Competições de Algoritmos e Programação UFRJ

Federal University of Rio de Janeiro

Todo mundo adora o Chris

Felipe Chen, Vinícius Lettiéri e Lucas Melick

2023 ICPC South America/Brazil First Phase

September 2, 2023

Contest (1)

Makefile
8 lines

```
CXX = g++
CXXFLAGS = -std=c++17 -O2 -Wall -Wextra -pedantic -Wshadow -
     Wformat=2 -Wfloat-equal -Wconversion -Wlogical-op -Wshift-
     overflow=2 -Wduplicated-cond -Wcast-qual -Wcast-align -Wno
     -unused-result -Wno-sign-conversion
DEBUGFLAGS = -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC -DLOCAL
      -fsanitize=address -fsanitize=undefined -fno-sanitize-
     recover=all -fstack-protector -D_FORTIFY_SOURCE=2

DEBUG = false
ifeq ($(DEBUG),true)
  CXXFLAGS += $(DEBUGFLAGS)
endif
```

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
```

hash-cpp.sh
5 lines

```
# Hashes a file, ignoring all whitespace, comments and defines.
     Use for
# verifying that code was correctly typed.
# First do: chmod +x ./hash-cpp.sh
# ./hash-cpp.sh *.cpp start end
sed -n $2','$3' p' $1 | sed '/^#w/d' | cpp -dD -P -
     fpreprocessed | tr -d '[:space:]' | md5sum |cut -c-6
```

Mathematics (2)

2.1 Equations

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

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

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

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 Recurrences

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

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

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

2.3 Trigonometry

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

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

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

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

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

2.4 Geometry

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

Pick's: A polygon on an integer grid strictly containing i lattice points and having b lattice points on the boundary has area $i+\frac{b}{2}-1$. (Nothing similar in higher dimensions)

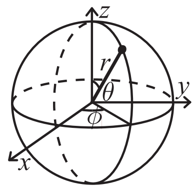
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x=\frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x=-\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x=1+\tan^2 x \qquad \frac{d}{dx}\arctan x=\frac{1}{1+x^2}$$
$$\int\tan ax=-\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax=\frac{\sin ax-ax\cos ax}{a^2}$$
$$\int e^{-x^2}=\frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int xe^{ax}dx=\frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

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

Green's theorem:
Let C be a positive, smooth, simple curve. D is a region bounded by C.

$$\oint_C (Pdx + Qdy) = \int\int_D (\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y})$$

To calculate area, $\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} = 1$, usually, picking $Q = \frac{1}{2}x$ and $P = -\frac{1}{2}y$ suffice.

Then we have

$$\frac{1}{2}\oint_C xdy - \frac{1}{2}\oint_C ydx$$

Line integral:
C given by $x = x(t), y = y(t), t \in [a, b]$, then

$$\oint_C f(x, y)ds = \int_a^b f(x(t), y(t))ds$$

where, $ds = \sqrt{(\frac{dx}{dt})^2 + (\frac{dy}{dt})^2}dt$ or $\sqrt{(1 + (\frac{dy}{dx})^2}dx$

2.5.1 XOR sum

$$\bigoplus_{x=0}^{n-1} x = \{0, n-1, 1, n\}[n \bmod 4]$$

$$\bigoplus_{x=l}^{r-1} x = \bigoplus_{a=0}^{r-1} a \oplus \bigoplus_{b=0}^{l-1} b$$

2.6 Sums

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

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

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

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

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

2.7 Series

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

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

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

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

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

2.8 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

$\mathrm{Bin}(n, p)$ 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 wich yields success with probability p is $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

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

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

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

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

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

2.8.2 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 $\mathrm{Exp}(\lambda)$, $\lambda > 0$.

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ 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 absorbing chain if
1. there is at least one absorbing state and
2. it is possible to go from any state to at least one absorbing state in a finite number of steps.
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)

order-statistic-tree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element.
Time: $\mathcal{O}(\log N)$

<bits/extc++.h> acfa21, 19 lines

```
template <typename K, typename V, typename Comp = std::less<K>>
using ordered_map = __gnu_pbds::tree<
  K, V, Comp,
  __gnu_pbds::rb_tree_tag,
  __gnu_pbds::tree_order_statistics_node_update
>;

template <typename K, typename Comp = std::less<K>>
using ordered_set = ordered_map<K, __gnu_pbds::null_type, Comp
    >;

void example() {
  ordered_set<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1); // num strictly smaller
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

dsu.h

Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$

7d5db8, 14 lines

```
struct UF {
  vector<int> e;
  UF(int n) : e(n, -1) {}
  bool same_set(int a, int b) { return find(a) == find(b); }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
  bool unite(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return 0;
    if (e[a] > e[b]) swap(a, b);
    e[a] += e[b]; e[b] = a;
    return 1;
  }
};
```

bipartite-dsu.h

Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$

07774d, 31 lines

```
struct DSU {
  vector<int> p, rk, color, bipartite;
  DSU(int n) : p(n), rk(n), color(n), bipartite(n, 1) {
    iota(p.begin(), p.end(), 0);
  };
  int find(int u) {
    if (u == p[u]) return u;
    int v = find(p[u]);
    color[u] ^= color[p[u]];
    return p[u] = v;
  }
  int find_color(int u) {
    find(u);
    return color[u];
  }
  // check if it doesn't create an odd cycle
  bool can(int u, int v) {
    return find(u) != find(v) || color[u] != color[v];
  }
  void unite(int u, int v) {
    int pu = find(u), pv = find(v);
    if (pu == pv) {
      if (color[u] == color[v]) bipartite[pu] = false;
      return;
    }
    if (rk[pu] < rk[pv]) swap(pu, pv);
    if (color[u] == color[v]) color[pv] ^= 1;
    p[pv] = pu, rk[pu] += (rk[pu] == rk[pv]);
    if (not bipartite[pv]) bipartite[pu] = false;
  }
};
```

dsu-rollback.h

Description: Disjoint-set data structure with undo.
Usage: `int t = uf.time(); ...; uf.rollback(t);`
Time: $\mathcal{O}(\log(N))$

7ddf1d, 21 lines

```
struct RollbackUF {
  vector<int> e; vector<pair<int,int>> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return st.size(); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool unite(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

monotonic-queue.h

Description: Structure that supports all operations of a queue and get the minimum/maximum active value in the queue. Useful for sliding window 1D and 2D. For 2D problems, you will need to pre-compute another matrix, by making a row-wise traversal, and calculating the min/max value beginning in each cell. Then you just make a column-wise traverse as they were each an independent array.
Time: $\mathcal{O}(1)$

da8783, 41 lines

```
template<typename T> struct monotonic_queue {
  vector<T> as, aas;
  vector<T> bs, bbs;
  void reserve(int N) {
    as.reserve(N); aas.reserve(N);
    bs.reserve(N); bbs.reserve(N);
  }
  void reduce() {
    while (!bs.empty()) {
      as.push_back(bs.back());
      aas.push_back(aas.empty() ? bs.back() : (bs.back() * aas.
          back()));
      bs.pop_back(); bbs.pop_back();
    }
  }
  T get() {
    if (as.empty()) reduce();
    return (bbs.empty() ? aas.back() : (aas.back() * bbs.back()
        ));
  }
  bool empty() const { return (as.empty() && bs.empty()); }
  int size() const { return int(as.size()) + int(bs.size()); }
  T front() {
    if (as.empty()) reduce();
    return as.back();
  }
  void push(const T& val) {
    bs.push_back(val);
    bbs.push_back(bbs.empty() ? val : (bbs.back() * val));
  }
  void pop() {
    if (as.empty()) reduce();
    as.pop_back();
    aas.pop_back();
  }
};

struct affine_t {
  int64_t b, c;
```

```
  affine_t operator*(affine_t rhs) {
    return {(rhs.b * b) % M, (rhs.b * c + rhs.c) % M};
  }
};
```

point-context.h

Description: Examples of Segment Tree

3f5b19, 36 lines

```
struct seg_node { // bbfc07
  int val;
  int mi, ma;
  seg_node() : mi(INT_MAX), ma(INT_MIN), val(0) {}
  seg_node(int x) : mi(x), ma(x), val(x) {}
  void merge(const seg_node& l, const seg_node& r) {
    val = l.val + r.val;
    mi = min(l.mi, r.mi);
    ma = max(l.ma, r.ma);
  }
  void update(int x) {
    mi = ma = val = x;
  }
  bool acc_min(int& acc, int x) const {
    if (x >= mi) return true;
    if (acc > mi) acc = mi;
    return false;
  }
  bool acc_max(int& acc, int x) const {
    if (x <= ma) return true;
    if (acc < ma) acc = ma;
    return false;
  }
};

// 1 + min of [a, N) <= x
auto find_min_right = [&](segtree<seg_node>& sg, int a, int x)
    -> int {
  int acc = INT_MAX;
  return sg.find_first(a, N, &seg_node::acc_min, acc, x);
};

// max of [0, a) >= x
auto find_max_left = [&](segtree<seg_node>& sg, int a, int x)
    -> int {
  int acc = INT_MIN;
  return sg.find_last(0, a, &seg_node::acc_max, acc, x);
};
```

rec-lazy-segtree.h

Description: Segment Tree with Lazy update (half-open interval).

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

fc41d6, 62 lines

```
template<class T> struct segtree_range {
  int N; vector<T> ts;
  segtree_range() {}
  explicit segtree_range(int N_) : segtree_range(vector<T>(N_,
      T(0))) {}
  template<class Q> explicit segtree_range(const vector<Q>& A)
      {
    const int N_ = int(A.size());
    N = (1 << __lg(2*N_-1)); ts.resize(2*N);
    for (int i = 0; i < N_; ++i) at(i) = T(A[i]);
    build();
  }
  T& at(int a) { return ts[a + N]; }
  void build() { for (int a = N; --a; ) merge(a); }
  inline void push(int a) { ts[a].push(ts[2*a], ts[2*a+1]); }
  inline void merge(int a) { ts[a].merge(ts[2*a], ts[2*a+1]); }
  template<class Op, class E, class F, class... Args>
  auto query(int v, int l, int r, int a, int b, Op op, E e, F f
      , Args&&... args) {
    if (l >= b || r <= a) return e();
    if (l >= a && r <= b) return (ts[v].*f)(args...);
    int m = (l + r)/2;
    push(v);
    return op(query(2*v, l, m, a, b, op, args...), query(2*v+1,
         m, r, a, b, op, args...));
  }
  template<class Op, class E, class F, class... Args>
  auto query(int a, int b, Op op, E e, F f, Args&&... args) {
    return query(1, 0, N, a, b, op, e, f, args...);
  }
  T query(int v, int l, int r, int a, int b) {
    if (l >= b || r <= a) return T();
    if (l >= a && r <= b) return ts[v];
    int m = (l + r)/2;
    push(v); T t;
    t.merge(query(2*v, l, m, a, b), query(2*v+1, m, r, a, b));
    return t;
  }
  T query(int a, int b) { return query(1, 0, N, a, b); }
  template<class F, class... Args> void update(int v, int l,
      int r, int a, int b, F f, Args&&... args) {
    if (l >= b || r <= a) return;
    if (l >= a && r <= b && (ts[v].*f)(args...)) return;
    int m = (l + r)/2;
    push(v);
    update(2*v, l, m, a, b, f, args...);
    update(2*v+1, m, r, a, b, f, args...);
    merge(v);
  }
  template<class F, class... Args>
  void update(int a, int b, F f, Args&&... args) {
    update(1, 0, N, a, b, f, args...);
  }
  template<class F, class... Args> int find_first(int v, int l,
       int r, int a, int b, F f, Args&&... args) {
    if (l >= b || r <= a || !(ts[v].*f)(args...)) return -1;
    if (l + 1 == r) return l;
    int m = (l + r)/2;
    push(v);
    int cur = find_first(2*v, l, m, a, b, f, args...);
    if (cur == -1) cur = find_first(2*v+1, m, r, a, b, f, args
        ...);
    return cur;
  }
  template<class F, class... Args>
  int find_first(int a, int b, F f, Args&&... args) {
    return find_first(1, 0, N, a, b, f, args...);
  }
};
```

lazy-context.h

Description: Examples of Segment Tree with Lazy update

6555fa, 194 lines

```
template<typename T = int64_t> struct seg_node {
  T val, lz_add, lz_set;
  int sz;
  bool to_set;
  seg_node(T n = 0) : val(n), lz_add(0), lz_set(0), sz(1),
      to_set(0) {}
  void push(seg_node& l, seg_node& r) {
    if (to_set) {
      l.assign(lz_set);
      r.assign(lz_set);
      lz_set = 0;
      to_set = false;
    }
    if (lz_add != 0) {
      l.add(lz_add);
      r.add(lz_add);
      lz_add = 0;
    }
  }
  void merge(const seg_node& l, const seg_node& r) {
    sz = l.sz + r.sz;
    val = l.val + r.val;
  }
  bool add(T v) {  // update range a[i] <- a[i] + v
    val += v * sz;
    lz_add += v; return true;
  }
  bool assign(T v) {   // update range a[i] <- v
    val = v * sz;
    lz_add = 0;
    lz_set = v;
    to_set = true; return true;
  }
  T get_sum() const { return val; } // sum a[l, r)
};

// update range a[i] <- a[i] + b * (i - s) + c
// assuming b and c are non zero, be careful
// get sum a[l, r)
template<typename T = int64_t> struct seg_node {
  T sum, lzB, lzC;
  int sz, idx;
  seg_node(int id = 0, T v = 0, int s = 0, T b = 0, T c = 0) :
    sum(v), lzB(b), lzC(c - s * b), idx(id), sz(1) {}
  void push(seg_node& l, seg_node& r) {
    l.add(lzB, lzC);
    r.add(lzB, lzC);
    lzB = lzC = 0;
  }
  void merge(const seg_node& l, const seg_node& r) {
    idx = min(l.idx, r.idx);
    sz = l.sz + r.sz;
    sum = l.sum + r.sum;
  }
  T sum_idx(T n) const { return n * (n + 1) / 2; }
  bool add(T b, T c) {
    sum += b * (sum_idx(idx + sz) - sum_idx(idx)) + sz * c;
    lzB += b;
    lzC += c; return true;
  }
  T get_sum() const { return sum; }
};

// update range a[i] <- b * a[i] + c
// get sum a[l, r)
struct seg_node {
  int sz; i64 sum, lzB, lzC;
  seg_node() : sz(1), sum(0), lzB(1), lzC(0) {}
  seg_node(i64 v) : sz(1), sum(v), lzB(1), lzC(0) {}
  void push(seg_node& l, seg_node& r) {
    l.add(lzB, lzC);
    r.add(lzB, lzC);
    lzB = 1, lzC = 0;
  }
  void merge(const seg_node& l, const seg_node& r) {
    sz = l.sz + r.sz;
    sum = l.sum + r.sum;
  }
  bool add(i64 b, i64 c) {
    sum = (b * sum + c * sz);
    lzB = (lzB * b);
    lzC = (lzC * b + c); return true;
```

```
  }
  i64 get_sum() const { return sum; }
};

// update range a[i] <- min(a[i], b);
// update range a[i] <- max(a[i], b);
// get val a[i]
struct seg_node {
  int mn, mx;
  int lz0, lz1;
  seg_node() : mn(INT_MAX), mx(INT_MIN), lz0(INT_MAX), lz1(INT_MIN) {}
  void push(seg_node& l, seg_node& r) {
    l.minimize(lz0);
    l.maximize(lz1);
    r.minimize(lz0);
    r.maximize(lz1);
    lz0 = INT_MAX;
    lz1 = INT_MIN;
  }
  void merge(const seg_node& l, const seg_node& r) {
    mn = min(l.mn, r.mn);
    mx = max(l.mx, r.mx);
  }
  bool minimize(int val) {
    mn = lz0 = min(lz0, val);
    mx = lz1 = min(lz0, lz1); return true;
  }
  bool maximize(int val) {
    mx = lz1 = max(lz1, val);
    mn = lz0 = max(lz0, lz1); return true;
  }
  pair<int, int> get() const { return {mx, mn}; }
};

template<typename T> struct lazy_t {
  T a, b, c;
  lazy_t() : a(0), b(-INF), c(+INF) {}
  lazy_t(T a, T b, T c) : a(a), b(b), c(c) {}
  void add(T val) {
    a += val, b += val, c += val;
  }
  void upd_min(T val) {
    if (b > val) b = val;
    if (c > val) c = val;
  }
  void upd_max(T val) {
    if (b < val) b = val;
    if (c < val) c = val;
  }
};
template<typename T = int64_t> struct seg_node {
  T mi, mi2, ma, ma2, sum;
  T cnt_mi, cnt_ma, sz;
  lazy_t<T> lz;
  seg_node() : mi(INF), mi2(INF), ma(-INF), ma2(-INF), sum(0), cnt_mi(0), cnt_ma(0), sz(0), lz() {}
  seg_node(T n) : mi(n), mi2(INF), ma(n), ma2(-INF), sum(n), cnt_mi(1), cnt_ma(1), sz(1), lz() {}
  void push(seg_node& l, seg_node& r) {
    if (!l.can_apply(lz) || !r.can_apply(lz)) return;
    lz = lazy_t<T>();
  }
  bool can_apply(const lazy_t<T>& f) {
    if (!add(f.a) || !upd_max(f.b) || !upd_min(f.c)) return false;
    return true;
  }
  void merge(const seg_node& l, const seg_node& r) {
    mi = min(l.mi, r.mi);
    mi2 = min((l.mi == mi) ? l.mi2 : l.mi, (r.mi == mi) ? r.mi2 : r.mi);
    cnt_mi = ((l.mi == mi) ? l.cnt_mi : 0) + ((r.mi == mi) ? r.cnt_mi : 0);
    ma = max(l.ma, r.ma);
    ma2 = max((l.ma == ma) ? l.ma2 : l.ma, (r.ma == ma) ? r.ma2 : r.ma);
    cnt_ma = ((l.ma == ma) ? l.cnt_ma : 0) + ((r.ma == ma) ? r.cnt_ma : 0);
    sum = l.sum + r.sum;
    sz = l.sz + r.sz;
  }
  bool add(T v) { // a_i = a_i + v
    if (v) {
      mi += v;
      if (mi2 < INF) mi2 += v;
      ma += v;
      if (ma2 > -INF) ma2 += v;
      sum += sz * v;
      lz.add(v);
    }
    return true;
  }
  bool upd_max(T v) { // a_i = max(a_i, v)
    if (v > -INF) {
      if (v >= mi2) return false;
      else if (v > mi) {
        if (ma == mi) ma = v;
        if (ma2 == mi) ma2 = v;
        sum += cnt_mi * (v - mi);
        mi = v;
        lz.upd_max(v);
      }
    }
    return true;
  }
  bool upd_min(T v) { // a_i = min(a_i, v)
    if (v < INF) {
      if (v <= ma2) return false;
      else if (v < ma) {
        if (ma == mi) mi = v;
        if (mi2 == ma) mi2 = v;
        sum -= cnt_ma * (ma - v);
        ma = v;
        lz.upd_min(v);
      }
    }
    return true;
  }
  T get_sum() const { return sum; } // sum a[l, r)
};
```

sparse-segtree.h

Description: Sparse Segment Tree with point update. Doesnt allocate storage for nodes with no data. Use BumpAllocator for better performance!

014f68, 39 lines

```
const int SZ = 1<<19;
template<class T> struct node_t {
  T delta = 0; node_t<T>* c[2];
  node_t() { c[0] = c[1] = nullptr; }
  void upd(int pos, T v, int L = 0, int R = SZ-1) { // add v
    if (L == pos && R == pos) { delta += v; return; }
    int M = (L + R)>>1;
    if (pos <= M) {
      if (!c[0]) c[0] = new node_t();
      c[0]->upd(pos, v, L, M);
    } else {
      if (!c[1]) c[1] = new node_t();
      c[1]->upd(pos, v, M+1, R);
    }
    delta = 0;
    for (int i = 0; i < 2; ++i) if (c[i]) delta += c[i]->delta;
  }
  T query(int lx, int rx, int L = 0, int R = SZ-1) { // query sum of segment
    if (rx < L || R < lx) return 0;
    if (lx <= L && R <= rx) return delta;
    int M = (L + R)>>1; T res = 0;
    if (c[0]) res += c[0]->query(lx, rx, L, M);
    if (c[1]) res += c[1]->query(lx, rx, M+1, R);
    return res;
  }
  void upd(int pos, node_t *a, node_t *b, int L = 0, int R = SZ-1) {
    if (L != R) {
      int M = (L + R)>>1;
      if (pos <= M) {
        if (!c[0]) c[0] = new node_t();
        c[0]->upd(pos, a ? a->c[0] : nullptr, b ? b->c[0] : nullptr, L, M);
      } else {
        if (!c[1]) c[1] = new node_t();
        c[1]->upd(pos,a ? a->c[1] : nullptr, b ? b->c[1] : nullptr, M+1, R);
      }
    }
    delta = (a ? a->delta : 0)+(b ? b->delta : 0);
  }
};
```

segtree-2d.h

Description: 2D Segment Tree.
Time: $\mathcal{O}\left(N \log^2 N\right)$ of memory, $\mathcal{O}\left(\log^2 N\right)$ per query

"sparse_seg_tree.h" 09098e, 25 lines

```
template<class T> struct Node {
  node_t<T> seg; Node* c[2];
  Node() { c[0] = c[1] = nullptr; }
  void upd(int x, int y, T v, int L = 0, int R = SZ-1) { // add v
    if (L == x && R == x) { seg.upd(y,v); return; }
    int M = (L+R)>>1;
    if (x <= M) {
      if (!c[0]) c[0] = new Node();
      c[0]->upd(x,y,v,L,M);
    } else {
      if (!c[1]) c[1] = new Node();
      c[1]->upd(x,y,v,M+1,R);
    }
    seg.upd(y,v); // only for addition
        // seg.upd(y,c[0]?&c[0]->seg:nullptr,c[1]?&c[1]->seg:nullptr);
  }
  T query(int x1, int x2, int y1, int y2, int L = 0, int R = SZ-1) { // query sum of rectangle
    if (x1 <= L && R <= x2) return seg.query(y1,y2);
    if (x2 < L || R < x1) return 0;
    int M = (L+R)>>1; T res = 0;
    if (c[0]) res += c[0]->query(x1, x2, y1, y2, L, M);
    if (c[1]) res += c[1]->query(x1, x2, y1, y2, M+1, R);
    return res;
  }
};
```

persistent-segtree.h

Description: Persistent implementation of a segment tree. This one compute the kth smallest element in a subarray $[a, b]$.

d277eb, 31 lines

```
struct segtree_t {
  struct snapshot {
    int cnt, linkl, linkr;
    snapshot() : cnt(0), linkl(0), linkr(0) {}
    snapshot(int _cnt, int l, int r) : cnt(_cnt), linkl(l),
         linkr(r) {}
  };
  int id;
  vector<snapshot> tree;
  segtree_t() {}
  segtree_t(int n) : id(1), tree(20*n) {}
  int update(int v, int l, int r, int x) {
    if (x < l || x > r) return v;
    if (l == r) {
      tree[id] = snapshot(1, 0, 0);
      return id++;
    }
    int m = (l + r) >> 1;
    int lx = update(tree[v].linkl, l, m, x);
    int rx = update(tree[v].linkr, m+1, r, x);
    tree[id] = snapshot(tree[lx].cnt + tree[rx].cnt, lx, rx);
    return id++;
  }
  int query(int a, int b, int l, int r, int k) { // kth
    if (l == r) return l;
    int m = (l + r) >> 1;
    int cnt = tree[tree[b].linkl].cnt - tree[tree[a].linkl].cnt
        ;
    if (k <= cnt)
      return query(tree[a].linkl, tree[b].linkl, l, m, k);
    return query(tree[a].linkr, tree[b].linkr, m+1, r, k-cnt);
  }
};
```

merge-sort-tree.h

Description: Build segment tree where each node stores a sorted version of the underlying range.
Time: $\mathcal{O}\left(\log^2 N\right)$

9216c7, 39 lines

```
struct merge_sort_tree {
  vector<int> v, ids;
  vector<vector<int>> tree;
  // f9aa52
  merge_sort_tree(vector<int> &v) : v(v), tree(4*(v.size()+1))
      {
    for(int i = 0; i < v.size(); ++i) ids.push_back(i);
    sort(ids.begin(), ids.end(), [&v](int i, int j) { return v[
        i] < v[j]; });
    build(1, 0, v.size()-1);
  }
  // 55ba58
  void build(int id, int left, int right) {
    if (left == right) tree[id].push_back(ids[left]);
    else {
      int mid = (left + right)>>1;
      build(id<<1, left, mid);
      build(id<<1|1, mid+1, right);
      tree[id] = vector<int>(right - left + 1);
      merge(tree[id<<1].begin(), tree[id<<1].end(),
        tree[id<<1|1].begin(), tree[id<<1|1].end(),
        tree[id].begin());
    }
  }
  // how many elements in this node have id in the range [a,b]
  int how_many(int id, int a, int b) {
    return (int)(upper_bound(tree[id].begin(), tree[id].end(),
        b)
      - lower_bound(tree[id].begin(), tree[id].end(), a));
  }
  // e575ae
  int query(int id, int left, int right, int a, int b, int x) {
    if (left == right) return v[tree[id].back()];
    int mid = (left + right)>>1;
    int lcount = how_many(id<<1, a, b);
    if (lcount >= x) return query(id<<1, left, mid, a, b, x);
    else return query(id<<1|1, mid+1, right, a, b, x - lcount);
  }
  int kth(int a, int b, int k) {
    return query(1, 0, v.size()-1, a, b, k);
  }
};
```

rmq.h

Description: Range Minimum/Maximum Queries on an array. Returns min(V[a], V[a + 1], ... V[b]) in constant time. Returns a pair that holds the answer, first element is the value and the second is the index.
Usage:
```
rmq_t<pair<int, int>> rmq(values);
// values is a vector of pairs {val(i), index(i)}
rmq.query(inclusive, exclusive);
rmq_t<pair<int, int>, greater<pair<int, int>>> rmq(values)
//max query
```
Time: $\mathcal{O}(|V|\log|V| + Q)$

8c53c5, 19 lines

```
template<typename T, typename Cmp=less<T>>
struct rmq_t : private Cmp {
  int N = 0;
  vector<vector<T>> table;
  const T& min(const T& a, const T& b) const { return Cmp::
      operator()(a, b) ? a : b; }
  rmq_t() {}
  rmq_t(const vector<T>& values) : N(int(values.size())), table
      (__lg(N) + 1) {
    table[0] = values;
    for (int a = 1; a < int(table.size()); ++a) {
      table[a].resize(N - (1 << a) + 1);
      for (int b = 0; b + (1 << a) <= N; ++b)
        table[a][b] = min(table[a-1][b], table[a-1][b + (1 << (
            a-1))]);
    }
  }
  T query(int a, int b) const {
    int lg = __lg(b - a);
    return min(table[lg][a], table[lg][b - (1 << lg) ]);
  }
};
```

fenwick-tree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$.

f79f29, 28 lines

```
template<typename T> struct FT { // 8b7639
  vector<T> s;
  FT(int n) : s(n) {}
  FT(const vector<T>& A) : s(A) {
    const int N = int(s.size());
    for (int a = 0; a < N; ++a) {
      if ((a | (a + 1)) < N) s[a | (a + 1)] += s[a];
    }
  }
  void update(int pos, T dif) { // a[pos] += dif
    for (; pos < (int)s.size(); pos |= pos + 1) s[pos] += dif;
  }
  T query(int pos) { // sum of values in [0, pos)
    T res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(T sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= (int)s.size() && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

fenwick-tree-2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Time: $\mathcal{O}\left(\log^2 N\right)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

"fenwick-tree.h" aebbdc, 25 lines

```
template<typename T> struct FT2 {
  vector<vector<int>> ys; vector<FT<T>> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < (int)ys.size(); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    for(auto &v : ys){
      sort(v.begin(), v.end());
      v.resize(unique(v.begin(),v.end()) - v.begin());
      ft.emplace_back(v.size());
    }
  }
  int ind(int x, int y) {
    return (int)(lower_bound(ys[x].begin(), ys[x].end(), y) -
        ys[x].begin()); }
  void update(int x, int y, T dif) {
    for (; x < ys.size(); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  T query(int x, int y) {
    T sum = 0;
    for (; x; x &= x - 1) sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

mo.h

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change `step` to add/remove the edge (a, c) and remove the initial `add` call (but keep `in`).
Time: $\mathcal{O}\left(N\sqrt{Q}\right)$

5ef29d, 49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vector<int> mo(vector<pair<int, int>> Q) { // d9247c
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vector<int> s(int(Q.size())), res = s;
#define K(x) pair<int, int>(x.first/blk, x.second ^ -(x.first/
    blk & 1))
  iota(s.begin(), s.end(), 0);
  sort(s.begin(), s.end(), [&](int s, int t){ return K(Q[s]) <
      K(Q[t]); });
  for (int qi : s) {
    auto q = Q[qi];
    while (L > q.first) add(--L, 0);
```

```
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}

vector<int> moTree(vector<array<int, 2>> Q, vector<vector<int
    >>& ed, int root=0){ // bbf891
  int N = int(ed.size()), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vector<int> s(int(Q.size())), res = s, I(N), L(N), R(N), in(N
      ), par(N);
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) -> void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
  iota(s.begin(), s.end(), 0);
  sort(s.begin(), s.end(), [&](int s, int t){ return K(Q[s]) <
      K(Q[t]); });
  for (int qi : s) for (int end = 0; end < 2; ++end) {
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
  else { add(c, end); in[c] = 1; } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
  return res;
}
```

line-container.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$

8b2ace, 29 lines

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

lichao.h

Description: Line Segments Li Chao Tree. Allows line add, segment add and point query.
Time: $\mathcal{O}(\log N)$ except for segment add $\mathcal{O}(\log^2 N)$

42896f, 62 lines

```
template<typename T, T L, T R>
struct lichao_t{
  static const T inf = numeric_limits<T>::max() / 2;
  bool first_best( T a, T b ){ return a < b; }
  T get_best( T a, T b ){  return first_best(a, b) ? a : b;  }
  struct line{ // 785930
    T m, b;
    T operator()( T x ){ return m*x + b; }
  };
  struct node{ // e92ef4
    line li;
    node *left, *right;
    node( line _li = {0, inf}): li(_li), left(nullptr), right(
        nullptr){}
    ~node(){
      delete left;
      delete right;
    }
  };
  node *root;
  lichao_t( line li = {0, inf} ): root ( new node(li) ) {}
  ~lichao_t(){ delete root; }
  T query( T x , node *cur , T l, T r){ // e3e758
    if(cur == nullptr) return inf;
    if(x < l || x > r) return inf;
    T mid = ( l + r ) >> 1;
    T ans =  cur->li(x);
    ans = get_best( ans , query(x, cur->left, l, mid) );
    ans = get_best( ans , query(x, cur->right, mid+1, r) );
    return ans;
  }
  T query( T x ){ return query( x, root, L, R ); }
  void add( line li, node *&cur, T l, T r){ // 0962ab
    if(cur == nullptr){
      cur = new node(li);
      return;
    }
    T mid = ( l + r ) >> 1;
    if( first_best( li(mid), cur->li(mid) ) )
      swap(li, cur->li);
    if( first_best( li(l), cur->li(l) ) )
      add(li, cur->left, l, mid);
    if( first_best( li(r), cur->li(r) ) )
      add(li, cur->right, mid + 1, r);
  }
  void add( T m, T b ){ add( {m, b}, root, L, R ); }
  void addSegment( line li, node *&cur, T l, T r, T lseg, T
      rseg){ // d1fcf2
    if(r < lseg || l > rseg) return;
    if(cur == nullptr) cur =  new node;
    if(lseg <= l && r <= rseg){
      add(li, cur, l, r);
      return;
    }
    T mid = ( l + r ) >> 1;
    if(l != r){
      addSegment(li, cur->left, l, mid, lseg, rseg);
      addSegment(li, cur->right, mid+1, r, lseg, rseg);
    }
  }
  void addSegment( T m, T b, T left, T right){
    addSegment( {m, b}, root, L, R, left, right);
  }
};
```

lichao-lazy.h

Description: Lazy Li Chao Tree. Allows line add, segment add, segment update and point query.
Time: $\mathcal{O}(\log N)$ except for segment add $\mathcal{O}(\log^2 N)$

5bf94a, 104 lines

```
template<typename T, T L, T R>
struct lichao_lazy{
  static const T inf = numeric_limits<T>::max() / 2;
  bool first_best( T a, T b ){ return a < b; }
  T get_best( T a, T b ){  return first_best(a, b) ? a : b;  }
  struct line{ // 88f949
    T m, b;
    T operator()( T x ){ return m*x + b; }
    void apply(line other){
      m += other.m;
      b += other.b;
    }
  };
  struct node{ // e6c99b
    line li, lazy;
    node *left, *right;
    node( line _li = {0, inf}): li(_li), lazy({0,0}), left(
        nullptr), right(nullptr){}
    void apply(line other){
      li.apply(other);
      lazy.apply(other);
    }
    ~node(){
      delete left;
      delete right;
    }
  };
  node *root;
  lichao_lazy( line li = {0, inf} ): root ( new node(li) ) {}
  ~lichao_lazy(){ delete root; }
  void propagateLazy(node *&cur){ // f09e7a
    if(cur == nullptr) return;
    if(cur->left == nullptr) cur->left = new node;
    if(cur->right == nullptr) cur->right = new node;
    cur->left->apply( cur-> lazy);
    cur->right->apply( cur-> lazy);
    cur->lazy = {0, 0};
  }
  T query( T x , node *cur , T l, T r){ // f56802
    if(x < l || x > r || l > r) return inf;
    if(cur == nullptr) return inf;
    T mid = ( l + r ) >> 1;
    if(l != r) propagateLazy(cur);
    T ans =  cur->li(x);
    ans = get_best( ans , query(x, cur->left, l, mid) );
    ans = get_best( ans , query(x, cur->right, mid+1, r) );
    return ans;
  }
  T query( T x ){ return query( x, root, L, R ); }
  void add( line li, node *&cur, T l, T r){ // 4191d1
    if(cur == nullptr){
      cur = new node(li);
      return;
    }
    T mid = ( l + r ) >> 1;
    propagateLazy(cur);
```

```
    if( first_best( li(mid), cur->li(mid) ) )
      swap(li, cur->li);
    if( first_best( li(l), cur->li(l) ) )
      add(li, cur->left, l, mid);
    if( first_best( li(r), cur->li(r) ) )
      add(li, cur->right, mid + 1, r);
  }
  void add( T m, T b ){ add( {m, b}, root, L, R ); }
  void propagateLine(node *&cur, T l, T r){ // 8d3255
    if(cur == nullptr) return;
    T mid = ( l + r ) >> 1;
    add(cur->li, cur->left, l, mid);
    add(cur->li, cur->right, mid+1, r);
    cur->li = {0, inf};
  }
  void addSegment( line li, node *&cur, T l, T r, T lseg, T
      rseg){ // 1a6dd3
    if(r < lseg || l > rseg) return;
    if(cur == nullptr) cur =  new node;
    if(lseg <= l && r <= rseg){
      add(li, cur, l, r);
      return;
    }
    T mid = ( l + r ) >> 1;
    if(l != r){
      propagateLazy(cur);
      addSegment(li, cur->left, l, mid, lseg, rseg);
      addSegment(li, cur->right, mid+1, r, lseg, rseg);
    }
  }
  void addSegment( T m, T b, T left, T right){
    addSegment( {m, b}, root, L, R, left, right);
  }
  void updateSegment( line li, node *&cur, T l, T r, T lseg, T
      rseg){ // cce50c
    if(r < lseg || l > rseg) return;
    if(cur == nullptr) cur =  new node;
    if(lseg <= l && r <= rseg){
      cur->apply(li);
      return;
    }
    T mid = ( l + r ) >> 1;
    propagateLazy(cur);
    propagateLine(cur, l, r);
    updateSegment(li, cur->left, l, mid, lseg, rseg);
    updateSegment(li, cur->right, mid+1, r, lseg, rseg);
  }
  void updateSegment( T m, T b, T left, T right){
    updateSegment( {m, b}, root, L, R, left, right);
  }
};
```

lichao-range.h

Description: Lazy Li Chao Tree. Allows line add, segment add, segment update (only linear coeficient) and range query.
Time: $\mathcal{O}(\log N)$ except for segment add $\mathcal{O}(\log^2 N)$

da0993, 120 lines

```
template<typename T, T L, T R>
struct lichao_range{
  static const T inf = numeric_limits<T>::max() / 2;
  static bool first_best( T a, T b ){ return a < b; }
  static T get_best( T a, T b ){  return first_best(a, b) ? a :
      b;  }
  struct line{ // 88f949
    T m, b;
    T operator()( T x ){ return m*x + b; }
    void apply(line other){
      m += other.m;
      b += other.b;
    }
  };
  struct node{ // 419efd
    line li, lazy;
    node *left, *right;
    T answer;
    node( line _li = {0, inf}): li(_li), lazy({0,0}), left(
        nullptr), right(nullptr), answer(inf){}
    void apply(T l, T r, line other){
      li.apply(other);
      lazy.apply(other);
      answer = get_best(inf, answer + other.b);
    }
    ~node(){
      delete left;
      delete right;
    }
  };
  node *root;
  lichao_range( line li = {0, inf} ): root ( new node(li) ) {}
  ~lichao_range(){ delete root; }
  void updateAnswer(node *&cur, T l, T r){ // 02ae1f
    if(cur == nullptr) return;
    cur->answer = inf;
    if(cur->left != nullptr) cur->answer = get_best(cur->answer
        , cur->left->answer);
    if(cur->right != nullptr) cur->answer = get_best(cur->
        answer, cur->right->answer);
    cur->answer = get_best(cur->answer, cur->li(l));
    cur->answer = get_best(cur->answer, cur->li(r));
  }
  void propagateLazy(node *&cur, T l, T r){ // 5da08d
    if(cur == nullptr) return;
    if(cur->left == nullptr) cur->left = new node;
    if(cur->right == nullptr) cur->right = new node;
    T mid = ( l + r ) >> 1;
    cur->left->apply( l, mid, cur-> lazy);
    cur->right->apply( mid+1, r, cur-> lazy);
    cur->lazy = {0, 0};
  }
  T query( node *cur , T l, T r, T lseg, T rseg){ // 72eb4e
    if(r < lseg || l > rseg) return inf;
    if(cur == nullptr) return inf;
    if(lseg <= l && r <= rseg) return cur->answer;
    T answer = get_best(cur->li(max(l, lseg)), cur->li(min(r,
        rseg)));
    if(l != r) propagateLazy(cur, l, r);
    T mid = ( l + r ) >> 1;
    answer = get_best(answer, query(cur->left, l, mid, lseg,
        rseg));
    answer = get_best(answer, query(cur->right, mid+1, r, lseg,
         rseg));
    updateAnswer(cur, l, r);
    return answer;
  }
  T query( T l, T r){ return query( root, L, R, l, r); }
  void add( line li, node *&cur, T l, T r){ // 74c963
    if(cur == nullptr){
      cur = new node(li);
      return;
    }
    T mid = ( l + r ) >> 1;
    propagateLazy(cur, l, r);
    if( first_best( li(mid), cur->li(mid) ) )
      swap(li, cur->li);
    if( first_best( li(l), cur->li(l) ) )
      add(li, cur->left, l, mid);
    if( first_best( li(r), cur->li(r) ) )
      add(li, cur->right, mid + 1, r);
    updateAnswer(cur, l, r);
  }
  void add( T m, T b ){ add( {m, b}, root, L, R ); }
  void propagateLine(node *&cur, T l, T r){ // 8d3255
    if(cur == nullptr) return;
    T mid = ( l + r ) >> 1;
    add(cur->li, cur->left, l, mid);
    add(cur->li, cur->right, mid+1, r);
    cur->li = {0, inf};
  }
  void addSegment( line li, node *&cur, T l, T r, T lseg, T
      rseg){ // 43e625
    if(r < lseg || l > rseg) return;
    if(cur == nullptr) cur =  new node;
    if(lseg <= l && r <= rseg){
      add(li, cur, l, r);
      return;
    }
    T mid = ( l + r ) >> 1;
    if(l != r){
      propagateLazy(cur, l, r);
      addSegment(li, cur->left, l, mid, lseg, rseg);
      addSegment(li, cur->right, mid+1, r, lseg, rseg);
    }
    updateAnswer(cur, l, r);
  }
  void addSegment( T m, T b, T left, T right){
    addSegment( {m, b}, root, L, R, left, right);
  }
  void updateSegment( T b, node *&cur, T l, T r, T lseg, T rseg
      ){ // ff8f3e
    if(r < lseg || l > rseg) return;
    if(cur == nullptr) cur =  new node;
    if(lseg <= l && r <= rseg){
      cur->apply(l, r, {0, b});
      return;
    }
    T mid = ( l + r ) >> 1;
    propagateLazy(cur, l, r);
    propagateLine(cur, l, r);
    updateSegment(b, cur->left, l, mid, lseg, rseg);
    updateSegment(b, cur->right, mid+1, r, lseg, rseg);
    updateAnswer(cur, l, r);
  }
  void updateSegment( T b, T left, T right){
    updateSegment( b , root, L, R, left, right);
  }
};
```

matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int> A(N, vector<int>(N));`

4c1476, 33 lines

```
template <typename T> struct Matrix : vector<vector<T>> {
  using vector<vector<T>>::vector;
  using vector<vector<T>>::size;
  int h() const { return int(size()); }
  int w() const { return int((*this)[0].size()); }
  Matrix operator*(const Matrix& r) const {
    assert(w() == r.h());
    Matrix res(h(), vector<T>(r.w()));
    for (int i = 0; i < h(); ++i) {
      for (int j = 0; j < r.w(); ++j) {
        for (int k = 0; k < w(); ++k) {
          res[i][j] += (*this)[i][k] * r[k][j];
        }
      }
    }
    return res;
```

```
  }
  friend vector<T> operator*(const Matrix<T>& A, const vector<T
      >& b) {
    int N = int(A.size()), M = int(A[0].size());
    vector<T> y(N);
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < M; ++j) y[i] += A[i][j] * b[j];
    return y;
  }
  Matrix& operator*=(const Matrix& r) { return *this = *this *
      r; }
  Matrix pow(int n) const {
    assert(h() == w());
    Matrix x = *this, r(h(), vector<T>(w()));
    for (int i = 0; i < h(); ++i) r[i][i] = T(1);
    while (n) { if (n & 1) r *= x; x *= x; n >>= 1; }
    return r;
  }
};
```

submatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}\left(N^2+Q\right)$ cd3f87, 13 lines

```
template<class T> struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = v.size(), C = v[0].size();
    p.assign(R+1, vector<T>(C+1));
    for (int r = 0; r < R; ++r)
      for (int c = 0; c < C; ++c)
        p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c
            ];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

wavelet.h

Description: Segment tree on values instead of indices.
Time: $\mathcal{O}(\log(n))$ 80ec5e, 130 lines

```
struct wavelet_t { // b26328
  struct BitVector { // space: 32N bits
    vector<int> _rank = {0};
    BitVector(vector<char> v = vector<char>()) {
      _rank.reserve(v.size() + 1);
      for (int d : v) _rank.push_back(_rank.back() + d);
    }
    int rank(bool f, int k) { return f ? _rank[k] : (k - _rank[
        k]); }
    int rank(bool f, int l, int r) { return rank(f, r) - rank(f
        , l); }
  };
  /*
    struct BitVector { // space: 1.5N bits
    vector<ull> v;
    vector<int> _rank;
    BitVector(vector<char> _v = vector<char>()) {
    int n = int(_v.size());
    v = vector<ull>((n + 63) / 64);
    _rank = vector<int>(v.size() + 1);
    for (int i = 0; i < n; i++) {
    if (_v[i]) {
    v[i / 64] |= 1ULL << (i % 64);
    _rank[i / 64 + 1]++;
    }
    }
    for (int i = 0; i < int(v.size()); i++) {
    _rank[i+1] += _rank[i];
    }
    }
    int rank(int k) {
    int a = _rank[k / 64];
    if (k % 64) a += __builtin_popcountll(v[k / 64] << (64 - k
        % 64));
    return a;
    }
    int rank(bool f, int k) { return f ? rank(k) : k - rank(k)
        ; }
    int rank(bool f, int l, int r) { return rank(f, r) - rank(
        f, l); }
    };
    */
  int n, lg = 1;
  vector<int> mid;
  vector<BitVector> data;
  wavelet_t(vector<int> v = vector<int>()) : n(int(v.size())) {
    int ma = 0;
    for (int x : v) ma = max(ma, x);
    while ((1 << lg) <= ma) lg++;
    mid = vector<int>(lg);
    data = vector<BitVector>(lg);
    for (int lv = lg - 1; lv >= 0; lv--) {
      vector<char> buf;
      vector<vector<int>> nx(2);
      for (int d : v) {
        bool f = (d & (1 << lv)) > 0;
        buf.push_back(f);
        nx[f].push_back(d);
      }
      mid[lv] = int(nx[0].size());
      data[lv] = BitVector(buf);
      v.clear();
      v.insert(v.end(), nx[0].begin(), nx[0].end());
      v.insert(v.end(), nx[1].begin(), nx[1].end());
    }
  }
  pair<int, int> succ(bool f, int a, int b, int lv) {
    int na = data[lv].rank(f, a) + (f ? mid[lv] : 0);
    int nb = data[lv].rank(f, b) + (f ? mid[lv] : 0);
    return {na, nb};
  }
  // count i, s.t. (a <= i < b) && (v[i] < u)
  int rank(int a, int b, int u) {
    if ((1 << lg) <= u) return b - a;
    int ans = 0;
    for (int lv = lg - 1; lv >= 0; lv--) {
      bool f = (u & (1 << lv)) > 0;
      if (f) ans += data[lv].rank(false, a, b);
      tie(a, b) = succ(f, a, b, lv);
    }
    return ans;
  }
  // k-th(0-indexed!) number in v[a..b]
  int select(int a, int b, int k) {
    int u = 0;
    for (int lv = lg - 1; lv >= 0; lv--) {
      int le = data[lv].rank(false, a, b);
      bool f = (le <= k);
      if (f) {
        u += (1 << lv);
        k -= le;
      }
      tie(a, b) = succ(f, a, b, lv);
    }
    return u;
  }
  // k-th(0-indexed!) largest number in v[a..b]
  int large_select(int a, int b, int k) {
    return select(a, b, b - a - k - 1);
  }
  // count i s.t. (a <= i < b) && (x <= v[i] < y)
  int count(int a, int b, int x, int y) {
    return rank(a, b, y) - rank(a, b, x);
  }
  // max v[i] s.t. (a <= i < b) && (v[i] < x)
  int pre_count(int a, int b, int x) {
    int cnt = rank(a, b, x);
    return cnt == 0 ? -1 : select(a, b, cnt - 1);
  }
  // min v[i] s.t. (a <= i < b) && (x <= v[i])
  int nxt_count(int a, int b, int x) {
    int cnt = rank(a, b, x);
    return cnt == b - a ? -1 : select(a, b, cnt);
  }
};

struct CompressWavelet { // 2447db
  wavelet_t wt;
  vector<int> v, vidx;
  int zip(int x) {
    return int(lower_bound(vidx.begin(), vidx.end(), x) - vidx.
        begin());
  }
  CompressWavelet(vector<int> _v = vector<int>()) : v(_v), vidx
      (v) {
    sort(vidx.begin(), vidx.end());
    vidx.erase(unique(vidx.begin(), vidx.end()), vidx.end());
    for (auto& d : v) d = zip(d);
    wt = Wavelet(v);
  }
  int rank(int a, int b, int u) { return wt.rank(a, b, zip(u));
      }
  int select(int a, int b, int k) { return vidx[wt.select(a, b,
      k)]; }
  int largest(int a, int b, int k) { return wt.large_select(a,
      b, k); }
  int count(int a, int b, int mi, int ma) { return wt.count(a,
      b, mi, ma); }
  int find_max(int a, int b, int x) { return wt.pre_count(a, b,
      x); }
  int find_min(int a, int b, int x) { return wt.nxt_count(a, b,
      x); }
};
```

range-color.h

Description: RangeColor structure, supports point queries and range updates, if C isn't int32_t change $freq$ to map
Time: $\mathcal{O}\left(\lg(L) * Q\right)$ d25555, 47 lines

```
template<class T = int64_t, class C = int32_t> struct
    RangeColor{
  struct Node{
    T left, right; C color;
    bool operator < (const Node &n) const{ return right < n.
        right; }
  };
  C minInf;
  set<Node> st;
  vector<T> freq;
  RangeColor(T first, T last, C maxColor, C iniColor = C(0)):
      minInf(first - T(1)), freq(maxColor + 1) {
```

```
    freq[iniColor] = last - first + T(1);
    st.insert({first, last, iniColor});
  }
  C query(T i) { //get color in position i
    auto p = st.upper_bound({T(0), i - T(1), minInf});
    return p->color;
  }
  void upd(T a, T b, C newColor) { //set newColor in [a, b]
    auto p = st.upper_bound({T(0), a - T(1), minInf});
    assert(p != st.end());
    T left = p->left, right = p->right;
    C old = p->color;
    freq[old] -= (right - left + T(1));
    p = st.erase(p);
    if (left < a){
      freq[old] += (a - left);
      st.insert({left, a - T(1), old});
    }
    if (b < right){
      freq[old] += (right - b);
      st.insert({b + T(1), right, old});
    }
    while ((p != st.end()) && (p->left <= b)){
      left = p->left, right = p->right;
      old = p->color;
      freq[old] -= (right - left + T(1));
      if (b < right){
        freq[old] += (right - b);
        st.erase(p);
        st.insert({b + T(1), right, old});
        break;
      } else  p = st.erase(p);
    }
    freq[newColor] += (b - a + T(1));
    st.insert({a, b, newColor});
  }
  T countColor(C x){ return freq[x]; }
};
```

implicit-treap.h

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

2d0d97, 96 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
struct node {
  int v, p, sz;
  node *l, *r;
  bool rev;
  node(int k) : v(k), p(rng()), l(nullptr), rev(0), r(nullptr),
       sz(0) {}
};
int sz(node *t) {
  if (t == nullptr) return 0;
  return t->sz;
}
void push(node *t) {
  if (t == nullptr) return;
  if (t->rev) {
    swap(t->l, t->r);
    if (t->l != nullptr) t->l->rev ^= t->rev;
    if (t->r != nullptr) t->r->rev ^= t->rev;
    t->rev = 0;
  }
}
void updsz(node *t) {
  if (t == nullptr) return;
  push(t); push(t->l); push(t->r);
  t->sz = sz(t->l) + sz(t->r) + 1;
}
void split(node *t, node *&l, node *&r, int k) { //k on left
  push(t);
  if (t == nullptr) l = r = nullptr;
  else if (k <= sz(t->l)) {
    split(t->l, l, t->l, k);
    r = t;
  }
  else {
    split(t->r, t->r, r, k-1-sz(t->l));
    l = t;
  }
  updsz(t);
}
void merge(node *&t, node *l, node *r) {
  push(l); push(r);
  if (l == nullptr) t = r;
  else if (r == nullptr) t = l;
  else if (l->p <= r->p) {
    merge(l->r, l->r, r);
    t = l;
  }
  else {
    merge(r->l, l, r->l);
    t = r;
  }
  updsz(t);
}
void add(node *&t, node *c, int k) {
  push(t);
  if (t == nullptr) t = c;
  else if (c->p >= t->p) {
    split(t, c->l, c->r, k);
    t = c;
  }
  else if (sz(t->l) >= k) add(t->l, c, k);
  else add(t->r, c, k-1-sz(t->l));
  updsz(t);
}
void del(node *&t, int k) {
  push(t);
  if (t == nullptr) return;
  if (sz(t->l) == k) merge(t, t->l, t->r);
  else if (sz(t->l) > k) del(t->l, k);
  else del(t->r, k);
  updsz(t);
}
void print(node *t) {
  if (r == nullptr) return;
  print(t->l);
  cout << t->v << ' ';
  print(t->r);
}

int main() {
  node *treap = nullptr;
  while(1) {
    int a;
    cin >> a;
    if (a == 1) {
      int c, d;
      cin >> c >> d;
      node *r = new node(d);
      add(treap, r, c);
    } else if (a == 2) {
      int d;
      cin >> d;
      del(treap, d);
    }
    print(treap);
  }
}
```

Numerical (4)

polynomial.h

84593c, 17 lines

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

poly-roots.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" 49396a, 23 lines

```
vector<double> poly_roots(Poly p, double xmin, double xmax) {
  if ((p.a).size() == 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(dr.begin(), dr.end());
  for(int i = 0; i < dr.size()-1; ++i) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign^(p(h) > 0)) {
      for(int it = 0; it < 60; ++it) { // 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;
}
```

poly-interpolate.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 $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

97a266, 13 lines

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

```
  double last = 0; temp[0] = 1;
  for(int k = 0; k < n; ++k) for(int i = 0; i < n; ++i) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

lagrange.h

Description: Lagrange interpolation over a finite field and some combo stuff
Time: $\mathcal{O}(N)$

"../number-theory/modular-arithmetic.h", "../number-theory/preparator.h" 84e6f3, 14 lines

```
template<typename T> struct interpolator_t {
  vector<T> pref, suff;
  interpolator_t(int N): pref(N), suff(N) {}
  T interpolate(const vector<T>& y, T x) {
    int N = int(y.size()); T res = 0;
    pref[0] = suff[N - 1] = 1;
    for (int i = 0; i+1 < N; ++i) pref[i+1] = pref[i] * (x-i);
    for (int i = N-1; i > 0; --i) suff[i-1] = suff[i] * (x-i);
    for (int i = 0, sgn = (N & 1 ? +1 : -1); i < N; ++i, sgn *=
         -1) {
      res += y[i] * sgn * pref[i] * suff[i] * invFac[i] *
             invFac[N-1-i];
    }
    return res;
  }
};
```

berlekamp-massey.h

Description: Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.
Usage: BerlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}
Time: $\mathcal{O}(N^2)$

"ModularArithmetic.h" 4c4a48, 19 lines

```
template <typename num>
vector<num> BerlekampMassey(const vector<num>& s) {
  int n = int(s.size()), L = 0, m = 0;
  vector<num> C(n), B(n), T;
  C[0] = B[0] = 1;
  num b = 1;
  for(int i = 0; i < n; i++) { ++m;
    num d = s[i];
    for (int j = 1; j <= L; j++) d += C[j] * s[i - j];
    if (d == 0) continue;
    T = C; num coef = d / b;
    for (int j = m; j < n; j++) C[j] -= coef * B[j - m];
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }
  C.resize(L + 1); C.erase(C.begin());
  for (auto& x : C) x = -x;
  return C;
}
```

linear-recurrence.h

Description: Bostan-Mori algorithm. Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: linear_rec({0, 1}, {1, 1}, k) // k'th Fibonacci number
Time: $\mathcal{O}(n \log n \log k)$

"../modular-arithmetic.h" e584bd, 18 lines

```
template<typename T>
T linear_rec(const vector<T>& S, const vector<T>& tr, ll K) {
  const int N = int(tr.size());
  vector<T> qs(N + 1); qs[0] = 1;
  for (int i = 0; i < N; ++i) qs[i + 1] = -tr[i];
  auto fs = fft.convolve(S, qs); fs.resize(N);
  for (; K; K /= 2) {
    auto qneg = qs;
    for (int i = 1; i <= N; i += 2) qneg[i] = -qneg[i];
    fs = fft.convolve(fs, qneg), qs = fft.convolve(qs, qneg);
    for (int i = 0; i < N; ++i) {
      fs[i] = fs[2 * i + (K & 1)];
      qs[i] = qs[2 * i];
    }
    qs[N] = qs[2*N]; fs.resize(N), qs.resize(N+1);
  }
  return fs[0];
}
```

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.

7bb98e, 7 lines

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

integrate-adaptive.h

Description: Fast integration using an adaptive Simpson's rule.
Usage: double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [&](double y) {
return quad(-1, 1, [&](double z) {
return x*x + y*y + z*z < 1; });});});

92dd79, 15 lines

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

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
  d c = (a + b) / 2;
  d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
  if (abs(T - S) <= 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
  return rec(f, a, b, eps, S(a, b));
}
```

gaussian-elimination.h

Time: $\mathcal{O}(\min(N, M)NM)$

"../data-structures/matrix.h" b33fca, 75 lines

```
template<typename T> struct gaussian_elimination {
  int N, M;
  Matrix<T> A, E;
  vector<int> pivot;
  int rank, nullity, sgn;
  gaussian_elimination(const Matrix<T>& A_) : A(A_) {
    N = A.size(), M = A[0].size();
    E = Matrix<T>(N, vector<T>(N));
    for (int i = 0; i < N; ++i) E[i][i] = 1;
    rank = 0, nullity = M, sgn = 0;
    pivot.assign(M, -1);
    for (int col = 0, row = 0; col < M && row < N; ++col) {
      int sel = -1;
      for (int i = row; i < N; ++i) {
        if (A[i][col] != 0) {
          sel = i;
          break;
        }
      }
      if (sel == -1) continue;
      if (sel != row) {
        sgn += 1;
        swap(A[sel], A[row]);
        swap(E[sel], E[row]);
      }
      for (int i = 0; i < N; ++i) {
        if (i == row) continue;
        T c = A[i][col] / A[row][col];
        for (int j = col; j < M; ++j)
          A[i][j] -= c * A[row][j];
        for (int j = 0; j < N; ++j)
          E[i][j] -= c * E[row][j];
      }
      pivot[col] = row++;
      ++rank, --nullity;
    }
  }
  pair<bool, vector<T>> solve(vector<T> b, bool reduced = false
      ) const {
    if (reduced == false) b = E * b;
    vector<T> x(M);
    for (int j = 0; j < M; ++j) {
      if (pivot[j] == -1) continue;
      x[j] = b[pivot[j]] / A[pivot[j]][j];
      b[pivot[j]] = 0;
    }
    for (int i = 0; i < N; ++i)
      if (b[i] != 0) return {false, x};
    return {true, x};
  }
  vector<vector<T>> kernel_basis() const {
    vector<vector<T>> basis;
    vector<T> e(M);
    for (int j = 0; j < M; ++j) {
      if (pivot[j] != -1) continue;
      e[j] = 1;
      auto y = solve(A * e, true).second;
      e[j] = 0, y[j] = -1;
      basis.push_back(y);
    }
    return basis;
  }
  Matrix<T> inverse() const {
    assert(N == M); assert(rank == N);
    Matrix<T> res(N, vector<T>(N));
    vector<T> e(N);
    for (int i = 0; i < N; ++i) {
      e[i] = 1;
      auto x = solve(e).second;
      for (int j = 0; j < N; ++j)
        res[j][i] = x[j];
      e[i] = 0;
    }
    return res;
  }
};
```

linear-solver-z2.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns true, or false if no solutions. Last column of a is b. c is the rank.
Time: $\mathcal{O}(n^2 m)$

7a24e1, 26 lines

```
typedef bitset<2010> bs;
bool gauss(vector<bs> a, bs& ans, int n) {
  int m = int(a.size()), c = 0;
  bs pos; pos.set();
  for (int j = n-1, i; j >= 0; --j) {
    for (i = c; i < m; ++i)
      if (a[i][j]) break;
    if (i == m) continue;
    swap(a[c], a[i]);
    i = c++; pos[j] = 0;
    for (int k = 0; k < m; ++k)
      if (a[k][j] && k != i)
        a[k] ^= a[i];
  }
  ans = pos;
  for(int i = 0; i < m; ++i) {
    int ac = 0;
    for (int j = 0; j < n ; ++j) {
      if (!a[i][j]) continue;
      if (!pos[j]) pos[j] = 1, ans[j] = ac^a[i][n];
      ac ^= ans[j];
    }
    if (ac != a[i][n]) return false;
  }
  return true;
}
```

char-poly.h

Description: Calculates the characteristic polynomial of a matrix. $\sum_{k=0}^{n} p(k)(-1)^{n-k}$
Time: $\mathcal{O}(N^3)$ and div-free is $\mathcal{O}(N^4)$

30bd65, 55 lines

```
// det(x I + a)
template<class T> vector<T> char_poly(const vector<vector<T>>&
    a) { // ed7ab1
  const int N = int(a.size()); auto b = a;
  for (int j = 0; j < N - 2; ++j) {
    for (int i = j + 1; i < N; ++i) {
      if (b[i][j]) {
        swap(b[j + 1], b[i]);
        for (int k = 0; k < N; ++k) swap(b[k][j + 1], b[k][i]);
        break;
      }
    }
    if (b[j + 1][j]) {
      const T r = 1 / b[j + 1][j];
      for (int i = j + 2; i < N; ++i) {
        const T s = r * b[i][j];
        for (int q = j; q < N; ++q) b[i][q] -= s * b[j + 1][q];
        for (int p = 0; p < N; ++p) b[p][j + 1] += s * b[p][i];
      }
    }
  }
  // fss[i] := det(x I_i + b[0..i][0..i])
  vector<vector<T>> fss(N + 1);
  fss[0] = {1};
  for (int i = 0; i < N; ++i) {
    fss[i + 1].assign(i + 2, 0);
    for (int k = 0; k <= i; ++k) fss[i + 1][k + 1] = fss[i][k];
    for (int k = 0; k <= i; ++k) fss[i + 1][k] += b[i][i] * fss
        [i][k];
    T q = 1;
    for (int j = i - 1; j >= 0; --j) {
      q *= -b[j + 1][j];
      const T s = q * b[j][i];
      for (int k = 0; k <= j; ++k) fss[i + 1][k] += s * fss[j][
          k];
    }
  }
  return fss[N];
}
// det(x I + a), division free
template<class T> vector<T> char_poly_div_free(const vector<
    vector<T>>& a) { // 693758
  const int N = int(a.size());
  vector<T> ps(N + 1, 0);
  ps[N] = 1;
  for (int h = N - 1; h >= 0; --h) {
    vector<vector<T>> sub(N, vector<T>(h + 1, 0));
    for (int i = N; i >= 1; --i)
      sub[i - 1][h] += ps[i];
    for (int i = N - 1; i >= 1; --i) for (int u = 0; u <= h; ++
        u) {
      for (int v = 0; v < h; ++v)
        sub[i - 1][v] -= sub[i][u] * a[u][v];
    }
    for (int i = N - 1; i >= 1; --i) for (int u = 0; u <= h; ++
        u) {
      ps[i] += sub[i][u] * a[u][h];
    }
  }
  return ps;
}
```

simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$.
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case. WARNING- segfaults on empty (size 0) max cx st Ax<=b, x>=0 do 2 phases; 1st check feasibility; 2nd check boundedness and ans

c3703c, 39 lines

```
vector<double> simplex(vector<vector<double>> A, vector<double>
     b, vector<double> c) {
  int n = A.size(), m = 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];
  double result = D[n][m];
  return x; // ans: D[n][m]
}
```

tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$

d0855f, 26 lines

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T> &super,
    const vector<T> &sub, vector<T> b) {
  int n = b.size(); vector<int> tr(n);
  for(int i = 0; i < n-1; ++i) {
    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;
}
```

polyominoes.h

Description: Generate all fixed polyominoes with at most n squares. poly[x] gives the polyominoes with x squares. Takes less than a sec if $n < 10$, around 2s if $n = 10$ and around 6s if $n = 11$.

580a1b, 34 lines

```cpp
const int LIM = 11;
using pii = pair<int,int>;
int dx[] = {0, 1, 0, -1};
int dy[] = {1, 0, -1, 0};
vector<vector<pii>> poly[LIM + 1];
void generate(int n = LIM){
  poly[1] = { { { 0 , 0 } } };
  for(int i = 2 ; i <= n; ++i) {
    set<vector<pii>> cur_om;
    for(auto &om : poly[i-1]) for(auto &p : om)
      for(int d = 0; d < 4; ++d) {
        int x = p.first + dx[d];
        int y = p.second + dy[d];
        if( ! binary_search(om.begin(), om.end(), pii(x,y)) ) {
          pii m = min(om[0], {x, y});
          pii new_cell(x - m.first, y - m.second);
          vector<pii> norm;
          norm.reserve(i);
          bool new_in = false;
          for(pii &c : om) {
            pii cur(c.first - m.first, c.second - m.second);
            if( ! new_in && cur > new_cell ) {
              new_in = true;
              norm.push_back(new_cell);
            }
            norm.push_back(cur);
          }
          if( ! new_in ) norm.push_back(new_cell);
          cur_om.insert(norm);
        }
      }
    poly[i].assign(cur_om.begin(), cur_om.end());
  }
}
```

4.1 Fourier transforms

fast-fourier-transform.h

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

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

d90ac9, 158 lines

```cpp
inline int nxt_pow2(int s) { return 1 << (s > 1 ? 32 -
     __builtin_clz(s-1) : 0); }
template <typename dbl> struct cplx {
  dbl x, y;
  cplx(dbl x_ = 0, dbl y_ = 0) : x(x_), y(y_) { }
  friend cplx operator+(cplx a, cplx b) { return cplx(a.x + b.x
       , a.y + b.y); }
  friend cplx operator-(cplx a, cplx b) { return cplx(a.x - b.x
       , a.y - b.y); }
  friend cplx operator*(cplx a, cplx b) { return cplx(a.x * b.x
       - a.y * b.y, a.x * b.y + a.y * b.x); }
  friend cplx conj(cplx a) { return cplx(a.x, -a.y); }
  friend cplx inv(cplx a) { dbl n = (a.x*a.x+a.y*a.y); return
       cplx(a.x/n,-a.y/n); }
};

template <typename T>
struct root_of_unity {};
template <typename dbl> struct root_of_unity<cplx<dbl>> {
  static cplx<dbl> f(int k) {
    static const dbl PI = acos(-1);
    dbl a = 2*PI/k;
    return cplx<dbl>(cos(a),sin(a));
  }
};

using M0 = modnum<998244353U>; // g = 3
using M1 = modnum<897581057U>; // g = 3
using M2 = modnum<880803841U>; // g = 26
using M3 = modnum<985661441U>; // g = 3
using M4 = modnum<943718401U>; // g = 7
using M5 = modnum<935329793U>; // g = 3
using M6 = modnum<918552577U>; // g = 5

constexpr unsigned primitive_root(unsigned M) {
  if (M == 880803841U) return 26U;
  else if (M == 943718401U) return 7U;
  else if (M == 918552577U) return 5U;
  else return 3U;
}
template<unsigned MOD> struct root_of_unity<modnum<MOD>> {
  static constexpr modnum<MOD> g0 = primitive_root(MOD);
  static modnum<MOD> f(int K) {
    assert((MOD-1)%K == 0);
    return g0.pow((MOD-1)/K);
  }
};

template<typename T> struct FFT {
  vector<T> rt; vector<int> rev;
  FFT() : rt(2, T(1)) {}
  void init(int N) {
    N = nxt_pow2(N);
    if (N > int(rt.size())) {
      rev.resize(N); rt.reserve(N);
      for (int a = 0; a < N; ++a) {
        rev[a] = (rev[a/2] | ((a&1)*N)) >> 1;
      }
      for (int k = int(rt.size()); k < N; k *= 2) {
        rt.resize(2*k);
        T z = root_of_unity<T>::f(2*k);
        for (int a = k/2; a < k; ++a) {
          rt[2*a] = rt[a];
          rt[2*a+1] = rt[a] * z;
        }
      }
    }
  }
  void fft(vector<T>& xs, bool inverse) const {
    int N = int(xs.size());
    int s = __builtin_ctz(int(rev.size())/N);
    if (inverse) reverse(xs.begin() + 1, xs.end());
    for (int a = 0; a < N; ++a) {
      if (a < (rev[a] >> s))
        swap(xs[a], xs[rev[a] >> s]);
    }
    for (int k = 1; k < N; k *= 2) {
      for (int a = 0; a < N; a += 2*k) {
        int u = a, v = u + k;
        for (int b = 0; b < k; ++b, ++u, ++v) {
          T z = rt[b + k] * xs[v];
          xs[v] = xs[u] - z;
          xs[u] = xs[u] + z;
        }
      }
    }
    if (inverse) {
      for (int a = 0; a < N; ++a)
        xs[a] = xs[a] * inv(T(N));
    }
  }
  vector<T> convolve(vector<T> as, vector<T> bs) {
    int N = int(as.size()), M = int(bs.size());
    int K = N + M - 1, S = nxt_pow2(K); init(S);
    if (min(N, M) <= 64) {
      vector<T> res(K);
      for (int u = 0; u < N; ++u)
        for (int v = 0; v < M; ++v)
          res[u + v] = res[u + v] + as[u] * bs[v];
      return res;
    } else {
      as.resize(S), bs.resize(S);
      fft(as, false); fft(bs, false);
      for (int i = 0; i < S; ++i) as[i] = as[i] * bs[i];
      fft(as, true); as.resize(K);
      return as;
    }
  }
};

FFT<M0> FFT0; FFT<M1> FFT1;
FFT<M2> FFT2; FFT<M3> FFT3;
FFT<M4> FFT4; FFT<M5> FFT5; FFT<M6> FFT6;

// M0 M1 = 896005221510021121 (> 4.48 * 10^{17}, > 2^58)
// M0 M1 M2 = 789204840662082423367925761 (> 7.892 * 10^26, > 2
     ^89)
// M0 M3 M4 M5 M6 =
     797766583174034668024539679147517452591562753 (> 7.977 *
     10^44, > 2^149)

// T = {unsigned, unsigned long long, modnum<M>}
template<class T, unsigned M0, unsigned M1, unsigned M2>
T garner(modnum<M0> a0, modnum<M1> a1, modnum<M2> a2) {
  static const modnum<M1> INV_M0_M1 = modnum<M1>(M0).inv();
  static const modnum<M2> INV_M0M1_M2 = (modnum<M2>(M0) * M1).
       inv();
  const modnum<M1> b1 = INV_M0_M1 * (a1 - a0.x);
  const modnum<M2> b2 = INV_M0M1_M2 * (a2 - (modnum<M2>(b1.x) *
       M0 + a0.x));
  return (T(b2.x) * M1 + b1.x) * M0 + a0.x;
}
template<class T, unsigned M0, unsigned M1, unsigned M2,
     unsigned M3, unsigned M4>
T garner(modnum<M0> a0, modnum<M1> a1, modnum<M2> a2, modnum<M3
     > a3, modnum<M4> a4) {
  static const modnum<M1> INV_M0_M1 = modnum<M1>(M0).inv();
  static const modnum<M2> INV_M0M1_M2 = (modnum<M2>(M0) * M1).
       inv();
  static const modnum<M3> INV_M0M1M2_M3 = (modnum<M3>(M0) * M1
       * M2).inv();
  static const modnum<M4> INV_M0M1M2M3_M4 = (modnum<M4>(M0) *
       M1 * M2 * M3).inv();
  const modnum<M1> b1 = INV_M0_M1 * (a1 - a0.x);
  const modnum<M2> b2 = INV_M0M1_M2 * (a2 - (modnum<M2>(b1.x) *
       M0 + a0.x));
  const modnum<M3> b3 = INV_M0M1M2_M3 * (a3 - ((modnum<M3>(b2.x
       ) * M1 + b1.x) * M0 + a0.x));
  const modnum<M4> b4 = INV_M0M1M2M3_M4 * (a4 - (((modnum<M4>(
       b3.x) * M2 + b2.x) * M1 + b1.x) * M0 + a0.x));
  return (((T(b4.x) * M3 + b3.x) * M2 + b2.x) * M1 + b1.x) * M0
       + a0.x;
}

// results must be in [-448002610255888384, 448002611254132736]
vector<long long> convolve(const vector<long long>& as, const
     vector<long long>& bs) {
```

```
  static constexpr unsigned M0 = M0::M, M1 = M1::M;
  static const modnum<M1> INV_M0_M1 = modnum<M1>(M0).inv();
  if (as.empty() || bs.empty()) return {};
  const int len_as = int(as.size()), len_bs = int(bs.size());
  vector<modnum<M0>> as0(len_as), bs0(len_bs);
  for (int i = 0; i < len_as; ++i) as0[i] = as[i];
  for (int i = 0; i < len_bs; ++i) bs0[i] = bs[i];
  const vector<modnum<M0>> cs0 = FFT0.convolve(as0, bs0);
  vector<modnum<M1>> as1(len_as), bs1(len_bs);
  for (int i = 0; i < len_as; ++i) as1[i] = as[i];
  for (int i = 0; i < len_bs; ++i) bs1[i] = bs[i];
  const vector<modnum<M1>> cs1 = FFT1.convolve(as1, bs1);
  vector<long long> cs(len_as + len_bs - 1);
  for (int i = 0; i < len_as + len_bs - 1; ++i) {
    const modnum<M1> d1 = INV_M0_M1 * (cs1[i] - cs0[i].x);
    cs[i] = (d1.x > M1 - d1.x)
      ? (-1ULL - (static_cast<unsigned long long>(M1 - 1U - d1.x) * M0 + (M0 - 1U - cs0[i].x)))
      : (static_cast<unsigned long long>(d1.x) * M0 + cs0[i].x);
  }
  return cs;
}
```

fast-subset-transform.h

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

5b9574, 16 lines

```
void FST(vector<int> &a, bool inv) {
  for (int n = a.size(), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) for(int j = i; j < i+step; ++j) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);                   // XOR
    }
  }
  if (inv) for(auto &x : a) x /= a.size(); // XOR only
}
vector<int> conv(vector<int> a, vector<int> b) {
  FST(a, 0); FST(b, 0);
  for(int i = 0; i < a.size(); ++i) a[i] *= b[i];
  FST(a, 1); return a;
}
```

poly-998244353.h

"finite-field-fft.h", "../number-theory/mod-sqrt.h", "../number-theory/preparator.h"

b9d4bf, 242 lines

```
using num = modnum<998244353U>;

FFT<998244353U, 3U, 23> fft_data;

template<unsigned M> struct Poly : public vector<modnum<M>> {
  Poly() {}
  explicit Poly(int n) : vector<modnum<M>>(n) {}
  Poly(const vector<modnum<M>> &vec) : vector<modnum<M>>(vec) {}
  Poly(std::initializer_list<modnum<M>> il) : vector<modnum<M>>(il) {}
  int size() const { return vector<modnum<M>>::size(); }
  num at(long long k) const { return (0 <= k && k < size()) ? (*this)[k] : 0U; }
  int ord() const { for (int i = 0; i < size(); ++i) if (int((*this)[i])) return i; return -1; }
  int deg() const { for (int i = size(); --i >= 0;) if (int((*this)[i])) return i; return -1; }
  Poly mod(int n) const { return Poly(vector<modnum<M>>(this->data(), this->data() + min(n, size()))); }
  friend std::ostream &operator<<(std::ostream &os, const Poly &fs) {
    os << "[";
    for (int i = 0; i < fs.size(); ++i) { if (i > 0) os << ", "; os << fs[i]; }
    return os << "]";
  }
  Poly &operator+=(const Poly &fs) { // d36be
    if (size() < fs.size()) this->resize(fs.size());
    for (int i = 0; i < fs.size(); ++i) (*this)[i] += fs[i];
    return *this;
  }
  Poly &operator-=(const Poly &fs) { // 1f585
    if (size() < fs.size()) this->resize(fs.size());
    for (int i = 0; i < fs.size(); ++i) (*this)[i] -= fs[i];
    return *this;
  }
  Poly &operator*=(const Poly &fs) { // 24a99
    if (this->empty() || fs.empty()) return *this = {};
    *this = fft_data.convolve(*this, fs);
    return *this;
  }
  Poly &operator*=(const num &a) { // ea9fb
    for (int i = 0; i < size(); ++i) (*this)[i] *= a;
    return *this;
  }
  Poly &operator/=(const num &a) { // 71618
    const num b = a.inv();
    for (int i = 0; i < size(); ++i) (*this)[i] *= b;
    return *this;
  }
  Poly &operator/=(const Poly &fs) { // 291cd
    auto ps = fs;
    if (size() < ps.size()) return *this = {};
    int s = int(size()) - int(ps.size()) + 1;
    int nn = 1; for (; nn < s; nn <<= 1) {}
    reverse(this->begin(), this->end());
    reverse(ps.begin(), ps.end());
    this->resize(nn); ps.resize(nn);
    ps = ps.inv();
    *this = *this * ps;
    this->resize(s); reverse(this->begin(), this->end());
    return *this;
  }
  Poly &operator%=(const Poly& fs) { // d6a38
    if (size() >= fs.size()) {
      Poly Q = (*this / fs) * fs;
      this->resize(fs.size() - 1);
      for (int x = 0; x < int(size()); ++x) (*this)[x] -= Q[x];
    }
    while (size() && this->back() == 0) this->pop_back();
    return *this;
  }
  Poly inv() const { // c47df7
    if (this->empty()) return {};
    Poly b({(*this)[0].inv()}), fs;
    b.reserve(2 * int(this->size()));
    while (b.size() < this->size()) {
      int len = 2 * int(b.size());
      b.resize(2 * len, 0);
      if (int(fs.size()) < 2 * len) fs.resize(2 * len, 0);
      fill(fs.begin(), fs.begin() + 2 * len, 0);
      copy(this->begin(), this->begin() + min(len, int(this->size())), fs.begin());
      fft_data.fft(b);
      fft_data.fft(fs);
      for (int x = 0; x < 2*len; ++x) b[x] = b[x] * (2 - fs[x] * b[x]);
      fft_data.inverse_fft(b);
      b.resize(len);
    }
    b.resize(this->size()); return b;
  }
  Poly differential() const { // 0b718
    if (this->empty()) return {};
    Poly f(max(size() - 1, 1));
    for (int x = 1; x < size(); ++x) f[x - 1] = x * (*this)[x];
    return f;
  }
  Poly integral() const { // 71d33
    if (this->empty()) return {};
    Poly f(size() + 1);
    for (int x = 0; x < size(); ++x) f[x + 1] = invs[x + 1] * (*this)[x];
    return f;
  }
  Poly log() const { // 6a365
    if (this->empty()) return {};
    Poly f = (differential() * inv()).integral();
    f.resize(size()); return f;
  }
  Poly exp() const { // 25174b
    Poly f = {1};
    if (this->empty()) return f;
    while (f.size() < size()) {
      int len = min(f.size() * 2, size());
      f.resize(len);
      Poly d(len);
      copy(this->begin(), this->begin() + len, d.begin());
      Poly g = d - f.log();
      g[0] += 1;
      f *= g;
      f.resize(len);
    }
    return f;
  }
  Poly pow(int N) const { // 48fee9
    Poly b(size());
    if (N == 0) { b[0] = 1; return b; }
    int p = 0;
    while (p < size() && (*this)[p] == 0) ++p;
    if (1LL * N * p >= size()) return b;
    num mu = ((*this)[p]).pow(N), di = ((*this)[p]).inv();
    Poly c(size() - N*p);
    for (int x = 0; x < int(c.size()); ++x) {
      c[x] = (*this)[x + p] * di;
    }
    c = c.log();
    for (auto& val : c) val *= N;
    c = c.exp();
    for (int x = 0; x < int(c.size()); ++x) {
      b[x + N*p] = c[x] * mu;
    }
    return b;
  }
  Poly sqrt(int N) const {  // 262e0
    if (!size()) return {};
    if (deg() == -1) return Poly(N);
    int p = 0;
    while (at(p) == 0 && p < size()) ++p;
    if (p >= N) return {0};
    Poly fs(2*N);
    copy(this->begin() + p, this->end(), fs.begin());
    auto v = mod_sqrt(fs.at(0).x, M);
```

```
  if (p & 1 || v.empty()) return {};
  fs.resize(size() - p/2);
  fs *= fs.front().inv();
  fs = v[0] * (fs.log() / 2).exp();
  fs.insert(fs.begin(), p/2, 0);
  return fs;
}
Poly operator+() const { return *this; }
Poly operator-() const {
  Poly fs(size());
  for (int i = 0; i < size(); ++i) fs[i] = -(*this)[i];
  return fs;
}
Poly operator+(const Poly &fs) const { return (Poly(*this) +=
     fs); }
Poly operator-(const Poly &fs) const { return (Poly(*this) -=
     fs); }
Poly operator*(const Poly &fs) const { return (Poly(*this) *=
     fs); }
Poly operator%(const Poly &fs) const { return (Poly(*this) %=
     fs); }
Poly operator/(const Poly &fs) const { return (Poly(*this) /=
     fs); }
Poly operator*(const num &a) const { return (Poly(*this) *= a
   ); }
Poly operator/(const num &a) const { return (Poly(*this) /= a
   ); }
friend Poly operator*(const num &a, const Poly &fs) { return
     fs * a; }
// multipoint evaluation/interpolation
friend Poly eval(const Poly& fs, const Poly& qs) { // da119a
  int N = int(qs.size());
  if (N == 0) return {};
  vector<Poly> up(2 * N);
  for (int x = 0; x < N; ++x) {
    up[x + N] = Poly({0-qs[x], 1});
  }
  for (int x = N-1; x >= 1; --x) {
    up[x] = up[2 * x] * up[2 * x + 1];
  }
  vector<Poly> down(2 * N);
  down[1] = fs % up[1];
  for (int x = 2; x < 2*N; ++x) {
    down[x] = down[x / 2] % up[x];
  }
  Poly y(N);
  for (int x = 0; x < N; ++x) {
    y[x] = (down[x + N].empty() ? 0 : down[x + N][0]);
  }
  return y;
}
friend Poly interpolate(const Poly& fs, const Poly& qs) { //
   798982
  int N = int(fs.size());
  vector<Poly> up(2 * N);
  for (int x = 0; x < N; ++x) {
    up[x + N] = Poly({0-fs[x], 1});
  }
  for (int x = N-1; x >= 1; --x) {
    up[x] = up[2 * x] * up[2 * x + 1];
  }
  Poly E = eval(up[1].differential(), fs);
  vector<Poly> down(2 * N);
  for (int x = 0; x < N; ++x) {
    down[x + N] = Poly({qs[x] * E[x].inv()});
  }
  for (int x = N-1; x >= 1; --x) {
    down[x] = down[2*x] * up[2*x+1] + down[2*x+1] * up[2*x];
  }
  return down[1];
}
friend Poly convolve_all(const vector<Poly>& fs, int l, int r
   ) {
  if (r - l == 1) return fs[l];
  else {
    int md = (l + r) / 2;
    return convolve_all(fs, l, md) * convolve_all(fs, md, r);
  }
}
Poly bernoulli(int N) const { // 145ab7
  N += 5;
  Poly fs(N);
  fs[1] = 1;
  fs = fs.exp();
  copy(fs.begin()+1, fs.end(), fs.begin());
  fs = fs.inv();
  for (int x = 0; x < N; ++x) fs[x] *= fac[x];
  fs.resize(N - 5);
  return fs;
}
// x(x - 1)(x - 2)...(x - N + 1)
Poly stirling_first(int N) const {
  if (N == 0) return {1};
  vector<Poly> P(N);
  for (int x = 0; x < N; ++x) P[x] = {-x, 1};
  return convolve_all(P, 0, N);
}
Poly stirling_second(int N) const {
  if (N == 0) return {1};
  Poly P(N), Q(N);
  for (int x = 0; x < N; ++x) {
    P[x] = (x & 1 ? -1 : 1) * invFac[x];
    Q[x] = num(x).pow(N-1) * invFac[x];
  }
  P *= Q;
  P.resize(N);
  return P;
}
};
```

sum-of-powers.h

Description: Computes monomials and sum of powers product certain polynomials. Check "General purpose numbers" section for more info. (Monomials) $pw(x) = x^d$ for a fixed d. $\sum_{x=0}^{\infty} r^x f(x)$. (degree of $f \leq d$). $\sum_{x=0}^{N-1} r^x f(x)$. (degree of $f \leq d$).

"../number-theory/modular-arithmetic.h", "/lagrange.h" 0b1060, 45 lines

```
vector<num> get_monomials(int N, long long d) {
  vector<int> pfac(N);
  for (int i = 2; i < N; ++i) pfac[i] = i;
  for (int p = 2; p < N; ++p) if (pfac[p] == p) {
    for (int m = 2*p; m < N; m += p) if (pfac[m] > p) pfac[m] =
         p;
  }
  vector<num> pw(N);
  for (int i = 0; i < N; ++i) {
    if (i <= 1 || pfac[i] == i) pw[i] = num(i).pow(d);
    else pw[i] = (pw[pfac[i]] * pw[i / pfac[i]]);
  }
  return pw;
}
num sum_of_power_limit(num r, int d, const vector<num>& fs) {
  interpolator_t<num> M(d + 2);
  vector<num> qs(d + 1); qs[0] = 1;
  for (int x = 1; x <= d; ++x) qs[x] = qs[x - 1] * r;
  num ans = 0, cur_sum = 0;
  for (int x = 0; x <= d; ++x) {
    cur_sum += qs[x] * fs[x];
    ans += cur_sum * invFac[d - x] * invFac[x + 1] * (((d - x)
        & 1) ? -1 : +1) * qs[d - x];
  }
  // ans is equivalent to invFac(d + 1) * dp(d+1), where
  // for all x in [0, d], dp(x + 1) := E(d, d-x) + dp(x) * r,
      dp(0) = 0.
  // with E being the eulerian number. Works in O(d^2).
  ans *= (1 - r).pow(-(d + 1)) * fac[d + 1];
  return ans;
}
num sum_of_power(num r, int d, const vector<num>& fs, long long
     N) {
  if (r == 0) return (0 < N) ? fs[0] : 0;
  interpolator_t<num> M(d + 10);
  vector<num> gs(d + 2); gs[0] = 0; num rr = 1;
  for (int x = 0; x <= d; ++x) {
    gs[x + 1] = gs[x] + rr * fs[x];
    rr *= r;
  }
  if (r == 1) return M.interpolate(gs, N);
  const num c = sum_of_power_limit(r, d, fs);
  const num r_inv = r.inv(); num rr_inv = 1;
  for (int x = 0; x <= d + 1; ++x) {
    gs[x] = rr_inv * (gs[x] - c);
    rr_inv *= r_inv;
  }
  return c + r.pow(N) * M.interpolate(gs, N);
}
```

4.1.1 General linear recurrences

If $a_n = \sum_{k=0}^{n-1} a_k b_{n-k}$, then $A(x) = \frac{a_0}{1-B(x)}$.

4.1.2 Polyominoes

How many free (rotation, reflection), one-sided (rotation) and fixed n-ominoes are there?

n	3	4	5	6	7	8	9	10
free	2	5	12	35	108	369	1.285	4.655
one-sided	2	7	18	60	196	704	2.500	9.189
fixed	6	19	63	216	760	2.725	9.910	36.446

4.1.3 Duality

max $c^T x$ sjt to $Ax \leq b$. Dual problem is min $b^T x$ sjt to $A^T x \geq c$. By strong duality, min max value coincides.

4.1.4 Strong duality

Given a linear problem Π_1: minimize $c^t x$, sjt to $Ax \leq b$, $x \geq 0$ we can define the linear problem dual standard Π_2 like the following: minimize $-b^t y$, sjt to $A^t y \geq c$. If Π_1 is satisfied then Π_2 is also satisfied and $c^t x = b^t y$. If Π_1 is not satisfied and unbounded, then Π_2 is not satisfied and unbounded. (OBS: Can't be both unbounded!)

4.1.5 Generating functions

A list of generating functions for useful sequences:

$(1,1,1,1,1,1,\dots)$	$\frac{1}{1-z}$
$(1,-1,1,-1,1,-1,\dots)$	$\frac{1}{1+z}$
$(1,0,1,0,1,0,\dots)$	$\frac{1}{1-z^2}$
$(1,0,\dots,0,1,0,1,0,\dots,0,1,0,\dots)$	$\frac{1}{1-z^2}$
$(1,2,3,4,5,6,\dots)$	$\frac{1}{(1-z)^2}$
$(1,\binom{m+1}{m},\binom{m+2}{m},\binom{m+3}{m},\dots)$	$\frac{1}{(1-z)^{m+1}}$
$(1,c,\binom{c+1}{2},\binom{c+2}{3},\dots)$	$\frac{1}{(1-z)^c}$
$(1,c,c^2,c^3,\dots)$	$\frac{1}{1-cz}$
$(0,1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\dots)$	$\ln\frac{1}{1-z}$

A neat manipulation trick is:

$$\frac{1}{1-z}G(z) = \sum_n \sum_{k \le n} g_k z^n$$

Number theory (5)

5.1 Modular arithmetic

modular-arithmetic.h

Description: Operators for modular arithmetic.

"mod-inv.h" 62c1a3, 34 lines

```
template<unsigned M_> struct modnum {
  static constexpr unsigned M = M_;
  using ll = int64_t; using ull = uint64_t; unsigned x;
  modnum& norm(unsigned a) { x = a < M ? a : a - M; return *
      this; }
  constexpr modnum(ll a = 0U) : x(unsigned((a %= ll(M)) < 0 ? a
       + ll(M) : a)) {}
  explicit operator int() const { return x; }
  modnum& operator+=(const modnum& a) { return norm(x + a.x); }
  modnum& operator-=(const modnum& a) { return norm(x - a.x + M
      ); }
  modnum& operator*=(const modnum& a) { x = unsigned(ull(x) * a
      .x % M); return *this; }
  modnum& operator/=(const modnum& a) { return (*this *= a.inv
      ()); }
  modnum operator+(const modnum& a) const { return (modnum(*
      this) += a); }
  modnum operator-(const modnum& a) const { return (modnum(*
      this) -= a); }
  modnum operator*(const modnum& a) const { return (modnum(*
      this) *= a); }
  modnum operator/(const modnum& a) const { return (modnum(*
      this) /= a); }
  template<typename T> friend modnum operator+(T a, const
      modnum& b) { return (modnum(a) += b); }
  template<typename T> friend modnum operator-(T a, const
      modnum& b) { return (modnum(a) -= b); }
  template<typename T> friend modnum operator*(T a, const
      modnum& b) { return (modnum(a) *= b); }
  template<typename T> friend modnum operator/(T a, const
      modnum& b) { return (modnum(a) /= b); }
  modnum operator+() const { return *this; }
  modnum operator-() const { return modnum() - *this; }
  modnum pow(ll e) const {
    if (e < 0) return inv().pow(-e);
    modnum b = x, xe = 1U;
    for (; e; e >>= 1) { if (e & 1) xe *= b; b *= b; }
    return xe;
  }
  modnum inv() const { return minv(x, M); }
  friend modnum inv(const modnum& a) { return a.inv(); }
  explicit operator bool() const { return x; }
  friend bool operator==(const modnum& a, const modnum& b) {
      return a.x == b.x; }
  friend bool operator!=(const modnum& a, const modnum& b) {
      return a.x != b.x; }
  friend ostream &operator<<(ostream& os, const modnum& a) {
      return os << a.x; }
  friend istream &operator>>(istream& in, modnum& n) { ll v_;
      in >> v_; n = modnum(v_); return in; }
};
```

pairnum-template.h

Description: Support pairs operations using modnum template. Pretty good for string hashing.

229a89, 42 lines

```
template <typename T, typename U> struct pairnum {
  T t; U u;
  pairnum() : t(0), u(0) {}
  pairnum(long long v) : t(v), u(v) {}
  pairnum(const T& t_, const U& u_) : t(t_), u(u_) {}
  friend std::ostream& operator << (std::ostream& out, const
      pairnum& n) { return out << '(' << n.t << ',' << ' ' <<
      n.u << ')'; }
  friend std::istream& operator >> (std::istream& in, pairnum&
      n) { long long v; in >> v; n = pairnum(v); return in; }
  friend bool operator == (const pairnum& a, const pairnum& b)
      { return a.t == b.t && a.u == b.u; }
  friend bool operator != (const pairnum& a, const pairnum& b)
      { return a.t != b.t || a.u != b.u; }
  pairnum inv() const {
    return pairnum(t.inv(), u.inv());
  }
  pairnum neg() const {
    return pairnum(t.neg(), u.neg());
  }
  pairnum operator- () const {
    return pairnum(-t, -u);
  }
  pairnum operator+ () const {
    return pairnum(+t, +u);
  }
  pairnum& operator += (const pairnum& o) {
    t += o.t; u += o.u;
    return *this;
  }
  pairnum& operator -= (const pairnum& o) {
    t -= o.t; u -= o.u;
    return *this;
  }
  pairnum& operator *= (const pairnum& o) {
    t *= o.t; u *= o.u;
    return *this;
  }
  pairnum& operator /= (const pairnum& o) {
    t /= o.t; u /= o.u;
    return *this;
  }
  friend pairnum operator + (const pairnum& a, const pairnum& b
      ) { return pairnum(a) += b; }
  friend pairnum operator - (const pairnum& a, const pairnum& b
      ) { return pairnum(a) -= b; }
  friend pairnum operator * (const pairnum& a, const pairnum& b
      ) { return pairnum(a) *= b; }
  friend pairnum operator / (const pairnum& a, const pairnum& b
      ) { return pairnum(a) /= b; }
};
```

preparator.h

Description: Precompute factorials and inverses

"modular-arithmetic.h" 91fedc, 13 lines

```
constexpr int LIM_INV = 1 << 20;
num invs[LIM_INV], fac[LIM_INV], invFac[LIM_INV];
struct ModIntPreparator {
  ModIntPreparator() {
    invs[1] = 1;
    for (int i = 2; i < LIM_INV; ++i) invs[i] = -((num::M / i)
        * invs[num::M % i]);
    fac[0] = invFac[0] = 1;
    for (int i = 1; i < LIM_INV; ++i) {
      fac[i] = fac[i - 1] * i;
      invFac[i] = invFac[i - 1] * invs[i];
    }
  }
} preparator;
```

mod-inv.h

Description: Find x such that $ax \equiv 1 (\bmod m)$. The inverse only exist if a and m are coprimes.

48d5fb, 4 lines

```
int minv(int a, int m) {
  a %= m; assert(a);
  return a == 1 ? 1 : int(m - int64_t(minv(m, a)) * m / a);
}
```

mod-sum.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.

decfb8, 17 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) {
    ull to2 = (to * k + c) / m;
    res += to * to2;
    res -= divsum(to2, m-1 - c, m, k) + to2;
  }
  return res;
}
lint modsum(ull to, lint c, lint k, lint m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

mod-mul.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`

59afa8, 11 lines

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

mod-sqrt.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, $\mathcal{O}\left(\log p\right)$ for most p

22df14, 39 lines

```
int jacobi(int64_t a, int64_t m) { // Jacobi symbol (a/m)
  int s = 1;
  if (a < 0) a = a % m + m;
  for (; m > 1; ) {
    a %= m;
    if (a == 0) return 0;
    const int r = __builtin_ctzll(a);
    if ((r & 1) && ((m + 2) & 4)) s = -s;
    a >>= r;
    if (a & m & 2) s = -s;
    swap(a, m);
  }
  return s;
}
vector<int64_t> mod_sqrt(int64_t a, int64_t p) {
  if (p == 2) return {a & 1};
  const int j = jacobi(a, p);
  if (j == 0) return {0};
  if (j == -1) return {};
  int64_t b, d;
  while (true) {
    b = xrand() % p;
    d = (b * b - a) % p;
    if (d < 0) d += p;
    if (jacobi(d, p) == -1) break;
  }
  int64_t f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
  for (int64_t e = (p + 1) >> 1; e; e >>= 1) {
    if (e & 1) {
      tmp = (g0 * f0 + d * ((g1 * f1) % p)) % p;
      g1 = (g0 * f1 + g1 * f0) % p;
      g0 = tmp;
    }
    tmp = (f0 * f0 + d * ((f1 * f1) % p)) % p;
    f1 = (2 * f0 * f1) % p;
    f0 = tmp;
  }
  return (g0 < p - g0) ? vector<int64_t>{g0, p - g0} : vector<
      int64_t>{p - g0, g0};
}
```

mul-order.h

Description: Find the smallest integer k such that $a^k \pmod m = 1$. $0 < k < m$.
Time: $\mathcal{O}\left(log(N)\right)$

"prime-factors.h", "mod-pow.h" 3d20e1, 12 lines

```
template<typename T> T mul_order(T a, T m) {
  if (__gcd(a, m) != 1) return 0;
  auto N = phi(m);
  auto primes = prime_factorize(N);
  T res = 1;
  for (auto &[p, e] : primes) {
    while (N % p == 0 && modpow(a, N/p, m) == 1) {
      N /= p;
    }
  }
  return N;
}
```

mod-range.h

Description: min $x \geq 0$ s.t. $l \leq ((ax) \mod m) \leq r$, $m > 0$, $a \geq 0$.

eb665e, 11 lines

```
template<typename T> T mod_range(T m, T a, T l, T r) {
  l = max(l, T(0));
  r = min(r, m - 1);
  if (l > r) return -1;
  a %= m;
  if (a == 0) return (l > 0) ? -1 : 0;
  const T k = (l + a - 1) / a;
  if (a * k <= r) return k;
  const T y = mod_range(a, m, a * k - r, a * k - l);
  return (y == -1) ? -1 : ((m * y + r) / a);
}
```

5.2 Primality

sieve.h

Description: Prime sieve for generating all primes up to a certain limit. $pfac[i]$ is the lowest prime factor of i. Also useful if you need to compute any multiplicative function.
Time: $\mathcal{O}\left(N\right)$

a76cb9, 24 lines

```
vector<int> run_sieve(int N) {
  vector<int> pfac(N + 1);
  vector<int> primes; primes.reserve(N+1);
  vector<int> mu(N + 1, -1); mu[1] = 1;
  vector<int> phi(N + 1); phi[1] = 1;
  for (int i = 2; i <= N; ++i) {
    if (!pfac[i]) {
      pfac[i] = i; primes.push_back(i);
      phi[i] = i - 1;
    }
    for (int p : primes) {
      if (p > N/i) break;
      pfac[p * i] = p;
      mu[p * i] *= mu[i];
      phi[p * i] = phi[i] * phi[p];
      if (i % p == 0) {
        mu[p * i] = 0;
        phi[p * i] = phi[i] * p;
        break;
      }
    }
  }
  return primes;
}
```

segmented-sieve.h

Description: Prime sieve for generating all primes smaller than S.
Time: S=1e9 $\approx$ 1.5s

68455e, 20 lines

```
const int S = 1e6;
bitset<S> isPrime;
vector<int> eratosthenes() {
  const int S = round(sqrt(S)), R = S/2;
  vector<int> pr = {2}, sieve(S+1); pr.reserve(int(S/log(S)
      *1.1));
  vector<pair<int,int>> 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;
    for (int i = 0; i < min(S, R - L); ++i)
      if (!block[i]) pr.push_back((L + i)*2 + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

miller-rabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to 2^{64}; for larger numbers, extend A randomly.
Time: 7 times the complexity of $a^b \mod c$.

"mod-mul.h" bbee97, 12 lines

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  vector<ull> A = {2, 325, 9375, 28178, 450775, 9780504,
      1795265022};
  ull 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;
}
```

pollard-rho.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.

"mod-mul.h", "extended-euclid.h", "miller-rabin.h" 6bf31f, 18 lines

```
ull pollard(ull n) {
  auto f = [n](ull x, ull k) { return modmul(x, x, n) + k; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x, i);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x, i), y = f(f(y, i), i);
  }
  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(), r.begin(), r.end());
  return l;
}
```

5.3 Divisibility

extended-euclid.h

Description: Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.

8b62c4, 6 lines

```
template<typename T>
T egcd(T a, T b, T &x, T &y) {
  if (!a) { x = 0, y = 1; return b; }
  T g = egcd(b % a, a, y, x);
  x -= y * (b/a); return g;
}
```

division-lemma.h

Description: This lemma let us exploit the fact tha the sequence (harmonic on integer division) has at most $2\sqrt{N}$ distinct elements, so we can iterate through every possible value of $\lfloor \frac{N}{i} \rfloor$, using the fact that the greatest integer j satisfying $\lfloor \frac{N}{i} \rfloor = \lfloor \frac{N}{j} \rfloor$ is $\lfloor \frac{N}{\lfloor \frac{N}{i} \rfloor} \rfloor$. This one computes the $\sum_{i=1}^{N} \lfloor \frac{N}{i} \rfloor i$.
Time: $\mathcal{O}\left(\sqrt{N}\right)$

b2c1ab, 15 lines

```
int res = 0;
for (int a = 1, b; a <= N; a = b + 1) { // floor
  b = N / (N / a);
  // quotient (N/a) and there are (b - a + 1) elements
```

```
  int l = b - a + 1, r = a + b; // l * r / 2 = sum(i, j)
  if (l & 1) r /= 2;
  else l /= 2;
  res += l * r * (N / a);
}
// [1, N), need to deal with case where a = N separately
for (int a = 1, b; a < N; a = b + 1) { // ceil
  const int k = (N - 1) / a + 1; // quotient k
  b = (N - 1) / (k - 1);
  int cnt = b - a + 1; // occur cnt times on interval [a, b]
}
```

prime-factors.h

Description: Find all prime factors of n.
Time: $\mathcal{O}(log(n))$

"sieve.h" 7a803a, 25 lines

```
template<typename T>
vector<pair<T, int>> prime_factorize(T n) {
  vector<pair<T, int>> factors;
  while(n != 1) {
    T p = pfac[n];
    int exp = 0;
    do {
      n /= p;
      ++exp;
    } while(n % p == 0);
    factors.push_back({p, exp});
  }
  for (T p : primes) {
    if (p * p > n) break;
    if (p * p == 0) {
      factors.push_back({p, 0});
      do {
        n /= p;
        ++factors.back().second;
      } while(n % p == 0);
    }
  }
  if (n > 1) factors.push_back({n, 1});
  return factors;
}
```

divisors.h

Description: Generate all factors of n given it's prime factorization.
Time: $\mathcal{O}\left(\frac{\sqrt{N}}{\log N}\right)$

"prime-factors.h" 0526d1, 15 lines

```
template<typename T> vector<T> get_divisors(T N) {
  auto factors = prime_factorize(N);
  vector<T> ans; ans.reserve(int(sqrtl(N) + 1));
  auto dfs = [&](auto&& self, auto& ans, T val, int depth) ->
       void {
    auto& [P, E] = factors[depth];
    if (depth == int(factors.size())) ans.push_back(val);
    else {
      T X = 1;
      for (int pw = 0; pw <= E; ++pw, X *= P) {
        self(self, ans, val * X, depth + 1);
      }
    }
  }; dfs(dfs, ans, 1, 0);
  return ans;
}
```

num-div.h

Description: Count the number of divisors of n. Requires having run Sieve up to at least sqrt(n).
Time: $\mathcal{O}(log(N))$

"sieve.h" be1146, 15 lines

```
template<typename T> T numDiv(T n) {
  T how_many = 1, prime_factors = 0;
  while(n != 1) {
    T p = lp[n];
    int exp = 0;
    do {
      n /= p;
      ++exp;
      ++prime_factors; //count prime factors!
    } while(n % p == 0);
    how_many *= 1ll * (exp + 1);
  }
  if (n != 1) ++prime_factors;
  return how_many;
}
```

sum-div.h

Description: Sum of all divisors of n.
Time: $\mathcal{O}(log(N))$

"sieve.h", "mod-pow.h" 0448f4, 13 lines

```
template<typename T> T divSum(T n) {
  T sum = 1;
  while (n > 1) {
    int exp = 0;
    T p = pfac[n];
    do {
      n /= p;
      ++exp;
    } while(n % p == 0);
    sum *= (modpow(p, exp + 1, mod) - 1)/(p - 1);
  }
  return sum;
}
```

phi-function.h

Description: *Euler's totient* or *Euler's phi* function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with n. The *cototient* is $n - \phi(n)$. $\phi(1) = 1$, 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 $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}\ \forall a$.

da7671, 7 lines

```
const int n = int(1e5)*5;
vector<int> phi(n);
void calculatePhi() {
  for(int i = 0; i < n; ++i) phi[i] = i&1 ? i : i/2;
  for(int i = 3; i < n; i += 2) if (phi[i] == i)
    for(int j = i; j < n; j += i) phi[j] -= phi[j]/i;
}
```

discrete-log.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 calculate the order of a. Assumes that $0^0 = 1$.
Time: $\mathcal{O}(\sqrt{m})$

"extended-euclid.h" 6c6eb0, 18 lines

```
template<typename T> T modLog(T a, T b, T m) {
  T k = 1, it = 0, g;
  while ((g = gcd(a, m)) != 1) {
    if (b == k) return it;
    if (b % g) return -1;
    b /= g; m /= g; ++it;
    k = k * a / g % m;
  }
  T n = sqrtl(m) + 1, f = 1, j = 1;
  unordered_map<T, T> A;
  while (j <= n) {
    f = f * a % m;
    A[f * b % m] = j++;
  }
  for(int i = 1; i <= n; ++i) if (A.count(k = k * f % m))
    return n * i - A[k] + it;
  return -1;
}
```

primitive-roots.h

Description: a is a primitive root mod n if for every number x coprime to n there is an integer z s.t. $x \equiv a^z \pmod{n}$. The number of primitive roots mod n, if there are any, is equal to $phi(phi(N))$. If m isnt prime, replace $m - 1$ by $phi(m)$.
Time: $\mathcal{O}(log(N))$

<sieve.h>, <prime-factors.h>, <mod-pow.h> 05729f, 6 lines

```
template<typename T> bool is_primitive(T a, T m) {
  vector<pair<T, T>> D = prime_factorize(m-1);
  for (auto p : D)
    if (modpow(a, (m-1)/p.first, m) == 1) return false;
  return true;
}
```

prime-counting.h

Description: Count the number of primes up to N. Also useful for sum of primes.
Time: $\mathcal{O}(N^{3/4}/\log N)$

05ef7c, 22 lines

```
struct primes_t {
  vector<ll> dp, w;
  ll pi(ll N) {
    const int sqrtN = int(sqrt(N));
    for (ll a = 1, b; a <= N; a = b+1) {
      b = N / (N / a);
      w.push_back(N/a);
    }
    auto get = [&](ll x) {
      if (x <= sqrtN) return int(x-1);
      return int(w.size() - N/x);
    };
    reverse(w.begin(), w.end()); dp.reserve(w.size());
    for (auto& x : w) dp.push_back(x-1);
    for (ll i = 2; i*i <= N; ++i) {
      if (dp[i-1] == dp[i-2]) continue;
      for (int j = int(w.size())-1; w[j] >= i*i; --j)
        dp[j] -= dp[get(w[j]/i)] - dp[i-2];
    }
    return dp.back();
  }
};
```

5.4 Chinese remainder theorem

chinese-remainder.h

Description: Chinese Remainder Theorem. `crt(a, m, b, n)` computes x such that $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, x will obey $0 \leq x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\mathcal{O}(\log(LCM(m)))$

"extended-euclid.h" ece59a, 7 lines

```
pair<ll, ll> crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = egcd(m, n, x, y);
  if ((a - b) % g != 0) return {0, -1};
  x = (b - a) % n * x % n / g * m + a;
  return {x + (x < 0 ? m*n/g : 0), m*n/g};
}
```

5.5 Fractions

fractions.h

Description: Template that helps deal with fractions. df1f1d, 31 lines

```
template<typename num = long long>
struct frac {
  num n, d;
  frac() : n(0), d(1) { }
  frac(num _n, num _d = 1): n(_n), d(_d){
    num g = gcd(n, d); n /= g, d /= g;
    if (d < 0) n *= -1, d *= -1;
    assert(d != 0);
  }
  friend bool operator<(const frac& l, const frac& r) { return
      l.n * r.d < r.n * l.d; }
  friend bool operator==(const frac& l, const frac& r) { return
       l.n == r.n && l.d == r.d; }
  friend bool operator!=(const frac& l, const frac& r) { return
       !(l == r); }
  friend frac operator+(const frac& l, const frac& r) {
    num g = gcd(l.d, r.d);
    return frac( r.d / g * l.n + l.d / g * r.n, l.d / g * r.d);
  }
  friend frac operator-(const frac& l, const frac& r) {
    num g = gcd(l.d, r.d);
    return frac( r.d / g * l.n - l.d / g * r.n, l.d / g * r.d);
  }
  friend frac operator*(const frac& l, const frac& r) { return
      frac(l.n * r.n, l.d * r.d); }
  friend frac operator/(const frac& l, const frac& r) { return
      l * frac(r.d, r.n); }
  friend frac& operator+=(frac& l, const frac& r) { return l =
      l+r; }
  friend frac& operator-=(frac& l, const frac& r) { return l =
      l-r; }
  template<class T> friend frac& operator*=(frac& l, const T& r
      ) { return l = l*r; }
  template<class T> friend frac& operator/=(frac& l, const T& r
      ) { return l = l/r; }
  friend ostream& operator<<(ostream& strm, const frac& a) {
    strm << a.n << "/" << a.d;
    return strm;
  }
};
```

continued-fractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.
For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.
Time: $\mathcal{O}(\log N)$ 61608b, 21 lines

```
typedef double dbl; // for N ~ 1e7; long double for N ~ 1e9
pair<lint, lint> approximate(dbl x, lint N) {
  lint LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; dbl y = x
      ;
  for (;;) {
    lint lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
       a = (lint)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 - (dbl)NP / (dbl)NQ) < abs(x - (dbl)P / (
          dbl)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (dbl)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

frac-binary-search.h

Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.
Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`
Time: $\mathcal{O}(\log(N))$ f83d46, 23 lines

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

5.5.1 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

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

5.5.2 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.5.3 Primitive Roots

It only exists when n is $2, 4, p^k, 2p^k$, where p odd prime. If g is a primitive root, all primitive roots are of the form g^k where $k, \phi(p)$ are coprime (hence there are $\phi(\phi(p))$ primitive roots).

5.5.4 Chicken McNugget theorem

Let x and y be two coprime integers, the greater integer that can't be written in the form of $ax + by$ is $\frac{(x-1)(y-1)}{2}$

5.6 Primes

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

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

5.6.1 Wilson's theorem

Let $n > 1$. Then $n|(n-1)! + 1$ iff n is prime.

5.6.2 Wolstenholme's theorem

Let $p > 3$ be a prime number. Then its numerator $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{p-1}$ is divisible by p^2.

5.6.3 Prime counting function ($\pi(x)$)

The prime counting function is asymptotic to $\frac{x}{\log x}$, by the prime number theorem.

x	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
$\pi(x)$	4	25	168	1.229	9.592	78.498	664.579	5.761.455

5.6.4 Sum of primes

For any multiplicative f:

$$S(n,p) = S(n, p-1) - f(p) \cdot (S(n/p, p-1) - S(p-1, p-1))$$

5.6.5 Moebius Function

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

Moebius Inversion:

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

Other useful formulas/forms:

$\sum_{d|n} \phi(d) = n$

$\sum_{\substack{i<n \\ \gcd(i,n)=1}} i = n\frac{\phi(n)}{2}$

$\sum_{a=1}^n \sum_{b=1}^n [\gcd(a,b) = 1] = \sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor^2$

$\sum_{a=1}^n \sum_{b=1}^n \gcd(a,b) = \sum_{d=1}^n d \sum_{d|x}^n \lfloor \frac{n}{x} \rfloor^2 \mu(\frac{x}{d})$

$\sum_{a=1}^n \sum_{b=a}^n \gcd(a,b) = \sum_{d=1}^n \sum_{d|x}^n \phi(\frac{x}{d}) d$

$\sum_{a=1}^n \sum_{b=1}^n \mathrm{lcm}(a,b) = \sum_{d=1}^n \mu(d) d \sum_{d|x}^n x \binom{\lfloor \frac{n}{x} \rfloor + 1}{2}^2$

$\sum_{a=1}^{n}\sum_{b=a+1}^{n}\operatorname{lcm}(a,b)=\sum_{d=1}^{n}\sum_{d|x}\phi(\frac{x}{d})\frac{x^2}{2d}$

$\sum_{a\in S}\sum_{b\in S}\gcd(a,b)=\sum_{d=1}^{n}(\sum_{x|d}\frac{d}{x}\mu(x))(\sum_{d|v}\text{freq[v]})^2$

$\sum_{a\in S}\sum_{b\in S}\operatorname{lcm}(a,b)=\sum_{d=1}^{n}(\sum_{x|d}\frac{x}{d}\mu(x))(\sum_{v\in S,d|v}v)^2$

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

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

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

5.6.6 Dirichlet Convolution

Given a function $f(x)$, let

$$(f*g)(x)=\sum_{d|x}g(d)f(x/d)$$

If the partial sums $s_{f*g}(n), s_g(n)$ can be computed in $O(1)$ and $s_f(1\dots n^{2/3})$ can be computed in $O\left(n^{2/3}\right)$ then all $s_f\left(\frac{n}{d}\right)$ can as well. Use

$$s_{f*g}(n)=\sum_{d=1}^{n}g(d)s_f(n/d).$$

$$\implies s_f(n)=\frac{s_{f*g}(n)-\sum_{d=2}^{n}g(d)s_f(n/d)}{g(1)}$$

1. If $f(x)=\mu(x)$ then $g(x)=1,(f*g)(x)=(x==1)$, and $s_f(n)=1-\sum_{i=2}^{n}s_f(n/i)$
2. If $f(x)=\phi(x)$ then $g(x)=1,(f*g)(x)=x$, and $s_f(n)=\frac{n(n+1)}{2}-\sum_{i=2}^{n}s_f(n/i)$

dirichlet-convolution.h

Description: Dirichlet convolution. Change f, gs and fgs accordingly. This example calculates $\phi(N)$.
Time: $\mathcal{O}\left(N^{\frac{2}{3}}\right)$

bd3050, 26 lines

```
template<typename T, typename V> struct dirichlet_convolution {
  V N; // ~ N^{2/3}
  T inv;
  vector<V> fs; // can be any multiplicative function
  vector<T> psum;
  unordered_map<V, T> mapa;
  V f(V x) { return fs[x]; }
  T gs(V x) { return x; }
  T fgs(V x) { return T(x) * (x + 1) / 2; }
  dirichlet_convolution(V _N, const vector<V>& F) : N(_N + 1),
      fs(F), psum(_N + 1) {
    inv = gs(1);
    for (V a = 0; a + 1 < N; ++a) {
      psum[a + 1] = f(a + 1) + psum[a];
    }
  }
  T query(V x) {
    if (x < N) return psum[x];
    if (mapa.find(x) != mapa.end()) return mapa[x];
    T ans = fgs(x);
    for (V a = 2, b; a <= x; a = b + 1) {
      b = x / (x / a);
      ans -= (gs(b) - gs(a - 1)) * query(x / a);
    }
    return mapa[x] = (ans / inv);
  }
};
```

5.6.7 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 $n<1e19$.

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		

int-perm.h

Description: Permutation -> integer conversion. (Not order preserving.)
Time: $\mathcal{O}(n)$

06f786, 6 lines

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

6.1.2 Binomials

- Sum of every element in the n-th row of pascal triangle is 2^n.
- The product of the elements in each row is $\frac{(n+1)^n}{n!}$
- $\sum_{k=0}^{n}\binom{n}{k}^2=\binom{2n}{n}$
- In a row p where p is a prime number, all the terms in that row except the 1s are multiples of p
- To count odd terms in row n, convert n to binary. Let x be the number of 1s in the binary representation. Then the number of odd terms will be 2^x
- Every entry in row 2^n-1 is odd

$$\binom{n}{k}=\frac{n}{k}\binom{n-1}{k-1}$$
$$\sum_{k=0}^{n}\binom{n}{k}k=n2^{n-1}$$
$$\sum_{m=0}^{n}\binom{m}{j}\binom{n-m}{k-j}=\binom{n+1}{k+1}$$
$$(x+y)^n=\sum_{k=0}^{n}\binom{n}{k}x^{n-k}y^k$$
$$\sum_{j=0}^{m}\binom{m}{j}^2=\binom{2m}{m}$$
$$2\sum_{i=L}^{R}\binom{n}{i}-\binom{n}{L}-\binom{n}{R}=\sum_{i=L+1}^{R}\binom{n+1}{i}$$
$$\binom{n}{k}\binom{n-k}{v}=\binom{n}{v}\binom{n-v}{k}$$
$$\sum_{j=0}^{k}\binom{m}{j}\binom{n-m}{k-j}=\binom{n}{k}$$
$$\sum_{r=0}m\binom{n+r}{r}=\binom{n+m+1}{m}$$
$$\sum_{m=k}^{n}\binom{m}{k}=\binom{n+1}{k+1}$$

rolling-binomial.h

Description: $\binom{n}{k}\pmod m$ in time proportional to the difference between (n,k) and the previous (n,k).

"../number-theory/preparator.h" d087bf, 14 lines

```
using i64 = int64_t;
const int mod = int(1e9) + 7;
struct Bin {
  int N = 0,  K = 0; i64 r = 1;
  void m(int a, int b) { r = r * a % mod * invs[b] % mod; }
  i64 choose(int n, int k) {
    if (k > n || k < 0) return 0;
    while(N < n) ++N, m(N, N - K);
    while(K < k) ++K, m(N - K + 1, K);
    while(K > k) m(K, N - K + 1), --K;
    while(N > n) m(N - K, N), --N;
    return r;
  }
};
```

lucas.h

Description: Lucas' thm: Let n,m be non-negative integers and p a prime. Write $n=n_kp^k+...+n_1p+n_0$ and $m=m_kp^k+...+m_1p+m_0$. Then $\binom{n}{m}\equiv\prod_{i=0}^{k}\binom{n_i}{m_i}\pmod p$. fact and ifact must hold pre-computed factorials / inverse factorials, e.g. from ModInv.h.
Time: $\mathcal{O}(\log_p m)$

"../number-theory/preparator.h" c55480, 10 lines

```
ll chooseModP(ll n, ll m, int p) {
  assert(m < 0 || m > n);
  ll c = 1;
  for (; m > 0; n /= p, m /= p) {
    lint n0 = n % p, m0 = m % p;
    if (n0 < m0) return 0;
    c = c * ((((fac[n0] * invFac[m0]) % p) * invFac[n0 - m0]) %
        p) % p;
  }
  return c;
}
```

multinomial.h

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

864cdb, 7 lines

```
lint multinomial(vector<int>& v) {
  lint c = 1, m = v.empty() ? 1 : v[0];
  for (int i = 1 < v.size(); ++i)
    for (int j = 0; j < v[i]; ++j)
      c = c * ++m / (j+1);
  return c;
}
```

6.1.3 Cycles

Let the number of n-permutations whose cycle lenghts all belong to the set S be denoted by $g_S(n)$

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

6.1.4 Involutions

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$

$$a(0) = a(1) = 1$$

1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152

6.1.5 Inclusion-Exclusion Principle

Let $A_1, A_2, ..., A_n$ be finite sets. Then $A_1 \cup A_2 \cup ... \cup A_n$ is

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{\substack{I \subseteq \{1,2,\ldots,n\} \\ I \neq \emptyset}} (-1)^{|I|+1} \left|\bigcap_{i \in I} A_i\right|$$

6.1.6 The twelvefold way (from Stanley)

How many functions $f\colon N \to X$ are there?

N	X	Any f	Injective	Surjective
dist.	dist.	x^n	$\frac{x!}{(x-n)!}$	$x!\left\{ {n \atop x} \right\}$
indist.	dist.	$\binom{x+n-1}{n}$	$\binom{x}{n}$	$\binom{n-1}{n-x}$
dist.	indist.	$\left\{ {n \atop 1} \right\} + \ldots + \left\{ {n \atop x} \right\}$	$[n \leq x]$	$\left\{ {n \atop k} \right\}$
indist.	indist.	$p_1(n) + \ldots p_x(n)$	$[n \leq x]$	$p_x(n)$

Where $\binom{a}{b} = \frac{1}{b!}(a)_b$, $p_x(n)$ is the number of ways to partition the integer n using x summand and $\left\{ {n \atop x} \right\}$ is the number of ways to partition a set of n elements into x subsets (aka Stirling number of the second kind).

6.1.7 Burnside

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

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

6.1.8 Derangements

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

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

6.2 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

6.3 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:

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

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.

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

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

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

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

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

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

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. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$.

$$\mathcal{B}_{n+1} = \sum_{k=0}^{n}\binom{n}{k}\mathcal{B}_k$$

Also possible to calculate using Stirling numbers of the second kind,

$$B_n = \sum_{k=0}^{n} S(n,k)$$

If p is 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: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$ # forests with exactly k rooted trees:

$$\binom{n}{k} k \cdot n^{n-k-1}$$

.

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 a $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children) or $2n+1$ elements.
- 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 subsequence.

6.3.8 Super Catalan numbers

The number of monotonic lattice paths of a nxn grid that do not touch the diagonal.

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$
$$S(1) = S(2) = 1$$

1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859

6.3.9 Motzkin numbers

Number of ways of drawing any number of nonintersecting chords among n points on a circle. Number of lattice paths from (0, 0) to (n, 0) never going below the x-axis, using only steps NE, E, SE.

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

$$M(0) = M(1) = 1$$

1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634

6.3.10 Narayana numbers

Number of lattice paths from (0,0) to (2n,0) never going below the x-axis, using only steps NE and SE, and with k peaks.

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

$$N(n,1) = N(n,n) = 1$$

$$\sum_{k=1}^{n} N(n,k) = C_n$$

1, 1, 1, 1, 3, 1, 1, 6, 6, 1, 1, 10, 20, 10, 1, 1, 15, 50

6.3.11 Schroder numbers

Number of lattice paths from (0, 0) to (n, n) using only steps N,NE,E, never going above the diagonal. Number of lattice paths from (0, 0) to (2n, 0) using only steps NE, SE and double east EE, never going below the x-axis. Twice the Super Catalan number, except for the first term.
1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098

6.3.12 Triangles

Given rods of length 1, ..., n,

$$T(n) = \frac{1}{24}\left\{\begin{array}{ll} n(n-2)(2n-5) & n \text{ even} \\ (n-1)(n-3)(2n-1) & n \text{ odd} \end{array}\right\}$$

is the number of distinct triangles (positive are) that can be constructed, i.e., the # of 3-subsets of $[n]$ s.t. $x \le y \le z$ and $z \ne x + y$.

6.4 Fibonacci

$Fib(x+y) = Fib(x+1)Fib(y) + Fib(x)Fib(y-1)$

$Fib(n+1)Fib(n-1) - Fib(n)^2 = (-1)^n$

$Fib(2n-1) = Fib(n)^2 - Fib(n-1)^2$

$\sum_{i=0}^{n} Fib(i) = Fib(n+2) - 1$

$\sum_{i=0}^{n} Fib(i)^2 = Fib(n)Fib(n+1)$

$\sum_{i=0}^{n} Fib(i)^3 = \frac{Fib(n)Fib(n+1)^2 - (-1)^n Fib(n-1) + 1}{2}$

6.5 Linear Recurrences

(i) $F_n = F_{n-1} + F_{n-2}$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} F_0 \\ F_1 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

(ii) $F_i = \sum_{j=1}^{K} C_j F_{i-j}$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & . & 0 \\ 0 & 0 & 1 & 0 & . & 0 \\ 0 & 0 & 0 & 1 & . & 0 \\ . & . & . & . & . & . \\ C_K & C_{K-1} & C_{K-2} & C_{K-3} & . & C_1 \end{bmatrix}\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ . \\ F_{K-1} \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ . \\ F_K \end{bmatrix}$$

(iii) $F_i = \sum_{j=1}^{K} C_j F_{i-j} + D$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & . & 0 & 0 \\ 0 & 0 & 1 & 0 & . & 0 & 0 \\ 0 & 0 & 0 & 1 & . & 0 & 0 \\ . & . & . & . & . & . & . \\ C_K & C_{K-1} & C_{K-2} & C_{K-3} & . & C_1 & 1 \\ 0 & 0 & 0 & 0 & . & 0 & 1 \end{bmatrix}\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ . \\ F_{K-1} \\ D \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ . \\ F_K \\ D \end{bmatrix}$$

6.6 Game Theory

A game can be reduced to Nim if it is a finite impartial game. Nim and its variants include:

6.6.1 Nim

Let $X = \bigoplus_{i=1}^{n} x_i$, then $(x_i)_{i=1}^{n}$ is a winning position iff $X \ne 0$. Find a move by picking k such that $x_k > x_k \oplus X$.

6.6.2 Misère Nim

Regular Nim, except that the last player to move *loses*. Play regular Nim until there is only one pile of size larger than 1, reduce it to 0 or 1 such that there is an odd number of piles. The second player wins $(a_1, \dots, a_n)$ if 1) there is a pile $a_i > 1$ and $\oplus_{i=1}^{n} a_i = 0$ or 2) all $a_i \le 1$ and $\oplus_{i=1}^{n} a_i = 1$.

6.6.3 Staircase Nim

Stones are moved down a staircase and only removed from the last pile. $(x_i)_{i=1}^{n}$ is an L-position if $(x_{2i-1})_{i=1}^{n/2}$ is (i.e. only look at odd-numbered piles).

6.6.4 Moore's Nim$_k$

The player may remove from at most k piles (Nim = Nim$_1$). Expand the piles in base 2, do a carry-less addition in base $k+1$ (i.e. the number of ones in each column should be divisible by $k+1$).

6.6.5 Dim$^+$

The number of removed stones must be a divisor of the pile size. The Sprague-Grundy function is $k+1$ where 2^k is the largest power of 2 dividing the pile size.

6.6.6 Aliquot Game

Same as above, except the divisor should be proper (hence 1 is also a terminal state, but watch out for size 0 piles). Now the Sprague-Grundy function is just k.

6.6.7 Nim (at most half)

Write $n+1 = 2^m y$ with m maximal, then the Sprague-Grundy function of n is $(y-1)/2$.

6.6.8 Lasker's Nim

Players may alternatively split a pile into two new non-empty piles. $g(4k+1) = 4k+1$, $g(4k+2) = 4k+2$, $g(4k+3) = 4k+4$, $g(4k+4) = 4k+3$ ($k \ge 0$).

6.6.9 Hackenbush on Trees

A tree with stalks $(x_i)_{i=1}^{n}$ may be replaced with a single stalk with length $\bigoplus_{i=1}^{n} x_i$.

nim-product.cpp

Description: Product of nimbers is associative, commutative, and distributive over addition (xor). Forms finite field of size 2^{2^k}. Application: Given 1D coin turning games G_1, G_2 $G_1 \times G_2$ is the 2D coin turning game defined as follows. If turning coins at $x_1, x_2, \dots, x_m$ is legal in G_1 and $y_1, y_2, \dots, y_n$ is legal in G_2, then turning coins at all positions (x_i, y_j) is legal assuming that the coin at (x_m, y_n) goes from heads to tails. Then the grundy function $g(x, y)$ of $G_1 \times G_2$ is $g_1(x) \times g_2(y)$.
Time: 64^2 xors per multiplication, memorize to speed up.

38fb87, 28 lines

```
using ull = uint64_t;
ull nim_prod[64][64];
ull nim_prod2(int i, int j) {
  if (nim_prod[i][j]) return nim_prod[i][j];
  if ((i & j) == 0) return nim_prod[i][j] = 1ull << (i|j);
  int a = (i&j) & -(i&j);
  return nim_prod[i][j] = nim_prod2(i ^ a, j) ^ nim_prod2((i ^
       a) | (a-1), (j ^ a) | (i & (a-1)));
}

void all_nim_prod() {
  for (int i = 0; i < 64; i++) {
    for (int j = 0; j < 64; j++) {
      if ((i & j) == 0) nim_prod[i][j] = 1ull << (i|j);
      else {
        int a = (i&j) & -(i&j);
        nim_prod[i][j] = nim_prod[i ^ a][j] ^ nim_prod[(i ^ a)
             | (a-1)][(j ^ a) | (i & (a-1))];
      }
    }
  }
}
ull get_nim_prod(ull x, ull y) {
  ull res = 0;
  for (int i = 0; i < 64 && (x >> i); ++i)
    if ((x >> i) & 1)
      for (int j = 0; j < 64 && (y >> j); ++j)
        if ((y >> j) & 1) res ^= nim_prod2(i, j);
  return res;
}
```

partitions.h

378a72, 16 lines

```
const int M = 998244353;
vector<int64_t> prep(int N) {
  vector<int64_t> dp(N); dp[0] = 1;
  for (int n = 1; n < N; ++n) {
    int64_t sum = 0;
    for (int k = 0, l = 1, m = n - 1; ;) {
      sum += dp[m]; if ((m -= (k += 1)) < 0) break;
      sum += dp[m]; if ((m -= (l += 2)) < 0) break;
```

```
      sum -= dp[m]; if ((m -= (k += 1)) < 0) break;
      sum -= dp[m]; if ((m -= (l += 2)) < 0) break;
    }
    if ((sum %= M) < 0) sum += M;
    dp[n] = sum;
  }
  return dp;
}
```

Graph (7)

7.1 Fundamentals

bellman-ford.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max |w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$

d37b03, 38 lines

```
const lint inf = LLONG_MAX;
struct edge_t { int a, b, w, s() { return a < b ? a : -a; }};
struct node_t { lint dist = inf; int prev = -1; };

void bellmanFord(vector<node_t>& nodes, vector<edge_t>& eds,
    int s) {
  nodes[s].dist = 0;
  sort(eds.begin(), eds.end(), [](edge_t a, edge_t b) { return
      a.s() < b.s(); });
  int lim = nodes.size() / 2 + 2; // /3+100 with shuffled
      vertices
  for(int i = 0; i < lim; ++i) for(auto &ed : eds) {
    node_t cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    lint d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  for(int i = 0; i < lim; ++i) for(auto &e : eds)
    if (nodes[e.a].dist == -inf) nodes[e.b].dist = -inf;
}

vector<int> negCyc(int n, vector<edge_t>& edges) {
  vector<int64_t> d(n); vector<int> p(n);
  int v = -1;
  for (int i = 0; i < n; ++i) {
    v = -1;
    for (edge_t &u : edges)
      if (d[u.b] > d[u.a] + u.w) {
        d[u.b] = d[u.a] + u.w;
        p[u.b] = u.a, v = u.b;
      }
    if (v == -1) return {};
  }
  for (int i = 0; i < n; ++i) v = p[v]; // enter cycle
  vector<int> cycle = {v};
  while (p[cycle.back()] != v) cycle.push_back(p[cycle.back()])
      ;
  return {cycle.rbegin(), cycle.rend()};
}
```

floyd-warshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge distances. Input is an distance matrix m, where $m[i][j]$ = inf if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, inf if no path, or -inf if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$

578e31, 16 lines

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

dijkstra.h

Description: Faster implementation of Dijkstra's algorithm. Makes very easy to handle SSSP on state graphs.
Time: $\mathcal{O}(N \log N)$

67beaf, 31 lines

```
#include<bits/extc++.h> // keep-include!!

template <class D> struct MinDist {
  vector<D> dist; vector<int> from;
};
template <class D, class E> // Weight type and Edge info
MinDist<D> Dijkstra(const vector<vector<E>>& g, int s, D inf =
    numeric_limits<D>::max()) {
  int N = int(g.size());
  vector<D> dist = vector<D>(N, inf);
  vector<int> par = vector<int>(N);
  struct state_t {
    D key;
    int to;
    bool operator<(state_t r) const { return key > r.key; }
  };
  __gnu_pbds::priority_queue<state_t> q;
  q.push(state_t{0, s});
  dist[s] = D(0);
  while (!q.empty()) {
    state_t p = q.top(); q.pop();
    if (dist[p.to] < p.key) continue;
    for (E nxt : g[p.to]) {
      if (p.key + nxt.second < dist[nxt.first]) {
        dist[nxt.first] = p.key + nxt.second;
        par[nxt.first] = p.to;
        q.push(state_t{dist[nxt.first], nxt.first});
      }
    }
  }
  return MinDist<D>{dist, par};
}
```

euler-walk.h

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

c1cf41, 16 lines

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

7.2 Network flow

push-relabel.h

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. id can be used to restore each edge and its amount of flow used.
Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$ Better for dense graphs - Slower than Dinic (in practice)

387a0c, 42 lines

```
template<typename flow_t = int> struct PushRelabel {
  struct edge_t { int dest, back; flow_t f, c; };
  vector<vector<edge_t>> g;
  vector<flow_t> ec;
  vector<edge_t*> cur;
  vector<vector<int>> hs; vector<int> H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
  void addEdge(int s, int t, flow_t cap, flow_t rcap = 0) { //
      d58501
    if (s == t) return;
    g[s].push_back({t, (int)g[t].size(), 0, cap});
    g[t].push_back({s, (int)g[s].size()-1, 0, rcap});
  }
  void addFlow(edge_t& e, flow_t f) { // 2f7969
    edge_t &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_t maxflow(int s, int t) { // 21100c
    int v = int(g.size()); H[s] = v; ec[t] = 1;
    vector<int> co(2*v); co[0] = v-1;
    for(int i = 0; i < v; ++i) cur[i] = g[i].data();
    for(auto& e : g[s]) addFlow(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() + g[u].size()) {
          H[u] = 1e9;
          for(auto &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)
            for(int i = 0; i < v; ++i) 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)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= int(g.size()); }
};
```

dinitz.h

Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max|\text{cap}|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching. To obtain each partition A and B of the cut look at lvl, for $v \subset A$, $lvl[v] > 0$, for $u \subset B$, $lvl[u] = 0$.

7c5dcd, 75 lines

```
template<typename T = int> struct Dinitz {
  struct edge_t { int to, rev; T c, f; };
  vector<vector<edge_t>> adj;
  vector<int> lvl, ptr, q;
  Dinitz(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  inline void addEdge(int a, int b, T c, T rcap = 0) { // 694
      aae
    adj[a].push_back({b, (int)adj[b].size(), c, 0});
    adj[b].push_back({a, (int)adj[a].size() - 1, rcap, 0});
  }
  T dfs(int v, int t, T f) { // 8ffe6b
    if (v == t || !f) return f;
    for (int &i = ptr[v]; i < int(adj[v].size()); ++i) {
      edge_t &e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (T p = dfs(e.to, t, min(f, e.c - e.f))) {
          e.f += p, adj[e.to][e.rev].f -= p;
          return p;
        }
    }
    return 0;
  }
  T maxflow(int s, int t) { // db2141
    T flow = 0; q[0] = s;
    for (int L = 0; L < 31; ++L) do { // 'int L=30' maybe
        faster for random data
      lvl = ptr = vector<int>(q.size());
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (edge_t &e : adj[v])
          if (!lvl[e.to] && (e.c - e.f) >> (30 - L))
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
      while (T p = dfs(s, t, numeric_limits<T>::max()/4)) flow
          += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int v) { return bool(lvl[v] != 0); }
  pair<T, vector<pair<int,int>>> minCut(int s, int t) { // 727
      b22
    T cost = maxflow(s,t);
    vector<pair<int,int>> cut;
    for (int i = 0; i < int(adj.size()); i++) for(edge_t &e :
        adj[i])
      if (lvl[i] && !lvl[e.to]) cut.push_back({i, e.to});
    return {cost, cut};
  }
};

struct flow_demand_t {
  int src, sink;
  vector<int> d;
  Dinitz<int> flower;
  flow_demand_t(int N) : src(N + 1), sink(N + 2), d(N + 3),
      flower(N + 3) {}
  void add_edge(int a, int b, int demand, int cap) {
    d[a] -= demand;
    d[b] += demand;
    flower.addEdge(a, b, cap - demand);
  }
  int get_flow() {
    const int INF = std::numeric_limits<int>::max():
    int x = 0, y = 0;
    flower.add_edge(N, N-1, INF);
    for (int i = 0; i <= N; ++i) {
      if (d[i] < 0) {
        flower.add_edge(i, sink, -d[i]);
        x += -d[i];
      }
      if (d[i] > 0) {
        flower.add_edge(src, i, d[i]);
        y += d[i];
      }
    }
    bool has_circulation = (flower.maxflow(src, sink) == x && x
        == y);
    if (!has_circulation) return -1;
    return flower.maxflow(N-1, N);
  }
};
```

min-cost-max-flow.h

Description: Min-cost max-flow. Assumes there is no negative cycle.
Time: $\mathcal{O}(F(V+E)logV)$, being F the amount of flow.

e7ff44, 73 lines

```
template<class flow_t, class cost_t> struct min_cost {
  static constexpr flow_t FLOW_EPS = 1e-10L;
  static constexpr flow_t FLOW_INF = std::numeric_limits<flow_t
      >::max();
  static constexpr cost_t COST_EPS = 1e-10L;
  static constexpr cost_t COST_INF = std::numeric_limits<cost_t
      >::max();
  int n, m;
  vector<int> ptr, nxt, zu;
  vector<flow_t> capa;
  vector<cost_t> cost;
  explicit min_cost(int n_) : n(n_), m(0), ptr(n_, -1) {}
  void add_edge(int u, int v, flow_t w, cost_t c) { // d482f5
    nxt.push_back(ptr[u]); zu.push_back(v); capa.push_back(w);
        cost.push_back( c); ptr[u] = m++;
    nxt.push_back(ptr[v]); zu.push_back(u); capa.push_back(0);
        cost.push_back(-c); ptr[v] = m++;
  }
  vector<cost_t> pot, dist;
  vector<bool> vis;
  vector<int> pari;
  // cost slopes[j] per flow when flows[j] <= flow <= flows[j +
      1]
  vector<flow_t> flows; vector<cost_t> slopes;
  //   The distance to a vertex might not be determined if it
      is >= dist[t].
  //   You can pass t = -1 to find a shortest path to each
      vertex.
  void shortest(int s, int t) { // e9bb0d
    using Entry = pair<cost_t, int>;
    priority_queue<Entry, vector<Entry>, std::greater<Entry>>
        que;
    for (int u = 0; u < n; ++u) { dist[u] = COST_INF; vis[u] =
        false; }
    for (que.emplace(dist[s] = 0, s); !que.empty(); ) {
      const cost_t c = que.top().first;
      const int u = que.top().second;
      que.pop();
      if (vis[u]) continue;
      vis[u] = true;
      if (u == t) return;
      for (int i = ptr[u]; ~i; i = nxt[i]) if (capa[i] >
          FLOW_EPS) {
        const int v = zu[i];
        const cost_t cc = c + cost[i] + pot[u] - pot[v];
        if (dist[v] > cc) { que.emplace(dist[v] = cc, v); pari[
            v] = i; }
      }
    }
  }
  pair<flow_t, cost_t> run(int s, int t, flow_t limFlow =
      FLOW_INF) {
    assert(0 <= limFlow);
    pot.assign(n, 0);
    while (true) {
      bool upd = false;
      for (int i = 0; i < m; ++i) if (capa[i] > FLOW_EPS) {
        const int u = zu[i ^ 1], v = zu[i];
        const cost_t cc = pot[u] + cost[i];
        if (pot[v] > cc + COST_EPS) { pot[v] = cc; upd = true;
            }
      }
      if (!upd) break;
    }
    dist.resize(n); vis.resize(n); pari.resize(n);
    flows.clear(); flows.push_back(0);
    slopes.clear();
    flow_t flow = 0; cost_t cost = 0;
    while (flow < limFlow) {
      shortest(s, t);
      if (!vis[t]) break;
      for (int u = 0; u < n; ++u) pot[u] += min(dist[u], dist[t
          ]);
      flow_t f = limFlow - flow;
      for (int v = t; v != s; ) {
        const int i = pari[v]; if (f > capa[i]) { f = capa[i];
            } v = zu[i ^ 1];
      }
      for (int v = t; v != s; ) {
        const int i = pari[v]; capa[i] -= f; capa[i ^ 1] += f;
            v = zu[i ^ 1];
      }
      flow += f;
      cost += f * (pot[t] - pot[s]);
      flows.push_back(flow); slopes.push_back(pot[t] - pot[s]);
    }
    return {flow, cost};
  }
};
```

7.3 Matching

hopcroft-karp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vector<int> btoa(m, -1); hopcroftKarp(g, btoa);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

d9a55d, 35 lines

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

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

bipartite-matching.h

Description: Fast Kuhn! Simple maximum cardinality bipartite matching algorithm. Better than hopcroftKarp in practice. Worst case is $O(VE)$ on an hairy tree. Shuffling the edges and vertices ordering should break some worst-case inputs.

Time: $\Omega(VE)$

1b4d72, 31 lines

```
struct bm_t {
  int N, M, T;
  vector<vector<int>> adj;
  vector<int> match, seen;
  bm_t(int a, int b) : N(a), M(a+b), T(0), adj(M),
  match(M, -1), seen(M, -1) {}
  void add_edge(int a, int b) { adj[a].push_back(b + N); }
  bool dfs(int cur) {
    if (seen[cur] == T) return false;
    seen[cur] = T;
    for (int nxt : adj[cur]) if (match[nxt] == -1) {
      match[nxt] = cur, match[cur] = nxt;
      return true;
    }
    for (int nxt : adj[cur]) if (dfs(match[nxt])) {
      match[nxt] = cur, match[cur] = nxt;
      return true;
    }
    return false;
  }
  int solve() {
    int res = 0;
    for (int cur = 1; cur; ) {
      cur = 0; ++T;
      for (int i = 0; i < N; ++i) if (match[i] == -1)
        cur += dfs(i);
      res += cur;
    }
    return res;
  }
};
```

weighted-matching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost.

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

7a2392, 31 lines

```
pair<int, vector<int>> hungarian(const vector<vector<int>> &a)
    {
  if (a.empty()) return {0, {}};
  int n = a.size() + 1, m = a[0].size() + 1;
  vector<int> u(n), v(m), p(m), ans(n - 1);
  for(int i = 1; i < n; ++i) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vector<int> dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do {
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      for(int j = 1; j < m; ++j) if (!done[j]) {
        auto cur = a[i0-1][j-1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      for(int j = 0; j < m; ++j) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  for(int j = 1; j < m; ++j) if (p[j]) ans[p[j]-1] = j-1;
  return {-v[0], ans}; // min cost
}
```

general-matching-dfs.h

Description: Maximum Matching for general graphs (undirected and non bipartite) using a crazy chinese heuristic(Yet to find any counter case). one-indexed based implementation, be careful. *it* represents how many iterations you wanna try, something between [5, 500] suffice.

Usage: `GeneralMatching G(N+1); G.addEdge(a+1, b+1);`
`int max_matching = G.solve(5);`

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

"../various/RandomNumbers.h" 596e90, 44 lines

```
struct GeneralMatching {
  int N, T;
  vector<vector<int>> edges;
  vector<int> seen, match;
  GeneralMatching(int _N) : N(_N), T(0), edges(N), seen(N),
       match(N) {}
  void addEdge(int a, int b) { // one-based!
    edges[a].push_back(b);
    edges[b].push_back(a);
  }
  bool dfs(int v) {
    if (v == 0) return true;
    seen[v] = T;
    shuffle(edges[v].begin(), edges[v].end(), rng);
    for (int u : edges[v]) {
      int to = match[u];
      if (seen[to] < T) {
        match[v] = u, match[u] = v, match[to] = 0;
        if (dfs(to)) return true;
        match[u] = to, match[to] = u, match[v] = 0;
      }
    }
    return false;
  }
  int solve(int it) {
    int res = 0;
    for (int t = 0; t < it; ++t) {
      for (int i = 1; i < N; ++i) {
        if (match[i]) continue;
        ++T;
        res += dfs(i);
      }
    }
    return res;
  }
  vector<array<int, 2>> get_edges(int it) {
    int ma = solve(it);
    vector<array<int, 2>> E; E.reserve(ma);
    for (int i = 1; i < N; ++i) {
      if (i > match[i] || match[i] <= 0) continue;
      E.push_back({i-1, match[i]-1});
    }
    return E;
  }
};
```

general-matching.h

Description: Maximum Matching for general graphs (undirected and non bipartite) using Edmond's Blossom Algorithm.

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

0b82ee, 68 lines

```
struct blossom_t {
  int t, n; // 1-based indexing!!
  vector<vector<int>> edges;
  vector<int> seen, parent, og, match, aux, Q;
  blossom_t(int _n) : n(_n), edges(n+1), seen(n+1),
  parent(n+1), og(n+1), match(n+1), aux(n+10), t(0) {}
  void addEdge(int u, int v) {
    edges[u].push_back(v);
    edges[v].push_back(u);
  }
  void augment(int u, int v) {
    int pv = v, nv; // flip states of edges on u-v path
    do {
      pv = parent[v]; nv = match[pv];
      match[v] = pv; match[pv] = v;
      v = nv;
    } while(u != pv);
  }
  int lca(int v, int w) { // find LCA in O(dist)
    ++t;
    while (1) {
      if (v) {
        if (aux[v] == t) return v; aux[v] = t;
        v = og[parent[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while (og[v] != a) {
      parent[v] = w; w = match[v]; // go other way around cycle
      if(seen[w] == 1) Q.push_back(w), seen[w] = 0;
      og[v] = og[w] = a;      // merge into supernode
      v = parent[w];
    }
  }
  bool bfs(int u) {
    for (int i = 1; i <= n; ++i) seen[i] = -1, og[i] = i;
    Q = vector<int>(); Q.push_back(u); seen[u] = 0;
    for(int i = 0; i < Q.size(); ++i) {
      int v = Q[i];
      for(auto &x : edges[v]) {
        if (seen[x] == -1) {
```

```
        parent[x] = v; seen[x] = 1;
        if (!match[x]) return augment(u, x), true;
        Q.push_back(match[x]); seen[match[x]] = 0;
      } else if (seen[x] == 0 && og[v] != og[x]) {
        int a = lca(og[v], og[x]);
        blossom(x, v, a); blossom(v, x, a);
      }
    }
  }
  return false;
}
int solve() {
  int ans = 0; // find random matching (not necessary,
  vector<int> V(n-1); iota(V.begin(), V.end(), 1);// constant
        improvement)
  shuffle(V.begin(), V.end(), mt19937(0x94949));
  for(auto &x : V) if(!match[x])
    for(auto &y : edges[x]) if (!match[y]) {
      match[x] = y, match[y] = x;
      ++ans; break;
    }
  for (int i = 1; i <= n; ++i)
    if (!match[i] && bfs(i)) ++ans;
  return ans;
}
};
```

max-independent-set.h

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

min-vertex-cover.h

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

"bipartite-matching.h" 31f695, 20 lines

```
vector<int> cover(bm_t& B, int N, int M) {
  int ma = B.solve();
  vector<bool> lfound(N, true), seen(N+M);
  for (int i = N; i < N+M; ++i) if (B.match[i] != -1)
    lfound[B.match[i]] = false;
  vector<int> q, cover;
  for (int i = 0; i < N; ++i) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int v = q.back(); q.pop_back();
    lfound[v] = true;
    for(int e : B.adj[v]) if (!seen[e] && B.match[e] != -1) {
      seen[e] = true;
      q.push_back(B.match[e]);
    }
  }
  for (int i = 0; i < N; ++i) if (!lfound[i]) cover.push_back(i
      );
  for (int i = N; i < N+M; ++i) if (seen[i]) cover.push_back(i)
      ;
  assert(cover.size() == ma);
  return cover;
}
```

min-edge-cover.h

Description: Finds a minimum edge cover in a bipartite graph. The size is the same as the number of vertices minus the size of a maximum matching. The mark vector represents who the vertices of set B has an edge to.
Usage: `vector<int> mark(n+m, -1);`
`auto cover = minEdgeCover(g, mark, n, m);`

"bipartite-matching.h" ad86e9, 13 lines

```
vector<pair<int,int>> minEdgeCover(bm_t& g, vector<int>& mark,
    int N, int M) {
  int ma = g.solve();
  vector<pair<int,int>> cover;
  for (int i = 0; i < N; ++i) {
    if (g.match[i] >= 0) cover.push_back({i, g.match[i]-N});
    else if (int(g.adj[i].size()))
      cover.push_back({i, g.adj[i][0] - N});
  }
  for (int i = N; i < N + M; ++i)
    if (g.match[i] == -1 && mark[i] >= 0)
      cover.push_back({mark[i], i - N});
  return cover;
}
```

min-path-cover.h

Description: Finds a minimum vertex-disjoint path cover in a dag. The size is the same as the number of vertices minus the size of a maximum matching.

"bipartite-matching.h" 212c5c, 15 lines

```
vector<vector<int>> minPathCover(bm_t& g, int N) {
  int how_many = int(g.adj.size()) - g.solve();
  vector<vector<int>> paths;
  for (int i = 0; i < N; ++i)
    if (g.match[i + N] == -1) {
      vector<int> path = {i};
      int cur = i;
      while (g.match[cur] >= 0) {
        cur = g.match[cur] - N;
        path.push_back(cur);
      }
      paths.push_back(path);
    }
  return paths;
}
```

7.4 DFS algorithms

dfs-tree.h

Description: Builds dfs tree. Find cut vertices and bridges.
Usage: `Call solve right after build the graph` 72963d, 52 lines

```
struct tree_t {
  int n, timer;
  vector<vector<int>> edges;
  vector<pair<int,int>> bridges;
  vector<int> parent, mindepth, depth, st, child;
  vector<bool> cut;
  tree_t(int N) : n(N), timer(0), edges(n), parent(n,-1),
  mindepth(n,-1), depth(n,-1), st(n,-1) {}
  void addEdge(int a, int b) {
    edges[a].push_back(b); edges[b].push_back(a);
  }
  void dfs(int v) {
    st[v] = timer;
    mindepth[v] = depth[v];
    for (int u : edges[v]) {
      if (u == parent[v]) continue;
      if (st[u] == timer) {
        mindepth[v] = min(mindepth[v], depth[u]);
        continue;
      }
      depth[u] = 1 + depth[v];
      parent[u] = v;
      dfs(u);
      mindepth[v] = min(mindepth[v], mindepth[u]);
    }
  }
  vector<pair<int,int>> find_bridges() {
    for (int i = 0; i < n; ++i)
      if (parent[i] != -1 && mindepth[i] == depth[i])
        bridges.push_back({parent[i], i});
    return bridges;
  }
  vector<bool> find_cut() {
    cut.resize(n), child.resize(n);
    for (int i = 0; i < n; ++i)
      if (parent[i] != -1 && mindepth[i] >= depth[parent[i]])
        cut[parent[i]] = 1;
    for (int i = 0; i < n; ++i)
      if (parent[i] != -1) child[parent[i]]++;
    for (int i = 0; i < n; ++i)
      if (parent[i] == -1 && child[i] < 2) cut[i] = 0;
    return cut;
  }
  void solve() {
    for (int i = 0; i < n; ++i)
      if (depth[i] == -1) {
        depth[i] = 0; parent[i] = -1;
        ++timer;
        dfs(i);
      }
  }
};
```

centroid-decomposition.h

Description: Divide and Conquer on Trees. dd21a1, 75 lines

```
template<typename T> struct centroid_t {
  int N;
  vector<vector<int>> adj;
  vector<vector<int>> dist; // dist to all ancestors
  vector<bool> blocked; // processed centroid
  vector<int> sz, depth, parent; // centroid parent
  centroid_t(int _n) : N(_n), adj(_n), dist(32 - __builtin_clz(
      _n), vector<int>(_n)),
  blocked(_n), sz(_n), depth(_n), parent(_n) {}
  void add_edge(int a, int b) {
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  void dfs_sz(int cur, int prv) {
    sz[cur] = 1;
    for (int nxt : adj[cur]) {
      if (nxt == prv || blocked[nxt]) continue;
      dfs_sz(nxt, cur);
      sz[cur] += sz[nxt];
    }
  }
  int find(int cur, int prv, int tsz) {
    for (int nxt : adj[cur])
      if (!blocked[nxt] && nxt != prv && 2*sz[nxt] > tsz)
        return find(nxt, cur, tsz);
    return cur;
  }
  void dfs_dist(int cur, int prv, int layer, int d) {
    dist[layer][cur] = d;
    for (int nxt : adj[cur]) {
      if (blocked[nxt] || nxt == prv) continue;
      dfs_dist(nxt, cur, layer, d + 1);
    }
  }
  void get_path(int cur, int prv, int d, vector<int>& cur_path)
      {
    cur_path.push_back(d);
    for (int nxt : adj[cur]) {
      if (nxt == prv || blocked[nxt]) continue;
      get_path(nxt, cur, d + 1, cur_path);
```

```
    }
  }
  // solve for each subtree (cnt := # of paths of length K
  // that goes through vertex cur)
  T solve_subtree(int cur, int prv, int K) {
    vector<T> dp(sz[prv] + 1); dp[0] = 1;
    T cnt = 0;
    for (int nxt : adj[cur]) {
      if (blocked[nxt]) continue;
      vector<int> path;
      get_path(nxt, cur, 1, path);
      for (int d : path) {
        if (d > K || K - d > sz[prv]) continue;
        cnt += dp[K - d];
      }

      for (int d : path) dp[d] += 1;
    }
    return cnt;
  }
  T decompose(int cur, int K, int layer = 0, int prv_root = -1)
      {
    dfs_sz(cur, -1);
    int root = find(cur, cur, sz[cur]);
    blocked[root] = true;
    depth[root] = layer;
    parent[root] = prv_root;
    dfs_dist(root, root, layer, 0);

    T res = solve_subtree(root, cur, K);

    for (int nxt : adj[root]) {
      if (blocked[nxt]) continue;
      res += decompose(nxt, K, layer + 1, root);
    }
    return res;
  }
};
```

tarjan.h

Description: Finds all strongly connected components in a directed graph.
Usage: `scc_t s(g); s.solve([&](const vector<int>& cc) {...});` visits all components in reverse topological order.
Time: $\mathcal{O}(E+V)$

48ded7, 34 lines

```
struct scc_t {
    int n, t, scc_num;
    vector<vector<int>> adj;
    vector<int> low, id, stk, in_stk, cc_id;
    scc_t(const vector<vector<int>>& g) : n(int(g.size())), t
        (0), scc_num(0),
    adj(g), low(n,-1), id(n,-1), in_stk(n, false), cc_id(n) {}
    template<class F> void dfs(int cur, F f) {
        id[cur] = low[cur] = t++;
        stk.push_back(cur);
        in_stk[cur] = true;
        for (int nxt : adj[cur]) {
            if (id[nxt] == -1) {
                dfs(nxt, f);
                low[cur] = min(low[cur], low[nxt]);
            } else if (in_stk[nxt])
                low[cur] = min(low[cur], id[nxt]);
        }
        if (low[cur] == id[cur]) {
            vector<int> cc; cc.reserve(stk.size());
            while (true) {
                int v = stk.back(); stk.pop_back();
                in_stk[v] = false;
                cc.push_back(v);
                cc_id[v] = scc_num;
                if (v == cur) break;
            } f(cc); scc_num ++;
        }
    }
    template<class F> void solve(F f) {
        stk.reserve(n);
        for (int r = 0; r < n; ++r)
            if (id[r] == -1) dfs(r, f);
    }
};
```

kosaraju.h

Description: Kosaraju's Algorithm, DFS twice to generate strongly connected components in topological order. a, b in same component if both $a \to b$ and $b \to a$ exist.
Time: $\mathcal{O}(V+E)$

25be07, 35 lines

```
struct Kosaraju_t {
  int n;
  vector<vector<int>> edges, redges;
  vector<bool> seen;
  vector<int> cnt_of, cnts;
  Kosaraju_t(const int &N) : n(N), edges(N), redges(N), seen(N)
      , cnt_of(N, -1) {}
  void addEdge(int a, int b) {
    edges[a].push_back(b);
    redges[b].push_back(a);
  }
  void dfs(int v) {
    seen[v] = true;
    for (int u : edges[v]) {
      if (seen[u]) continue;
      dfs(u);
    }
    toposort.push_back(v);
  }
  void dfs_fix(int v, int w) {
    cnt_of[v] = x;
    for (int u : redges[v]) {
      if (cnt_of[u] == -1) dfs_fix(u, w);
    }
  }
  void solve() {
    for (int i = 0; i < n; ++i)
      if (seen[i] == false) dfs(i);
    reverse(toposort.begin(), toposort.end());
    for (int u : toposort) {
      if (cnt_of[u] != -1) continue;
      dfs_fix(u, u);
      cnts.push_back(u);
    }
  }
};
```

bcc.h

Description: Finds all biconnected components in an undirected graph. In a biconnected component there are at least two distinct paths between any two nodes or the component is a bridge. Note that a node can be in several components. *blockcut* constructs the block cut tree of given graph. The first nodes represents the blocks, the others represents the articulation points.
Usage: `int e_id = 0; vector<pair<int, int>> g(N);`
`for (auto [a,b] : edges) {`
`g[a].emplace_back(b, e_id);`
`g[b].emplace_back(a, e_id++); }`
`bcc_t b(g); b.solve([&](const vector<int>& edges_id) {...});`
Time: $\mathcal{O}(E+V)$

11efd8, 63 lines

```
struct bcc_t{
    using pii = pair<int, int>; // v, e_id
    int n, t;
    vector<vector<pii>> adj;
    vector<int> low, id, stk, is_art;
    bcc_t(const vector<vector<pii>> &g) : n(int(g.size())), t
        (0),
    adj(g), low(n,-1), id(n,-1), is_art(n) {}
    template<class F> void dfs(int cur, int e_par, F f){
        id[cur] = low[cur] = t++;
        stk.push_back(e_par);
        int c = 0;
        for (auto [nxt, e_id] : adj[cur]) {
            if (id[nxt] == -1) {
                dfs(nxt, e_id, f);
                low[cur] = min(low[cur], low[nxt]);
                c ++;
                if (low[nxt] >= id[cur]) {
                    is_art[cur] = true;
                    auto top = find(stk.rbegin(), stk.rend(),
                        e_id);
                    vector<int> cc(stk.rbegin(), next(top));
                    f(cc);
                    stk.resize(stk.size() - cc.size());
                }
            }
            else if (e_id != e_par) {
                low[cur] = min(low[cur], id[nxt]);
                if (id[nxt] < id[cur]) stk.push_back(e_id);
            }
        }
        if(e_par == -1) is_art[cur] = (c > 1) ? true : false;
    }
    template<class F> void solve(F f) {
        stk.reserve(n);
        for (int r = 0; r < n; ++r)
            if (id[r] == -1) dfs(r, -1, f);
    }
    auto blockcut(const vector<pii> &edges){
        vector<vector<int>> cc;
        vector<int> cc_id(n);
        solve( [&](const vector<int> &c) {
            set<int> vc;
            for(int e : c){
                auto [a, b] = edges[e];
                cc_id[a] = cc_id[b] = int(cc.size());
                vc.insert(a);
                vc.insert(b);
            }
            cc.emplace_back(vc.begin(), vc.end());
        } );
        for(int a = 0; a < n; a++) if(is_art[a]) {
            cc_id[a] = int(cc.size());
            cc.push_back({a});
        }
        int bcc_num = int(cc.size());
        vector<vector<int>> tree(bcc_num);
        for(int c = 0; c < bcc_num && 1 < int(cc[c].size()); ++
            c)
            for(int a : cc[c]) if(is_art[a]) {
                tree[c].push_back(cc_id[a]);
                tree[cc_id[a]].push_back(c);
            }
        return make_tuple(cc_id, cc, tree);
    }
};
```

2sat.h

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

Usage: `TwoSat ts(number of boolean variables);`
```
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.set_value(2); // Var 2 is true
ts.at_most_one({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars
```
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

"tarjan.h" 10ff6d, 42 lines

```
struct TwoSat {
    int N;
    vector<vector<int>> gr;
    vector<int> values; // 0 = false, 1 = true
    TwoSat(int n = 0) : N(n), gr(2*n) {}
    int add_var() { // (optional)
        gr.emplace_back();
        gr.emplace_back();
        return N++;
    }
    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j);
        gr[f].push_back(j^1);
        gr[j].push_back(f^1);
    }
    void implies(int f, int j) { either(~f, j); }
    void set_value(int x) { either(x, x); }
    void at_most_one(const vector<int>& li) { // (optional)
        if (int(li.size()) <= 1) return;
        int cur = ~li[0];
        for (int i = 2; i < int(li.size()); ++i) {
            int next = add_var();
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }
    bool solve() {
        scc_t s(gr);
        s.solve([](const vector<int> &v){ return; } );
        values.assign(N, -1);
        for (int i = 0; i < N; ++i) if (s.cc_id[2*i] == s.cc_id
            [2*i+1]) return 0;
        for (int i = 0; i < N; ++i){
            if (s.cc_id[2*i] < s.cc_id[2*i+1]) values[i] =
                false;
            else values[i] = true;
        }
        return 1;
    }
};
```

7.5 Heuristics

maximal-cliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Possible optimization: on the top-most recursion level, ignore 'cands', and go through nodes in order of increasing degree, where degrees go down as nodes are removed.

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

57e107, 12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B> &eds, F f, B P = ~B(), B X={}, B R={}) {
  if (!P.any()) { if (!X.any()) f(R); return; }
  auto q = (P | X)._Find_first();
  auto cands = P & ~eds[q];
  for(int i = 0; i < eds.size(); ++i) if (cands[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

maximum-clique.h

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

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

261d2e, 49 lines

```
using vb = vector<bitset<40>>;
struct Maxclique {
  double limit = 0.025, pk = 0;
  struct Vertex { int i, d = 0; };
  using vv = vector<Vertex>;
  vb e;
  vv V;
  vector<vector<int>> C;
  vector<int> qmax, q, S, old;
  void init(vv& r) {
    for(auto& v : r) v.d = 0;
    for(auto& v : r) for(auto& j : r) v.d += e[v.i][j.i];
    sort(r.begin(), r.end(), [](auto a, auto b) { return a.d >
        b.d; });
    int mxD = r[0].d;
    for(int i = 0; i < int(r.size()); ++i) r[i].d = min(i, mxD)
        + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (int(R.size())) {
      if (int(q.size()) + R.back().d <= int(qmax.size()))
          return;
      q.push_back(R.back().i);
      vv T;
      for(auto& v : R) if (e[R.back().i][v.i]) T.push_back({v.i
          });
      if (int(T.size())) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(int(qmax.size()) - int(q.
            size()) + 1, 1);
        C[1].clear(), C[2].clear();
        for(auto& v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(C[k].begin(), C[k].end(), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        for(int k = mnk; k <= mxk; ++k) for(int i : C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (int(q.size()) > int(qmax.size())) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vector<int> maxClique() { init(V), expand(V); return qmax; }
  Maxclique(vb conn) : e(conn), C(int(e.size())+1), S(int(C.
      size())), old(S) {
    for(int i = 0; i < int(e.size()); ++i) V.push_back({i});
  }
};
```

chromatic-number.h

Description: Compute the chromatic number of a graph. Minimum number of colors needed to paint the graph in a way s.t. if two vertices share an edge, they must have distinct colors.

Time: $\mathcal{O}\left(N2^N\right)$

ea44b7, 33 lines

```
template<class T> int min_colors(int N, const T& gr) {
  vector<int> adj(N);
  for (int a = 0; a < N; ++a) {
    for (int b = a + 1; b < N; ++b) {
      if (!gr[a][b]) continue;
      adj[a] |= (1 << b);
      adj[b] |= (1 << a);
    }
  }
  static vector<unsigned> dp(1 << N), buf(1 << N), w(1 << N);
  for (int mask = 0; mask < (1 << N); ++mask) {
    bool ok = true;
    for (int i = 0; i < N; ++i) if (mask & 1 << i) {
      if (adj[i] & mask) ok = false;
    }
    if (ok) dp[mask]++;
    buf[mask] = 1;
    w[mask] = __builtin_popcount(mask) % 2 == N % 2 ? 1 : -1;
  }
  for (int i = 0; i < N; ++i) {
    for (int mask = 0; mask < (1 << N); ++mask) if (!(mask & 1
        << i)) {
      dp[mask^(1 << i)] += dp[mask];
    }
  }
  for (int colors = 1; colors <= N; ++colors) {
    unsigned S = 0;
    for (int mask = 0; mask < (1 << N); ++mask) {
      S += (buf[mask] *= dp[mask]) * w[mask];
    }
    if (S) return colors;
  }
  assert(false);
}
```

cycle-counting.cpp

Description: Counts 3 and 4 cycles

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

77cb6f, 47 lines

```
int count_cycles(const vector<vector<int>>& adj, const vector<
    int>& deg) {
  const int N = int(adj.size());
  vector<int> idx(N), loc(N);
  iota(idx.begin(), idx.end(), 0);
  sort(idx.begin(), idx.end(), [&](const int& a, const int& b)
      { return deg[a] < deg[b]; });
  for (int i = 0; i < N; ++i) loc[idx[i]] = i;

  vector<vector<int>> gr(N);
  for (int a = 0; a < N; ++a) {
    for (int b : adj[a]) {
      if (loc[a] < loc[b]) gr[a].push_back(b);
    }
  }

  int cycle3 = 0;
```

```
  {
    vector<bool> seen(N, false);
    for (int a = 0; a < N; ++a) {
      for (int b : gr[a]) seen[b] = true;
      for (int b : gr[a]) {
        for (int c : gr[b]) {
          if (seen[c]) {
            cycle3 += 1;
          }
        }
      }
      for (int b : gr[a]) seen[b] = false;
    }
  }
  int cycle4 = 0;
  {
    vector<int> cnt(N);
    for (int a = 0; a < N; ++a) {
      for (int b : adj[a]) {
        for (int c : gr[b]) {
          if (loc[a] < loc[c]) {
            cycle4 += cnt[c];
            cnt[c]++;
          }
        }
      }
      for (int b : adj[a]) for (int c : gr[b]) cnt[c] = 0;
    }
  }

  return cycle3;
}
```

edge-coloring.h

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

13a6e5, 32 lines

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

7.6 Trees

lca-binary-lifting.h

Description: Solve lowest common ancestor queries using binary jumps. Can also find the distance between two nodes.
Time: $\mathcal{O}(N \log N + Q \log N)$

cc5b6d, 53 lines

```
struct lca_t {
  int logn{0}, preorderpos{0};
  vector<int> invpreorder, height;
  vector<vector<int>> jump, edges;
  lca_t(int n, vector<vector<int>>& adj) :
    edges(adj), height(n), invpreorder(n) {
      while((1<<(logn+1)) <= n) ++logn;
      jump.assign(n+1, vector<int>(logn+1, 0));
      dfs(0, -1, 0);
    }
  void dfs(int v, int p, int h) {
    invpreorder[v] = preorderpos++;
    height[v] = h;
    jump[v][0] = p < 0 ? v : p;
    for (int l = 1; l <= logn; ++l)
      jump[v][l] = jump[jump[v][l-1]][l-1];
    for (int u : edges[v]) {
      if (u == p) continue;
      dfs(u, v, h + 1);
    }
  }
  int climb(int v, int dist) {
    for (int l = 0; l <= logn; ++l)
      if (dist&(1<<l)) v = jump[v][l];
    return v;
  }
  int query(int a, int b) {
    if (height[a] < height[b]) swap(a, b);
    a = climb(a, height[a] - height[b]);
    if (a == b) return a;
    for (int l = logn; l >= 0; --l)
      if (jump[a][l] != jump[b][l])
        a = jump[a][l], b = jump[b][l];
    return jump[a][0];
  }
  int dist(int a, int b) {
    return height[a] + height[b] - 2 * height[query(a,b)];
  }
  bool is_parent(int p, int v) {
    if (height[p] > height[v]) return false;
    return p == climb(v, height[v] - height[p]);
  }
  bool on_path(int x, int a, int b) {
    int v = query(a, b);
    return is_parent(v, x) && (is_parent(x, a) || is_parent(x,
        b));
  }
  int get_kth_on_path(int a, int b, int k) {
    int v = query(a, b);
    int x = height[a] - height[v], y = height[b] - height[v];
    if (k < x) return climb(a, k);
    else return climb(b, x + y - k);
  }
};
```

lca-euler-tour.h

Description: Data structure for computing lowest common ancestors and build Euler Tour in a tree. Edges should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q + Q \log)$

7da0bf, 164 lines

```
struct small_lca_t {
  int T = 0;
  vector<int> time, path, walk, depth;
  rmq_t<int> rmq;
  small_lca_t(vector<vector<int>> &edges) : time(int(edges.size
      ())),
  depth(time), rmq((dfs(edges,0,-1), walk)) {}
  void dfs(vector<vector<int>> &edges, int v, int p) {
    time[v] = T++;
    for(int u : edges[v]) {
      if (u == p) continue;
      depth[u] = depth[v] + 1;
      path.push_back(v), walk.push_back(time[v]);
      dfs(edges, u, v);
    }
  }
  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
  }
};

struct lca_t {
  int N;
  vector<vector<int>> adj;
  vector<int> parent, depth, sz;
  vector<int> euler_tour, timer;
  vector<int> tour_in, tour_out, postorder;
  vector<int> idx, rev_idx;
  vector<int> heavy_root;
  rmq_t<pair<int,int>> rmq;
  int next_idx = 0, rev_next_idx = 0;
  bool built = false;
  lca_t() : N(0) {}
  lca_t(vector<vector<int>>& _adj, int root = -1, bool
       build_rmq = true) :
    N(int(_adj.size())), adj(_adj), parent(N, -1), depth(N), sz
        (N), timer(N),
    tour_in(N), tour_out(N), postorder(N), idx(N), heavy_root(N
        ),
    built(false) {

      if (0 <= root && root < N) pre_dfs(root, -1);

      euler_tour.reserve(2 * N);

      for (int i = 0; i < N; ++i)
        if (parent[i] == -1) {
          if (i != root) pre_dfs(i, -1);
          dfs(i, false);
          euler_tour.push_back(-1);
        }

      rev_idx = idx;
      reverse(rev_idx.begin(), rev_idx.end());
      assert(int(euler_tour.size()) == 2 * N);
      vector<pair<int, int>> euler_tour_depths;
      euler_tour_depths.reserve(euler_tour.size());

      int id = 0;
      for (int cur : euler_tour) {
        euler_tour_depths.push_back({cur == -1 ? cur : depth[
            cur], id++});
```

```
    }

    if (build_rmq) rmq = rmq_t<pair<int, int>>(
        euler_tour_depths);
    built = true;
  }

  void pre_dfs(int cur, int par) {
    parent[cur] = par;
    depth[cur] = (par == -1 ? 0 : 1 + depth[par]);
    adj[cur].erase(remove(adj[cur].begin(), adj[cur].end(), par
        ), adj[cur].end());
    sz[cur] = 1;
    for (int nxt : adj[cur]) {
      pre_dfs(nxt, cur);
      sz[cur] += sz[nxt];
    }
    if (!adj[cur].empty()) {
      auto w = max_element(adj[cur].begin(), adj[cur].end(),
          [&](int a, int b) { return sz[a] < sz[b]; });
      swap(*adj[cur].begin(), *w);
    }
  }

  void dfs(int cur, bool heavy) {
    heavy_root[cur] = heavy ? heavy_root[parent[cur]] : cur;
    timer[cur] = int(euler_tour.size());
    euler_tour.push_back(cur);
    idx[next_idx] = cur;
    tour_in[cur] = next_idx++;
    bool heavy_child = true;
    for (int next : adj[cur]) {
      dfs(next, heavy_child);
      euler_tour.push_back(cur);
      heavy_child = false;
    }
    tour_out[cur] = next_idx;
    postorder[cur] = rev_next_idx++;
  }

  pair<int, array<int, 2>> get_diameter() const {
    assert(built);
    pair<int, int> u_max = {-1, -1};
    pair<int, int> ux_max = {-1, -1};
    pair<int, array<int, 2>> uxv_max = {-1, {-1, -1}};
    for (int cur : euler_tour) {
      if (cur == -1) break;
      u_max = max(u_max, {depth[cur], cur});
      ux_max = max(ux_max, {u_max.first - 2 * depth[cur], u_max
          .second});
      uxv_max = max(uxv_max, {ux_max.first + depth[cur], {
          ux_max.second, cur}});
    }
    return uxv_max;
  }

  int query(int a, int b) const {
    if (a == b) return a;
    a = timer[a], b = timer[b];
    if (a > b) swap(a, b);
    return euler_tour[rmq.query(a, b).second];
  }

  bool is_ancestor(int a, int b) const {
    return tour_in[a] <= tour_in[b] && tour_in[b] < tour_out[a
        ];
  }

  bool on_path(int x, int a, int b) const {
    return (is_ancestor(x, a) || is_ancestor(x, b)) &&
        is_ancestor(query(a, b), x);
  }

  int dist(int a, int b) const {
    return depth[a] + depth[b] - 2 * depth[query(a, b)];
  }

  int child_ancestor(int a, int b) const {
    assert(a != b); assert(is_ancestor(a, b));
    // Note: this depends on rmq_t breaking ties by latest
        index.
    int child = euler_tour[rmq.query(timer[a], timer[b]).second
        + 1];
    assert(parent[child] == a);
    assert(is_ancestor(child, b));
    return child;
  }

  int get_kth_ancestor(int a, int k) const {
    while (a >= 0) {
      int root = heavy_root[a];
      if (depth[root] <= depth[a] - k) return idx[tour_in[a] -
          k];
      k -= depth[a] - depth[root] + 1;
      a = parent[root];
    }
    return a;
  }

  int get_kth_node_on_path(int a, int b, int k) const {
    int lca = query(a, b);
    int x = depth[a] - depth[lca], y = depth[b] - depth[lca];
    assert(0 <= k && k <= x + y);
    if (k < x) return get_kth_ancestor(a, k);
    else return get_kth_ancestor(b, x + y - k);
  }

  int get_common_node(int a, int b, int c) const {
    // Return the deepest node among lca(a, b), lca(b, c), and
        lca(c, a).
    int x = query(a, b), y = query(b, c), z = query(c, a);
    x = depth[y] > depth[x] ? y : x;
    x = depth[z] > depth[x] ? z : x;
    return x;
  }
};
```

compress-tree.h

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

"LCA.h" ae0a91, 20 lines

```
vector<pair<int,int>> compressTree(lca_t &lca, const vector<int
    >& subset) {
  static vector<int> rev; rev.resize(lca.time.size());
  vector<int> li = subset, &T = lca.time;
  auto cmp = [&](int a, int b) { return T[a] < T[b]; };
  sort(li.begin(), li.end(), cmp);
  int m = li.size()-1;
  for (int i = 0; i < m; ++i) {
    int a = li[i], b = li[i+1];
    li.push_back(lca.lca(a, b));
  }
  sort(li.begin(), li.end(), cmp);
  li.erase(unique(li.begin(), li.end()), li.end());
  for (int i = 0; i < int(li.size()); ++i) rev[li[i]] = i;
  vector<pair<int,int>> ret = {{0, li[0]}};
  for (int i = 0; i < li.size()-1; ++i) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.lca(a, b)], b);
  }
  return ret;
}
```

heavylight.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code supports commutative segtree modifications/queries on paths, edges and subtrees. Takes as input the full adjacency list with pairs of (vertex, value). USE_EDGES being true means that values are stored in the edges and are initialized with the adjacency list, otherwise values are stored in the nodes and are initialized to the T defaults value.
Time: $\mathcal{O}\left((\log N)^2\right)$

"../data-structures/lazy-segtree.h" 053b35, 70 lines

```
template<bool use_edges> struct HLD_t {
  int N, T{};
  vector<vector<int>> adj;
  vector<int> sz, depth, chain, par, in, out, preorder;
  HLD_t() {}
  HLD_t(const vector<vector<int>>& G, int r = 0) : N(int(G.size
      ())), adj(G),
  sz(N), depth(N), chain(N), par(N, -1), in(N), out(N),
      preorder(N) {
    dfs_sz(r); chain[r] = r; dfs_hld(r); }
  void dfs_sz(int cur) {
    if (~par[cur]) {
      adj[cur].erase(find(adj[cur].begin(), adj[cur].end(), par
          [cur]));
    }
    sz[cur] = 1;
    for (auto& nxt : adj[cur]) {
      par[nxt] = cur; depth[nxt] = 1 + depth[cur];
      dfs_sz(nxt); sz[cur] += sz[nxt];
      if (sz[nxt] > sz[adj[cur][0]]) swap(nxt, adj[cur][0]);
    }
  }
  void dfs_hld(int cur) {
    in[cur] = T++;
    preorder[in[cur]] = cur;
    for (auto& nxt : adj[cur]) {
      chain[nxt] = (nxt == adj[cur][0] ? chain[cur] : nxt);
      dfs_hld(nxt);
    }
    out[cur] = T;
  }
  int lca(int a, int b) {
    while (chain[a] != chain[b]) {
      if (in[a] < in[b]) swap(a, b);
      a = par[chain[a]];
    }
    return (in[a] < in[b] ? a : b);
  }
  bool is_ancestor(int a, int b) { return in[a] <= in[b] && in[
      b] < out[a]; }
  int climb(int a, int k) {
    if (depth[a] < k) return -1;
    int d = depth[a] - k;
    while (depth[chain[a]] > d) a = par[chain[a]];
    return preorder[in[a] - depth[a] + d];
  }
  int kth_on_path(int a, int b, int K) {
    int m = lca(a, b);
    int x = depth[a] - depth[m], y = depth[b] - depth[m];
    if (K > x + y) return -1;
    return (x > K ? climb(a, K) : climb(b, x + y - K));
```

```
  }
  // bool is true if path should be reversed (only for
      noncommutative operations)
  const vector<tuple<bool, int, int>>& get_path(int a, int b)
      const {
    static vector<tuple<bool, int, int>> L, R;
    L.clear(); R.clear();
    while (chain[a] != chain[b]) {
      if (depth[chain[a]] > depth[chain[b]]) {
        L.push_back({true, in[chain[a]], in[a] + 1});
        a = par[chain[a]];
      } else {
        R.push_back({false, in[chain[b]], in[b] + 1});
        b = par[chain[b]];
      }
    }
    if (depth[a] > depth[b]) {
      L.push_back({true, in[b] + use_edges, in[a] + 1});
    } else {
      R.push_back({false, in[a] + use_edges, in[b] + 1});
    }
    L.insert(L.end(), R.rbegin(), R.rend());
    return L;
  }
};
```

tree-isomorphism.h

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

92e59f, 51 lines

```
map<vector<int>, int> delta;
struct tree_t {
  int n;
  pair<int,int> centroid;
  vector<vector<int>> edges;
  vector<int> sz;
  tree_t(vector<vector<int>>& graph) :
    edges(graph), sz(edges.size()) {}
  int dfs_sz(int v, int p) {
    sz[v] = 1;
    for (int u : edges[v]) {
      if (u == p) continue;
      sz[v] += dfs_sz(u, v);
    }
    return sz[v];
  }
  int dfs(int tsz, int v, int p) {
    for (int u : edges[v]) {
      if (u == p) continue;
      if (2*sz[u] <= tsz) continue;
      return dfs(tsz, u, v);
    }
    return centroid.first = v;
  }
  pair<int,int> find_centroid(int v) {
    int tsz = dfs_sz(v, -1);
    centroid.second = dfs(tsz, v, -1);
    for (int u : edges[centroid.first]) {
      if (2*sz[u] == tsz)
        centroid.second = u;
    }
    return centroid;
  }
  int hash_it(int v, int p) {
    vector<int> offset;
    for (int u : edges[v]) {
      if (u == p) continue;
      offset.push_back(hash_it(u, v));
    }
    sort(offset.begin(), offset.end());
    if (!delta.count(offset))
      delta[offset] = int(delta.size());
    return delta[offset];
  }
  lint get_hash(int v = 0) {
    pair<int,int> cent = find_centroid(v);
    lint x = hash_it(cent.first, -1), y = hash_it(cent.second,
        -1);
    if (x > y) swap(x, y);
    return (x << 30) + y;
  }
};
```

7.6.1 Sqrt Decomposition

HLD generally suffices. If not, here are some common strategies:

- Rebuild the tree after every $\sqrt{N}$ queries.
- Consider vertices with $>$ or $<$ $\sqrt{N}$ degree separately.
- For subtree updates, note that there are $O(\sqrt{N})$ distinct sizes among child subtrees of any vertex.

Block Tree: Use a DFS to split edges into contiguous groups of size $\sqrt{N}$ to $2\sqrt{N}$.

Mo's Algorithm for Tree Paths: Maintain an array of vertices where each one appears twice, once when a DFS enters the vertex (`st`) and one when the DFS exists (`en`). For a tree path $u \leftrightarrow v$ such that `st[u]<st[v]`,

- If u is an ancestor of v, query `[st[u],st[v]]`.
- Otherwise, query `[en[u],st[v]]` and consider $lca(u, v)$ separately.

7.7 Functional Graphs

functional-graph.h

Description: finds the directions of the edges of given functional graph, returns pair of *parent* and *indegree* of each vertex. Useful together with functional-digraph.h.

46a635, 41 lines

```
pair<vector<int>, vector<int>> make_functional_digraph(const
      vector<vector<int>> &g, vector<int> deg){
  int n = int(g.size());
  vector<int> par(n), indeg(n);
  vector<bool> vis(n);
  queue<int> q;
  for(int u=0; u<n; u++)
    if(deg[u] == 1)
      q.push(u);
  while(!q.empty()){
    int u = q.front();
    q.pop();
    vis[u] = true;
    for(int v: g[u]){
      if(vis[v]) continue;
      par[u] = v;
      indeg[v]++;
      deg[v]--;
      if(deg[v] == 1)
        q.push(v);
    }
  }
  for(int u=0; u<n; u++){
    if(vis[u]) continue;
    int cur = u, nxt = -1;
    while(nxt != u){
      vis[cur] = true;
      nxt = -1;
      for(int x: g[u])
        if(!vis[x]){
          nxt = x;
          break;
        }
      if(nxt == -1)
        nxt = u;
      indeg[nxt]++;
      par[cur] = nxt;
      cur = nxt;
    }
  }
  return {par, indeg};
}
```

functional-digraph.h

Description: Called lumberjack technique, solve functional graphs problems for digraphs, it's also pretty good for dp on trees. Consists in go cutting the leaves until there is no leaves, only cycles. For that we keep a processing queue of the leaves, note that during this processing time we go through all the childrens of v before reaching a vertex v, therefore we can compute some infos about the children, like subtree of a given vertex

Usage: `Lumberjack<10010> g; g.init(par, indeg);`
`// (Be careful with the size of cyles when declared locally!)`

df0687, 69 lines

```
template<int T> struct Lumberjack {
  int n, numcycle;
  vector<int> subtree, order, par, cycle;
  vector<int> parincycles, idxcycle, sz, st;
  vector<int> depth, indeg, cycles[T];
  vector<bool> seen, incycle, leaf;
  void init(vector<int>& _par, vector<int>& _indeg) {
    n = int(_par.size());
    par = _par;
    indeg = _indeg;
    order.resize(0);
    subtree.assign(n, 0);
    seen.assign(n, false);
    sz = st = subtree;
    parincycles = cycle = sz;
    idxcycle = depth = sz;
    incycle = leaf = seen;
    bfs();
  }
  void find_cycle(int u){
    int idx= ++numcycle, cur = 0, p = u;
    st[idx] = u;
    sz[idx] = 0;
    cycles[idx].clear();
    while (!seen[u]) {
      seen[u] = incycle[u] = 1;
      parincycles[u] = u;
      cycle[u] = idx;
      idxcycle[u] = cur;
      cycles[idx].push_back(u);
      ++sz[idx];
      depth[u] = 0;
      ++subtree[u];
      u = par[u];
      ++cur;
    }
```

```
  }
  void bfs() {
    queue<int> q;
    for (int i = 0; i < n; ++i)
      if (!indeg[i]){
        seen[i] = leaf[i] = true;
        q.push(i);
      }
    while(!q.empty()){
      int v = q.front(); q.pop();
      order.push_back(v);
      ++subtree[v];
      int curpar = par[v];
      indeg[curpar]--;
      subtree[curpar] += subtree[v];
      if(!indeg[curpar]){
        q.push(curpar);
        seen[curpar] = true;
      }
    }
    numcycle = 0;
    for (int i = 0; i < n; ++i)
      if (!seen[i]) find_cycle(i);
    for(int i = order.size()-1; i >= 0; --i){
      int v = order[i], curpar = par[v];
      parincycles[v] = parincycles[curpar];
      cycle[v] = cycle[curpar];
      incycle[v] = false;
      idxcycle[v] = -1;
      depth[v] = 1 + depth[curpar];
    }
  }
};
```

7.8 Other

directed-mst.h

Description: Edmonds' algorithm for finding the weight of the minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

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

"../data-structures/dsu-rollback.h" dedbb2, 59 lines

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

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

7.9 Theorems

7.9.1 Landau

There are a tournament with outdegree $d_1 \le d_2 \le \ldots \le d_n$ iff:

- $d_1 + d_2 + \ldots + d_n = \binom{n}{2}$
- $d_1 + d_2 + \ldots + d_k \ge \binom{k}{2} \quad \forall 1 \le k \le n.$

In order to build, lets make 1 point to $2, 3, \ldots, d_1 + 1$ and we follow recursively

7.9.2 Euler's theorem

Let V, A and F be the number of vertices, edges and faces of connected planar graph, $V - A + F = 2$

7.9.3 Eulerian Cycles

The number of Eulerian cycles in a *directed* graph G is:

$$t_w(G) \prod_{v \in G} (\deg v - 1)!,$$

where $t_w(G)$ is the number of arborescences ("directed spanning" tree) rooted at w (Check Number of Spanning Trees)

7.9.4 Dilworth's theorem

For any partially ordered set, the sizes of the max antichain and of the min chain decomposition are equal. Equivalent to Konig's theorem on the bipartite graph (U, V, E) where $U = V = S$ and (u, v) is an edge when $u < v$. Those vertices outside the min vertex cover in both U and V form a max antichain

7.9.5 König-Egervary theorem

For Bipartite Graphs, the number of edges in the maximum matching is greater than or equal the number of vertices in the minimum cover

Maximum Weight Closure

Given a vertex-weighted directed graph G. Turn the graph into a flow network, adding weight ∞ to each edge. Add vertices S, T. For each vertex v of weight w, add edge (S, v, w) if $w \ge 0$, or edge $(v, T, -w)$ if $w < 0$. Sum of positive weights minus minimum $S - T$ cut is the answer. Vertices reachable from S are in the closure. The maximum-weight closure is the same as the complement of the minimum-weight closure on the graph with edges reversed.

7.9.6 Maximum Weighted Independent Set in a Bipartite Graph

This is the same as the minimum weighted vertex cover. Solve this by constructing a flow network with edges $(S, u, w(u))$ for $u \in L$, $(v, T, w(v))$ for $v \in R$ and (u, v, ∞) for $(u, v) \in E$. The minimum S, T-cut is the answer. Vertices adjacent to a cut edge are in the vertex cover.

7.9.7 Tutte-Berge formula

The theorem states that the size of a maximum matching of a graph $G = (V, E)$ equals $\frac{1}{2} \min_{U \subseteq V} (|U| - \mathtt{odd}(G - U) + |V|)$, where $\mathtt{odd}$(H) counts how many of the connected components of the graph H have an odd number of vertices.

7.9.8 Tutte's theorem

A graph $G = (V, A)$ has a perfect matching iff for all subset U of V, the induced subgraph by $V \setminus U$ has at most $|U|$ connected components with odd number of vertices.

7.9.9 Number of Spanning Trees

Define Laplacian Matrix as $L = D - A$, D being a Diagonal Matrix with $D_{i,i} = deg(i)$ and A an Adjacency Matrix. Create an $N \times N$ Laplacian 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).

7.9.10 Tutte Matrix

- A graph has a perfect matching iff the *Tutte* matrix has a non-zero determinant.
- The rank of the *Tutte* matrix is equal to twice the size of the maximum matching. The maximum cost matching can be found by polynomial interpolation.

7.9.11 Menger's theorem

- Vertices: A graph is k-connected iff all pairwise vertices are connected to at least k internally disjoint paths.
- Edges: A graph is called k-edge-connected if the removal of at least k edges of the graph keeps it connected. A graph is k-edge-connected iff for all pairwise vertices u and v, exist k paths which link u to v without sharing an edge.

Geometry (8)

8.1 Geometric primitives

Point.h

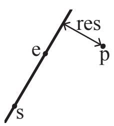
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

f90ade, 27 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)); }
};
using P = Point<double>;
```

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.

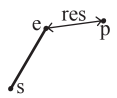


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

Description:
Returns the shortest distance between point p and the line segment from point s to e.
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;

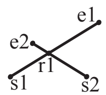


"Point.h" ae751a, 5 lines

```
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" f6be16, 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 {s.begin(), s.end()};
}
```

SegmentIntersectionQ.h

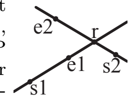
Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

"Point.h" 1ff4ba, 16 lines

```
template<class P>
bool segmentIntersectionQ(P s1, P e1, P s2, P e2) {
  if (e1 == s1) {
    if (e2 == s2) return e1 == e2;
    swap(s1,s2); swap(e1,e2);
  }
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = d.cross(v1), a2 = d.cross(v2);
  if (a == 0) { // parallel
    auto b1 = s1.dot(v1), c1 = e1.dot(v1),
       b2 = s2.dot(v1), c2 = e2.dot(v1);
    return !a1 && max(b1,min(b2,c2)) <= min(c1,max(b2,c2));
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  return (0 <= a1 && a1 <= a && 0 <= a2 && a2 <= a);
}
```

LineIntersection.h

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



Usage: auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;

"Point.h" a01f81, 8 lines

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

LineProjectionReflection.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab insted. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
  P v = b - a;
  return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

SideOf.h

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

"Point.h" 3af81c, 9 lines

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

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

OnSegment.h

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

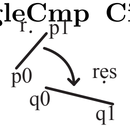
"Point.h" c597e8, 3 lines

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

LinearTransformation.h

Description:

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



"Point.h" 03a306, 6 lines

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

Angle.h

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

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

0f0602, 34 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)};
}
```

AngleCmp.h

Description: Useful utilities for dealing with angles of rays from origin. OK for integers, only uses cross product. Doesn't support (0,0).

6edd25, 22 lines

```
template <class P>
bool sameDir(P s, P t) {
  return s.cross(t) == 0 && s.dot(t) > 0;
}
// checks 180 <= s..t < 360?
template <class P>
bool isReflex(P s, P t) {
  auto c = s.cross(t);
  return c ? (c < 0) : (s.dot(t) < 0);
}
// operator < (s,t) for angles in [base,base+2pi)
template <class P>
bool angleCmp(P base, P s, P t) {
  int r = isReflex(base, s) - isReflex(base, t);
  return r ? (r < 0) : (0 < s.cross(t));
}
// is x in [s,t] taken ccw? 1/0/-1 for in/border/out
template <class P>
int angleBetween(P s, P t, P x) {
  if (sameDir(x, s) || sameDir(x, t)) return 0;
  return angleCmp(s, x, t) ? 1 : -1;
}
```

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" c64785, 10 lines

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

CircleTangents.h

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

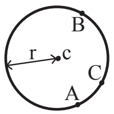
"Point.h" b0153d, 13 lines

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

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" c98102, 8 lines

```
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" 8ab87f, 19 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(ps.begin(),ps.end(), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  for(int i = 0; i < ps.size(); ++i)
    if ((o - ps[i]).dist() > r * EPS) {
      o = ps[i], r = 0;
      for(int j = 0; j < i; ++j) if ((o - ps[j]).dist() > r *
          EPS) {
        o = (ps[i] + ps[j]) / 2;
        r = (o - ps[i]).dist();
        for(int k = 0; k < j; ++k)
          if ((o - ps[k]).dist() > r * EPS) {
            o = ccCenter(ps[i], ps[j], ps[k]);
            r = (o - ps[i]).dist();
          }
      }
    }
  return {o, r};
}
```

CircleUnion.h

Description: Computes the circles union total area

fd65da, 86 lines

```
struct CircleUnion {
    static const int maxn = 1e5 + 5;
    const double PI = acos((double)-1.0);
    int n;
    double x[maxn], y[maxn], r[maxn];
    int covered[maxn];
    vector<pair<double, double>> seg, cover;
    double arc, pol;
    inline int sign(double x) {return x < -EPS ? -1 : x > EPS;}
    inline int sign(double x, double y) {return sign(x - y);}
    inline double sqr(const double x) {return x * x;}
    inline double dist(double x1, double y1, double x2, double
        y2) {return sqrt(sqr(x1 - x2) + sqr(y1 - y2));}
    inline double angle(double A, double B, double C) {
        double val = (sqr(A) + sqr(B) - sqr(C)) / (2 * A * B);
        if (val < -1) val = -1;
        if (val > +1) val = +1;
        return acos(val);
    }
    CircleUnion() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void init() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void add(double xx, double yy, double rr) {
        x[n] = xx, y[n] = yy, r[n] = rr, covered[n] = 0, n++;
    }
    void getarea(int i, double lef, double rig) {
        arc += 0.5 * r[i] * r[i] * (rig - lef - sin(rig - lef))
            ;
        double x1 = x[i] + r[i] * cos(lef), y1 = y[i] + r[i] *
            sin(lef);
        double x2 = x[i] + r[i] * cos(rig), y2 = y[i] + r[i] *
            sin(rig);
```

```
        pol += x1 * y2 - x2 * y1;
    }
    double calc() {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < i; j++)
                if (!sign(x[i] - x[j]) && !sign(y[i] - y[j]) &&
                      !sign(r[i] - r[j])) {
                    r[i] = 0.0;
                    break;
                }
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i != j && sign(r[j] - r[i]) >= 0 && sign(
                       dist(x[i], y[i], x[j], y[j]) - (r[j] - r[i
                       ])) <= 0) {
                    covered[i] = 1;
                    break;
                }
        for (int i = 0; i < n; i++) {
            if (sign(r[i]) && !covered[i]) {
                seg.clear();
                for (int j = 0; j < n; j++)
                    if (i != j) {
                        double d = dist(x[i], y[i], x[j], y[j])
                            ;
                        if (sign(d - (r[j] + r[i])) >= 0 ||
                            sign(d - abs(r[j] - r[i])) <= 0)
                          continue;
                        double alpha = atan2(y[j] - y[i], x[j]
                            - x[i]);
                        double beta = angle(r[i], d, r[j]);
                        pair<double, double> tmp(alpha - beta,
                            alpha + beta);
                        if (sign(tmp.first) <= 0 && sign(tmp.
                            second) <= 0)
                            seg.push_back(pair<double, double
                                >(2 * PI + tmp.first, 2 * PI +
                                 tmp.second));
                        else if (sign(tmp.first) < 0) {
                            seg.push_back(pair<double, double
                                >(2 * PI + tmp.first, 2 * PI))
                                ;
                            seg.push_back(pair<double, double
                                >(0, tmp.second));
                        }
                        else seg.push_back(tmp);
                    }
                sort(seg.begin(), seg.end());
                double rig = 0;
                for (vector<pair<double, double> >::iterator
                       iter = seg.begin(); iter != seg.end();
                       iter++) {
                    if (sign(rig - iter->first) >= 0)
                        rig = max(rig, iter->second);
                    else {
                        getarea(i, rig, iter->first);
                        rig = iter->second;
                    }
                }
                if (!sign(rig)) arc += r[i] * r[i] * PI;
                else getarea(i, rig, 2 * PI);
            }
        }
        return pol / 2.0 + arc;
    }
} ccu;
```

CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>

"Point.h", "lineDistance.h", "LineProjectionReflection.h" debf86, 8 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  double h2 = r*r - a.cross(b,c)*a.cross(b,c)/(b-a).dist2();
  if (h2 < 0) return {};
  P p = lineProj(a, b, c), h = (b-a).unit() * sqrt(h2);
  if (h2 == 0) return {p};
  return {p - h, p + h};
}
```

CircleCircleArea.h

Description: Calculates the area of the intersection of 2 circles 8bf2b6, 12 lines

```
template<class P>
double circleCircleArea(P c, double cr, P d, double dr) {
  if (cr < dr) swap(c, d), swap(cr, dr);
  auto A = [&](double r, double h) {
    return r*r*acos(h/r)-h*sqrt(r*r-h*h);
  };
  auto l = (c - d).dist(), a = (l*l + cr*cr - dr*dr)/(2*l);
  if (l - cr - dr >= 0) return 0; // far away
  if (l - cr + dr <= 0) return M_PI*dr*dr;
  if (l - cr >= 0) return A(cr, a) + A(dr, l-a);
  else return A(cr, a) + M_PI*dr*dr - A(dr, a-l);
}
```

CirclePolygonIntersection.h

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

"Point.h" cf9deb, 18 lines

```
#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;
  for (int i = 0; i < ps.size(); ++i)
    sum += tri(ps[i] - c, ps[(i + 1) % ps.size()] - c);
  return sum;
}
```

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" f9442d, 12 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = p.size();
  for(int i = 0; i < n; ++i) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict; // change to
      // -1 if u need to detect points in the boundary
    //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 the area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" 3794ee, 17 lines

```
template<class T>
T polygonArea(vector<Point<T>> &v) {
  T a = v.back().cross(v[0]);
  for(int i = 0; i < v.size()-1; ++i)
    a += v[i].cross(v[i+1]);
  return abs(a)/2.0;
}

Point<T> polygonCentroid(vector<Point<T>> &v) { // not tested
  Point<T> cent(0,0); T area = 0;
  for(int i = 0; i < v.size(); ++i) {
    int j = (i+1) % (v.size()); T a = cross(v[i], v[j]);
    cent += a * (v[i] + v[j]);
    area += a;
  }
  return cent/area/(T)3;
}
```

PolygonCenter.h

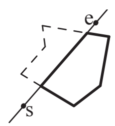
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 26a00f, 8 lines

```
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = v.size() - 1; i < v.size(); 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.



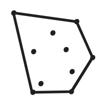
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h" 7df36f, 11 lines

```
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  for(int i = 0; i < poly.size(); ++i) {
    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 indices 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" 3612d7, 12 lines

```
vector<P> convexHull(vector<P> pts) {
  if (pts.size() <= 1) return pts;
  sort(pts.begin(), pts.end());
  vector<P> h(pts.size()+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(pts.begin(), pts.end
      ()))
    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/colinear points).

0e0c1f, 11 lines

```
array<P, 2> hullDiameter(vector<P> S) {
  int n = S.size(), j = n < 2 ? 0 : 1;
  pair<lint, array<P, 2>> res({0, {S[0], S[0]}});
  for(int i = 0; i < j; ++i)
    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 colinear 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" 7b8514, 12 lines

```
bool inHull(const vector<P> &l, P p, bool strict = true) {
  int a = 1, b = l.size() - 1, r = !strict;
  if (l.size() < 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;
}
```

minkowski-sum.h

Description: Minkowski sum of two convex polygons given in ccw order.

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

71a25a, 20 lines

```
vector<P> minkowski_sum(vector<P> A, vector<P> B) {
  if (int(A.size()) > int(B.size())) swap(A, B);
  if (A.empty()) return {};
  if (int(A.size()) == 1) {
    for (auto& b : B) b = b + A.front();
    return B;
  }
  rotate(A.begin(), min_element(A.begin(), A.end()), A.end());
  rotate(B.begin(), min_element(B.begin(), B.end()), B.end());
  A.push_back(A[0]); A.push_back(A[1]);
  B.push_back(B[0]); B.push_back(B[1]);
  const int N = int(A.size()), M = int(B.size());
  vector<P> ans; ans.reserve(N+M);
  for (int i = 0, j = 0; i+2 < N || j+2 < M;) {
    ans.push_back(A[i] + B[j]);
    auto sgn = (A[i+1] - A[i]).cross(B[j+1] - B[j]);
    i += (sgn >= 0); j += (sgn <= 0);
  }
  return ans;
}
```

PolyUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. Guaranteed to be precise for integer coordinates up to 3e7. If epsilons are needed, add them in sideOf as well as the definition of sgn.

Time: $\mathcal{O}(N^2)$, where N is the total number of points

"Point.h", "sideOf.h" a45bd4, 33 lines

```
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  for(int i = 0; i < poly.size(); ++i)
    for(int v = 0; v < poly[i].size(); ++v) {
      P A = poly[i][v], B = poly[i][(v + 1) % poly[i].size()];
      vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
      for(int j = 0; j < poly.size(); ++j) if (i != j) {
        for(int u = 0; u < poly[j]; ++u) {
          P C = poly[j][u], D = poly[j][(u + 1) % poly[j].size
              ()];
          int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
          if (sc != sd) {
            double sa = C.cross(D, A), sb = C.cross(D, B);
            if (min(sc, sd) < 0)
              segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
          } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))
              >0){
            segs.emplace_back(rat(C - A, B - A), 1);
            segs.emplace_back(rat(D - A, B - A), -1);
          }
        }
      }
      sort(segs.begin(), segs.end());
      for(auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
      double sum = 0;
      int cnt = segs[0].second;
      for(int j = 1; j < segs.size(); ++j) {
        if (!cnt) sum += segs[j].first - segs[j - 1].first;
        cnt += segs[j].second;
      }
      ret += A.cross(B) * sum;
    }
  return ret / 2;
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear 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: $\mathcal{O}(N + Q \log n)$

"Point.h" 65ebb6, 39 lines

```
typedef array<P, 2> Line;
#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
int extrVertex(vector<P>& poly, P dir) {
  int n = poly.size(), left = 0, right = n;
  if (extr(0)) return 0;
  while (left + 1 < right) {
    int m = (left + right) / 2;
    if (extr(m)) return m;
    int ls = cmp(left + 1, left), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(left, m)) ? right : left
        ) = m;
  }
  return left;
}

#define cmpL(i) sgn(line[0].cross(poly[i], line[1]))
array<int, 2> lineHull(Line line, vector<P>& poly) {
  int endA = extrVertex(poly, (line[0] - line[1]).perp());
  int endB = extrVertex(poly, (line[1] - line[0]).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  for(int i = 0; i < 2; ++i) {
    int left = endB, right = endA, n = poly.size();
    while ((left + 1) % n != right) {
      int m = ((left + right + (left < right ? 0 : n)) / 2) % n
          ;
      (cmpL(m) == cmpL(endB) ? left : right) = m;
    }
    res[i] = (left + !cmpL(right)) % 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) % poly.size()) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

HalfPlane.h

Description: Halfplane intersection area

"Point.h", "lineIntersection.h" c0a94b, 70 lines

```
#define eps 1e-8
typedef Point<double> P;

struct Line {
  P P1, P2;
  // Right hand side of the ray P1 -> P2
  explicit Line(P a = P(), P b = P()) : P1(a), P2(b) {};
  P intpo(Line y) {
    pair<int, P> r = lineInter(P1, P2, y.P1, y.P2);
    assert(r.first == 1);
    return r.second;
  }
  P dir() { return P2 - P1; }
  bool contains(P x) {
    return (P2 - P1).cross(x - P1) < eps;
  }
  bool out(P x) { return !contains(x); }
};

template<class T>
bool mycmp(Point<T> a, Point<T> b) {
  // return atan2(a.y, a.x) < atan2(b.y, b.x);
  if (a.x * b.x < 0) return a.x < 0;
  if (abs(a.x) < eps) {
    if (abs(b.x) < eps) return a.y > 0 && b.y < 0;
    if (b.x < 0) return a.y > 0;
    if (b.x > 0) return true;
  }
  if (abs(b.x) < eps) {
    if (a.x < 0) return b.y < 0;
```

```
    if (a.x > 0) return false;
  }
  return a.cross(b) > 0;
}

bool cmp(Line a, Line b) { return mycmp(a.dir(), b.dir()); }

double Intersection_Area(vector <Line> b) {
  sort(b.begin(), b.end(), cmp);
  int n = b.size();
  int q = 1, h = 0, i;
  vector<Line> c(b.size() + 10);
  for (i = 0; i < n; i++) {
    while (q < h && b[i].out(c[h].intpo(c[h - 1]))) h--;
    while (q < h && b[i].out(c[q].intpo(c[q + 1]))) q++;
    c[++h] = b[i];
    if (q < h && abs(c[h].dir().cross(c[h - 1].dir())) < eps) {
      if (c[h].dir().dot(c[h - 1].dir()) > 0) {
        h--;
        if (b[i].out(c[h].P1))  c[h] = b[i];
      }else {
        // The area is either 0 or infinite.
        // If you have a bounding box, then the area is
            definitely 0.
        return 0;
      }
    }
  }
  while (q < h-1 && c[q].out(c[h].intpo(c[h - 1]))) h--;
  while (q < h-1 && c[h].out(c[q].intpo(c[q + 1]))) q++;
  // Intersection is empty. This is sometimes different from
      the case when
  // the intersection area is 0.
  if (h - q <= 1) return 0;
  c[h + 1] = c[q];
  vector<P> s;
  for (i = q; i <= h; i++) s.push_back(c[i].intpo(c[i + 1]));
  s.push_back(s[0]);
  double ans = 0;
  for (i = 0; i < (int) s.size()-1; i++) ans += s[i].cross(s[i
      + 1]);
  return ans / 2;
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" 32b14f, 16 lines

```
pair<P, P> closest(vector<P> v) {
  assert(v.size() > 1);
  set<P> S;
  sort(v.begin(), v.end(), [](P a, P b) { return a.y < b.y; });
  pair<int64_t, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for(P &p : v) {
    P d{1 + (int64_t)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" 915562, 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 the box is wider than high (not best
          heuristic...)
      sort(vp.begin(),vp.end(), 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 = vp.size()/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({vp.begin(), vp.
      end()})) {}

  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);
  }
};
```

DelaunayTriangulation.h

Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are colinear or any four are on the same circle, behavior is undefined.
Time: $\mathcal{O}(n^2)$

"Point.h", "3dHull.h" f6175a, 10 lines

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (ps.size() == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0)
      ;
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  for(auto &p : ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (ps.size() > 3) for(auto &t: hull3d(p3)) if ((p3[t.b]-p3[t
      .a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

FastDelaunay.h

Description: Fast Delaunay triangulation. 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" a1f392, 89 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 {
  bool mark; Q o, rot; P p;
  P F() { return r()->p; }
  Q r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return rot->r()->o->rot; }
};
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 q0 = new Quad{0,0,0,orig}, q1 = new Quad{0,0,0,arb},
    q2 = new Quad{0,0,0,dest}, q3 = new Quad{0,0,0,arb};
  q0->o = q0; q2->o = q2; // 0-0, 2-2
  q1->o = q3; q3->o = q1; // 1-3, 3-1
  q0->rot = q1; q1->rot = q2;
  q2->rot = q3; q3->rot = q0;
  return q0;
}
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) + 1) / 2;
  tie(ra, A) = rec({s.begin(), s.begin() + half});
  tie(B, rb) = rec({s.begin() + half, s.end()});
  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 = 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(pts.begin(), pts.end());  assert(unique(pts.begin(), pts
      .end()) == pts.end());
  if (pts.size() < 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;
}
```

RectangleUnionArea.h

Description: Sweep line algorithm that calculates area of union of rectangles in the form $[x1, x2) \times [y1, y2)$

Usage: `vector<pair<int,int>, pair<int,int>> rectangles;`
`rectangles.push_back({{x1, x2}, {y1, y2}});`
`lint result = area(rectangles);`

529ff1, 51 lines

```
struct seg_node{
  int val, cnt, lz;
  seg_node(int n = INT_MAX, int c = 0): val(n), cnt(c), lz(0)
      {}
  void push(seg_node& l, seg_node& r){
    if(lz){
      l.add(lz);
      r.add(lz);
      lz = 0;
    }
  }
  void merge(const seg_node& l, const seg_node& r){
    if(l.val < r.val) val = l.val, cnt = l.cnt;
    else if(l.val > r.val) val = r.val, cnt = r.cnt;
    else val = l.val, cnt = l.cnt + r.cnt;
  }
  void add(int n){
    val += n;
    lz += n;
  }
  int get_sum(){ return (val ? 0 : cnt); }
};
// x1 y1 x2 y2
lint solve(const vector<array<int, 4>>&v){
  vector<int>ys;
  for(auto& [a, b, c, d] : v){
    ys.push_back(b);
    ys.push_back(d);
  }
  sort(ys.begin(), ys.end());
  ys.erase(unique(ys.begin(), ys.end()), ys.end());
  vector<array<int, 4>>e;
  for(auto [a, b, c, d] : v){
    b = int(lower_bound(ys.begin(), ys.end(), b) - ys.begin());
    d = int(lower_bound(ys.begin(), ys.end(), d) - ys.begin());
    e.push_back({a, b, d, 1});
    e.push_back({c, b, d, -1});
  }
  sort(e.begin(), e.end());
  int m = (int)ys.size();
  segtree_range<seg_node>seg(m-1);
  for(int i=0;i<m-1;i++) seg.at(i) = seg_node(0, ys[i+1] - ys[i
      ]);
  seg.build();
  int last = INT_MIN, total = ys[m-1] - ys[0];
  lint ans = 0;
  for(auto [x, y1, y2, c] : e){
    ans += (lint)(total - seg.query(0, m-1).get_sum()) * (x -
        last);
    last = x;
    seg.update(y1, y2, &seg_node::add, c);
  }
  return ans;
}
```

8.5 3D

PolyhedronVolume.h

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

832599, 6 lines

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

Point3D.h

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

8058ae, 32 lines

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*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" 3ed613, 48 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(A.size() >= 4);
  vector<vector<PR>> E(A.size(), vector<PR>(A.size(), {-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);
  };
  for(int i=0;i<4;i++) for(int j=i+1;j<4;j++) for(k=j+1;k<4;k
      ++)
    mf(i, j, k, 6 - i - j - k);
  for(int i=4; i<A.size();++i) {
    for(int j=0;j<FS.size();++j) {
      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 = FS.size();
    for(int j=0;j<nw;j++) {
```

```
    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(auto &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. 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.

611f07, 8 lines

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

Strings (9)

kmp.h

Description: failure[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> -1,0,0,1,0,1,2,3). Can be used to find all occurrences of a pattern in a text.
Time: $\mathcal{O}(n)$

4eb73b, 9 lines

```
vector<int> prefix_function(const string& S) {
  vector<int> fail = {-1}; fail.reserve(S.size());
  for (int i = 0; i < int(S.size()); ++i) {
    int j = fail.back();
    while (j != -1 && S[i] != S[j]) j = fail[j];
    fail.push_back(j+1);
  }
  return fail;
}
```

duval.h

Description: A string is called simple (or a Lyndon word), if it is strictly smaller than any of its own nontrivial suffixes.
Time: $\mathcal{O}(N)$

d9b2cb, 27 lines

```
template <typename T>
pair<int, vector<string>> duval(int n, const T &s) {
  assert(n >= 1);
  // s += s  //uncomment if you need to know the min cyclic
      string
  vector<string> factors; // strings here are simple and in non
      -inc order
  int i = 0, ans = 0;
  while (i < n) { // until n/2 to find min cyclic string
    ans = i;
    int j = i + 1, k = i;
    while (j < n + n && !(s[j % n] < s[k % n])) {
      if (s[k % n] < s[j % n]) k = i;
      else k++;
      j++;
    }
    while (i <= k) {
      factors.push_back(s.substr(i, j-k));
      i += j - k;
    }
  }
  return {ans, factors};
  // returns 0-indexed position of the least cyclic shift
  // min cyclic string will be s.substr(ans, n/2)
}
template <typename T>
pair<int, vector<string>> duval(const T &s) {
  return duval((int) s.size(), s);
}
```

z-algorithm.h

Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

7c8c64, 16 lines

```
vector<int> Z(const string& S) {
  vector<int> z(S.size());
  int l = -1, r = -1;
  for(int i = 1; i < int(S.size()); ++i) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < int(S.size()) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r) l = i, r = i + z[i];
  }
  return z;
}
vector<int> get_prefix(string a, string b) {
  string str = a + '@' + b;
  vector<int> k = z(str);
  return vector<int>(k.begin() + int(a.size())+1, k.end());
}
```

manacher.h

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

87e1f0, 13 lines

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

min-rotation.h

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

2a08fd, 8 lines

```
int min_rotation(string s) {
  int a=0, N=s.size(); s += s;
  for(int b = 0; b < N; ++b) for(int i =0; i < N; ++i) {
    if (a+i == b || s[a+i] < s[b+i]) {b += max(0, i-1); break;}
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
}
```

xor-trie.h

Description: Query get the maximum possible xor between an integer X and every possible subarray. Just insert zero and for each prefix xor, insert it in the trie and query for max xor. The answer is the maximum possible value for each prefix query.

714ffb, 28 lines

```
template<int K = 31> struct trie_t {
  vector<array<int, 2>> trie;
  trie_t() : trie(1, {-1, -1}) {}
  void add(int val) {
    int cur = 0;
    for (int a = K; a >= 0; --a) {
      int b = (val >> a) & 1;
      if (trie[cur][b] == -1) {
        trie[cur][b] = size(trie);
        trie.push_back({-1, -1});
      }
      cur = trie[cur][b];
    }
  }
  int max_xor(int val) {
    int cur = 0, mask = 0;
    for (int a = K; a >= 0; --a) {
      int b = (val >> a) & 1;
      if (trie[cur][!b] == -1) {
        cur = trie[cur][b];
      } else {
        mask |= (1 << a);
        cur = trie[cur][!b];
      }
    }
    return mask;
  }
};
```

hashing.h

d0abe8, 38 lines

```
const int maxn = 400001;
const int mod = 1004669333, base = 33, inv_base = 121778101;
vector<int> base_pow(maxn + 1), inv_base_pow(maxn + 1);
void prep() { // 5c2398
  base_pow[0] = 1;
  for (int i = 1; i <= maxn; ++i)
    base_pow[i] = (lint)base_pow[i - 1] * base % mod;
  inv_base_pow[0] = 1;
  for (int i = 1; i <= maxn; ++i)
    inv_base_pow[i] = (lint)inv_base_pow[i - 1] * inv_base %
        mod;
}
struct hashes_t { // f1dd26
  string s;
  int n;
  vector<int> acc_hash, acc_inv_hash;
  hashes_t(const string &_s): s(_s), n(s.size()), acc_hash(n +
      1, 0) ,acc_inv_hash(n + 1, 0) { // 127dd9
    for (int i = 0; i < n; ++i) {
      acc_hash[i + 1] =
        (acc_hash[i] + (lint)base_pow[i] * (s[i] - 'a' + 1)) %
            mod;
      acc_inv_hash[i + 1] =
        (acc_inv_hash[i] + (lint)inv_base_pow[i] * (s[i] - 'a'
            + 1)) % mod;
    }
  }
  int get_hash(int a, int b) { // 04a73b
    assert(a <= b);
    int hash = acc_hash[b + 1] - acc_hash[a];
```

```
    if (hash < 0) hash += mod;
    hash = (lint)hash * inv_base_pow[a] % mod;
    return hash;
  }
  int get_inv_hash(int a, int b) { // d3dfd9
    assert(a <= b);
    int hash = acc_inv_hash[b + 1] - acc_inv_hash[a];
    if (hash < 0) hash += mod;
    hash = (lint)hash * base_pow[b] % mod;
    return hash;
  }
};
```

modnum-double-hashing.h

Description: Simple, short and efficient hashing using pairs to reduce load factor.

"../number-theory/modular-arithmetic.h", "../number-theory/pairnum-template.h" 3bcdb0, 40 lines

```
using num = modnum<int(1e9)+7>;
using hsh = pairnum<num, num>;
const hsh BASE(163, 311);
// uniform_int_distribution<int> MULT_DIST(0.1*MOD,0.9*MOD);
// constexpr hsh BASE(MULT_DIST(rng), MULT_DIST(rng));
struct hash_t { // c9d6c0
  int n;
  string str;
  vector<hsh> hash, basePow;
  hash_t(const string& s) : n(s.size()), str(s), hash(n+1),
       basePow(n+1) { // dd1f3f
    basePow[0] = 1;
    for (int i = 1; i <= n; ++i) basePow[i] = basePow[i-1] *
        BASE;
    for (int i = 0; i < n; ++i)
      hash[i+1] = hash[i] * BASE + hsh(s[i]);
  }
  hsh get_hash(int left, int right) { // 302ee0
    assert(left <= right);
    return hash[right] - hash[left] * basePow[right - left];
  }
  int lcp(hash_t &other) { // 5eb9e2
    int left = 0, right = min(str.size(), other.str.size());
    while (left < right) {
      int mid = (left + right + 1)/2;
      if (hash[mid] == other.hash[mid]) left = mid;
      else right = mid-1;
    }
    return left;
  }
};
vector<int> rabinkarp(string t, string p) { // c11cfc
  vector<int> matches;
  hsh h(0, 0);
  for (int i = 0; i < p.size(); ++i)
    h = BASE * h + hsh(p[i]);
  hash_t result(t);
  for (int i = 0; i + p.size() <= t.size(); ++i)
    if (result.get_hash(i, i + p.size()) == h)
      matches.push_back(i);
  return matches;
}
```

aho-corasick.h

5159f5, 41 lines

```
const int sigma = 26;
array<int, sigma> init;
for (int i = 0; i < sigma; i++) init[i] = -1;
vector<array<int, sigma>> trie(1, init);
vector<int> out(1, -1), parent(n, -1), ids(n);
for (int i = 0; i < n; i++) {
  int cur = 0;
  for (char ch : s[i]) {
    int c = ch - 'a';
    if (trie[cur][c] == -1) {
      trie[cur][c] = (int)trie.size();
      trie.push_back(init);
      out.push_back(-1);
    }
    cur = trie[cur][c];
  }
  if (out[cur] == -1) out[cur] = i;
  ids[i] = out[cur];
}
vector<int> bfs; bfs.reserve(trie.size());
vector<int> f(trie.size());
for (int c = 0; c < sigma; c++) {
  if (trie[0][c] == -1) trie[0][c] = 0;
  else bfs.push_back(trie[0][c]);
}
for (int z = 0; z < (int)bfs.size() ; z++) {
  int cur = bfs[z];
  for (int c = 0; c < sigma; c++) {
    if (trie[cur][c] == -1) {
      trie[cur][c] = trie[f[cur]][c];
    }
    else {
      int nxt = trie[cur][c];
      int fail = trie[f[cur]][c];
      if (out[nxt] == -1) out[nxt] = out[fail];
      else parent[out[nxt]] = out[fail];
      f[nxt] = fail;
      bfs.push_back(nxt);
    }
  }
}
```

suffix-array.h

Description: Builds suffix array for a string, first element is the size of the string. The lcp function calculates longest common prefixes for neighbouring strings in suffix array. The returned vector is of size $n+1$.

Time: $\mathcal{O}(N \log N)$ where N is the length of the string for creation of the SA. $\mathcal{O}(N)$ for longest common prefixes.

<../data-structures/rmq.h> 5393da, 47 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());
struct suffix_array_t {
  int N, H; vector<int> sa, invsa, lcp;
  rmq_t<pair<int, int>> rmq;
  bool cmp(int a, int b) { return invsa[a + H] < invsa[b + H];
       }
  void ternary_sort(int a, int b) {
    if (a == b) return;
    int md = sa[a+rng() % (b-a)], lo = a, hi = b;
    for (int i = a; i < b; ++i) if (cmp(sa[i], md)) swap(sa[i],
         sa[lo++]);
    for (int i = b-1; i >= lo; --i) if (cmp(md, sa[i])) swap(sa
        [i], sa[--hi]);
    ternary_sort(a, lo);
    for (int i = lo; i < hi; ++i) invsa[sa[i]] = hi-1;
    if (hi-lo == 1) sa[lo] = -1;
    ternary_sort(hi, b);
  }
  suffix_array_t() {}
  template<typename I>
  suffix_array_t(I begin, I end): N(int(end - begin)+1), sa(N)
       {
    vector<int> v(begin, end); v.push_back(INT_MIN);
    invsa = v; iota(sa.begin(), sa.end(), 0);
    H = 0; ternary_sort(0, N);
    for (H = 1; H <= N; H *= 2)
      for (int j = 0, i = j; i != N; i = j)
        if (sa[i] < 0) {
          while (j < N && sa[j] < 0) j += -sa[j];
          sa[i] = -(j - i);
        }
        else { j = invsa[sa[i]] + 1; ternary_sort(i, j); }
    for (int i = 0; i < N; ++i) sa[invsa[i]] = i;
    lcp.resize(N-1); int K = 0;
    for (int i = 0; i < N-1; ++i) {
      if (invsa[i] > 0) while (v[i + K] == v[sa[invsa[i] - 1] +
           K]) ++K;
      lcp[invsa[i]-1] = K; K = max(K - 1, 0);
    }
    vector<pair<int, int>> lcp_index(N-1);
    for (int i = 0; i < N-1; ++i) lcp_index[i] = {lcp[i], 1 + i
        };
    rmq = rmq_t<pair<int, int>>(std::move(lcp_index));
  }
  pair<int, int> rmq_query(int a, int b) const { return rmq.
       query(a, b); }
  pair<int, int> get_split(int a, int b) const { return rmq.
       query(a, b-1); }
  int get_lcp(int a, int b) const {
    if (a == b) return N - a;
    a = invsa[a], b = invsa[b];
    if (a > b) swap(a, b);
    return rmq_query(a + 1, b + 1).first;
  }
};
```

suffix-tree.h

Description: Builds suffix-tree informations based by emulating it over the suffix-array and lcp, root of the tree represents the special character (size of string for suffix-array), can therefore be ignored when calculating stuff.

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

<../data-structures/rmq.h>, "suffix-array.h" 31aacf, 46 lines

```
struct suffix_tree_t {
  int N, V;
  vector<vector<int>> ch;
  vector<array<int, 2>> sa_range;
  vector<int> leaves, par, depth;
  vector<int> suff_link;
  vector<bool> is_unique_link, has_unique_child;
  suffix_array_t us;
  suffix_tree_t() {}
  suffix_tree_t(string S) : N(int(S.size())), V(0), ch(2*N+1),
  sa_range(2*N+1), leaves(N+1), par(2*N+1), depth(2*N+1),
  us(S.begin(), S.end()) { dfs(0, N+1, -1); }
  void dfs(int a, int b, int prv) {
    int cur = V++;
    par[cur] = prv;
    if (prv != -1) ch[prv].push_back(cur);
    sa_range[cur] = {a, b};
    if (b - a == 1) {
      leaves[us.sa[a]] = cur;
      depth[cur] = N - us.sa[a];
    } else {
      int d = us.get_split(a, b).first;
      depth[cur] = d;
      int mi = a;
      while (b - mi >= 2) {
        auto [nd, nmi] = us.get_split(mi, b);
        if (nd != d) break;
        dfs(mi, nmi, cur);
        mi = nmi;
      }
      dfs(mi, b, cur);
    }
```

```
  }
  void build_links() {
    suff_link.resize(V, -1), is_unique_link.resize(V),
         has_unique_child.resize(V);
    for (int i = 0; i < N; ++i) {
      for (int cur = leaves[i], link = leaves[i+1];; cur = par[
           cur]) {
        if (cur == 0 || suff_link[cur] != -1) break;
        suff_link[cur] = link;
        is_unique_link[cur] = (sa_range[cur][1] - sa_range[cur
             ][0]) == (sa_range[link][1] - sa_range[link][0]);
        if (is_unique_link[cur]) has_unique_child[link] = true;
        while (~link && depth[link] + 1 > depth[par[cur]]) link
              = par[link];
      }
    }
  }
};
```

suffix-automaton.h

Description: Suffix automaton

c4406e, 38 lines

```
template<int offset = 'a'> struct array_state {
  array<int, 26> as;
  array_state() { fill(begin(as), end(as), ~0); }
  int& operator[](char c) { return as[c - offset]; }
  int count(char c) { return (~as[c - offset] ? 1 : 0); }
};

template<typename Char, typename state = map<Char, int>> struct
     suffix_automaton {
  struct node_t {
    int len, link; int64_t cnt;
    state next;
  };
  int N, cur;
  vector<node_t> nodes;
  suffix_automaton() : N(1), cur(0), nodes{node_t{0, -1, 0,
       {}}} {}
  node_t& operator[](int v) { return nodes[v]; };
  void append(Char c) {
    int v = cur; cur = N++;
    nodes.push_back(node_t{nodes[v].len + 1, 0, 1, {}});
    for (; ~v && !nodes[v].next.count(c); v = nodes[v].link) {
      nodes[v].next[c] = cur;
    }
    if (~v) {
      const int u = nodes[v].next[c];
      if (nodes[v].len + 1 == nodes[u].len) {
        nodes[cur].link = u;
      } else {
        const int clone = N++;
        nodes.push_back(nodes[u]);
        nodes[clone].len = nodes[v].len + 1;
        nodes[u].link = nodes[cur].link = clone;
        for (; ~v && nodes[v].next[c] == u; v = nodes[v].link)
             {
          nodes[v].next[c] = clone;
        }
      }
    }
  }
};
```

9.1 Suffix Automaton

9.1.1 Number of different substrings

Is the number of paths in the automaton starting at the root.

$$d(v) = 1 + \sum_{v \to w} d(w)$$

9.1.2 Total lenght of different substrings

Is the sum of children answers and paths starting at each children.

$$ans(v) = \sum_{v \to w} d(w) + ans(w)$$

9.1.3 Lexicographically K-th substring

Is the K-th lexicographically path, so you can search using the number of paths from each state

9.1.4 Smallest cyclic shift

Construct for string $S + S$. Greedily search the minimal character.

9.1.5 Number of occurrences

For each state not created by cloning, initialize $cnt(v) = 1$. Then, just do a dfs to calculate $cnt(v)$

$$cnt(link(v)) += cnt(v)$$

9.1.6 First occurence position

When we create a new state cur do $first(pos) = len(cur) - 1$. When we clone q as $clone$ do $first(clone) = first(q)$. Answer is $first(v) - size(P) + 1$, where v is the state of string P

9.1.7 All occurence positions

From $first(v)$ do a dfs using suffix link, from $link(u)$ go to u.

Various (10)

10.1 Intervals

interval-container.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: $\mathcal{O}(\log N)$

f47dfb, 22 lines

```
set<pair<int,int>>::iterator addInterval(set<pair<int,int>> &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<pair<int,int>> &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);
}
```

interval-cover.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: $\mathcal{O}(N \log N)$

133eb4, 19 lines

```
template<class T>
vector<int> cover(pair<T, T> G, vector<pair<T, T>> I) {
  vector<int> S(I.size()), R;
  iota(S.begin(), S.end(), 0);
  sort(S.begin(), S.end(), [&](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 = {cur, -1};
    while (at < I.size() && I[S[at]].first <= cur) {
      mx = max(mx, {I[S[at]].second, S[at]});
      at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

constant-intervals.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: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

753a4c, 19 lines

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

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

10.2 Misc. algorithms

floor.h

e7f4c9, 7 lines

```
template<typename T> T mfloor(T a, T b) {
  return a / b - (((a ^ b) < 0 && a % b != 0) ? 1 : 0);
}

template<typename T> T mceil(T a, T b) {
  return a / b + (((a ^ b) > 0 && a % b != 0) ? 1 : 0);
}
```

basis-manager.h

Description: A list of basis values sorted in decreasing order, where each value has a unique highest bit.

d5bcd3, 41 lines

```
const int BITS = 60;

template<typename T> struct xor_basis {
  int N = 0;
  array<T, BITS> basis;

  T min_value(T start) const {
    if (N == BITS) return 0;
    for (int i = 0; i < N; ++i)
      start = min(start, start ^ basis[i]);
    return start;
  }
  T max_value(T start = 0) const {
    if (N == BITS) return ((T) 1 << BITS) - 1;
    for (int i = 0; i < N; ++i)
      start = max(start, start ^ basis[i]);
    return start;
  }
  bool add(T x) {
    x = min_value(x);
    if (x == 0) return false;
    basis[N++] = x;
    // Insertion sort.
    for (int k = N - 1; k > 0 && basis[k] > basis[k - 1]; k--)
      swap(basis[k], basis[k - 1]);
    return true;
  }
  void merge(const xor_basis<T>& other) {
    for (int i = 0; i < other.n && N < BITS; i++)
      add(other.basis[i]);
  }
  void merge(const xor_basis<T>& a, const xor_basis<T>& b) {
    if (a.N > b.N) {
      *this = a;
      merge(b);
    } else {
      *this = b;
      merge(a);
    }
  }
};
```

ternary-search.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). If you are dealing with real numbers, you'll need to pick $m_1 = (2a+b)/3.0$ and $m_2 = (a+2b)/3.0$. Consider setting a constant number of iterations for the search, usually $[200, 300]$ iterations are sufficient for problems with error limit as 10^{-6}.

Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`

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

35ef73, 12 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;
  }
  for(int i = a+1; i <= b; ++i)
    if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

merge-sort.h

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

fac159, 25 lines

```
vector<int> merge(vector<int> &values, int l, int r) {
  static vector<int> result(values.size());
  int i = l, j = l + (r - l)/2;
  int mid = j, k = i, inversions = 0;
  while (i < mid && j < r) {
    if (values[i] < values[j]) result[k++] = values[i++];
    else {
      result[k++] = values[j++];
      inversions += (mid - i);
    }
  }
  while (i < mid) result[k++] = values[i++];
  while (j < r) result[k++] = values[j++];
  for (k = l; k < r; ++k) values[k] = result[k];
  return result;
}

vector<int> msort(vector<int> &values, int l, int r) {
  if (r - l > 1) {
    int mid = l + (r - l)/2;
    msort(values, l, mid); msort(values, mid, r);
    return merge(values, l, r);
  }
  return {};
}
```

radix-sort.h

Description: Radix Sort Algorithm.

Time: $\mathcal{O}(NK)$ where K is the number of bits in the largest element of the array to be sorted.

889884, 54 lines

```
struct identity {
  template<typename T>
    T operator()(const T &x) const {
      return x;
    }
};
template<typename T, typename T_extract_key = identity>
void radix_sort(vector<T> &data, int bits_per_pass = 10, const
     T_extract_key &extract_key = identity()) {
  if (data.size() < 256) {
    sort(data.begin(), data.end(), [&](const T &a, const T &b)
         {
      return extract_key(a) < extract_key(b);
    });
    return;
  }
  using T_key = decltype(extract_key(data.front()));
  T_key minimum = numeric_limits<T_key>::max();
  for (T &x : data) minimum = min(minimum, extract_key(x));
  int max_bits = 0;
  for (T &x : data) {
    T_key key = extract_key(x);
    max_bits = max(max_bits, key == minimum ? 0 : 64 -
         __builtin_clzll(key - minimum));
  }
  int passes = max((max_bits + bits_per_pass / 2) /
       bits_per_pass, 1);
  if (32 - __builtin_clz(data.size()) <= 1.5 * passes) {
    sort(data.begin(), data.end(), [&](const T &a, const T &b)
         {
      return extract_key(a) < extract_key(b);
    });
    return;
  }
  vector<T> buffer(data.size());
  vector<int> counts;
  int bits_so_far = 0;
  for (int p = 0; p < passes; p++) {
    int bits = (max_bits + p) / passes;
    counts.assign(1 << bits, 0);
    for (T &x : data) {
      T_key key = extract_key(x) - minimum;
      counts[(key >> bits_so_far) & ((1 << bits) - 1)]++;
    }
    int count_sum = 0;
    for (int &count : counts) {
      int current = count;
      count = count_sum;
      count_sum += current;
    }
    for (T &x : data) {
      T_key key = extract_key(x) - minimum;
      int key_section = (key >> bits_so_far) & ((1 << bits) -
           1);
      buffer[counts[key_section]++] = x;
    }
    swap(data, buffer);
    bits_so_far += bits;
  }
}
```

postfix-notation-solver.h

Description: Solves postfix (Reverse Polish) notation equation to solve prefix notation equation reverse e and change (i) and (ii)

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

bf1f57, 31 lines

```
template<typename T, typename P, typename F>
T postfixSolver(const vector<P> &e, const set<P> &ops, F ptot){
  vector<T> stk;
  for(auto cur: e)
    if(ops.count(cur)){
      T c;
      //operations:
      if(cur == "-"){
        T b = stk.back(); // (i) T a = stk.back();
        stk.pop_back();
        T a = stk.back(); //(ii) T b = stk.back();
        stk.pop_back();
        c = a - b;
      }
      else if(cur == "NOT"){
```

```
      T a = stk.back();
      stk.pop_back();
      c = ~a;
    }
    stk.push_back(c);
  } else
    stk.push_back(ptot(cur));
  return stk.back();
}
//example postfix:
vector<string> e = {"13", "14", "-", "NOT"};
int ans = postfixSolver<int>( e, {"-", "NOT"}, [](const string
    &s){ return stoi(s); } );
//example prefix:
vector<string> e = {"NOT", "-", "13", "14"};
reverse(e.begin(), e.end()); // DON'T FORGET!!!!!
int ans = postfixSolver<int>( e, {"-", "NOT"}, [](const string
    &s){ return stoi(s); } );
```

count-triangles.h

Description: Counts x, y >= 0 such that Ax + By <= C.

8d67b3, 8 lines

```
lint count_triangle(lint A, lint B, lint C) {
  if (C < 0) return 0;
  if (A > B) swap(A, B);
  lint p = C / B;
  lint k = B / A;
  lint d = (C - p * B) / A;
  return count_triangle(B - k * A, A, C - A * (k * p + d + 1))
      + (p + 1) * (d + 1) + k * p * (p + 1) / 2;
}
```

karatsuba.h

Description: Faster-than-naive convolution of two sequences: $c[x] = \sum a[i]b[x-i]$. Uses the identity $(aX + b)(cX + d) = acX^2 + bd + ((a + c)(b + d) - ac - bd)X$. Doesn't handle sequences of very different length welint. See also FFT, under the Numerical chapter.

Time: $\mathcal{O}(N^{1.6})$

37b858, 30 lines

```
int size(int s) { return s > 1 ? 32-__builtin_clz(s-1) : 0; }

void karatsuba(lint *a, lint *b, lint *c, lint *t, int n) {
  int ca = 0, cb = 0;
  for(int i = 0; i < n; ++i) ca += !!a[i], cb += !!b[i];
  if (min(ca, cb) <= 1500/n) { // few numbers to multiply
    if (ca > cb) swap(a, b);
    for(int i = 0; i < n; ++i)
      if (a[i]) for(int j = 0; j < n; ++j) c[i+j] += a[i]*b[j];
  }
  else {
    int h = n >> 1;
    karatsuba(a, b, c, t, h); // a0*b0
    karatsuba(a+h, b+h, c+n, t, h); // a1*b1
    for(int i = 0; i < h; ++i) a[i] += a[i+h], b[i] += b[i+h];
    karatsuba(a, b, t, t+n, h); // (a0+a1)*(b0+b1)
    for(int i = 0; i < h; ++i) a[i] -= a[i+h], b[i] -= b[i+h];
    for(int i = 0; i < n; ++i) t[i] -= c[i]+c[i+n];
    for(int i = 0; i < n; ++i) c[i+h] += t[i], t[i] = 0;
  }
}

vector<lint> conv(vector<lint> a, vector<lint> b) {
  int sa = a.size(), sb = b.size(); if (!sa || !sb) return {};
  int n = 1<<size(max(sa,sb)); a.resize(n), b.resize(n);
  vector<lint> c(2*n), t(2*n);
  for(int i = 0; i < 2*n; ++i) t[i] = 0;
  karatsuba(&a[0], &b[0], &c[0], &t[0], n);
  c.resize(sa+sb-1); return c;
}
```

count-inversions.h

Description: Count the number of inversions to make an array sorted. Merge sort has another approach.

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

"../data-structures/fenwick-tree.h" 7e4bc9, 6 lines

```
FT<lint> bit(n);
lint inv = 0;
for (int i = n-1; i >= 0; --i) {
  inv += bit.query(values[i]); // careful with the interval
  bit.update(values[i], 1);    // [0, x) or [0, x] ?
}
```

histogram.h

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

a77bf4, 17 lines

```
int max_area(const vector<int>& height) {
  const int N = int(height.size());
  vector<int> L(N), R(N);
  for (int i = N-1; i >= 0; --i) {
    R[i] = i+1;
    while (R[i] < N && height[i] <= height[R[i]]) R[i] = R[R[i
        ]];
  }
  for (int i = 0; i < N; ++i) {
    L[i] = i-1;
    while (L[i] >= 0 && height[i] <= height[L[i]]) L[i] = L[L[i
        ]];
  }
  int area = 0;
  for (int i = 0; i < N; ++i) {
    area = max(area, int64_t(R[i] - L[i] - 1) * heigh[i]);
  }
  return area;
}
```

date-manipulation.h

088459, 44 lines

```
string week_day_str[7] = {"Sunday", "Monday", "Tuesday", "
    Wednesday", "Thursday", "Friday", "Saturday"};
string month_str[13] = {"", "January", "February", "March", "
    April", "May", "June", "July", "August", "September", "
    October", "November", "December"};
map<string, int> week_day_int = {{"Sunday", 0}, {"Monday", 1},
    {"Tuesday", 2}, {"Wednesday", 3}, {"Thursday", 4}, {"
    Friday", 5}, {"Saturday", 6}};
map<string, int> month_int = {{"January", 1}, {"February", 2},
    {"March", 3}, {"April", 4}, {"May", 5}, {"June", 6}, {"
    July", 7}, {"August", 8}, {"September", 9}, {"October",
    10}, {"November", 11}, {"December", 12}};
int month[2][13] = {{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
     30, 31}, {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30,
    31}};

/* O(1) - Checks if year y is a leap year. */
bool leap_year(int y){
  return (y % 4 == 0 && y % 100 != 0) || y % 400 == 0;
}

/* O(1) - Increases the day by one. */
void update(int &d, int &m, int &y){
  if (d == month[leap_year(y)][m]){
    d = 1;
    if (m == 12) {
      m = 1;
      y++;
    }
    else m++;
  }
  else d++;
}

int intToDay(int jd) { return jd % 7; }
int dateToInt(int y, int m, int d) {
  return 1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}
void intToDate(int jd, int &y, int &m, int &d) {
  int x, n, i, j;
  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}
```

n-queens.h

Description: NQueens

e97e9e, 43 lines

```
int ans;
bitset<30> rw, ld, rd; //2*MAX_N -1
bitset<30> iniqueens; //2*MAX_N -1
vector<int> col;
void init(int n){
  ans=0;
  rw.reset();
  ld.reset();
  rd.reset();
  col.assign(n,-1);
}
void init(int n, vector<pair<int,int>> initial_queens){
  //it does NOT check if initial queens are at valid positions
  init(n);
  iniqueens.reset();
  for(pair<int,int> pos: initial_queens){
    int r=pos.first, c= pos.second;
    rw[r] = ld[r-c+n-1] = rd[r+c]=true;
    col[c]=r;
    iniqueens[c] = true;
  }
}
void backtracking(int c, int n){
  if(c==n){
    ans++;
    for(int r:col) cout<<r+1<<" ";
    cout<<"\n";
    return;
  }
  else if(iniqueens[c]){
    backtracking(c+1,n);
  }
  else for(int r=0;r<n;r++){
    if(!rw[r] && !ld[r-c+n-1] && !rd[r+c]){
      // if(board[r][c]!=blocked && !rw[r] && !ld[r-c+n-1] && !
          rd[r+c]){ // if there are blocked possitions
      rw[r] = ld[r-c+n-1] = rd[r+c]=true;
      col[c]=r;
      backtracking(c+1,n);
      col[c]=-1;
      rw[r] = ld[r-c+n-1] = rd[r+c]=false;
    }
  }
```

```
}
```

sudoku-solver.h

6be906, 41 lines

```
int N,m; // N = n*n, m = n; where n equal number of rows or
    columns
array<array<int, 10>, 10> grid;
struct SudokuSolver {
  bool UsedInRow(int row,int num){
    for(int col = 0; col < N; ++col)
      if(grid[row][col] == num) return true;
    return false;
  }
  bool UsedInCol(int col,int num){
    for(int row = 0; row < N; ++row)
      if(grid[row][col] == num) return true;
    return false;
  }
  bool UsedInBox(int row_0,int col_0,int num){
    for(int row = 0; row < m; ++row)
      for(int col = 0; col < m; ++col)
        if(grid[row+row_0][col+col_0] == num) return true;
    return false;
  }
  bool isSafe(int row,int col,int num){
    return !UsedInRow(row,num) && !UsedInCol(col,num) && !
        UsedInBox(row-row%m,col-col%m,num);
  }
  bool find(int &row,int &col){
    for(row = 0; row < N; ++row)
      for(col = 0; col < N; ++col)
        if(grid[row][col] == 0) return true;
    return false;
  }
  bool Solve(){
    int row, col;
    if(!find(row,col)) return true;
    for(int num = 1; num <= N; ++num){
      if(isSafe(row,col,num)){
        grid[row][col] = num;
        if(Solve()) return true;
        grid[row][col] = 0;
      }
    }
    return false;
  }
};
```

floyd-cycle.h

Description: Detect loop in a list. Consider using mod template to avoid overflow.
Time: $\mathcal{O}(n)$

b456ab, 10 lines

```
template<class F>
pair<int,int> find(int x0, F f) {
  int t = f(x0), h = f(t), mu = 0, lam = 1;
  while (t != h) t = f(t), h = f(f(h));
  h = x0;
  while (t != h) t = f(t), h = f(h), ++mu;
  h = f(t);
  while (t != h) h = f(h), ++lam;
  return {mu, lam};
}
```

10.3 Dynamic programming

divide-and-conquer-dp.h

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

2cef33, 50 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  lint f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, lint v) { res[ind] = {k, v}; }
  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<lint, int> best(LLONG_MAX, LO);
    for(int k = max(LO,lo(mid)); k <= min(HI,hi(mid)); ++k)
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};

struct DP { // Modify at will:
  vector<int>a, freq;
  vector<lint>old, cur;
  lint cnt;
  int lcur, rcur;
  DP(const vector<int>&_a, int n): a(_a), freq(n), old(n+1,
      linf), cur(n+1, linf), cnt(0), lcur(0), rcur(0){}
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  void add(int k, int c){ cnt += freq[a[k]]++; }
  void del(int k, int c){ cnt -= --freq[a[k]]; }
  lint C(int l, int r){
    while(lcur > l) add(--lcur, 0);
    while(rcur < r) add(rcur++, 1);
    while(lcur < l) del(lcur++, 0);
    while(rcur > r) del(--rcur, 1);
    return cnt;
  }
  lint f(int ind, int k) { return old[k] + C(k, ind); }

  void store(int ind, int k, lint v) { cur[ind] = v; }
  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<lint, int> best(LLONG_MAX, LO);
    for(int k = max(LO,lo(mid)); k <= min(HI,hi(mid)); ++k)
      best = min(best, make_pair(f(mid, k), k));

    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second);
    rec(mid+1, R, best.second, HI);
  }
};
```

knuth-dp.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 $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

cht-dp.h

Description: Transforms dp of the form (or similar) $dp[i] = min_{j<i}(dp[j] + b[j]*a[i])$. Time goes from $O(n^2)$ to $O(n\log n)$, if using online line container, or $O(n)$ if lines are inserted in order of slope and queried in order of x. To apply try to find a way to write the factor inside minimization as a linear function of a value related to i. Everything else related to j will become constant.

edit-distance.h

Description: Find the minimum numbers of edits required to convert string s into t. Only insertion, removal and replace operations are allowed.

f0f083, 32 lines

```
int edit_dist(string &s, string &t) {
  const int n = int(s.size()), m = int(t.size());
  vector<vector<int>> dp(n+1, vector<int>(m+1, n+m+2));
  vector<vector<int>> prv(n+1, vector<int>(m+1, 0));
  dp[0][0] = 0;
  for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= m; j++) {
      if (i < n) { // remove
        int cnd = dp[i][j] + 1;
        if (cnd < dp[i+1][j]) {
          dp[i+1][j] = cnd;
          prv[i+1][j] = 1;
        }
      }
      if (j < m) { // insert
        int cnd = dp[i][j] + 1;
        if (cnd < dp[i][j+1]) {
          dp[i][j+1] = cnd;
          prv[i][j+1] = 2;
        }
      }
      if (i < n && j < m) { // modify
        int cnd = dp[i][j] + (s[i] != t[j]);
        if (cnd < dp[i+1][j+1]) {
          dp[i+1][j+1] = cnd;
          prv[i+1][j+1] = 3;
        }
      }
    }
  }
  return dp[n][m];
}
```

LIS.h

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

0675f2, 17 lines

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

digit-dp.h

Description: Compute how many # between 1 and N have K distinct digits in the base L without leading zeros;
Usage: auto hex_to_dec = [&](char c) -> int {
return ('A' <= c && c <= 'F' ? (10 + c - 'A') : (c - '0'));
};
digit_dp<modnum<int(1e9) + 7>, hex_to_dec>(N, K);
Time: $\mathcal{O}(NK)$

8eca47, 32 lines

```
template<typename T, class F> T digit_dp(const string& S, int K
    , F& L) {
  const int base = 16;
  const int len = int(S.size());

  vector<bool> w(base);
  vector<vector<T>> dp(len + 1, vector<T>(base + 2));

  int cnt = 0;
  for (int d = 0; d < len; ++d) {
    // adding new digit to numbers wiht prefix < s
    for (int x = 0; x <= base; ++x) {
      dp[d + 1][x] += dp[d][x] * x;
      dp[d + 1][x + 1] += dp[d][x] * (base - x);
    }
    // adding strings whith prefix only 0's and last digit != 0
    if (d) dp[d + 1][1] += (base - 1);
    // adding prefix equal to s and last digit < s, first digit
         cannot be 0
    for (int x = 0; x < L(S[d]); ++x) {
      if (d == 0 && x == 0) continue;
      if (w[x]) dp[d + 1][cnt] += 1;
      else dp[d + 1][cnt + 1] += 1;
    }
    // marking if the last digit appears in the prefix of s
    if (w[L(S[d])] == false) {
      w[L(S[d])] = true;
      cnt++;
    }
  }
  // adding string k
  dp[len][cnt] += 1;
  return dp[len][K];
}
```

LCS.h

Description: Finds the longest common subsequence.
Memory: $\mathcal{O}(nm)$.
Time: $\mathcal{O}(nm)$ where n and m are the lengths of the sequences.

463080, 14 lines

```
template<class T> T lcs(const T &X, const T &Y) {
  int a = X.size(), b = Y.size();
  vector<vector<int>> dp(a+1, vector<int>(b+1));
  for(int i = 1; i <= a; ++i) for(int j = 1; j <= b; j++)
    dp[i][j] = X[i-1]==Y[j-1] ? dp[i-1][j-1]+1 :
      max(dp[i][j-1],dp[i-1][j]);
  int len = dp[a][b];
  T ans(len, 0);
  while (a && b)
    if (X[a-1] == Y[b-1]) ans[--len] = X[--a], --b;
    else if (dp[a][b-1] > dp[a-1][b]) --b;
    else --a;
  return ans;
}
```

knapsack-unbounded.h

Description: Knapsack problem but now take the same item multiple items is allowed.
Time: $\mathcal{O}(N \log N)$

1acfe0, 10 lines

```
int knapsack(vector<int> &v, vector<int> &w, int total) {
  vector<int> dp(total+1, -1);
  int result = 0; dp[0] = 0;
  for (int i = 0; i <= total; ++i) for (int j = 0; j < n; ++j)
    if (w[j] <= i && dp[i - w[j]] >= 0)
      dp[i] = max(dp[i], dp[i - w[j]] + v[j]);
  int result = 0;
  for (int i = 0; i <= total; ++i) result = max(result, dp[i]);
  return result;
}
```

knapsack-bounded.h

Description: You are given n types of items, each items has a weight and a quantity. Is possible to fill a knapsack with capacity k using any subset of items?
Time: $\mathcal{O}(Wn)$

9bddad, 15 lines

```
vector<int> how_many(n+1), dp(k+1);
dp[0] = 1;
for (int i = 1; i <= n; ++i) cin >> how_many[i];
for (int i = 1; i <= n; ++i) {
  for (int j = k-items[i]; j >= 0; --j) {
    if (dp[j]) {
      int x = 1;
      while (x <= how_many[i] &&
        j + x*items[i] <= k && !dp[j + x*items[i]]) {
        dp[j + x*items[i]] = 1;
        ++x;
      }
    }
  }
}
```

knapsack-bounded-costs.h

Description: You are given n types of items, you have $e[i]$ items of i-th type, and each item of i-th type weight $w[i]$ and cost $c[i]$. What is the minimal cost you can get by picking some items weighing at most W in total?
Time: $\mathcal{O}(Wn)$

<MinQueue.h> 3ade3c, 28 lines

```
const int maxn = 1000;
const int maxm = 100000;
const int inf = 0x3f3f3f;

minQueue<int> q[maxm];

array<int, maxm> dp; // the minimum cost dp[i] I need to pay in
     order to fill the knapsack with total weight i
int w[maxn], e[maxn], c[maxn]; // weight, number, cost

int main() {
  int n, m;
  cin >> n >> m;
  for (int i = 1; i <= n; i++) cin >> w[i] >> c[i] >> e[i];
  for (int i = 1; i <= m; i++) dp[i] = inf;
  for (int i = 1; i <= n; i++) {
    for (int j = 0; j < w[i]; j++) q[j].clear();
    for (int j = 0; j <= m; j++) {
      minQueue<int> &mq = q[j % w[i]];
      if (mq.size() > e[i]) mq.pop();
      mq.add(c[i]);
      mq.push(dp[j]);
      dp[j] = mq.getMin();
    }
  }
  cout << "Minimum value i can pay putting a total weight " <<
     m << " is " << dp[m] << '\n';
  for (int i = 0; i <= m; i++) cout << dp[i] << " " << i << '\n
     ';
  cout << "\n";
}
```

knapsack-bitset.h

Description: Find first value greater than m that cannot be formed by the sums of numbers from v.

25166e, 11 lines

```
bitset<int(1e7)> dp, dp1;
int knapsack(vector<int> &items, int n, int m) {
  dp[0] = dp1[0] = true;
  for (int i = 0; i < n; ++i) {
    dp1 <<= items[i];
    dp |= dp1;
    dp1 = dp;
  }
  dp.flip();
  return dp._Find_next(m);
}
```

two-max-equal-sum.h

Description: Two maximum equal sum disjoint subsets, s[i] = 0 if v[i] wasn't selected, s[i] = 1 if v[i] is in the first subset and s[i] = 2 if v[i] is in the second subset
Time: $\mathcal{O}(n * S)$

d66110, 15 lines

```
pair<int, vector<int>> twoMaxEqualSumDS(vector<int> &v){
  const int n = int(v.size());
  const int sum = accumulate(v.begin(), v.end(), 0);
  vector<int> dp(2*sum + 1, INT_MIN/2), newdp(2*sum + 1), s(n);
  vector<vector<int>> rec(n, vector<int>(2*sum + 1));
  int i; dp[sum] = 0;
  for(i = 0; i < n; i++, swap(dp, newdp))
    for(int a, b, d = v[i]; d <= 2*sum - v[i]; d++){
      newdp[d] = max({dp[d], a = dp[d - v[i]] + v[i], b = dp[d
          + v[i]]});
      rec[i][d] = newdp[d] == a ? 1 : newdp[d] == b ? 2 : 0;
    }
  for(int j = i-1, d = sum; j >= 0 ; j--)
    d += (s[j] = rec[j][d]) ? s[j] == 2 ? v[j] : -v[j] : 0;
  return {dp[sum], s};
}
```

max-zero-submatrix.h

Description: Computes the area of the largest submatrix that contains only 0s
Time: $\mathcal{O}(NM)$

d7bff2, 18 lines

```
const int MAXN = 100, MAXM = 100;
array<array<int, MAXN>, MAXM> A, H;
int solve(int N, int M) {
  stack<int, vector<int>> s; int ret = 0;
  for (int j = 0; j < M; j++) for (int i = N - 1; i >= 0; i--)
      H[i][j] = A[i][j] ? 0 : 1 + (i == N - 1 ? 0 : H[i + 1][j
      ]);
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
      int minInd = j;
      while (!s.empty() && H[i][s.top()] >= H[i][j]) {
        ret = max(ret, (j - s.top()) * (H[i][s.top()]));
        minInd = s.top(); s.pop(); H[i][minInd] = H[i][j];
      }
      s.push(minInd);
    }
    while (!s.empty()) ret = max(ret, (M - s.top()) * H[i][s.
        top()]); s.pop();
  }
  return ret;
}
```

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` violations 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

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 for loops and optimizes floating points better (assumes associativity and turns off denormals).
- `#pragma GCC target ("avx,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).

fast-input.h

Description: Returns an integer. Usage requires your program to pipe in input from file. Can replace calls to gc() with getchar_unlocked() if extra speed isn't necessary (60% slowdown).
Usage: `./a.out < input.txt`
Time: About 5x as fast as cin/scanf.
b31afb, 18 lines

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

bump-allocator.h

Description: When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.
745db2, 8 lines

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

bump-allocator-stl.h

Description: BumpAllocator for STL containers.
Usage: `vector<vector<int, small<int>>> ed(N);`
bb66d4, 14 lines

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

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

hashmap.h

Description: Faster/better hash maps, taken from CF
09a72f, 19 lines

```
#include<bits/extc++.h>
struct splitmix64_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x^(x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x^(x >> 27)) * 0x94d049bb133111eb;
    return x^(x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = std::chrono::
        steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

template <typename K, typename V, typename Hash =
    splitmix64_hash>
using hash_map = __gnu_pbds::gp_hash_table<K, V, Hash>;

template <typename K, typename Hash = splitmix64_hash>
using hash_set = hash_map<K, __gnu_pbds::null_type, Hash>;
```

unrolling.h

520e76, 5 lines

```
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if needed
while (i + 4 <= to) { F F F F }
while (i < to) F
```

fast-mod.h

Description: Compute $a\%b$ about 4 times faster than usual, where b is constant but not known at compile time. Fails for $b = 1$.
c977c5, 10 lines

```
typedef unsigned long long ull;
typedef __uint128_t L;
struct FastMod {
  ull b, m;
  FastMod(ull b) : b(b), m(ull((L(1) << 64) / b)) {}
  ull reduce(ull a) {
    ull q = (ull)((L(m) * a) >> 64), r = a - q * b;
    return r >= b ? r - b : r;
  }
};
```

custom-comparator.h

1e3970, 6 lines

```
auto cmp = [](const kind_t& a, const kind_t& b) {
    return a.func() < b.func();
};
set<kind_t, decltype(cmp)> my_set(cmp);
map<kind_t, int, decltype(cmp)> my_map(cmp);
priority_queue<kind_t, vector<kind_t>, decltype(cmp)> my_pq(cmp
    );
```

10.6 Bit Twiddling Hack

hacks.h

829b7d, 22 lines

```
// Iterate on non-empty submasks of a bitmask.
for (int s = m; s > 0; s = (m & (s - 1)))
// Iterate on non-zero bits of a bitset B.
for (int j = B._Find_next(0); j < MAXV; j = B._Find_next(j))

ll next_perm(ll v) { // compute next perm i.e.
  ll t = v | (v-1);  // 00111,01011,01101,10011 ...
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1))
      ;
}
template<typename F>  // All subsets of size k of {0..N-1}
void iterate_k_subset(ll N, ll k, F f){
  ll mask = (1ll << k) - 1;
  while (!(mask & 1ll<<N)) { f(mask);
    ll t = mask | (mask-1);
    mask = (t+1) | (((~t & -~t) - 1) >> (__builtin_ctzll(mask)
        +1));
  }
}
template<typename F> // All subsets of set
void iterate_mask_subset(ll set, F f){ ll mask = set;
  do  f(mask), mask = (mask-1) & set;
  while (mask != set);
}
```

bitset.h

Description: Some bitset functions
b9f55a, 17 lines

```
int main() {
  bitset<100> bt;
  cin >> bt;
  cout << bt[0] << "\n";
  cout << bt.count() << "\n"; // number of bits set
  cout << (~bt).none() << "\n"; // return true if has no bits
      set
  cout << (~bt).any() << "\n"; // return true if has any bit
      set
  cout << (~bt).all() << "\n"; // retun true if has all bits
      set
  cout << bt._Find_first() << "\n"; // return first set bit
  cout << bt._Find_next(10) << "\n";// returns first set bit
      after index i
  cout << bt.flip() << '\n'; // flip the bitset
```

```
  cout << bt.test(3) << '\n'; // test if the ith bit of bt is
       set
  cout << bt.reset(3) << '\n'; // reset the ith bit
  cout << bt.set() << '\n'; // turn all bits on
  cout << bt.set(4, 1) << '\n'; // set the 4th bit to value 1
  cout << bt << "\n";
}
```

10.7 Random Numbers

random-numbers.h

Description: An example on the usage of generator and distribution. Use shuffle instead of random shuffle.

b28ebe, 9 lines

```
mt19937 rng(random_device{}());
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
     count());
shuffle(permutation.begin(), permutation.end(), rng);
uniform_int_distribution<int> uid(1, 100); // [1, 100]
     inclusive!
uniform_real_distribution<double> urd(1, 100);
unsigned xrand() {
  static unsigned x = 314159265, y = 358979323, z = 846264338,
       w = 327950288;
  unsigned t = x ^ x << 11; x = y; y = z; z = w; return w = w ^
        w >> 19 ^ t ^ t >> 8;
}
```