1) continue;
    ans++;
    while (j != -1) {
      int p = pai[j], pp = match[p];
      match[p] = j;
      match[j] = p;
      j = pp;
    }
  }
  return ans;
}
```

7.3 Trees

LinkCutTree.cpp

d41d8c, 208 lines

```
const int N = 1e5 + 9;

struct node {
  int p = 0, c[2] = {0, 0}, pp = 0;
  bool flip = 0;
  int sz = 0, ssz = 0, vsz = 0; // sz -> aux tree size, ssz =
       subtree size in rep tree, vsz = virtual tree size
  long long val = 0, sum = 0, lazy = 0, subsum = 0, vsum = 0;
  node() {}
  node(int x) {
    val = x; sum = x;
    sz = 1; lazy = 0;
    ssz = 1; vsz = 0;
    subsum = x; vsum = 0;
  }
};
struct LCT {
  vector<node> t;
  LCT() {}
  LCT(int n) : t(n + 1) {}

  // <independant splay tree code>
  int dir(int x, int y) { return t[x].c[1] == y; }
  void set(int x, int d, int y) {
    if (x) t[x].c[d] = y, pull(x);
    if (y) t[y].p = x;
  }
  void pull(int x) {
    if (!x) return;
    int &l = t[x].c[0], &r = t[x].c[1];
    push(l); push(r);
    t[x].sum = t[l].sum + t[r].sum + t[x].val;
    t[x].sz = t[l].sz + t[r].sz + 1;
    t[x].ssz = t[l].ssz + t[r].ssz + t[x].vsz + 1;
    t[x].subsum = t[l].subsum + t[r].subsum + t[x].vsum + t[x].
        val;
  }
  void push(int x) {
    if (!x) return;
    int &l = t[x].c[0], &r = t[x].c[1];
    if (t[x].flip) {
      swap(l, r);
      if (l) t[l].flip ^= 1;
      if (r) t[r].flip ^= 1;
      t[x].flip = 0;
    }
    if (t[x].lazy) {
      t[x].val += t[x].lazy;
      t[x].sum += t[x].lazy * t[x].sz;
      t[x].subsum += t[x].lazy * t[x].ssz;
      t[x].vsum += t[x].lazy * t[x].vsz;
      if (l) t[l].lazy += t[x].lazy;
      if (r) t[r].lazy += t[x].lazy;
      t[x].lazy = 0;
    }
  }
  void rotate(int x, int d) {
    int y = t[x].p, z = t[y].p, w = t[x].c[d];
    swap(t[x].pp, t[y].pp);
    set(y, !d, w);
    set(x, d, y);
    set(z, dir(z, y), x);
  }
  void splay(int x) {
    for (push(x); t[x].p;) {
      int y = t[x].p, z = t[y].p;
      push(z); push(y); push(x);
      int dx = dir(y, x), dy = dir(z, y);
      if (!z) rotate(x, !dx);
      else if (dx == dy) rotate(y, !dx), rotate(x, !dx);
      else rotate(x, dy), rotate(x, dx);
    }
  }
  // </independant splay tree code>

  // making it a root in the rep. tree
  void make_root(int u) {
    access(u);
    int l = t[u].c[0];
    t[l].flip ^= 1;
    swap(t[l].p, t[l].pp);
    t[u].vsz += t[l].ssz;
    t[u].vsum += t[l].subsum;
    set(u, 0, 0);
  }
  // make the path from root to u a preferred path
  // returns last path-parent of a node as it moves up the tree
  int access(int _u) {
    int last = _u;
    for (int v = 0, u = _u; u; u = t[v = u].pp) {
      splay(u); splay(v);
      t[u].vsz -= t[v].ssz;
      t[u].vsum -= t[v].subsum;
      int r = t[u].c[1];
      t[u].vsz += t[r].ssz;
      t[u].vsum += t[r].subsum;
      t[v].pp = 0;
      swap(t[r].p, t[r].pp);
      set(u, 1, v);
      last = u;
    }
    splay(_u);
    return last;
  }
  void link(int u, int v) { // u -> v
    // assert(!connected(u, v));
    make_root(v);
    access(u); splay(u);
    t[v].pp = u;
    t[u].vsz += t[v].ssz;
    t[u].vsum += t[v].subsum;
  }
  void cut(int u) { // cut par[u] -> u, u is non root vertex
    access(u);
    assert(t[u].c[0] != 0);
    t[t[u].c[0]].p = 0;
    t[u].c[0] = 0;
    pull(u);
  }
  // parent of u in the rep. tree
  int get_parent(int u) {
    access(u); splay(u); push(u);
    u = t[u].c[0]; push(u);
    while (t[u].c[1]) {
      u = t[u].c[1]; push(u);
    }
    splay(u);
    return u;
  }
  // root of the rep. tree containing this node
  int find_root(int u) {
    access(u); splay(u); push(u);
    while (t[u].c[0]) {
      u = t[u].c[0]; push(u);
    }
    splay(u);
    return u;
  }
  bool connected(int u, int v) {
    return find_root(u) == find_root(v);
  }
  // depth in the rep. tree
  int depth(int u) {
    access(u); splay(u);
    return t[u].sz;
  }
```

```
  int lca(int u, int v) {
    // assert(connected(u, v));
    if (u == v) return u;
    if (depth(u) > depth(v)) swap(u, v);
    access(v);
    return access(u);
  }
  int is_root(int u) {
    return get_parent(u) == 0;
  }
  int component_size(int u) {
    return t[find_root(u)].ssz;
  }
  int subtree_size(int u) {
    int p = get_parent(u);
    if (p == 0) {
      return component_size(u);
    }
    cut(u);
    int ans = component_size(u);
    link(p, u);
    return ans;
  }
  long long component_sum(int u) {
    return t[find_root(u)].subsum;
  }
  long long subtree_sum(int u) {
    int p = get_parent(u);
    if (p == 0) {
      return component_sum(u);
    }
    cut(u);
    long long ans = component_sum(u);
    link(p, u);
    return ans;
  }
  // sum of the subtree of u when root is specified
  long long subtree_query(int u, int root) {
    int cur = find_root(u);
    make_root(root);
    long long ans = subtree_sum(u);
    make_root(cur);
    return ans;
  }
  // path sum
  long long query(int u, int v) {
    int cur = find_root(u);
    make_root(u); access(v);
    long long ans = t[v].sum;
    make_root(cur);
    return ans;
  }
  void upd(int u, int x) {
    access(u); splay(u);
    t[u].val += x;
  }
  // add x to the nodes on the path from u to v
  void upd(int u, int v, int x) {
    int cur = find_root(u);
    make_root(u); access(v);
    t[v].lazy += x;
    make_root(cur);
  }
}t[2];
```

Block-Cut-Tree.cpp
Description: Block cut tree
Time: $\mathcal{O}(n+m)$
d41d8c, 65 lines

```
struct Bct {
    int T;
    vector<int> tin, low, stk, art, id, splits;
    vector<vector<int>> adj, g, comp, up;
    int n, sz, m;
    void build(int _n, int _m) {
        n = _n, m = _m;
        adj.resize(n);
    }
    void add_edge(int u, int v) {
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }
    void dfs(int u, int p) {
        low[u] = tin[u] = ++T;
        stk.emplace_back(u);
        for (auto v : adj[u]) {
            if (tin[v] == -1) {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if (low[v] >= tin[u]) {
                    int x;
                    sz++;
                    do {
                        assert(stk.size());
                        x = stk.back();
                        stk.pop_back();
                        comp[x].emplace_back(sz);
                    } while (x != v);
                    comp[u].emplace_back(sz);
                }
            } else if (v != p) {
                low[u] = min(low[u], tin[v]);
            }
        }
    }
    inline bool is_articulation_point(int u) { return art[id[u
        ]]; }
    inline int number_of_splits(int u) { return splits[id[u]];
         }
    void work() {
        T = sz = 0;
        stk.clear();
        tin.resize(n, -1);
        comp.resize(n);
        low.resize(n);
        for (int i = 0; i < n; i++)
            if (tin[i] == -1) dfs(i, 0);
        art.resize(sz + n + 1);
        splits.resize(n + sz + 1, 1);
        id.resize(n);
        g.resize(sz + n + 1);
        for (int i = 0; i < n; i++) {
            if ((int)comp[i].size() > 1) {
                id[i] = ++sz;
                art[id[i]] = 1;
                splits[id[i]] = (int)comp[i].size();
                for (auto u : comp[i]) {
                    g[id[i]].emplace_back(u);
                    g[u].emplace_back(id[i]);
                }
            } else if (comp[i].size()) {
                id[i] = comp[i][0];
            }
        }
    }
};
```

7.4 Math

7.4.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--, mat[b][b]++` (and `mat[b][a]--, mat[a][a]++` if G is undirected). Remove the ith 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).

7.4.2 Erdős–Gallai theorem

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

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

Geometry (8)

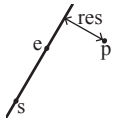
8.1 Geometric primitives

Point.h
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)
d41d8c, 28 lines

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

lineDistance.h

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

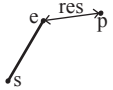


"Point.h" d41d8c, 4 lines

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

SegmentDistance.h

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



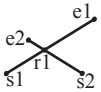
Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```

"Point.h" d41d8c, 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.



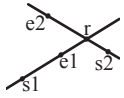
Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```

"Point.h", "OnSegment.h" d41d8c, 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" d41d8c, 8 lines

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

sideOf.h

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

Usage:
```
bool left = sideOf(p1,p2,q)==1;
```

"Point.h" d41d8c, 9 lines

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

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

OnSegment.h

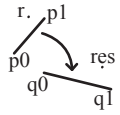
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" d41d8c, 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" d41d8c, 6 lines

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

Angle.h

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

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

d41d8c, 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" d41d8c, 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" d41d8c, 13 lines

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

CirclePolygonIntersection.h

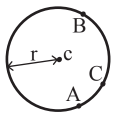
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" d41d8c, 19 lines

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

circumcircle.h

Description:



The circumcirle 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" d41d8c, 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" d41d8c, 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" d41d8c, 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" d41d8c, 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

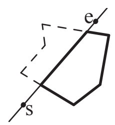
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" d41d8c, 9 lines

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

PolygonCut.h

Description:



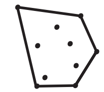
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" d41d8c, 13 lines

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

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" d41d8c, 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--; 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" d41d8c, 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" d41d8c, 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 a line.
Time: $\mathcal{O}(\log n)$

"Point.h" d41d8c, 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;
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" d41d8c, 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;
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

"Point.h" d41d8c, 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ...}, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h" d41d8c, 88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
```

```
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

8.5 3D

PolyhedronVolume.h

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

d41d8c, 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.

d41d8c, 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*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: $\mathcal{O}\left(n^2\right)$

"Point3D.h" d41d8c, 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);
  return FS;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

d41d8c, 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);
}
```

Strings (9)

AhoCorasick.cpp

Description: query retorna o somatorio do numero de matches de todas as stringuinhas na stringona

Time: insert in $\mathcal{O}\left(|s| \log SIGMA\right)$.

d41d8c, 40 lines

```
namespace aho {
  map<char, int> to[MAX];
  int link[MAX], idx, term[MAX], exit[MAX], sobe[MAX];

  void insert(string& s) {
    int at = 0;
    for (char c : s) {
      auto it = to[at].find(c);
      if (it == to[at].end()) at = to[at][c] = ++idx;
      else at = it->second;
    }
    term[at]++, sobe[at]++;
  }
#warning nao esquece de chamar build() depois de inserir
  void build() {
    queue<int> q;
    q.push(0);
    link[0] = exit[0] = -1;
    while (q.size()) {
      int i = q.front(); q.pop();
      for (auto [c, j] : to[i]) {
        int l = link[i];
        while (l != -1 and !to[l].count(c)) l = link[l];
        link[j] = l == -1 ? 0 : to[l][c];
        exit[j] = term[link[j]] ? link[j] : exit[link[j]];
        if (exit[j]+1) sobe[j] += sobe[exit[j]];
        q.push(j);
      }
    }
  }
  int query(string& s) {
    int at = 0, ans = 0;
    for (char c : s){
      while (at != -1 and !to[at].count(c)) at = link[at];
      at = at == -1 ? 0 : to[at][c];
      ans += sobe[at];
    }
    return ans;
  }
}
```

SuffixArray.cpp

Description: kasai recebe o suffix array e calcula $lcp[i]$, o lcp entre $s[sa[i], ..., n-1]$ e $s[sa[i+1], .., n-1]$.
Time: $\mathcal{O}(N \log N)$. Kasai in $\mathcal{O}(N)$.

d41d8c, 34 lines

```
vector<int> suffix_array(string s) {
  s += "$";
  int n = s.size(), N = max(n, 260);
  vector<int> sa(n), ra(n);
  for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

  for(int k = 0; k < n; k ? k *= 2 : k++) {
    vector<int> nsa(sa), nra(n), cnt(N);

    for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n, cnt[ra[
        i]]++;
    for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
    for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

    for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i
        ]] !=
      ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
    ra = nra;
    if (ra[sa[n-1]] == n-1) break;
  }
  return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
  int n = s.size(), k = 0;
  vector<int> ra(n), lcp(n);
  for (int i = 0; i < n; i++) ra[sa[i]] = i;

  for (int i = 0; i < n; i++, k -= !!k) {
    if (ra[i] == n-1) { k = 0; continue; }
    int j = sa[ra[i]+1];
    while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
    lcp[ra[i]] = k;
  }
  return lcp;
}
```

Manacher.cpp

Description: manacher recebe um vetor de T e retorna o vetor com tamanho dos palindromos ret[2*i] = tamanho do maior palindromo centrado em i ret[2*i+1] = tamanho maior palindromo centrado em i e i+1
Time: all in $\mathcal{O}(N)$.

d41d8c, 43 lines

```
template<typename T> vector<int> manacher(const T& s) {
  int l = 0, r = -1, n = s.size();
  vector<int> d1(n), d2(n);
  for (int i = 0; i < n; i++) {
    int k = i > r ? 1 : min(d1[l+r-i], r-i);
    while (i+k < n && i-k >= 0 && s[i+k] == s[i-k]) k++;
    d1[i] = k--;
    if (i+k > r) l = i-k, r = i+k;
  }
  l = 0, r = -1;
  for (int i = 0; i < n; i++) {
    int k = i > r ? 0 : min(d2[l+r-i+1], r-i+1); k++;
    while (i+k <= n && i-k >= 0 && s[i+k-1] == s[i-k]) k++;
    d2[i] = --k;
    if (i+k-1 > r) l = i-k, r = i+k-1;
  }
  vector<int> ret(2*n-1);
  for (int i = 0; i < n; i++) ret[2*i] = 2*d1[i]-1;
  for (int i = 0; i < n-1; i++) ret[2*i+1] = 2*d2[i+1];
  return ret;
}
```

```
// verifica se a string s[i..j] eh palindromo
template<typename T> struct palindrome {
  vector<int> man;

  palindrome(const T& s) : man(manacher(s)) {}
  bool query(int i, int j) {
    return man[i+j] >= j-i+1;
  }
};

// tamanho do maior palindromo que termina em cada posicao
template<typename T> vector<int> pal_end(const T& s) {
  vector<int> ret(s.size());
  palindrome<T> p(s);
  ret[0] = 1;
  for (int i = 1; i < s.size(); i++) {
    ret[i] = min(ret[i-1]+2, i+1);
    while (!p.query(i-ret[i]+1, i)) ret[i]--;
  }
  return ret;
}
```

Lyndon.cpp

Description: Duval algorithm to find the Lyndon factorization of a string. Also contains a function to find the minimum cyclic shift of a string.
Time: All $\mathcal{O}(n)$

d41d8c, 35 lines

```
vector<string> duval(string const &s) {
    int n = s.size();
    int i = 0;
    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}

string min_cyclic_shift(string s) {
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) i += j - k;
    }
    return s.substr(ans, n / 2);
}
```

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

d41d8c, 23 lines

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

d41d8c, 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: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

d41d8c, 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

TernarySearch.h

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

d41d8c, 11 lines

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

LIS.h

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

d41d8c, 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 <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$

d41d8c, 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: $\mathcal{O}((N + (hi - lo))\log N)$

d41d8c, 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^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^(1 << b)];` computes all sums of subsets.

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 $[0, 2b)$.

d41d8c, 8 lines

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

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.

d41d8c, 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 - 480;
  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.

d41d8c, 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*) {}
```