

National Taiwan University

nontoi

FHVirus, nathanlee726, hhhhaura

| 1 Contest | 1 | <pre>#define print() ((void)0) #endif</pre> |
|--|----------|---|
| 2 Mathematics | 1 | troubleshoot.txt |
| 3 Data structures | 3 | Pre-submit: 52 lines |
| 4 Numerical | 4 | Write a few simple test cases if sample is not enough. Are time limits close? If so, generate max cases. Is the memory usage fine? |
| 5 Number theory | 8 | Could anything overflow? Make sure to submit the right file. |
| 6 Combinatorial | 10 | Wrong answer: Print your solution! Print debug output, as well. Are you clearing all data structures between test cases? |
| 7 Graph | 11 | Can your algorithm handle the whole range of input? Read the full problem statement again. Do you handle all corner cases correctly? |
| 8 Geometry | 17 | Have you understood the problem correctly? Any uninitialized variables? |
| 9 Strings | 21 | Any overflows? Confusing N and M, i and j, etc.? Are you sure your algorithm works? |
| 10 Various | 23 | What special cases have you not thought of? Are you sure the STL functions you use work as you think? Add some assertions, maybe resubmit. |
| $\underline{\text{Contest}}$ (1) | | Create some testcases to run your algorithm on. Go through the algorithm for a simple case. Go through this list again. |
| .bashrc | 0.11 | Explain your algorithm to a teammate. |
| alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 -fsanitize=undefined,address -DNONTOI' xmodmap -e 'clear Lock' -e 'keycode 0x42 = Escape' | 3 lines | Ask the teammate to look at your code. Go for a small walk, e.g. to the toilet. Is your output format correct? (including whitespace) Rewrite your solution from the start or let a teammate do it. |
| .vimrc | 5 lines | Runtime error: Have you tested all corner cases locally? |
| se nu ru cul cin aw ai is ts=4 sw=4 noeb bg=dark sy on "Select region and then type :Hash to hash your selection." Useful for verifying that there aren't mistypes. ca Hash w !cpp -dD -P -fpreprocessed \ tr -d '[:space:]' \ | o mes | Any uninitialized variables? Are you reading or writing outside the range of any vector? Any assertions that might fail? Any possible division by 0? (mod 0 for example) Any possible infinite recursion? Invalidated pointers or iterators? |
| template.cpp 1 | 3 lines | Are you using too much memory? Debug with resubmits (e.g. remapped signals, see Various). |
| <pre>#include <bits stdc++.h=""> using namespace std; #define rep(i, a, b) for(int i = a; i < (b); ++i) #define all(x) begin(x), end(x) #define sz(x) (int)(x).size() typedef long long ll; typedef pair<int, int=""> pii; typedef vector<int> vi;</int></int,></bits></pre> | | Time limit exceeded: Do you have any possible infinite loops? What is the complexity of your algorithm? Are you copying a lot of unnecessary data? (References) How big is the input and output? (consider scanf) Avoid vector, map. (use arrays/unordered_map) What do your teammates think about your algorithm? |
| <pre>int main() { cin.tie(0)->sync_with_stdio(0); cin.exceptions(cin.failbit); }</pre> | | Memory limit exceeded: What is the max amount of memory your algorithm should need? Are you clearing all data structures between test cases? |
| , | | cmp.sh |
| debug.cpp Description: Debug tool. #ifdef NONTOI | 11 lines | # A script that checks the correctness of sol.cpp # using testdata generated by gen.cpp and bru.cpp # as a reference Outputs back if found |
| <pre>#Irder NONIOT #define debug(args) LKJ("\033[1;32m["#args"]\033[0m", args template<class i=""> void LKJ(I&&x){ cerr << x << endl; } template<class classt="" i,=""> void LKJ(I&&x, T&&t) { cerr << x << ", ", LKJ(t); } template<class i=""> void print(I a, I b) { while (a != b) cerr << *a++ << ' '; cerr << endl;} #else #define debug() ((void)0)</class></class></class></pre> | 3) | <pre># as a reference. Outputs hack if found. #!/bin/bash source ~/.bashrc && shopt -s expand_aliases c gen.cpp -o g && c bru.cpp -o b && c sol.cpp -o s for i in {1100000}; do echo \$i && ./g>i && ./b<i>a && ./s<i>o && diff -y a o if [\$? == 1]; then echo \$i; cat i; break; fi done</i></i></pre> |

echo Done.

$| \, \underline{\text{Mathematics}} | \, (2)$

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.

$$ax + by = e$$

$$cx + dy = f$$

$$\Rightarrow x = \frac{ed - bf}{ad - bc}$$

$$y = \frac{af - ec}{ad - bc}$$

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

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

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

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

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

Length of median (divides triangle into two equal-area triangles):
$$m = \frac{1}{r} \sqrt{\frac{3k^2 + 3c^2 - c^2}{r^2}}$$

 $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$ Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b+c} \right)^2 \right]}$$
Law of sines: $\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$
Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

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

2.4.2 Quadrilaterals

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

$$4A = 2ef \cdot \sin \theta = \sqrt{4e^2f^2 - F^2} = F|\tan \theta| \ (\theta \neq \frac{\pi}{2})$$

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

2.4.3 Spherical coordinates

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

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:

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

2.6 Sums

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

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

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

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

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

.bashrc .vimrc template debug troubleshoot cmp

2.7 Series

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

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

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

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

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

2.8 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

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

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$
$$\mu = np, \ \sigma^2 = np(1-p)$$

Bin(n, p) is approximately Po(np) for small p.

First success distribution

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

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

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

Poisson distribution

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

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

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

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is 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 $\operatorname{Exp}(\lambda), \lambda > 0.$

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

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

Normal distribution

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

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

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

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

NTU nontoi OrderStatisticTree HashMap LazySegmentTree UnionFindRollback LineContainer Treap

```
2.9 Markov chains
```

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

A Markov chain is a discrete random process with the property

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_i/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

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). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$.

transition probability is uniform among all neighbors, π_i is

two sets **A** and **G**, such that all states in **A** are absorbing $(p_{ii} = 1)$, and all states in G leads to an absorbing state in 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 $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$.

A Markov chain is an A-chain if the states can be partitioned into

Data structures (3)

proportional to node i's degree.

```
OrderStatisticTree.h
```

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

782797, 16 lines

#include <bits/extc++.h> using namespace __gnu_pbds;

template<class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

void example() { Tree<int> t, t2; t.insert(8); auto it = t.insert(10).first; assert(it == t.lower_bound(9)); assert(t.order_of_key(10) == 1); assert (t.order of key (11) == 2); assert(*t.find_by_order(0) == 8); t.join(t2); // assuming T < T2 or T > T2, merge t2 into t

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but $\sim 3x$ faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if proyided). #include <bits/extc++.h>

// To use most bits rather than just the lowest ones: struct chash { // large odd number for C

```
const uint64_t C = 11(4e18 * acos(0)) | 71;
 11 operator()(11 x) const { return __builtin_bswap64(x*C); }
__qnu_pbds::qp_hash_table<11,int,chash> h({},{},{},{},{1<<16});</pre>
LazySegmentTree.h
Description: ZKW implementation with ACL style nodes. [l, r).
Usage: SGT<Val, Tag> sgt(n);
Time: \mathcal{O}(N + Q \log N).
struct Val {
 int v:
 Val(int v = 0) : v(v) {} // must return identity element
  Val operator + (const Val& o) const {
   return Val(max(v, o.v)); }
  // merge two Vals, order is important
struct Tag
 int t;
 Tag(int t = 0) : t(t) {} // must return identity element
 Tag operator + (const Tag& o) const { return Tag(t + o.t); }
  // compose two Tags, order is important
 Val operator() (Val v) const { return Val(v.v + t); }
 // apply the Tag to v
int bc(int u) { return u <= 1 ? 1 : (2 << __lq(u-1)); }</pre>
template <class V, class T> struct SGT {
 int n; vector<V> val; vector<T> tag;
  SGT(int n): n(bc(n)), val(n*2), tag(n*2) {}
  SGT(const vector\langle V \rangle \& V): n(bc(sz(V))), val(n*2), tag(n*2) {
   rep (i, 0, sz(v)) val[i+n] = v[i];
   for (int i = n; --i; ) val[i] = val[i*2] + val[i*2+1];
  void upd(int u, T t)
  { val[u] = t(val[u]); if (u < n) tag[u] = tag[u] + t; }
  { while (u /= 2) \ val[u] = tag[u] (val[u*2] + val[u*2+1]); } }
  void push (int u) {
    for (int h = __lg(n) +1, i; --h;) {
     i = u \gg h;
     upd(i * 2, tag[i]);
     upd(i * 2 + 1, tag[i]);
     tag[i] = T();
  void set(int p, V v)
  { push(p += n); val[p] = v; pull(p); }
  V query(int 1, int r) {
    for (push(1+=n), push((r+=n)-1); 1 < r; 1 /= 2, r /= 2) {
     if (1 & 1) rl = rl + val[l++];
     if (r & 1) rr = val[--r] + rr;
   return rl + rr;
 void modify(int 1, int r, T t) {
   int t1 = (1 += n), tr = (r += n) - 1;
   for (push(t1), push(tr); 1 < r; 1 >>= 1, r >>= 1) {
     if (1 & 1) upd(1++, t);
     if (r & 1) upd(--r, t);
   pull(tl); pull(tr);
UnionFindRollback.h
Description: Disjoint-set data structure with undo. If undo is not needed,
```

skip (A)'s: st, time() and rollback().

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

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

```
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]); }</pre>
  int time() { return sz(st); } // (A)
  void rollback(int t) { // (A)
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]}); // (A)
    st.push_back({b, e[b]}); // (A)
    e[a] += e[b]; e[b] = a;
    return true;
LineContainer.h
Description: Container where you can add lines of the form kx+m, and query
maximum values at points x. Useful for dynamic programming ("convex hull
Time: \mathcal{O}(\log N)
struct Line {
 mutable 11 k, m, p; // minimum: change to k > o.k;
 bool operator<(const Line& o) const { return k < o.k; }</pre>
 bool operator<(ll x) const { return p < x; }</pre>
struct LineContainer : multiset<Line, less<>>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  11 div(11 a, 11 b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
 bool isect(iterator x, iterator y) {
    if (y == end()) return x \rightarrow p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf; // <
    else x -> p = div(y -> m - x -> m, x -> k - y -> k);
    return x->p >= y->p;
  void add(ll k, ll m) {
    auto z = insert(\{k, m, 0\}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() \&\& isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() \&\& (--x)->p >= y->p)
      isect(x, erase(y));
  ll query(ll x) {
    assert(!empty());
    auto 1 = *lower_bound(x);
    return 1.k * x + 1.m;
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)
                                                          dd38d5, 50 lines
struct Node {
 Node *1 = 0, *r = 0;
 int val, c = 1, y; // maybe not use rand()!
```

Node(int val) : val(val), y(rand()) {}

void pull();

de4ad0, 21 lines | int cnt(Node* n) { return n ? n->c : 0; }

struct RollbackUF {

799620, 55 lines

vi e; vector<pii> st; // (A)

```
template < class F > void each (Node * n, F f)
{ if (n) { each (n->1, f); f (n->val); each (n->r, f); } }
pair<Node*, Node*> split(Node* n, int k) {
if (!n) return {};
 if (cnt(n->1) >= k) { // "n->val>= k" for lower_bound(k)}
   auto pa = split(n->1, k);
   n->1 = pa.second;
   n->pull();
   return {pa.first, n};
 auto pa = split(n->r, k - cnt(n->1) - 1); // and just "k"
 n->r = pa.first;
 n->pull();
 return {n, pa.second};
Node* merge(Node* 1, Node* r) {
if (!1) return r;
 if (!r) return 1;
 if (1->y > r->y) {
   1->r = merge(1->r, r);
   1->pull();
   return 1;
 r->1 = merge(1, r->1);
 r->pull();
 return r;
Node* ins(Node* t, Node* n, int pos) { // O-base
auto pa = split(t, pos);
 return merge (merge (pa.first, n), pa.second);
// Example application: move the range \lceil l, r \rceil to index k
// void\ move(Node*&\ t,\ int\ l,\ int\ r,\ int\ k) {
// Node *a, *b, *c;
// tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
// if (k \le l) t = merge(ins(a, b, k), c);
// else t = merge(a, ins(c, b, k - r));
RMQ.h
Description: Range Minimum Queries on an array. Returns min(V[a], V[a +
1], ... V[b - 1]) in constant time.
Usage: RMQ rmg(values); rmg.query(inclusive, exclusive);
Time: \mathcal{O}\left(|V|\log|V|+Q\right)
                                                        05c3c0, 16 lines
template <class T>
struct RMO {
 vector<vector<T>> a;
 RMQ(const vector<T>& v) : a(1, v) {
   for (int p = 1, k = 1; p * 2 \le sz(v); p *= 2, ++k) {
     a.emplace_back(sz(v) - p * 2 + 1);
     rep(j, 0, sz(a[k]))
       a[k][j] = min(a[k-1][j], a[k-1][j+p]);
 T query(int 1, int r) {
   assert(1 < r):
   int d = 31 - builtin clz(r - 1);
   return min(a[d][l], a[d][r - (1 << d)]);</pre>
```

};

MoQueries.h

void Node::pull() { c = cnt(l) + cnt(r) + 1; }

```
TSP through the queries, and moving from one query to the next by adding/re-
moving points at the ends. If values are on tree edges, change step to add/re-
move the edge (a, c) and remove the initial add call (but keep in).
Time: \mathcal{O}\left(N\sqrt{Q}\right)
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
vi mo(vector<pii> Q) {
 int L = 0, R = 0, blk = 350; // \sim N/sqrt(Q)
 vi s(sz(0)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
 iota(all(s), 0);
  sort(all(s), [\&](int s, int t) \{ return K(Q[s]) < K(Q[t]); \});
 for (int qi : s) {
   pii q = Q[qi];
   while (L > q.first) add(--L, 0);
   while (R < g.second) add(R++, 1);
   while (L < q.first) del(L++, 0);
   while (R > g.second) del(--R, 1);
   res[qi] = calc();
 return res:
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0) {
 int N = sz(ed), pos[2] = {}, blk = 350; // \sim N/sqrt(Q)
 vi s(sz(Q)), 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(all(s), 0);
 sort(all(s), [\&](int s, int t) \{ return K(Q[s]) < K(Q[t]); \});
 for (int qi : s) rep(end, 0, 2) {
   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] \le L[a] \&\& R[a] \le 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;
Numerical (4)
```

Description: Answer interval or tree path queries by finding an approximate

4.1 Polynomials and recurrences

int n() const { return (int) size(); }

```
Polynomial.h
"NumberTheoreticTransform.h", "../number-theory/ModPow.h", "../number-theory/ModSqrt.h"
template <11 mod, 11 root> struct Poly : v1 {
 typedef Poly P;
 static int bc(int n)
 { return n <= 1 ? 1 : 1 << (32 - __builtin_clz(n - 1)); }
 static NTT<mod, root> ntt; // coefficients in [0, P)
 explicit Poly(int n = 1) : v1(n) {}
```

```
static ll modsub(ll a, ll b) { return a-b + (a<b?mod:0); }</pre>
static ll modinv(ll x) { return modpow(x, mod - 2, mod); }
Poly(const v1 &v) : v1(v) {}
Poly(const P &p, int m) : v1(m)
{ copy_n(p.data(), min(sz(p), m), data()); }
P &irev() { return reverse(all()), *this; }
P &isz(int s) { return resize(s), *this; }
P & iadd (const P & o) { // n() = o.n()
  rep(i,0,n()) (*this)[i] = ((*this)[i] + o[i]) % mod;
  return *this; }
P &imul(ll k) {
  rep(i, 0, n()) (*this)[i] = ((*this)[i] * k) % mod;
  return *this; }
P &imul(P v) {
  rep(i,0,n()) (*this)[i] = ((*this)[i] * v[i]) % mod;
  return *this; }
  while (n() > 1 and back() == 0) pop_back();
  return *this; }
P Mul(const P &o) const {
  const auto s = bc(n() + sz(o) - 1), inv = modinv(s);
  P x(*this, s), y(o, s), out(s);
  ntt(x.data(), s), ntt(y.data(), s);
  rep(i,0,s) out[-i & (s-1)] = x[i] * y[i] % mod * inv % mod;
  ntt(out.data(), s);
  return out.isz(n() + sz(o) - 1);
P Inv() const { assert(*begin() != 0);
  if (n() == 1) return vl{modinv(*begin())};
  const auto s = bc(n() * 2), inv = modinv(s);
  P x = P(*this, (n()+1)/2).Inv().isz(s), y(*this,s), out(s);
  ntt(x.data(), s), ntt(y.data(), s);
  rep(i, 0, s) {
    11 \&t = out[-i \& (s - 1)];
    t = x[i] * modsub(2, x[i]*y[i]%mod) % mod * inv % mod;
  ntt(out.data(), s);
  return out.isz(n());
P sqimp() const { // coef[0] \setminus in [1, mod)^2
  if (n() == 1) return v1{modsqrt((*this)[0], mod)};
  P x = P(*this, (n() + 1) / 2).Sgrt().isz(n());
  return x.iadd(Mul(x.Inv()).isz(n())).imul(mod / 2 + 1);
P Sqrt() const { // returns { -1 } on fail
  int m = 0; while (m < n() and (*this)[m] == 0) ++m;
  if (m == n()) return P(n());
  if (m % 2 or modsqrt((*this)[m], mod) == -1) return v1{-1};
  P p = P(vl{data() + m, data() + n()}, n() - m/2).sqimp();
  return p.irev().isz(sz(p) + m / 2).irev();
pair<P, P> DivMod(P o) const { // {0} for 0
 if (n() < sz(o)) return {vl{0}, *this};</pre>
  const int s = n() - sz(o) + 1;
  P x(o); x.irev().isz(s);
  P y(*this); y.irev().isz(s);
  PQ = y.Mul(x.Inv()).isz(s).irev();
  x = o.Mul(Q), y = *this;
  rep(i, 0, n()) y[i] = modsub(y[i], x[i]);
  return {Q, y.isz(max(1, sz(o)-1))};
P Dx() const {
 P ret(n() - 1);
  rep(i, 0, sz(ret)) ret[i] = (i + 1) * (*this)[i + 1] % mod;
  return ret.isz(max(1, sz(ret)));
```

PolyRoots BerlekampMassey LinearRecurrence GoldenSectionSearch

```
NTU nontoi
```

P Sx() const {

P ret(n() + 1);

P Ln() const { // coef[0] == 1

P Exp() const { // coef[0] == 0

if (n() == 1) return v1{1};

return x.Mul(y).isz(n());

P Pow(const string &K) const {

for (char c : K) {

nk2 % = mod - 1;

const int m = sz(x);

vector<P> up(m * 2);

vector<P> dn(m * 2);

const int m = sz(x);

 $// a_n = \sum_{i=1}^n a_i(n-i)$

return dn[1];

C[k] = 1;

return up;

return v:

int nz = 0; 11 nk = 0, nk2 = 0;

nk2 = nk2 * 10 + c - '0';

P tmul(int nn, const P &rhs) const {

P Y = Mul(rhs).isz(n() + nn - 1);

static vector<P> tree(const vl &x) {

for (int i = m - 1; i > 0; --i)

up[i] = up[i * 2].Mul(up[i * 2 + 1]);

const int m = sz(x); if (!m) return {};

vector<P> up = tree(x), dn(m * 2); vl z = up[1].Dx().eval(x, up);

rep(i, 0, m) $dn[m + i] = vl\{z[i]\};$

 $P C(k + 1), W(v1\{1\}, k), M = v1\{0, 1\};$

rep(i, 1, k + 1) C[k - i] = (c[i] == 0 ? 0 : mod - c[i]);

for (int i = m - 1; i > 0; --i)

while (nz < n() && !(*this)[nz]) ++nz;</pre>

if (nk2 * nz >= n()) **return** P(n());

if (!nk && !nk2) return P(v1{1}, n());

nk = (nk * 10 + c - '0') % mod;

P y = x.Ln(); y[0] = mod - 1;

return Dx().Mul(Inv()).Sx().isz(n());

```
rep(i, 0, n()) ret[i+1] = modinv(i + 1) * (*this)[i] % mod;
 P x = P(*this, (n() + 1) / 2).Exp().isz(n());
 rep(i, 0, n()) y[i] = modsub((*this)[i], y[i]);
 P x = vl(data() + nz, data() + n() - nz * (nk2 - 1));
 return x.imul(modinv(x0)).Ln().imul(nk).Exp().
   imul(modpow(x0, nk2, mod)).irev().isz(n()).irev();
 return P({Y.data() + n() - 1, Y.data() + Y.n()});
 rep(i, 0, m) up[m + i] = vl{(x[i] ? mod - x[i] : 0), 1};
v1 eval(const v1 &x, const vector<P> &up) const {
 dn[1]=P(up[1]).irev().isz(n()).Inv().irev().tmul(m,*this);
 rep(i,2,m*2) dn[i] = up[i^1].tmul(up[i].n()-1, dn[i/2]);
 vl y(m); rep(i, 0, m) y[i] = dn[m + i][0];
vl Eval(const vl &x) const { return eval(x, tree(x)); }
static P Interpolate (const vl &x, const vl &y) { // 1e5, 1.4s
 rep(i, 0, m) z[i] = y[i] * modinv(z[i]) % mod;
  dn[i]=dn[i*2].Mul(up[i*2+1]).iadd(dn[i*2+1].Mul(up[i*2]));
static ll LinearRecursion(const vl &a, const vl &c, ll n) {
 const int k = sz(a); assert(sz(c) == k + 1);
```

```
while (n) {
      if (n % 2) W = W.Mul(M).DivMod(C).second;
      n /= 2; M = M.Mul(M).DivMod(C).second;
    11 \text{ ret} = 0;
    rep(i, 0, k) ret = (ret + W[i] * a[i]) % mod;
  P TaylorShift(ll c) const {
    P fac(n()), caf(n()); fac[0] = 1;
    rep (i, 1, n()) fac[i] = fac[i-1] * i % mod;
    rep (i, 0, n()) caf[i] = modinv(fac[i]);
    P x = P(*this).imul(fac), y(n()); ll w = 1;
    rep (i, 0, n()) y[i] = w * caf[i] % mod, w = w * c % mod;
    return x.irev().Mul(y).isz(n()).irev().imul(caf);
  P SamplingShift(int m, ll c) const {
    const int k = max(n(), m);
    P fac(k), caf(k); fac[0] = 1;
    rep (i, 1, k) fac[i] = fac[i-1] * i % mod;
    rep (i, 0, k) caf[i] = modinv(fac[i]);
    P x = P(*this).imul(caf), y = caf;
    rep (i, 0, n()) if (i & 1)
     y[i] = (y[i] == 0 ? 0 : mod - y[i]);
    x = x.Mul(y).isz(n()).imul(fac).irev();
    rep (i, 0, n()) y[i] = caf[i] * w % mod,
        w = w * modsub(c, i) % mod;
    x = x.Mul(y).isz(n()).irev().imul(caf).isz(m);
   y = caf; y = y.isz(m);
    return x.Mul(y).isz(m).imul(fac);
};
PolyRoots.h
Description: Finds the real roots to a polynomial.
Usage: polyRoots(\{\{2,-3,1\}\},-1e9,1e9) // solve x^2-3x+2=0
Time: \mathcal{O}\left(n^2\log(1/\epsilon)\right)
"Polynomial.h"
                                                         b00bfe, 23 lines
vector<double> polyRoots(Poly p, double xmin, double xmax) {
 if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Polv der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i, 0, sz(dr) - 1) {
   double 1 = dr[i], h = dr[i+1];
    bool sign = p(1) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (1 + h) / 2, f = p(m);
        if ((f \le 0) ^ sign) 1 = m;
        else h = m;
      ret.push_back((1 + h) / 2);
 return ret;
BerlekampMassev.h
Description: Recovers any n-order linear recurrence relation from the first
2n terms of the recurrence. Useful for guessing linear recurrences after brute-
```

forcing the first terms. Should work on any field, but numerical stability for

floats is not guaranteed. Output will have size $\leq n$.

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

```
"../number-theory/ModPow.h"
const 11 mod = 5;
vector<ll> berlekampMassey(vector<ll> s) {
 int n = sz(s), L = 0, m = 0;
 vector<ll> C(n), B(n), T;
 C[0] = B[0] = 1;
 11 b = 1;
 rep(i,0,n) { ++m;
   11 d = s[i] % mod;
    rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C; ll coef = d * modpow(b, mod-2, mod) % mod;
    rep(j, m, n) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
   L = i + 1 - L; B = T; b = d; m = 0;
 C.resize(L + 1); C.erase(C.begin());
 for (11& x : C) x = (mod - x) % mod;
 return C;
LinearRecurrence.h
Description: Generates the k'th term of an n-order linear recurrence S[i]
\sum_{i} S[i-j-1]tr[j], given S[0... \ge n-1] and tr[0...n-1]. Faster than matrix
multiplication. Useful together with Berlekamp-Massey.
Usage: linearRec(\{0, 1\}, \{1, 1\}, k) // k'th Fibonacci number
Time: \mathcal{O}\left(n^2 \log k\right)
typedef vector<ll> Poly;
11 linearRec(Poly S, Poly tr, 11 k) {
 int n = sz(tr);
  auto combine = [&](Poly a, Poly b) {
   Poly res(n * 2 + 1);
    rep(i, 0, n+1) rep(j, 0, n+1)
      res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
 };
 Poly pol(n + 1), e(pol);
 pol[0] = e[1] = 1;
 for (++k; k; k /= 2) {
   if (k % 2) pol = combine(pol, e);
   e = combine(e, e);
 rep(i, 0, n) res = (res + pol[i + 1] * S[i]) % mod;
 return res;
```

3a4764, 21 lines

4.2 Optimization

GoldenSectionSearch.h

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

Description: Finds the argument minimizing the function f in the interval [a,b] assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is eps. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version. Usage: double func(double x) { return 4+x+.3*x*x; }

```
double xmin = gss(-1000, 1000, func);
Time: \mathcal{O}\left(\log((b-a)/\epsilon)\right)
                                                                                  d7b114, 15 lines
```

template <class F> double gss(double a, double b, F f) { NTU nontoi HillClimbing Integrate IntegrateAdaptive Simplex Duality Determinant IntDeterminant SolveLinear

Description: Solves a general linear maximization problem: maximize $c^T x$

subject to $Ax \leq b$, $x \geq 0$. Returns -inf if there is no solution, inf if there are

arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input

```
double f1 = f(x1), f2 = f(x2);
 while (b-a > eps)
   if (f1 < f2) { //change to > to find maximum
     b = x2; x2 = x1; f2 = f1;
     x1 = b - r*(b-a); f1 = f(x1);
     a = x1; x1 = x2; f1 = f2;
     x2 = a + r*(b-a); f2 = f(x2);
 return a;
HillClimbing.h
Description: Poor man's optimization for unimodal functions.

8eeeaf, 14 lines
typedef array<double, 2> P;
template < class F > pair < double, P > hillClimb (P start, F f) {
 pair<double, P> cur(f(start), start);
 for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
   rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
     P p = cur.second;
     p[0] += dx*jmp;
     p[1] += dy * jmp;
     cur = min(cur, make_pair(f(p), p));
 return cur;
Integrate.h
Description: Simple integration of a function over an interval using Simpson's
rule. The error should be proportional to h^4, although in practice you will want
to verify that the result is stable to desired precision when epsilon changes 4756fc. Times
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
 double h = (b - a) / 2 / n, v = f(a) + f(b);
 rep(i,1,n*2)
  v += f(a + i*h) * (i&1 ? 4 : 2);
 return v * h / 3;
IntegrateAdaptive.h
Description: Fast integration using an adaptive Simpson's rule.
Usage: double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [&](double y)
return quad(-1, 1, [&](double z) {
return x*x + y*y + z*z < 1; {);};};
                                                          92dd79, 15 lines
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
template <class F>
d rec(F& f, d a, d b, d eps, d S) {
dc = (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)</pre>
   return T + (T - S) / 15;
 return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
template<class F>
d \text{ quad}(d \text{ a, } d \text{ b, } F \text{ f, } d \text{ eps} = 1e-8)  {
return rec(f, a, b, eps, S(a, b));
Simplex.h
```

double r = (sqrt(5)-1)/2, eps = 1e-7;

double x1 = b - r*(b-a), x2 = a + r*(b-a);

```
vector is set to an optimal x (or in the unbounded case, an arbitrary solution
fulfilling the constraints). Numerical stability is not guaranteed. For better
performance, define variables such that x = 0 is viable.
Usage: simplex.init(n, m);
simplex.a[i][j] = aij;
int val = simplex.solve();
Time: \mathcal{O}(NM * \#pivots), where a pivot may be e.g. an edge relaxation.
\mathcal{O}\left(2^{N}\right) in the general case.
                                                         cc78c7, 66 lines
static constexpr long double eps = 1e-7;
template <class T = double>
struct Simplex {
 int n, m;
  vector<int> 1, d;
  vector<vector<T>> a;
  vector<T> b, c, sol;
  bool eq(T a, T b) { return fabs(a - b) < eps; }</pre>
  bool ls(T a, T b) { return a < b && !eq(a, b); }
  void init(int p, int q) {
   n = p; m = q; v = 0;
   1.assign(m, 0); b = 1;
   d.assign(n, 0); c = sol = d;
    a.assign(m, vector<T>(n, 0));
  void pivot(int x,int y) {
    swap(l[x], d[y]);
    T k = a[x][y]; a[x][y] = 1;
    vector<int> nz;
    rep (i, 0, n) {
      a[x][i] /= k;
      if(!eq(a[x][i], 0)) nz.push_back(i);
    b[x] /= k;
    rep (i, 0, m) {
     if(i == x || eq(a[i][y], 0)) continue;
      k = a[i][y]; a[i][y] = 0;
      b[i] = k*b[x];
      for(int j : nz) a[i][j] -= k * a[x][j];
    if(eq(c[y], 0)) return;
    k = c[y]; c[y] = 0;
    v += k * b[x];
    for(int i : nz) c[i] -= k * a[x][i];
  // 0: found solution, 1: no feasible solution, 2: unbounded
  int solve() {
    rep (i, 0, n) d[i] = i;
    rep (i, 0, m) l[i] = n+i;
    while (1) { // Eliminating negative b[i]
      int x = -1, y = -1;
      rep (i, 0, m) if (ls(b[i], 0) && (x == -1 || b[i] < b[x]))
           x = i;
      if(x == -1) break;
      rep (i, 0, n) if (ls(a[x][i], 0) && (y == -1 || a[x][i] < a
           [x][y])) y = i;
      if(y == -1) return 1;
      pivot(x, v);
    while(1) {
      int x = -1, y = -1;
      rep (i, 0, n)
       if (ls(0, c[i]) \&\& (y == -1 || c[i] > c[y])) y = i;
      if(y == -1) break;
```

```
if (ls(0, a[i][y]) \&\& (x == -1 || b[i]/a[i][y] < b[x]/a[x]
               ][y])) x = i;
       if(x == -1) return 2;
       pivot(x, y);
    rep (i, 0, m) if(l[i] < n) sol[l[i]] = b[i];
Duality.h
Description: Finds the Dual problem of an LP Maximize Z = c^T x \leftrightarrow \text{Mini-}
mize W = y^T b s.t. Ax \le b \leftrightarrow s.t. A^T y \ge c and x \ge 0 \leftrightarrow and y \ge 0. variables
to constraints, constraints to variables. weak duality property: any feasible
solution x of a primal problem and any feasible solution y of the dual problem
dual problem satisfies c^T x \leq b^T y. Strong duality property: any feasible solu-
tion x of a primal problem and any feasible solution y of the dual problem dual
problem satisfies \vec{c}^T x = \vec{b}^T y.
4.3
       Matrices
```

Determinant.h

rep (i, 0, m)

```
Description: Calculates determinant of a matrix. Destroys the matrix.
```

```
Time: \mathcal{O}(N^3)
double det(vector<vector<double>>& a) {
 int n = sz(a); double res = 1;
 rep(i,0,n) {
   int b = i;
   rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
   if (i != b) swap(a[i], a[b]), res *= -1;
   res *= a[i][i];
   if (res == 0) return 0;
   rep(j,i+1,n) {
     double v = a[j][i] / a[i][i];
      if (v != 0) rep(k, i+1, n) a[j][k] -= v * a[i][k];
 return res;
```

IntDeterminant.h

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

```
3313dc, 18 lines
const 11 mod = 12345;
11 det(vector<vector<11>>& a) {
 int n = sz(a); ll ans = 1;
 rep(i,0,n) {
    rep(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step}
        ll t = a[i][i] / a[j][i];
        if (t) rep(k,i,n)
         a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans \star = -1;
    ans = ans * a[i][i] % mod;
   if (!ans) return 0;
  return (ans + mod) % mod;
```

SolveLinear.h **Description:** Solves A * x = b. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost, B are the m - rank solutions for Ax = 0. T has to have +, -, *, /, val

rep(i,0,n) {

T bv = T(0);

break:

int n = sz(A), m = sz(x), rank = 0, br, bc;

br = r, bc = c, bv = A[r][c];

rep(j,0,n) swap(A[j][i], A[j][bc]);

rep(j, i+1, m) A[i][j] = A[i][j] / A[i][i];

if (n) assert(sz(A[0]) == m);

vi col(m); iota(all(col), 0);

if (A[r][c].val() > 0)

rep(r,i,n) rep(c,i,m)

if (bv.val() == 0) {

swap(A[i], A[br]);

swap(b[i], b[br]);

A[i][i] = T(1);

x.assign(m, T(0));

rep(i, rank, m) {

// Get Homo Solutions

vector<T> sol(m);

B.push_back(sol);

// matrix A is lost

x[col[i]] = b[i];

SolveLinearBinary.h

typedef bitset<1000> bs;

 $assert(m \le sz(x));$

if (br == n) {

swap(A[i], A[br]);

swap(b[i], b[br]);

swap(col[i], col[bc]);

int n = sz(A), rank = 0, br;

vi col(m); iota(all(col), 0);

rep(j,i,n) if(b[j]) return -1;

int bc = (int)A[br]._Find_next(i-1);

rep(j,0,n) **if** $(A[j][i] != A[j][bc]) {$

Time: $O(n^2m)$

rep(i,0,n) {

break:

for (**int** i = rank; i--;) {

rep(j,0,i) b[j] = b[j] - A[j][i] * b[i];

for (br=i; br<n; ++br) if (A[br].any()) break;</pre>

b[i] = b[i] / A[i][i];

rank++;

B.clear();

swap(col[i], col[bc]);

b[i] = b[i] / A[i][i];

rep(j,0,n) **if**(j != i){

b[j] = b[j] - fac * b[i];

T fac = A[j][i];

```
template < class T > int SolveLinear (vector < vector < T >> & A,
   vector<vector<T>>& B, vector<T>& b, vector<T>& x) {
     rep(j,i,n) if (abs(b[j].val()) > 0) return -1;
     rep(k,i,m) A[j][k] = A[j][k] - (fac*A[i][k]);
   rep(j, 0, rank) sol[col[j]] = (A[j][i] * T(-1));
   rep(j, rank, m) sol[col[j]] = (j == i ? 1 : 0);
 return rank; // (multiple solutions if rank < m)
Description: Solves Ax = b over \mathbb{F}_2. If there are multiple solutions, one is
returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
                                                           fa2d7a, 34 lines
int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
```

650f96, 47 lines

```
rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
   rank++;
  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
   x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  return rank; // (multiple solutions if rank < m)
MatrixInverse.h
Description: Invert matrix A. Returns rank; result is stored in A unless sin-
gular (rank < n). Can easily be extended to prime moduli; for prime powers,
repeatedly set A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k} where A^{-1} starts as the in-
verse of A mod p, and k is doubled in each step.
Time: \mathcal{O}\left(n^3\right)
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;</pre>
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] = f*A[i][k];
      rep(k,0,n) tmp[j][k] \rightarrow f*tmp[i][k];
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] \rightarrow v*tmp[i][k];
  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
MatrixInverse-mod.h
Description: Invert matrix A modulo a prime. Returns rank; result is
stored in A unless singular (rank < n). For prime powers, repeatedly set
A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k} where A^{-1} starts as the inverse of A mod
p, and k is doubled in each step.
Time: \mathcal{O}\left(n^3\right)
"../number-theory/ModPow.h"
                                                            4a65ea, 36 lines
```

int matInv(vector<vector<11>>& A) {

int n = sz(A); vi col(n);

SolveLinearBinary MatrixInverse MatrixInverse-mod Tridiagonal

A[j].flip(i); A[j].flip(bc);

```
vector<vector<ll>> tmp(n, vector<ll>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
   int r = i, c = i;
    rep(j,i,n) rep(k,i,n) if (A[j][k]) {
      r = j; c = k; goto found;
    return i;
found:
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n) swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c])
    swap(col[i], col[c]);
    11 v = modpow(A[i][i], mod - 2, mod);
    rep(j,i+1,n) {
      11 f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k, i+1, n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
      rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    11 v = A[j][i];
    rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod;
  rep(i,0,n) rep(i,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0 ? mod :
  return n;
Tridiagonal.h
Description: x = \text{tridiagonal}(d, p, q, b) solves the equation system
                                    0
                                                    0
       b_1
                        d_1 \quad p_1
                                                               x_1
                    q_0
                                                    0
       b_2
                    0
                            d_2
                        q_1
                                    p_2
                                                               x_2
       b_3
                                                               x_3
                    0
                        0 ...
                                   q_{n-3} d_{n-2} p_{n-2}
                        0
This is useful for solving problems on the type
                  a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, 1 \le i \le n,
where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from
      \{a_i\} = \text{tridiagonal}(\{1, -1, -1, ..., -1, 1\}, \{0, c_1, c_2, ..., c_n\},
                        \{b_1, b_2, \ldots, b_n, 0\}, \{a_0, d_1, d_2, \ldots, d_n, a_{n+1}\}\}
Fails if the solution is not unique.
If |d_i| > |p_i| + |q_{i-1}| for all i, or |d_i| > |p_{i-1}| + |q_i|, or the matrix is positive
definite, the algorithm is numerically stable and neither tr nor the check for
diag[i] == 0 is needed.
Time: \mathcal{O}(N)
                                                              8f9fa8, 26 lines
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
  int n = sz(b); vi tr(n);
  rep(i, 0, n-1) {
    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
      b[i+1] -= b[i] * diag[i+1] / super[i];
      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];</pre>
      diag[i+1] = sub[i]; tr[++i] = 1;
    } else {
      diag[i+1] -= super[i]*sub[i]/diag[i];
```

if (a.empty() || b.empty()) return {};

vl res(sz(a) + sz(b) - 1);

```
if (tr[i]) {
     swap(b[i], b[i-1]);
     diag[i-1] = diag[i];
     b[i] /= super[i-1];
   } else {
     b[i] /= diag[i];
     if (i) b[i-1] -= b[i]*super[i-1];
 return b;
4.4 Fourier transforms
FastFourierTransform.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 vec-
tors: 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| (\sim 1s for N = 2^{22})
                                                             00ced6, 35 lines
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
 int n = sz(a), L = 31 - __builtin_clz(n);
 static vector<complex<long double>> R(2, 1);
 static vector<C> rt(2, 1); // (^ 10% faster if double)
 for (static int k = 2; k < n; k \neq 2) {
   R.resize(n); rt.resize(n);
   auto x = polar(1.0L, acos(-1.0L) / k);
   rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
 vi rev(n);
 rep(i,0,n) \ rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
 rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
 for (int k = 1; k < n; k *= 2)
   for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
     Cz = rt[j+k] * a[i+j+k]; // (25\% faster if hand-rolled)
     a[i + j + k] = a[i + j] - z;
     a[i + j] += z;
vd conv(const vd& a, const vd& b) {
 if (a.empty() || b.empty()) return {};
 vd res(sz(a) + sz(b) - 1);
 int L = 32 - \underline{\quad} builtin_clz(sz(res)), n = 1 << L;
 vector<C> in(n), out(n);
 copy(all(a), begin(in));
 rep(i,0,sz(b)) in[i].imag(b[i]);
 fft(in);
 for (C& x : in) x *= x;
 rep(i,0,n) out[i] = in[-i & (n-1)] - conj(in[i]);
 rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
 return res;
FastFourierTransformMod.h
Description: Higher precision FFT, can be used for convolutions modulo arbi-
trary integers as long as N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14} (in practice 10^{16} or higher).
Inputs must be in [0, mod).
Time: \mathcal{O}(N \log N), where N = |A| + |B| (twice as slow as NTT or FFT)
"FastFourierTransform.h"
                                                             b82773, 22 lines
typedef vector<ll> v1;
```

template<int M> vl convMod(const vl &a, const vl &b) {

b[i+1] -= b[i] * sub[i] / diag[i];

for (int i = n; i--;) {

```
int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));</pre>
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i, 0, sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i \& (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  fft (outl), fft (outs);
  rep(i,0,sz(res)) {
    11 \text{ av} = 11(\text{real}(\text{outl}[i]) + .5), \text{ cv} = 11(\text{imag}(\text{outs}[i]) + .5);
    11 \text{ bv} = 11(\text{imag}(\text{outl}[i]) + .5) + 11(\text{real}(\text{outs}[i]) + .5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  return res;
NumberTheoreticTransform.h
Description: ntt(a) computes \hat{f}(k) = \sum_{x} a[x]g^{xk} for all k, where g = \sum_{x} a[x]g^{xk}
\operatorname{root}^{(mod-1)/N}. N must be a power of 2. Useful for convolution modulo spe-
cific nice primes of the form 2^a b + 1, where the convolution result has size
at most 2^a. For arbitrary modulo, see FFTMod. conv(a, b) = c, where
c[x] = \sum a[i]b[x-i]. For manual convolution: NTT the inputs, multiply point-
wise, divide by n, reverse(start+1, end), NTT back. Inputs must be in [0, mod).
Time: \mathcal{O}(N \log N)
"../number-theory/ModPow.h"
typedef vector<11> v1;
template <11 mod, 11 root>
struct NTT {
  void operator()(ll a[], int n) {
    int L = 31 - __builtin_clz(n);
    static v1 rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
      rt.resize(n);
      11 z[] = {1, modpow(root, mod >> s, mod)};
      rep(i, k, 2 * k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
      for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
        11 z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
        a[i + j + k] = ai - z + (z > ai ? mod : 0);
         ai += (ai + z >= mod ? z - mod : z);
const 11 mod = (119 \ll 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 \ll 21 (same root). The last two are > 10^9.
NTT<mod, root> ntt;
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = sz(a) + sz(b) - 1, n = 1 << (32 - _builtin_clz(s));
  11 \text{ inv} = \text{modpow}(n, \text{mod} - 2, \text{mod});
  vl x(a), y(b), out(n);
  x.resize(n); y.resize(n);
  ntt(x.data(), n); ntt(y.data(), n);
  rep(i, 0, n) out[-i & (n-1)] = x[i] * y[i] % mod * inv % mod;
  ntt(out.data(), n);
  return {begin(out), begin(out) + s};
```

```
FastSubsetTransform.h
Description: Transform to a basis with fast convolutions of the form c[z] =
\sum_{z=x\oplus y}a[x]\cdot b[y], where \oplus is one of AND, OR, XOR. The size of a must be
a power of two.
Time: \mathcal{O}(N \log N)
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);
  if (inv) for (int& x : a) x /= sz(a); // XOR only
vi conv(vi a, vi b) {
 FST(a, 0); FST(b, 0);
 rep(i, 0, sz(a)) a[i] *= b[i];
 FST(a, 1); return a;
Number theory (5)
5.1 Modular arithmetic
Description: Operators for modular arithmetic.
template <class T, T mod>
struct mint {
  // static int MOD = mod; for dynamic mod
  mint operator+(mint o) { return mint(v + o.v); }
  mint operator-(mint o) { return mint(v - o.v); }
  mint operator* (mint o) // maybe __int128_t
  { return mint((T) ((ll) v * o.v % mod)); }
  mint operator/(mint o) { return *this * o.inv(); }
    T x, y, g = euclid(v, mod, x, y);
    assert(g == 1); return mint(x);
```

Modular Arithmetic.h

```
"Euclid.h"
```

```
mint(T v = 0) : v(v + ((v < 0) - (v >= mod)) * mod) {}
 mint pow(ll e) {
   mint r(1), x(*this);
   for (; e; e >>= 1, x = x * x)
     if (e & 1) r = r * x;
   return r:
 T val() { return v; }
using mi = mint<int, 998244353>;
```

de7dcf, 23 lines

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

ModPow.h

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

```
Time: \mathcal{O}(\sqrt{m})
                                       c040b8, 11 lines
11 modLog(ll a, ll b, ll m) {
```

```
Divisibility
Description: Prime sieve for generating all primes smaller than LIM.
                                                                           Euclid.h
                                                            6b2912, 20 lines
                                                                           Description: Finds two integers x and y, such that ax + by = \gcd(a, b). If you
                                                                           just need gcd, use the built in \_gcd instead. If a and b are coprime, then x is
                                                                           the inverse of a \pmod{b}.
                                                                           ll euclid(ll a, ll b, ll &x, ll &y) {
                                                                             if (!b) return x = 1, y = 0, a;
  vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1));
                                                                             11 d = euclid(b, a % b, y, x);
                                                                             return y -= a/b * x, d;
    for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;</pre>
                                                                           CRT.h
                                                                           Description: Chinese Remainder Theorem.
                                                                           crt (a, m, b, n) computes x such that x \equiv a \pmod{m}, x \equiv b \pmod{n}. If
                                                                           |a| < m and |b| < n, x will obey 0 \le x < \text{lcm}(m, n). Assumes mn < 2^{62}
                                                                           Time: \log(n)
      for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;</pre>
                                                                           "Euclid.h"
                                                                                                                                        04d93a, 7 lines
                                                                           11 crt(l1 a, l1 m, l1 b, l1 n) {
                                                                             if (n > m) swap(a, b), swap(m, n);
                                                                             ll x, y, q = euclid(m, n, x, y);
                                                                             assert((a - b) % g == 0); // else no solution
                                                                             x = (b - a) % n * x % n / q * m + a;
                                                                             return x < 0 ? x + m*n/q : x;
```

return 1;

l.insert(l.end(), all(r));

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

5.3.1 Bézout's identity

ax + by = d

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

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

```
phiFunction.h
```

```
Description: Euler's \phi function is defined as \phi(n) := \# of positive integers
\leq n that are coprime with n. \phi(1) = 1, p prime \Rightarrow \phi(p^k) = (p-1)p^{k-1},
m, n \text{ 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) = \phi(n) \phi(n)
(p_1-1)p_1^{k_1-1}...(p_r-1)p_r^{k_r-1}. \ \phi(n)=n\cdot \prod_{p\mid n} (1-1/p).
\sum_{d|n} \phi(d) = n, \ \sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1
```

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

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

```
const int LIM = 5000000;
int phi[LIM];
void calculatePhi() {
 rep(i, 0, LIM) phi[i] = i&1 ? i : i/2;
 for (int i = 3; i < LIM; i += 2) if(phi[i] == i)</pre>
    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;</pre>
```

5.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with p, q < N. It will obey |p/q - x| < 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.

```
5.2 Primality
FastEratosthenes.h
Time: LIM=1e9 \approx 1.5s
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
  const int S = (int)round(sqrt(LIM)), R = LIM / 2;
  for (int i = 3; i <= S; i += 2) if (!sieve[i]) {</pre>
    cp.push_back(\{i, i * i / 2\});
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto &[p, idx] : cp)
    rep(i, 0, min(S, R - L))
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  for (int i : pr) isPrime[i] = 1;
  return pr;
MillerRabin.h
Description: Deterministic Miller-Rabin primality test. Guaranteed to work
for numbers up to 7 \cdot 10^{18}; for larger numbers, use Python and extend A ran-
Time: 7 times the complexity of a^b \mod c.
     p = modmul(p, p, n);
```

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

Description: Pollard-rho randomized factorization algorithm. Returns prime

if (modpow(a, (p-1)/2, p) != 1) return -1; if (p % 4 == 3) return modpow(a, (p+1)/4, p); $// a^{(n+3)/8} \text{ or } 2^{(n+3)/8} * 2^{(n-1)/4} \text{ works if } p \% 8 == 5$ **while** (modpow(n, (p - 1) / 2, p) != p - 1) ++n;11 b = modpow(a, s, p), g = modpow(n, s, p); if (m == 0) return min(x, p - x); 11 gs = modpow(g, 1LL << (r - m - 1), p);

f164c2, 24 lines

unordered_map<11, 11> A;

if (e == b % m) return j; **if** (__gcd(m, e) == __gcd(m, b))

return n * i - A[e];

Time: $\log(m)$, with a large constant.

typedef unsigned long long ull;

ull to2 = (to * k + c) / m;

ll modsum(ull to, ll c, ll k, ll m) {

Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

 $x^2 = a \pmod{p}$ (-x gives the other solution).

11 x = modpow(a, (s + 1) / 2, p);

for (m = 0; m < r && t != 1; ++m)

Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}\left(\log p\right)$ for most p

ll modsgrt(ll a, ll p) { // return -1 on fail

ll ret = a * b - M * ull(1.L / M * a * b);

return ret + M * (ret < 0) - M * (ret >= (11)M);

typedef unsigned long long ull;

a % = p; if (a < 0) a += p;

if (a == 0) **return** 0;

11 s = p - 1, n = 2;

while (s % 2 == 0)

++r, s /= 2;

for (;; r = m) {

t = t * t % p;

q = qs * qs % p;x = x * gs % p;

b = b * g % p;

11 t = b;

int r = 0, m;

ull modmul(ull a, ull b, ull M) {

A[e * b % m] = j++;

return -1;

ModSum.h

k %= m; c %= m;

ModSgrt.h

if (!k) return res;

C = ((C % m) + m) % m;

k = ((k % m) + m) % m;

while $(j \le n \& \& (e = f = e * a % m) != b % m)$

rep(i,2,n+2) **if** (A.count(e = e * f % m))

Description: Sums of mod'ed arithmetic progressions.

ull divsum(ull to, ull c, ull k, ull m) {

ull res = k / m * sumsq(to) + c / m * to;

ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);

return to * c + k * sumsq(to) - m * divsum(to, c, k, m);

Description: Calculate $a \cdot b \mod c$ (or $a^b \mod c$) for $0 < a, b < c < 7.2 \cdot 10^{18}$

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t.

modsum(to, c, k, m) = $\sum_{i=0}^{\text{to}-1} (ki+c)\%m$. divsum is similar but for floored

factors of a number, in arbitrary order (e.g. $2299 \rightarrow \{11, 19, 11\}$). **Time:** $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors. "ModMulLL.h", "MillerRabin.h" 8990a6, 19 lines mt19937_64 mt((unsigned) chrono::system_clock::now(). time since epoch().count()); ull pollard(ull n) { ull c = 1, p = 2, q, x = 2, y = 2, t = 0; auto f = [&c,n] (ull x) { return modmul(x, x, n) + c; }; **while** (t ++ % 128 **or** __gcd(p, n) == 1) { **if** (x == y) c = mt() % (n - 1) + 1, y = f(x = 2);**if** ((q = modmul(p, x > y ? x - y : y - x, n))) p = q;x = f(x); y = f(f(y));return __gcd(p, 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);

Time: $O(\log N)$

typedef double d; // for $N \sim 1e7$; long double for $N \sim 1e9$

a = (ll) floor(y), b = min(a, lim),NP = b*P + LP, NQ = b*Q + LQ;

make_pair(NP, NQ) : make_pair(P, Q);

if $(abs(y = 1/(y - (d)a)) > 3*(d)N) {$

finds an exact solution, in which case N can be removed.

return {NP, NQ}; LP = P; P = NP;

LO = O; O = NO;

FracBinarvSearch.h

struct Frac { ll p, q; };

if (f(lo)) return lo; assert(f(hi));

adv += step;

hi.p += lo.p * adv;hi.q += lo.q * adv;dir = !dir;

return dir ? hi : lo;

swap(lo, hi); A = B; B = !!adv;

adv -= step; si = 2;

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

template<class F> Frac fracBS(F f, ll N) { bool dir = 1, A = 1, B = 1;

while (A || B)

11 LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;

11 lim = min(P ? (N-LP) / P : inf, O ? (N-LO) / O : inf),

// If b > a/2, we have a semi-convergent that gives us a

// better approximation; if b = a/2, we *may* have one. // Return {P, Q} here for a more canonical approximation.

return (abs (x - (d) NP / (d) NQ) < abs <math>(x - (d) P / (d) Q)) ?

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

Usage: fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}

Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N)

11 adv = 0, step = 1; // move hi if dir, else lo

Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q}; if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {

for (int si = 0; step; (step *= 2) >>= si) {

pair<ll, ll> approximate(d x, ll N) {

27ab3e, 25 lines

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

5.7 Estimates

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

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

Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

$$\begin{split} & \sum_{d|n} \mu(d) = [n=1] \text{ (very useful)} \\ & g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d) \\ & g(n) = \sum_{1 < m < n} f(\left\lfloor \frac{n}{m} \right\rfloor) \Leftrightarrow f(n) = \sum_{1 < m < n} \mu(m) g(\left\lfloor \frac{n}{m} \right\rfloor) \end{split}$$

Combinatorial (6)

Permutations

6.1.1 Factorial

| т.т | 10 | icuoi i | aı | | | | | |
|-----|----|---------|------|--------|---------|----------|---------|---------|
| | n | 1 2 3 | 4 | 5 6 | 7 | 8 | 9 | 10 |
| | n! | 1 2 6 | 24 1 | 20 720 | 5040 | 40320 3 | 62880 3 | 628800 |
| | n | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| | n! | 4.0e7 | 4.8e | 86.2e9 | 8.7e1 | 0.1.3e12 | 2.1e13 | 3.6e14 |
| | n | 20 | 25 | 30 | 40 5 | 0 100 | 150 | 171 |
| | n! | 2e18 | 2e25 | 3e32 8 | 6e47 3e | 64 9e15 | 7 6e262 | >DBL_MA |

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table. Time: $\mathcal{O}(n)$

```
int permToInt(vi& v) {
 int use = 0, i = 0, r = 0;
  for (int x:v) r = r * ++i + \underline{\quad} builtin_popcount (use & -(1<<x)),
                                             // (note: minus, not \sim!)
```

6.1.2 Cycles

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

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

6.1.3 Derangements Permutations of a set such that none of the elements appear in

their original position.

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

6.1.4 Burnside's lemma

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

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

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

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

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

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.

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

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

6.2.2 Lucas, $11235711152230627 \sim 2e5 \sim 2e8$

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{6.2.3} \equiv \prod_{k=0}^{k} \binom{n_i}{m_i} \pmod{p}.$

multinomial.h

return c;

c = c * ++m / (j+1);

Description: Computes $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$ 11 c = 1, m = v.empty() ? 1 : v[0];rep(i,1,sz(v)) rep(j,0,v[i])

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

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

 $1\,000\,000.$

NTU nontoi

BellmanFord FloydWarshall TopoSort Dinic

int n = sz(m);

void floydWarshall(vector<vector<ll>>& m) {

rep(i, 0, n) m[i][i] = min(m[i][i], OLL);

m[i][j] = min(m[i][j], newDist);

if (m[i][k] != inf && m[k][j] != inf) {

auto newDist = max(m[i][k] + m[k][j], -inf);

rep(k, 0, n) if (m[k][k] < 0) rep(i, 0, n) rep(j, 0, n)

rep(k, 0, n) rep(i, 0, n) rep(j, 0, n)

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:

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

Euler-Maclaurin formula for infinite sums:

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

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

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) = $1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$ For p prime,

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

```
6.3.6 Labeled unrooted trees
```

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

6.3.7 Catalan numbers

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

$$C_0 = 1, \ C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \ C_{n+1} = \sum_{n=1}^{\infty} C_n C_{n-n}$$

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

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

Graph (7)

7.1 Fundamentals

BellmanFord.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: $\mathcal{O}(VE)$

```
830a8f, 23 lines
const 11 inf = LLONG MAX:
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};</pre>
struct Node { ll dist = inf; int prev = -1; };
void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
 nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });</pre>
  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
  rep(i,0,lim) for (Ed ed : eds) {
   Node cur = nodes[ed.a], &dest = nodes[ed.b];
   if (abs(cur.dist) == inf) continue;
   11 d = cur.dist + ed.w;
   if (d < dest.dist) {</pre>
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
  rep(i,0,lim) for (Ed e : eds) {
   if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
```

FlovdWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] = \inf$ 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-Time: $\mathcal{O}\left(N^3\right)$

```
const ll inf = 1LL << 62;</pre>
```

```
TopoSort.h
Description: Topological sorting. Given is an oriented graph. Output is an
ordering of vertices, such that there are edges only from left to right. If there
are cycles, the returned list will have size smaller than n – nodes reachable from
```

if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;

11

cycles will not be returned. Time: $\mathcal{O}(|V| + |E|)$ 66a137, 14 lines vi topoSort(const vector<vi>& gr) { vi indeg(sz(gr)), ret;

```
for (auto& li : gr) for (int x : li) indeg[x]++;
queue < int > q; // use priority_queue for lexic. largest ans.
rep(i, 0, sz(gr)) if (indeg[i] == 0) q.push(i);
while (!q.empty()) {
  int i = q.front(); // top() for priority queue
  ret.push_back(i);
  for (int x : gr[i])
    if (--indeq[x] == 0) q.push(x);
return ret;
```

7.2 Network flow

Dinic.h

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

```
struct Dinic {
 struct E {
   11 flow() { return max(oc - c, 011); } // if you need flows
 int n;
 vi le, it, q;
  vector<vector<E>> adj;
 Dinic(int n): n(n), le(n), it(n), g(n), adj(n) {}
 void add(int u, int v, 11 c, 11 rc = 0) {
   adj[u].push_back({v, sz(adj[v]), c, c});
   adj[v].push\_back({u, sz(adj[u]) - 1, rc, rc});
 11 dfs(int u, int t, 11 f) {
   if (u == t || !f) return f;
   for (int &i = it[u]; i < sz(adj[u]); ++i) {</pre>
     auto &[v, r, c, oc] = adj[u][i];
     if (le[v] == le[u] + 1)
        if (ll p = dfs(v, t, min(f, c))) {
          c -= p, adj[v][r].c += p;
          return p:
   return 0;
 11 flow(int s, int t) {
   11 res = 0; q[0] = s;
    rep(L,0,31) do { // 'rep(L,30,31)' maybe faster for random
        data
      le = it = vi(sz(q));
```

int qi = 0, qe = le[s] = 1;

while (qi < qe && !le[t]) {

q[qe++] = v, le[v] = le[u] + 1;

while (ll p = dfs(s, t, LLONG_MAX)) res += p;

for (auto [v, r, c, oc]: adj[u]) if (!le[v] && c >> (30 -

int u = q[qi++];

} while (le[t]);

```
return res;
 bool inSCut(int u) { return le[u] != 0; }
PushRelabel.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.
Time: \mathcal{O}\left(V^2\sqrt{E}\right)
                                                        0ae1d4, 48 lines
struct PushRelabel {
 struct Edge {
   int dest, back;
   11 f. c:
 vector<vector<Edge>> g;
 vector<11> ec;
 vector<Edge*> cur;
 vector<vi> hs; vi H;
 PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
 void addEdge(int s, int t, ll cap, ll rcap=0) {
   if (s == t) return;
   g[s].push_back({t, sz(g[t]), 0, cap});
   g[t].push_back({s, sz(g[s])-1, 0, rcap});
 void addFlow(Edge& e, ll f) {
   Edge &back = g[e.dest][e.back];
   if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
   e.f += f; e.c -= f; ec[e.dest] += f;
   back.f -= f; back.c += f; ec[back.dest] -= f;
 11 calc(int s, int t) {
   int v = sz(q); H[s] = v; ec[t] = 1;
   vi co(2*v); co[0] = v-1;
   rep(i,0,v) cur[i] = g[i].data();
   for (Edge& e : q[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() + sz(g[u])) {
         for (Edge& 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)</pre>
           rep(i, 0, v) if (hi < H[i] && H[i] < v)
             --co[H[i]], H[i] = v + 1;
         hi = H[u];
       } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
         addFlow(*cur[u], min(ec[u], cur[u]->c));
       else ++cur[u];
 bool leftOfMinCut(int a) { return H[a] >= sz(g); }
```

```
MinCostMaxFlow.h
Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges
are not. If costs can be negative, call setpi before maxflow, but note that neg-
ative cost cycles are not supported. To obtain the actual flow, look at positive
values only.
Time: Approximately \mathcal{O}(E^2)
                                                         fe85cc, 81 lines
#include <bits/extc++.h>
const 11 INF = numeric limits<11>::max() / 4;
typedef vector<ll> VL;
struct MCMF {
  int N;
  vector<vi> ed, red;
  vector<VL> cap, flow, cost;
  vi seen:
  VL dist, pi;
  vector<pii> par;
  MCMF (int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}
  void addEdge(int from, int to, 11 cap, 11 cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;
    __gnu_pbds::priority_queue<pair<11, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({0, s});
    auto relax = [&](int i, ll cap, ll cost, int dir) {
      11 val = di - pi[i] + cost;
      if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i], i});
        else q.modify(its[i], {-dist[i], i});
    };
    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (int i : ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
      for (int i : red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0);
    rep(i, 0, N) pi[i] = min(pi[i] + dist[i], INF);
  pair<11, 11> maxflow(int s, int t) {
    11 totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      11 fl = INF;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x][p]);
      totflow += fl;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        if (r) flow[p][x] += fl;
        else flow[x][p] -= fl;
```

```
rep(i, 0, N) rep(j, 0, N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (int to : ed[i]) if (cap[i][to])
          if ((v = pi[i] + cost[i][to]) < pi[to])</pre>
            pi[to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
};
MinCut.h
Description: After running max-flow, the left side of a min-cut from s to t
is given by all vertices reachable from s, only traversing edges with positive
residual capacity.
GlobalMinCut.h
Description: Find a global minimum cut in an undirected graph, as repre-
sented by an adjacency matrix.
Time: O(V^3)
                                                          8b0<u>e19, 21 lines</u>
pair<int, vi> globalMinCut(vector<vi> mat) {
 pair<int, vi> best = {INT_MAX, {}};
 int n = sz(mat);
 vector<vi> co(n);
 rep(i,0,n) co[i] = {i};
 rep(ph,1,n) {
   vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) \rightarrow O(E log V) with prio. queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i, 0, n) w[i] += mat[t][i];
    best = min(best, \{w[t] - mat[t][t], co[t]\});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i, 0, n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  return best;
GomorvHu.h
Description: Given a list of edges representing an undirected flow graph, re-
turns edges of the Gomory-Hu tree. The max flow between any pair of vertices
is given by minimum edge weight along the Gomory-Hu tree path.
Time: \mathcal{O}(V) Flow Computations
"PushRelabel.h"
                                                          0418b3, 13 lines
typedef array<11, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
 vector<Edge> tree;
 vi par(N);
```

rep(i,1,N) {

return tree;

rep(j,i+1,N)

PushRelabel D(N); // Dinic also works

for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);

if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;

tree.push_back({i, par[i], D.calc(i, par[i])});

7.3 Matching

```
hopcroftKarp.h
Description: Fast bipartite matching algorithm. Graph g should be a list of
```

NTU nontoi

neighbors of the left partition, and btoa should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. btoa[i] will be the match for vertex i on the right side, or -1 if it's not matched. Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);

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

```
f612e4, 42 lines
bool dfs(int a, int L, vector<vi>& q, vi& btoa, vi& A, vi& B) {
 if (A[a] != L) return 0;
 A[a] = -1;
 for (int b : q[a]) if (B[b] == L + 1) {
   B[b] = 0;
   if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
     return btoa[b] = a, 1;
 return 0:
int hopcroftKarp(vector<vi>& g, vi& btoa) {
 int res = 0;
 vi A(g.size()), B(btoa.size()), cur, next;
 for (;;) {
   fill(all(A), 0);
   fill(all(B), 0);
   cur.clear();
   for (int a : btoa) if (a != -1) A[a] = -1;
```

```
for (int lay = 1;; lay++) {
 bool islast = 0;
 next.clear();
  for (int a : cur) for (int b : g[a]) {
    if (btoa[b] == -1) {
     B[b] = lay;
     islast = 1;
    else if (btoa[b] != a && !B[b]) {
     B[b] = lay;
     next.push_back(btoa[b]);
  if (islast) break;
 if (next.empty()) return res;
  for (int a : next) A[a] = lay;
  cur.swap(next);
rep(a, 0, sz(q))
 res += dfs(a, 0, g, btoa, A, B);
```

rep(a, 0, sz(q)) **if**(A[a] == 0) cur.push_back(a);

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and btoa should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. btoa[i] will

```
be the match for vertex i on the right side, or -1 if it's not matched.
Usage: vi btoa(m, -1); dfsMatching(g, btoa);
Time: \mathcal{O}\left(VE\right)
                                                           522b98, 22 lines
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
if (btoa[j] == -1) return 1;
 vis[j] = 1; int di = btoa[j];
 for (int e : g[di])
   if (!vis[e] && find(e, g, btoa, vis)) {
     btoa[e] = di;
     return 1;
 return 0;
int dfsMatching(vector<vi>& g, vi& btoa) {
```

```
vi vis:
 rep(i, 0, sz(g)) {
   vis.assign(sz(btoa), 0);
   for (int j : q[i])
     if (find(j, g, btoa, vis)) {
       btoa[j] = i;
       break;
 return sz(btoa) - (int)count(all(btoa), -1);
MinimumVertexCover.h
```

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

mum independent set. "DFSMatching.h" vi cover(vector<vi>& q, int n, int m) {

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

WeightedMatching.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. Requires $N \leq M$. Time: $\mathcal{O}\left(N^2M\right)$ 1e0fe9, 31 lines pair<int, vi> hungarian(const vector<vi> &a) { if (a.empty()) return {0, {}};

```
int n = sz(a) + 1, m = sz(a[0]) + 1;
vi u(n), v(m), p(m), ans(n-1);
rep(i,1,n) {
 p[0] = i;
  int j0 = 0; // add "dummy" worker 0
  vi dist(m, INT_MAX), pre(m, -1);
  vector<bool> done(m + 1);
  do { // diikstra
    done[j0] = true;
    int i0 = p[j0], j1, delta = INT_MAX;
    rep(j,1,m) if (!done[j]) {
      auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
      if (cur < dist[j]) dist[j] = cur, pre[j] = j0;</pre>
      if (dist[j] < delta) delta = dist[j], j1 = j;</pre>
    rep(j,0,m) {
      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];
```

```
rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
 return {-v[0], ans}; // min cost
GeneralMatching.h
Description: Matching for general graphs. Fails with probability N/mod.
Time: \mathcal{O}(N^3)
vector<pii> generalMatching(int N, vector<pii>& ed) {
  vector<vector<ll>> mat(N, vector<ll>(N)), A;
  for (pii pa : ed) {
   int a = pa.first, b = pa.second, r = rand() % mod;
   mat[a][b] = r, mat[b][a] = (mod - r) % mod;
  int r = matInv(A = mat), M = 2*N - r, fi, fj;
  assert (r % 2 == 0);
 if (M != N) do {
   mat.resize(M, vector<ll>(M));
   rep(i,0,N) {
     mat[i].resize(M);
     rep(j,N,M) {
       int r = rand() % mod;
        mat[i][j] = r, mat[j][i] = (mod - r) % mod;
 } while (matInv(A = mat) != M);
  vi has(M, 1); vector<pii> ret;
  rep(it,0,M/2) {
   rep(i,0,M) if (has[i])
      rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
        fi = i; fj = j; goto done;
    } assert(0); done:
    if (fj < N) ret.emplace_back(fi, fj);</pre>
    has[fi] = has[fj] = 0;
    rep(sw,0,2) {
      ll a = modpow(A[fi][fj], mod-2);
      rep(i,0,M) if (has[i] && A[i][fj]) {
        11 b = A[i][fj] * a % mod;
        rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
```

p[j0] = p[j1], j0 = j1;

7.4 DFS algorithms

swap(fi,fj);

return ret;

SCC.h

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

```
Usage: scc(graph, [&](vi& v) { ... }) visits all components
in reverse topological order. comp[i] holds the component
index of a node (a component only has edges to components with
lower index). ncomps will contain the number of components.
Time: \mathcal{O}(E+V)
                                                        76b5c9, 24 lines
vi val, comp, z, cont;
int Time, ncomps;
template < class G, class F > int dfs (int j, G& g, F& f) {
```

int low = val[j] = ++Time, x; z.push_back(j);

low = min(low, val[e] ?: dfs(e,g,f));

for (auto e : g[j]) if (comp[e] < 0)</pre>

```
Description: Eulerian undirected/directed path/cycle algorithm. Input
```

```
x = z.back(); z.pop_back();
     comp[x] = ncomps;
     cont.push_back(x);
   } while (x != j);
   f(cont); cont.clear();
   ncomps++;
 return val[j] = low;
template < class G, class F > void scc(G& q, F f) {
 int n = sz(g);
 val.assign(n, 0); comp.assign(n, -1);
Time = ncomps = 0;
 rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
BiconnectedComponents.h
Description: Finds all biconnected components in an undirected graph, and
runs a callback for the edges in each. In a biconnected component there are at
least two distinct paths between any two nodes. Note that a node can be in
several components. An edge which is not in a component is a bridge, i.e., not
part of any cycle.
Usage: int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) \{...\});
Time: \mathcal{O}\left(E+V\right)
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
int me = num[at] = ++Time, e, y, top = me;
 for (auto pa : ed[at]) if (pa.second != par) {
   tie(y, e) = pa;
   if (num[y]) {
     top = min(top, num[y]);
     if (num[y] < me)
       st.push_back(e);
   } else {
     int si = sz(st);
     int up = dfs(y, e, f);
     top = min(top, up);
     if (up == me) {
       st.push back(e);
       f(vi(st.begin() + si, st.end()));
       st.resize(si);
     else if (up < me) st.push_back(e);</pre>
     else { /* e is a bridge */ }
 return top;
template<class F>
void bicomps(F f) {
num.assign(sz(ed), 0);
rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
Description: Calculates a valid assignment to boolean variables a,
```

b, c,... to a 2-SAT problem, so that an expression of the type

fiable. Negated variables are represented by bit-inversions ($\sim x$).

if (low == val[j]) {

```
ts.atMostOne(\{0, \sim 1, 2\}); // <= 1 of vars 0, \sim 1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars
Time: \mathcal{O}(N+E), where N is the number of boolean variables, and E is the
number of clauses.
                                                         5f9706, 56 lines
struct TwoSat {
 int N:
  vector<vi> gr:
  vi values; // 0 = false, 1 = true
  TwoSat(int n = 0) : N(n), qr(2*n) {}
  int addVar() { // (optional)
   gr.emplace_back();
    gr.emplace_back();
    return N++;
  void either(int f, int j) {
   f = \max(2 * f, -1 - 2 * f);
    j = \max(2*j, -1-2*j);
   gr[f].push_back(j^1);
   gr[j].push_back(f^1);
  void setValue(int x) { either(x, x); }
  void atMostOne(const vi& li) { // (optional)
   if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = \sim next;
    either(cur, ~li[1]);
  vi val, comp, z; int time = 0;
  int dfs(int i) {
   int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
     low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
     x = z.back(); z.pop_back();
      comp[x] = low;
     if (values[x>>1] == -1)
       values[x>>1] = x&1;
   } while (x != i);
    return val[i] = low;
  bool solve() {
   values.assign(N, -1);
   val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1:
};
```

Usage: TwoSat ts(number of boolean variables);

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

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

```
if (it == end) { ret.push_back(x); s.pop_back(); continue; }
   tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
  for (int x : D) if (x < 0 \mid | sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
DominatorTree.h
Description: Build dominator tree of graph adj. 0 base. Root's dominator is
itself, and unreachable nodes have -1 as dominator.
Time: \mathcal{O}\left(M\log N\right)
                                                         ac6fad, 41 lines
struct DominatorTree {
 vi p, semi, ord, dom, f, val;
  vector<vi> adj, pre, bkt;
  void dfs(int u) {
   semi[u] = sz(ord);
   ord.push_back(u);
    for (int v : adj[u]) {
      if (semi[v] == -1) p[v] = u, dfs(v);
      pre[v].push_back(u);
  int eval(int u, int t = 0) {
   if (f[u] == -1) return t ? -1 : u;
    if (int p = eval(f[u], 1); p != -1) {
      if (semi[val[f[u]]] < semi[val[u]])</pre>
        val[u] = val[f[u]];
      f[u] = p;
      return t ? p : val[u];
    } return t ? f[u] : val[u];
  DominatorTree(int N, const vector<vi>& adj, int r): p(N, -1),
  semi(p), dom(p), f(p), val(N), adj(adj), pre(N), bkt(N) {
   iota(all(val), 0);
    dfs(r);
    for (int i = sz(ord); --i; ) {
      int u = ord[i];
      for (int v : pre[u])
        semi[u] = min(semi[u], semi[eval(v)]);
      bkt[ord[semi[u]]].push_back(u);
      f[u] = p[u];
      for (int v : bkt[p[u]]) {
        int w = eval(v);
        dom[v] = semi[w] < semi[v] ? w : p[u];
      bkt[p[u]].clear();
    dom[r] = r;
    for (int u: ord) if (dom[u] != ord[semi[u]])
      dom[u] = dom[dom[u]];
};
```

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/-

vi eulerWalk (vector<vector<pii>>>& gr, int nedges, int src=0) {

int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);

path exists. To get edge indices back, add .second to s and ret.

D[src]++; // to allow Euler paths, not just cycles

vi D(n), its(n), eu(nedges), ret, s = {src};

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

int n = sz(qr);

while (!s.empty()) {

7.5 Coloring

vi edgeColoring(int N, vector<pii> eds) {

```
EdgeColoring.h
```

Description: Given a simple, undirected graph with max degree D, computes a (D+1)-coloring of the edges such that no neighboring edges share a color. (Dcoloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.) Time: $\mathcal{O}(NM)$ e210e2, 31 lines

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

7.6 Heuristics

MaximalCliques.h

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

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

b0d5b1, 12 lines

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

MaximumClique.h

struct Maxclique {

double limit=0.025, pk=0;

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

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

```
Runs faster for sparse graphs.
```

```
typedef vector<bitset<200>> vb;
```

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

MaximumIndependentSet.h

struct Vertex { int i, d=0; }; typedef vector<Vertex> vv;

vb e;

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

7.7 Trees

BinaryLifting.h

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

```
Time: construction \mathcal{O}(N \log N), queries \mathcal{O}(\log N)
                                                            bfce85, 25 lines
vector<vi> treeJump(vi& P){
 int on = 1, d = 1;
 while (on < sz(P)) on *= 2, d++;
 vector<vi> jmp(d, P);
  rep(i,1,d) rep(j,0,sz(P))
    jmp[i][j] = jmp[i-1][jmp[i-1][j]];
  return jmp;
int jmp(vector<vi>& tbl, int nod, int steps){
 rep(i,0,sz(tbl))
   if(steps&(1<<i)) nod = tbl[i][nod];
 return nod;
```

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

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most |S|-1) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself. Time: $\mathcal{O}(|S| \log |S|)$ 9775a0, 21 lines

```
"LCA.h"
```

typedef vector<pair<int, int>> vpi;

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

void dfsHld(int v) {

HLD.h **Description:** Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0. Time: $\mathcal{O}\left((\log N)^2\right)$

```
"../data-structures/LazySegmentTree.h"
                                                        cbb1fc, 46 lines
template <bool VALS EDGES> struct HLD {
 int N, tim = 0;
  vector<vi> adi:
  vi par, siz, depth, rt, pos;
  SGT<Val, Tag> tree;
  HLD(vector<vi> adj_)
   : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1), depth(N),
      rt(N), pos(N), tree(N) { dfsSz(0); dfsHld(0); }
  void dfsSz(int v) {
   if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
    for (int& u : adj[v]) {
      par[u] = v, depth[u] = depth[v] + 1;
      dfsSz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
```

```
pos[v] = tim++;
   for (int u : adj[v]) {
     rt[u] = (u == adj[v][0] ? rt[v] : u);
     dfsHld(u);
 template <class B> void process(int u, int v, B op) {
   for (; rt[u] != rt[v]; v = par[rt[v]]) {
     if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
     op(pos[rt[v]], pos[v] + 1);
   if (depth[u] > depth[v]) swap(u, v);
   op(pos[u] + VALS_EDGES, pos[v] + 1);
 void modifyPath(int u, int v, int val) {
   process(u, v, [&](int 1, int r) { tree.modify(1, r, val); });
 int queryPath(int u, int v) { // Modify depending on problem
   int res = -1e9;
   process(u, v, [&](int l, int r) {
       res = max(res, tree.query(1, r).v);
   return res;
 int querySubtree(int v) { // modifySubtree is similar
   return tree.query(pos[v] + VALS_EDGES, pos[v] + siz[v]).v;
};
LinkCutTree.h
Description: Represents a forest of unrooted trees. You can add and remove
edges (as long as the result is still a forest), and check whether two nodes are
in the same tree.
Time: All operations take amortized O(\log N).
struct Node { // Splay tree. Root's pp contains tree's parent.
Node *p = 0, *pp = 0, *c[2];
 bool flip = 0;
 Node() { c[0] = c[1] = 0; fix(); }
 void fix() {
   if (c[0]) c[0]->p = this;
   if (c[1]) c[1]->p = this;
   // (+ update sum of subtree elements etc. if wanted)
 void pushFlip() {
   if (!flip) return;
   flip = 0; swap(c[0], c[1]);
   if (c[0]) c[0]->flip ^= 1;
   if (c[1]) c[1]->flip ^= 1;
 int up() { return p ? p->c[1] == this : -1; }
 void rot(int i, int b) {
   int h = i ^ b;
   Node *x = c[i], *y = b == 2 ? x : x -> c[h], *z = b ? y : x;
   if ((y->p = p)) p->c[up()] = y;
   c[i] = z -> c[i ^ 1];
   if (b < 2) {
     x->c[h] = y->c[h ^ 1];
     z \rightarrow c[h ^1] = b ? x : this;
   y \rightarrow c[i ^1] = b ? this : x;
   fix(); x->fix(); y->fix();
   if (p) p->fix();
   swap(pp, y->pp);
 void splay() {
   for (pushFlip(); p; ) {
     if (p->p) p->p->pushFlip();
     p->pushFlip(); pushFlip();
     int c1 = up(), c2 = p->up();
```

if (c2 == -1) p->rot(c1, 2);

```
else p->p->rot(c2, c1 != c2);
 Node* first() {
   pushFlip();
   return c[0] ? c[0]->first() : (splay(), this);
struct LinkCut {
 vector<Node> node:
 LinkCut(int N) : node(N) {}
 void link(int u, int v) { // add \ an \ edge \ (u, \ v)
    assert(!connected(u, v));
   makeRoot (&node[u]);
    node[u].pp = &node[v];
  void cut (int u, int v) { // remove an edge (u, v)
   Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x-pp ?: x-c[0]));
    if (x->pp) x->pp = 0;
      x->c[0] = top->p = 0;
      x \rightarrow fix();
  bool connected (int u, int v) { // are u, v in the same tree?
   Node * nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  void makeRoot(Node* u) {
    access (u):
    u->splay();
    if(u->c[0]) {
      u - > c[0] - > p = 0;
      u - c[0] - flip ^= 1;
      u - > c[0] - > pp = u;
      u - > c[0] = 0;
      u \rightarrow fix();
   void push_down(Node* u) {
        u->pushFlip();
        if(u->c[0])push_down(u->c[0]);
        if (u->c[1]) push_down (u->c[1]);
 Node* access(Node* u) {
   u->splay();
        if(u->c[1]){
            u - > c[1] - > p = 0;
            u -> c[1] -> pp = u;
            u -> c[1] = 0;
            u \rightarrow fix();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp - c[1] - p = 0; pp - c[1] - pp = pp; 
      pp->c[1] = u; pp->fix(); u = pp;
   return u;
};
DirectedMST.h
```

```
Edge key;
 Node *1, *r;
  ll delta;
  void prop() {
   key.w += delta;
   if (1) 1->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->1, (a->r = merge(b, a->r)));
  return a;
void pop(Node*\& a) { a->prop(); a = merge(a->1, a->r); }
pair<11, vi> dmst(int n, int r, vector<Edge>& g) {
 RollbackUF uf(n);
 vector<Node*> heap(n);
  for (Edge e : q) heap[e.b] = merge(heap[e.b], new Node{e});
 11 \text{ res} = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, \{-1,-1\}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  rep(s,0,n) {
   int u = s, qi = 0, w;
    while (seen[u] < 0) {
     if (!heap[u]) return {-1,{}};
     Edge e = heap[u] - > top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
       Node * cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
    rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
  for (auto& [u,t,comp] : 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;
  rep(i,0,n) par[i] = in[i].a;
  return {res, par};
```

struct Edge { int a, b; ll w; };

struct Node

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1. Time: $\mathcal{O}\left(E\log V\right)$

"../data-structures/UnionFindRollback.h"

39e620, 60 lines

7.8 Math

7.8.1 Number of Spanning Trees

Create an $N \times N$ matrix mat, and for each edge $a \to b \in G$, do

mat[a][b]--, mat[b][b]++(and mat[b][a]--, mat [a] [a] ++ if G is undirected). Remove the ith row and column

and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

7.8.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 > \cdots > d_n$ exists iff

 $d_1 + \cdots + d_n$ is even and for every $k = 1 \dots n$,

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

Geometry (8)

8.1 Geometric primitives Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.) 47ec0a, 29 lines

```
// for double: (x > eps) - (x < -eps)
template <class T> int sgn(T x) \{ return (x > 0) - (x < 0); \}
template<class T>
struct Point {
 typedef Point P;
 T x, v;
 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); }</pre>
 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 a, P b) const { return (a-*this).cross(b-*this); }

P unit() const { return *this/dist(); } // makes dist()=1

P perp() const { return P(-y, x); } // rotates +90 degrees

// returns point rotated 'a' radians ccw around the origin

double dist() const { return sqrt((double)dist2()); }

return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }

return os << "(" << p.x << "," << p.y << ")"; }

friend ostream& operator<<(ostream& os, P p) {</pre>

LineDistance.h

Description:

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

T cross(P p) const { return x*p.y - y*p.x; }

double angle() const { return atan2(y, x); }

P normal() const { return perp().unit(); }

T dist2() const { return x*x + y*y; }

P rotate(double a) const {

// angle to x-axis in interval [-pi, pi]

f6bf6b, 3 lines

```
SideOf.h
template<class P>
double lineDist(const P& a, const P& b, const P& p)
{ return (double) (b-a).cross(p-a)/(b-a).dist(); }
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, e^2\}
(0,0)} is returned. The wrong position will be returned if P
is Point<|l> and the intersection point does not have integer
coordinates. Products of three coordinates are used in inter- \(^{\sigma}\)
mediate 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;</pre>
"Point.h"
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};
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;
                                                                              const P& q0, const P& q1, const P& r) {
                                                            7b886d, 6 lines
                                                                            P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
template <class P>
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d, max(.0, (p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
SegmentIntersection.h
Description:
If a unique intersection point between the line segments going
from s1 to e1 and from s2 to e2 exists then it is returned.
                                                                          P lineProj(P a, P b, P p, bool refl=false) {
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<|l> and the in-
tersection point does not have integer coordinates. Products
of three coordinates are used in intermediate steps so watch
                                                                           Angle.h
out for overflow if using int or long long.
Usage: vector<P> inter = seqInter(s1,e1,s2,e2);
if (sz(inter) == 1)
cout << "segments intersect at " << inter[0] << endl;</pre>
"Point.h", "OnSegment.h"
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 (sqn(oa) * sqn(ob) < 0 && sqn(oc) * sqn(od) < 0)
    return { (a * ob - b * oa) / (ob - oa) };
  set <P> s:
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
                                                                            Angle t90() const { return \{-y, x, t + (half() \&\& x >= 0)\}; \}
                                                                            Angle t180() const { return {-x, -y, t + half()}; }
```

Description: Returns where p is as seen from s towards e. $1/0/-1 \Leftrightarrow \text{left/on}$ 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;

```
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 1 = (e-s).dist()*eps;
 return (a > 1) - (a < -1);
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>. template < class P > bool on Segment (P s, P e, P p) { return $sgn(p.cross(s, e)) == 0 && sgn((s-p).dot(e-p)) <= 0; }$

LinearTransformation.h Description:

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

typedef Point < double > P; P linearTransformation(const P& p0, const P& p1,

```
return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
LineProjectionReflection.h
```

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. 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. template<class P>

```
P v = b - a;
return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
```

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
struct Angle {
 int x, y;
 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 t360() const { return {x, y, t + 1}; }
bool operator<(Angle a, Angle b) {</pre>
 // add a.dist2() and b.dist2() to also compare distances
 return make_tuple(a.t, a.half(), a.y * (ll)b.x) <</pre>
         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;</pre>
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
 int tu = b.t - a.t; a.t = b.t;
 return \{a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)\};
8.2 Circles
CircleIntersection.h
Description: Computes the pair of points at which two circles intersect. Re-
turns false in case of no intersection.
                                                          70ff65, 10 lines
typedef Point < double > P;
vector<P> circleInter(P a,P b,double r1,double r2) {
if (a == b) { assert(r1 != r2); return {}; }
P \text{ 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 {};
 P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
 return {mid + per, mid - per};
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"
                                                          b2c9e<u>1</u>, 13 lines
typedef Point <double > 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 {};</pre>
 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;
Description: Finds the intersection between a circle and a line. Returns a vec-
tor of either 0, 1, or 2 intersection points. P is intended to be Point < double >.
```

typedef Point < double > P;

```
vector<P> circleLine(P c, double r, P a, P b) {
 P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
 double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
 if (h2 < 0) return {};
 if (h2 == 0) return {p};
 P h = ab.unit() * sqrt(h2);
 return {p - h, p + h};
CirclePolygonIntersection.h
Description: Returns the area of the intersection of a circle with a ccw poly-
gon.
Time: \mathcal{O}(n)
"../../content/geometry/Point.h"
                                                         a1ee63, 19 lines
typedef Point < double > P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
 auto tri = [&](P p, P q) {
   auto r2 = r * r / 2;
   Pd = 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;</pre>
   auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
   if (t < 0 || 1 <= s) return arg(p, q) * r2;</pre>
   Pu = p + d * s, v = p + d * t;
   return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i, 0, sz(ps))
   sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
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.

```
1caa3a, 9 lines
```

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

MinimumEnclosingCircle.h

"Point.h"

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

```
"Circumcircle.h"
                                                        09dd0a, 1<u>7 lines</u>
pair<P, double> mec(vector<P> ps) {
 shuffle(all(ps), mt19937(time(0)));
 P \circ = ps[0];
 double r = 0, EPS = 1 + 1e-8;
 rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
   o = ps[i], r = 0;
   rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
     o = (ps[i] + ps[j]) / 2;
     r = (o - ps[i]).dist();
     rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
```

```
return {o, r};
```

8.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow. **Usage:** $vector < P > v = \{P\{4,4\}, P\{1,2\}, P\{2,1\}\};$

```
bool in = inPolygon(v, P\{3, 3\}, false);
Time: \mathcal{O}(n)
```

```
"Point.h", "OnSegment.h", "SegmentDistance.h"
                                                         2bf504, 11 lines
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
 int cnt = 0, n = sz(p);
 rep(i,0,n) {
   P q = p[(i + 1) % n];
   if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) \le eps) return !strict;
   cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  return cnt;
```

PolygonArea.h

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

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

PolygonCenter.h

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

```
"Point.h"
                                                         9706dc, 9 lines
typedef Point < double > P;
P polygonCenter(const vector<P>& v) {
 P res(0, 0); double A = 0;
 for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
   res = res + (v[i] + v[j]) * v[j].cross(v[i]);
   A += v[j].cross(v[i]);
  return res / A / 3;
```

PolygonCut.h

if (side)

res.push_back(cur);

Description:

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

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

```
"Point.h", "LineIntersection.h"
                                                          f2b7d4, 13 lines
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
 vector<P> res;
 rep(i,0,sz(poly)) {
   P cur = poly[i], prev = i ? poly[i-1] : poly.back();
   bool side = s.cross(e, cur) < 0;</pre>
    if (side != (s.cross(e, prev) < 0))</pre>
      res.push_back(lineInter(s, e, cur, prev).second);
```

```
19
```

```
PolygonUnion.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. (Ep-
silon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)
Time: \mathcal{O}(N^2), where N is the total number of points
"Point.h", "SideOf.h"
typedef Point <double > P;
double rat(P a, P b) { return sqn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
 double ret = 0;
 rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
  P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
   vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
   rep(j,0,sz(poly)) if (i != j) {
     rep(u,0,sz(polv[i])) {
       P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
       int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
         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 && sqn((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(all(segs));
   for (auto& s : seqs) s.first = min(max(s.first, 0.0), 1.0);
   double sum = 0;
   int cnt = segs[0].second;
   rep(j,1,sz(segs)) {
     if (!cnt) sum += seqs[j].first - seqs[j - 1].first;
     cnt += segs[j].second;
   ret += A.cross(B) * sum;
 return ret / 2;
8.4 Convex Hull
```

ConvexHull.h

"Point.h"

Description:

return res;

Returns a vector of the points of the convex hull in counterclockwise order. Points on the edge of the hull between two other points are not considered part of the hull. Time: $\mathcal{O}(n \log n)$



b50acd, 13 lines

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

${ m HullDiameter.h}$

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

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

"Point.h" 10aada, 11 lines

```
template <class P>
array<P, 2> hullDiameter(vector<P> S) {
 int n = sz(S), j = n < 2 ? 0 : 1;
 pair<11, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j) for (;; j = (j + 1) % n) {
   res = \max(\text{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 or-

der, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included. Time: $\mathcal{O}(\log N)$

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

LineHullIntersection.h

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

```
Time: \mathcal{O}(\log n)
"Point.h"
                                                         7cf45b, 39 lines
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
   int m = (lo + hi) / 2;
   if (extr(m)) return m;
   int 1s = cmp(1o + 1, 1o), ms = cmp(m + 1, m);
    (ls < ms \mid | (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
 return lo;
#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
 int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
 if (cmpL(endA) < 0 \mid \mid cmpL(endB) > 0)
   return {-1, -1};
  array<int, 2> res;
  rep(i, 0, 2) {
   int lo = endB, hi = endA, n = sz(poly);
```

while ((lo + 1) % n != hi) {

```
(cmpL(m) == cmpL(endB) ? lo : hi) = m;
 res[i] = (lo + !cmpL(hi)) % n;
  swap (endA, endB);
if (res[0] == res[1]) return {res[0], -1};
if (!cmpL(res[0]) && !cmpL(res[1]))
  switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
    case 0: return {res[0], res[0]};
    case 2: return {res[1], res[1]};
return res;
```

int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;

MinkowskiSum.h Description: Output the minkowski sum for two convex hull. For distance of

two convex hull A and B, calculate distance from O(0, 0) to minkowskiSum(A, "Point.h" ba4047, 16 lines

```
template<class P>
vector<P> minkowskiSum(vector<P> a, vector<P> b) {
 int n = sz(a), m = sz(b);
  rotate(begin(a), min_element(all(a)), end(a));
  rotate(begin(b), min_element(all(b)), end(b));
  a.push_back(a[0]); a.push_back(a[1]);
 b.push_back(b[0]); b.push_back(b[1]);
  vector<P> res;
  for (int i = 0, j = 0; i < n or j < m; ) {
   res.push_back(a[i] + b[j]);
   auto c = sgn((a[i+1] - a[i]).cross(b[j+1] - b[j]));
   i += (i < n \text{ and } c >= 0);
   j += (j < m \text{ and } c <= 0);
  return res:
```

HalfPlaneInter.h

Description: Given n segments (s, t), returns polygon sides representing the intersection of left side of the segments.

```
bc38c7, 35 lines
template <class P> struct Seq { P s, t; };
template <class S>
bool xleft(const S& o, const S& a, const S& b) {
 auto [03, 04] = make_pair(o.s.cross(o.t, b.s),
     o.s.cross(o.t, b.t)); // C^2
  auto [a3, a4] = make_pair(a.s.cross(a.t, b.s),
     a.s.cross(a.t, b.t));
  if (a3 - a4 < 0) a3 *= -1, a4 *= -1;
 return (__int128) o4 * a3 - (__int128) o3 * a4 > 0; // C^4
template <class P>
int cmp (const P& a, const P& b, const bool same = true) {
 int na = (a < P(0, 0)), nb = (b < P(0, 0));
 if (na != nb) return na < nb;</pre>
 if (sqn(a.cross(b)) != 0) return sqn(a.cross(b)) > 0;
  return same ? a.dist2() < b.dist2() : -1;</pre>
template<class S>
vector<S> halfPlaneInter(vector<S> ss) {
```

return (t != -1 ? t : sgn(a.s.cross(a.t, b.s)) < 0);

if ((ss[i-1].t - ss[i-1].s).cross(ss[i].t - ss[i].s) == 0)

while (qt-qh>1 and !xleft(ss[i], dq[qt-2], dq[qt-1])) --qt;

sort(all(ss), [&](S a, S b) -> int {

int n = sz(ss), qh = 0, qt = 1;

vector < S > dq(n); dq[0] = ss[0];

rep(i, 1, n) {

continue;

int t = cmp(a.t - a.s, b.t - b.s, 0);

```
while (qt-qh>1 \text{ and } !xleft(ss[i], dq[qh], dq[qh+1])) ++qh;
   dq[qt++] = ss[i];
 while (qt-qh>2 and !xleft(dq[qh], dq[qt-2], dq[qt-1])) --qt;
 return {begin(dq) + qh, begin(dq) + qt};
8.5 Misc. Point Set Problems
ClosestPair.h
Description: Finds the closest pair of points.
Time: \mathcal{O}(n \log n)
                                                        ac41a6, 17 lines
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
assert(sz(v) > 1);
 sort(all(v), [](P a, P b) { return a.y < b.y; });
 pair<11, pair<P, P>> ret{LLONG MAX, {P(), P()}};
 int j = 0;
 for (P p : v) {
   P d{1 + (ll)sgrt(ret.first), 0};
   while (v[j].y \le 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;
ManhattanMST.h
Description: Given N points, returns up to 4*N edges, which are guaran-
teed to contain a minimum spanning tree for the graph with edge weights
w(p,q) = |p.x - q.x| + |p.y - q.y|. Edges are in the form (distance, src, dst).
Use a standard MST algorithm on the result to find the final MST.
Time: \mathcal{O}(N \log N)
"Point.h"
                                                         df6f59, 23 lines
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
 vi id(sz(ps));
 iota(all(id), 0);
 vector<array<int, 3>> edges;
 rep(k,0,4) {
   sort(all(id), [&](int i, int j) {
        return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
   map<int, int> sweep;
   for (int i : id) {
     for (auto it = sweep.lower_bound(-ps[i].y);
               it != sweep.end(); sweep.erase(it++)) {
       int j = it->second;
       P d = ps[i] - ps[j];
       if (d.y > d.x) break;
       edges.push_back(\{d.y + d.x, i, j\});
     sweep[-ps[i].y] = i;
   for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
 return edges;
KDTree.h
Description: KD-tree (2d, can be extended to 3d)
                                                        bac5b0, 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; }</pre>

```
bool on_y(const P& a, const P& b) { return a.y < b.y; }</pre>
struct Node {
 P pt; // if this is a leaf, the single point in it
 T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;
  T distance (const P& p) { // min squared distance to a point
   T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
};
struct KDTree {
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}
  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      /\!/ uncomment if we should not find the point itself:
      // if (p = node \rightarrow 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)</pre>
      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);
};
FastDelaunav.h
Description: Fast Delaunay triangulation. Each circumcircle contains none
of the input points. There must be no duplicate points. If all points are on
a line, no triangles will be returned. Should work for doubles as well, though
there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1],
t[0][2], t[1][0], \dots, all counter-clockwise.
Time: \mathcal{O}(n \log n)
"Point.h"
                                                          eefdf5, 88 lines
typedef Point<11> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
```

```
struct Quad {
 Q rot, o; P p = arb; bool mark;
 P& F() { return r()->p; }
 O& r() { return rot->rot; }
 Q prev() { return rot->o->rot; }
 0 next() { return r()->prev(); }
bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
 111 p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r -> 0; r -> r() -> r() = r;
  rep(i,0,4) r = r - rot, r - p = arb, r - o = i & 1 ? <math>r : r - r();
 r->p = orig; r->F() = dest;
  return r;
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
Q connect(Q a, Q b) {
 Q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
pair<0,0> 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)
 O A. B. ra. rb:
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B\rightarrow p.cross(H(A)) < 0 \&\& (A = A\rightarrow 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())) {
     0 t = e \rightarrow dir; \
     splice(e, e->prev()); \
     splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \setminus
  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());
     base = connect(base->r(), LC->r());
 return { ra, rb };
```

```
vector<P> triangulate(vector<P> pts) {
 sort(all(pts)); assert(unique(all(pts)) == pts.end());
 if (sz(pts) < 2) return {};
 Q e = rec(pts).first;
 vector<Q> q = \{e\};
 int qi = 0;
 while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
 q.push_back(c->r()); c = c->next(); } while (c != e); }
8.501; ptVoronoi;Diagram
Perform half-plane intersection on bisectors of triangles from
FastDelaunay. Deal with cases that all points are collinear (no
triangles) carefully.
```

$8.6 \quad 3D$

```
\operatorname{PolyhedronVolume.h}
```

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

3058c3, 6 lines

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
 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

```
template < class T > struct Point 3D {
 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 {</pre>
   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 sgrt((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"
                                                        5b45fc, 49 lines
typedef Point3D<double> P3;
struct PR {
 void ins(int x) { (a == -1 ? a : b) = x; }
 void rem(int x) { (a == x ? a : b) = -1; }
 int cnt() { return (a != -1) + (b != -1); }
struct F { P3 q; int a, b, c; };
vector<F> hull3d(const vector<P3>& A) {
 assert(sz(A) >= 4);
 vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
 vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
   P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
   if (q.dot(A[1]) > q.dot(A[i]))
      q = q * -1;
   F f{q, i, j, k};
   E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
   FS.push_back(f);
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
   mf(i, j, k, 6 - i - j - k);
 rep(i,4,sz(A)) {
   rep(j, 0, sz(FS)) {
     F f = FS[i];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
   int nw = sz(FS);
   rep(j,0,nw) {
     F f = FS[j];
```

SphericalDistance.h

return FS;

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

#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);

A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);

C(a, b, c); C(a, c, b); C(b, c, a);

for (F& it : FS) if ((A[it.b] - A[it.a]).cross(

```
double sphericalDistance(double f1, double t1,
   double f2, double t2, double radius) {
  double dx = \sin(t2) \cdot \cos(f2) - \sin(t1) \cdot \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: f[x] = the longest prefix i, so that s(0:i) is a suffix of s(0:x).
When A_i \neq B_{i+1} move j to f[j].
Time: \mathcal{O}(|A| + |B|)
                                                                        45ef23, 21 lines
```

```
vi fail_function(string &s) {
 int n = sz(s);
 vi f(n, -1); // leave an additional space
 rep(i, 1, n) {
   int cur = f[i - 1];
   while (cur != -1 && s[cur + 1] != s[i]) cur = f[cur];
   cur += (s[cur + 1] == s[i]);
   f[i] = cur;
  return f;
int KMP (string &a, string &b) {
 vi f = fail_function(b);
 int j = -1, ans = 0;
 rep(i, 0, sz(a)) {
   while(j != -1 \&\& b[j + 1] != a[i]) j = f[j];
   j += (b[j + 1] == a[i]);
   ans += (j == sz(b));
  return ans;
```

ZFunc.h

Description: z[x] computes the length of the longest common prefix of s[i:]and s, except z[0] = 0. (abacaba -> 0010301) Time: $\mathcal{O}(N)$ d7a1f0, 10 lines

```
vi Zfunc(const string &s) {
 int n = sz(s), l = 1, r = 0;
 vi z(n, n);
 rep(i, 1, n) {
   z[i] = max(0, min(z[i-1], r-i+1));
   while(i + z[i] < n \&\& s[i + z[i]] == s[z[i]])
     1 = i, r = i + z[i], z[i]++;
 return z;
```

Description: Computes the longest palindromic subsequence in string. even indices are padded with char '.'. Time: $\mathcal{O}(N)$

d589dc, 14 lines

```
int Manacher(string &s) {
 string t = ".";
 rep(i, 0, sz(s) - 1) t += s[i] + '.';
 int n = sz(t), 1 = 0, r = 0;
 vi v(n, 1);
 rep(i, 1, n - 1) {
   v[i] = max(1, min(v[1 + 1 - i], r - i + 1));
   while(0 \le i - v[i] \&\& i + v[i] < n \&\&
       t[i + v[i]] == t[i - v[i]]) {
     l = i, r = i + v[i], v[i] ++;
 return *max_element(all(v)) - 1;
```

```
return a:
SuffixArray.h
Description: Builds suffix array for a string. sa[i] is the starting index of
the suffix which is i'th in the sorted suffix array. The returned vector is of
size n+1, and sa[0] = n. The lcp array contains longest common prefixes
for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]),
lcp[0] = 0. The input string must not contain any zero bytes.
Time: \mathcal{O}(N \log N)
                                                          0850d4, 23 lines
struct SuffixArray {
vi sa, lcp, rank;
 SuffixArray(string& s, int lim=256) {
   int n = sz(s) + 1, k = 0, a, b;
   vi \times (all(s)+1), v(n), ws(max(n, lim));
   sa = lcp = rank = y, iota(all(sa), 0);
   for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
     p = j, iota(all(y), n - j);
     rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
     fill(all(ws), 0);
     rep(i,0,n) ws[x[i]]++;
     rep(i,1,lim) ws[i] += ws[i-1];
     for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
     swap(x, y), p = 1, x[sa[0]] = 0;
     rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] \&\& y[a + j] == y[b + j]) ? p - 1 : p++;
   rep(i,1,n) rank[sa[i]] = i;
   for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)</pre>
     for (k \&\& k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
SuffixAutomaton.h
Description: Compressed form of all substrings of string S. link – the longest
suffix of current substring with different endpos. endpos(t) - the set of all po-
sitions in the string s, in which the occurrences of t end. Follow link from last
to obtain all terminal states.
Usage: Number of different substrings.
Smallest cyclic shift (S + S).
Number of occurrences.
Shortest non-appearing string.
LCS (substring) of multiple strings.
Time: \mathcal{O}\left(|S|\right)
                                                          c64732, 37 lines
struct SAM {
const int P = 100000;
 vector<map<char, int>> ch;
 vector<int> len, link;
 int sz, last;
 void init_() {
   sz = 1, last = 0;
   ch.assign(P * 2, map<char, int>());
   len.assign(P * 2, 0);
   link.assign(P \star 2, -1);
 void extend(char c) {
   int cur = sz ++;
```

Description: Finds the lexicographically smallest rotation of a string.

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

if $(a+k == b \mid | s[a+k] < s[b+k]) \{b += max(0, k-1); break; \}$

MinRotation.h

int minRotation(string s) {

int a=0, N=sz(s); s += s;

len[cur] = len[last] + 1;

if (s[a+k] > s[b+k]) { a = b; **break**; }

rep(b,0,N) rep(k,0,N) {

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

```
int p = last;
    while(p != -1 && !ch[p].count(c)) {
      ch[p][c] = cur;
      p = link[p];
    if(p == -1) link[cur] = 0;
    else {
      int q = ch[p][c];
      if(len[p] + 1 == len[q]) link[cur] = q;
      else {
         int cl = sz ++;
         ch[cl] = ch[q];
         len[cl] = len[p] + 1;
         link[cl] = link[q];
         while (p != -1 && ch[p].count (c) && ch[p][c] == q) {
          ch[p][c] = cl, p = link[p];
         link[q] = link[cur] = cl;
    last = cur;
};
AhoCorasick.h
Description: Aho-Corasick automaton, used for multiple pattern matching.
Initialize with AhoCorasick ac(patterns); the automaton start node will be at
index 0. find(word) returns for each position the index of the longest word that
ends there, or -1 if none. findAll(-, word) finds all words (up to N\sqrt{N} many
if no duplicate patterns) that start at each position (shortest first). Duplicate
patterns are allowed; empty patterns are not. To find the longest words that
start at each position, reverse all input. For large alphabets, split each symbol
into chunks, with sentinel bits for symbol boundaries.
Time: construction takes \mathcal{O}(26N), where N = \text{sum of length of patterns}.
find(x) is \mathcal{O}(N), where N = length of x. findAll is \mathcal{O}(NM).
                                                            119c29, 72 lines
struct AhoCorasick {
  enum { P = 26, st = 'a'};
  struct node { // zero-based
    array<int, P> ch = {0};
    int fail = 0, cnt = 0, dep = 0;
  };
  int cnt;
  vector<node> v;
  vector<int> ans;
  void init_(int mx) {
    v.clear();
    cnt = 1, v.resize(mx);
    v[0].fail = 0;
  void insert(string s) {
    int p = 0, dep = 1;
    for(auto i : s) {
      int c = i - st;
      if(!v[p].ch[c]) {
        v[cnt].dep = dep;
         v[p].ch[c] = cnt ++;
```

p = v[p].ch[c], dep ++;

void build(vector<string> s) {

for(auto i : s) insert(i);

for(int i = 0; i < P; i ++) {</pre>

if(v[0].ch[i]) q.push(v[0].ch[i]);

v[p].cnt ++;

queue<int> q;

q.pop();

while(q.size()) {

int p = q.front();

```
while(cur && !v[cur].ch[i]) cur = v[cur].fail;
        if(v[cur].ch[i]) cur = v[cur].ch[i];
        v[to].fail = cur;
        v[to].cnt += v[cur].cnt;
        q.push(to);
 void traverse(string s) {
   int p = 0;
   ans.assign(cnt, 0);
    for(auto i : s) {
      int c = i - st;
      while(p && !v[p].ch[c]) p = v[p].fail;
      if(v[p].ch[c]) {
       p = v[p].ch[c];
        ans[p] ++, v[p].cnt;
    vector<int> ord(cnt, 0);
    iota(all(ord), 0);
    sort(all(ord), [&](int a, int b) { return v[a].dep > v[b].dep
    for(auto i : ord) ans[v[i].fail] += ans[i];
  int go(string s) {
    int p = 0;
    for(auto i : s) {
      int c = i - st;
      assert (v[p].ch[c]);
     p = v[p].ch[c];
   return ans[p];
};
Hashing.h
Description: Airthmetic mod 2^{64} - 1.
                                                       2d2a67, 44 lines
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
typedef uint64_t ull;
struct H {
 ull x; H(ull x=0) : x(x) {}
 H operator+(H o) { return x + o.x + (x + o.x < x); }
 H operator-(H o) { return *this + ~o.x; }
 H operator*(H o) { auto m = ( uint128 t)x * o.x;
   return H((ull)m) + (ull)(m >> 64); }
 ull get() const { return x + !~x; }
 bool operator==(H o) const { return get() == o.get(); }
 bool operator<(H o) const { return get() < o.get(); }</pre>
static const H C = (11)1e11+3; // (order ~ 3e9; random also ok)
struct HashInterval {
 vector<H> ha, pw;
 HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
   pw[0] = 1;
   rep(i, 0, sz(str))
     ha[i+1] = ha[i] * C + str[i],
     pw[i+1] = pw[i] * C;
 H hashInterval(int a, int b) { // hash [a, b)
   return ha[b] - ha[a] * pw[b - a];
```

for(int i = 0; i < P; i ++) if(v[p].ch[i]) {</pre>

int to = v[p].ch[i], cur = v[p].fail;

if (mx.second == -1) return {};

cur = mx.first:

R.push_back (mx.second);

```
vector<H> getHashes(string& str, int length) {
if (sz(str) < length) return {};</pre>
 H h = 0, pw = 1;
 rep(i,0,length)
  h = h * C + str[i], pw = pw * C;
 vector<H> ret = {h};
 rep(i,length,sz(str)) {
  ret.push_back(h = h * C + str[i] - pw * str[i-length]);
 return ret;
H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
Various (10)
10.1 Intervals
IntervalContainer.h
Description: Add and remove intervals from a set of disjoint intervals. Will
```

merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

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

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

void removeInterval(set<pii>& is, int L, int R) {

```
IntervalCover.h
```

if (L == R) return;

auto r2 = it->second;

else (int&)it->second = L;

auto it = addInterval(is, L, R);

if (it->first == L) is.erase(it);

if (R != r2) is.emplace(R, r2);

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: $O(N \log N)$

```
9e9d8d, 19 lines
```

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

```
return R;
ConstantIntervals.h
Description: Split a monotone function on [from, to) into a minimal set of
half-open intervals on which it has the same value. Runs a callback g for each
             constantIntervals(0, sz(v), [&](int x){return v[x];},
[&] (int lo, int hi, T val)\{\ldots\});
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) {
    q(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;</pre>
 int i = from; auto p = f(i), q = f(to-1);
 rec(from, to-1, f, g, i, p, q);
 g(i, to, q);
10.2 Misc. algorithms
```

TernarySearch.h

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

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

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum $S \le t$ such that S is the sum of some subset of the weights.

```
Time: O(N \max(w_i))
int knapsack(vi w, int t) {
 int a = 0, b = 0, x;
 while (b < sz(w) && a + w[b] <= t) a += w[b++];
 if (b == sz(w)) return a;
 int m = *max_element(all(w));
 vi u, v(2*m, -1);
 v[a+m-t] = b;
 rep(i,b,sz(w)) {
   u = v;
```

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

10.3 Dynamic programming KnuthDP.h

```
Description: When doing DP on intervals: a[i][j] = \min_{i < k < j} (a[i][k] + 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 \leq b \leq c \leq d. Consider also: LineContainer (ch. Data structures),
monotone queues, ternary search.
Time: \mathcal{O}(N^2)
```

DivideAndConquerDP.h

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

```
d38d2b, 18 lines
struct DP { // Modify at will:
 int lo(int ind) { return 0; }
 int hi(int ind) { return ind; }
 11 f(int ind, int k) { return dp[ind][k]; }
 void store(int ind, int k, ll v) { res[ind] = pii(k, v); }
 void rec(int L, int R, int LO, int HI) {
   if (L >= R) return;
   int mid = (L + R) >> 1;
   pair<11, int> best(LLONG_MAX, LO);
   rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
     best = min(best, make pair(f(mid, k), k));
   store(mid, best.second, best.first);
   rec(L, mid, LO, best.second+1);
   rec(mid+1, R, best.second, HI);
 void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
```

10.4 Debugging tricks

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

10.5 Optimization tricks

__builtin_ia32_ldmxcsr(40896); disables denormals (which make floats 20x slower near their minimum value).

10.5.1 Bit hacks

- x & -x is the least bit in x.
- for (int x = m; x;) { --x &= m; ... } loops over all subset masks of m (except m itself).

```
• c = x\&-x, r = x+c; (((r^x) >> 2)/c) | r is the
  next number after x with the same number of bits set.
```

```
• rep(b, 0, K) rep(i, 0, (1 << K))
    if (i & 1 << b) D[i] += D[i^(1 << b)];
  computes all sums of subsets.
```

10.5.2 Pragmas

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

Description: Compute a%b about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod{b}$ in the range [0, 2b). 751a02, 8 lines

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

FastInput.h

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

Usage: ./a.out < input.txt

Time: About 5x as fast as cin/scanf.

7b3c70, 17 lines

```
inline char gc() { // like getchar()
 static char buf[1 << 16];</pre>
 static size_t bc, be;
 if (bc >= be) {
  buf[0] = 0, bc = 0;
  be = fread(buf, 1, sizeof(buf), stdin);
 return buf[bc++]; // returns 0 on EOF
int readInt() {
while ((a = gc()) < 40);
if (a == '-') return -readInt();
while ((c = gc()) >= 48) a = a * 10 + c - 480;
return a - 48;
```

BumpAllocator.h

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

```
// Either globally or in a single class:
static char buf[450 << 201;
void* operator new(size_t s) {
static size_t i = sizeof buf;
assert(s < i);
 return (void*)&buf[i -= s];
void operator delete(void*) {}
```

```
SmallPtr.h
Description: A 32-bit pointer that points into BumpAllocator memory.
template < class T> struct ptr {
 unsigned ind;
 ptr(T*p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
   assert (ind < sizeof buf);
 T& operator*() const { return *(T*)(buf + ind); }
 T* operator->() const { return &**this; }
 T& operator[](int a) const { return (&**this)[a]; }
 explicit operator bool() const { return ind; }
BumpAllocatorSTL.h
Description: BumpAllocator for STL containers.
Usage: vector<vector<int, small<int>>> ed(N);
                                                       bb66d4, 14 lines
char buf[450 << 20] alignas(16);</pre>
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) {}
```

Techniques (A)

techniques.txt

159 lines

Recursion Divide and conquer Finding interesting points in N log N Algorithm analysis Master theorem Amortized time complexity Greedy algorithm Scheduling Max contiquous subvector sum Invariants Huffman encoding Graph theory Dynamic graphs (extra book-keeping) Breadth first search Depth first search * Normal trees / DFS trees Dijkstra's algorithm MST: Prim's algorithm Bellman-Ford Konig's theorem and vertex cover Min-cost max flow Lovasz toggle Matrix tree theorem Maximal matching, general graphs Hopcroft-Karp Hall's marriage theorem Graphical sequences Floyd-Warshall Euler cycles Flow networks * Augmenting paths * Edmonds-Karp Bipartite matching Min. path cover Topological sorting Strongly connected components Cut vertices, cut-edges and biconnected components Edge coloring * Trees Vertex coloring * Bipartite graphs (=> trees) * 3^n (special case of set cover) Diameter and centroid K'th shortest path Shortest cycle Dynamic programming Knapsack Coin change Longest common subsequence Longest increasing subsequence Number of paths in a dag Shortest path in a dag Dynprog over intervals Dynprog over subsets Dynprog over probabilities Dynprog over trees 3^n set cover Divide and conquer Knuth optimization Convex hull optimizations RMQ (sparse table a.k.a 2^k-jumps) Bitonic cycle Log partitioning (loop over most restricted) Combinatorics Computation of binomial coefficients Pigeon-hole principle

Inclusion/exclusion Catalan number Pick's theorem Number theory Integer parts Divisibility Euclidean algorithm Modular arithmetic * Modular multiplication * Modular inverses * Modular exponentiation by squaring Chinese remainder theorem Fermat's little theorem Euler's theorem Phi function Frobenius number Quadratic reciprocity Pollard-Rho Miller-Rabin Hensel lifting Vieta root jumping Game theory Combinatorial games Game trees Mini-max Nim Games on graphs Games on graphs with loops Grundy numbers Bipartite games without repetition General games without repetition Alpha-beta pruning Probability theory Optimization Binary search Ternary search Unimodality and convex functions Binary search on derivative Numerical methods Numeric integration Newton's method Root-finding with binary/ternary search Golden section search Matrices Gaussian elimination Exponentiation by squaring Sorting Radix sort Geometry Coordinates and vectors * Cross product * Scalar product Convex hull Polygon cut Closest pair Coordinate-compression Ouadtrees KD-trees All segment-segment intersection Sweeping Discretization (convert to events and sweep) Angle sweeping Line sweeping Discrete second derivatives Strings Longest common substring Palindrome subsequences Knuth-Morris-Pratt Tries Rolling polynomial hashes Suffix array

Suffix tree Aho-Corasick Manacher's algorithm Letter position lists Combinatorial search Meet in the middle Brute-force with pruning Best-first (A*) Bidirectional search Iterative deepening DFS / A* Data structures LCA (2^k-jumps in trees in general) Pull/push-technique on trees Heavy-light decomposition Centroid decomposition Lazy propagation Self-balancing trees Convex hull trick (wcipeg.com/wiki/Convex_hull_trick) Monotone queues / monotone stacks / sliding queues Sliding queue using 2 stacks Persistent segment tree