Massachusetts Institute of Technology

MIT $(^w^)$

Benjamin Qi, Spencer Compton, Zhezheng Luo

adapted from KACTL and MIT NULL

2019-12-21

Contest (1)

templateShort.cpp

37 lines

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

typedef long long ll;
typedef pair<int, int> pi;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<pi> vpi;

#define FOR(i,a,b) for (int i = (a); i < (b); ++i)
#define F0R(i,a) FOR(i,0,a)
#define ROF(i,a,b) for (int i = (b)-1; i >= (a); --i)
#define R0F(i,a) ROF(i,0,a)
#define trav(a,x) for (auto& a: x)

#define sz(x) (int)x.size()
#define all(x) begin(x), end(x)
#define rsz resize

#define mp make_pair
#define pb push_back
#define f first
#define s second

const int MOD = 1e9+7; // 998244353; // = (119<<23)+1
const int MX = 2e5+5;

template<class T> bool ckmin(T& a, const T& b) {
  return a > b ? a = b, 1 : 0; }
template<class T> bool ckmax(T& a, const T& b) {
  return a < b ? a = b, 1 : 0; }

mt19937 rng((uint32_t)chrono::steady_clock::now().
  ↪time_since_epoch().count());

int main() {
  ios_base::sync_with_stdio(0); cin.tie(0);
}
```

.bashrc

6 lines

```
co() { # on mac, add -Wl,-stack_size -Wl,0x10000000
    g++ -std=c++11 -O2 -Wall -Wextra -o $1 $1.cpp
}
run() {
    co $1 && ./$1
}
```

hash.sh

3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp-9 -dD -P -fpreprocessed|tr -d '[:space:]'|md5sum|cut -c-6
```

troubleshoot.txt

61 lines

```
Pre-submit:
Write down your thoughts, even if they don't completely solve
  ↪the problem.
Stay organized (don't leave papers all over the place)!
Give your variables (and files) useful names!
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Remove debug output.
Make sure to submit the right file.
You should know what your code is doing ...

Wrong answer:
Read the full problem statement again.
Have you understood the problem correctly?
Are you sure your algorithm works?
Try writing a slow (but correct) solution.
Can your algorithm handle the whole range of input?
Did you consider corner cases (n=1) or other special cases?
Print your solution! Print debug output, as well.
Is your output format correct? (including whitespace)
Are you clearing all data structures between test cases?
Any uninitialized variables?
Any undefined behavior (array out of bounds)?
Any overflows or NaNs (shifting ll by 64 bits or more)?
Confusing N and M, i and j, etc.?
Confusing ++i and i++?
Make sure that you deal correctly with numbers close to (but
  ↪not) zero.
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some test cases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Rewrite your solution from the start or let a teammate do it.

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

Time limit exceeded:
Do you have any possible infinite loops?
What's your complexity? Extended TL does not mean that
  ↪something simple (like NlogN) isn't intended.
Are you copying a lot of unnecessary data? (References)
Avoid vector, map. (use arrays/unordered_map)
How big is the input and output? (consider FastI)
What do your teammates think about your algorithm?

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

FastI.h

Description: fast input for chinese contests
Time: ~300ms faster for 10^6 long longs

38cbac, 22 lines

```
namespace fastI {
  const int BSZ = 100000;
  char nc() { // get next char
    static char buf[BSZ], *p1 = buf, *p2 = p1;
    if (p1 == p2) {
      p1 = buf; p2 = buf+fread(buf,1,BSZ,stdin);
      if (p1 == p2) return EOF;
    }
    return *p1++;
  }
  bool blank(char ch) { return ch == ' ' || ch == '\n'
            || ch == '\r' || ch == '\t'; }
  template<class T> void ri(T& x) { // read int or ll
    char ch; int sgn = 1;
    while ((ch = nc()) > '9' || ch < '0')
      if (ch == '-') sgn *= -1;
    x = ch-'0';
    while ((ch = nc()) >= '0' && ch <= '9') x = x*10+ch-'0';
    x *= sgn;
  }
}
using namespace fastI;
```

Mathematics (2)

2.1 Equations

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

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.

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

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

2.3 Trigonometry

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

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

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

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

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

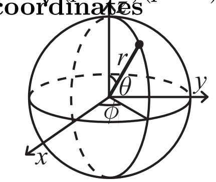
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

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

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

2.7 Series

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

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ 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,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

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

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

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

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

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

Data Structures (3)

3.1 STL

MapComparator.h

Description: custom comparator for map / set

Usage: set<int,cmp> s; map<int,int,cmp> m;

ae81c4, 5 lines

```
struct cmp {
  bool operator()(const int& l, const int& r) const {
    return l > r; // sort items in decreasing order
  }
};
```

HashMap.h

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

Usage: ht<int,int> h({},{},{},{},{1<<16}); // reserve memory for 1<<16 elements

<ext/pb_ds/assoc_container.hpp> a1d018, 11 lines

```
using namespace __gnu_pbds;

struct chash { // use most bits rather than just the lowest
    ones
  const uint64_t C = ll(2e18*PI)+71; // large odd number
  const int RANDOM = rng();
  ll operator()(ll x) const {
    return __builtin_bswap64((x^RANDOM)*C); }
};
template<class K,class V> using ht = gp_hash_table<K,V,chash>;
template<class K,class V> V get(ht<K,V>& u, K x) {
  return u.find(x) == end(u) ? 0 : u[x]; }
```

PQ.h

Description: Priority queue w/ modification. Use for Dijkstra?

<bits/extc++.h> 1ad0e6, 9 lines

```
pqExample() {
  __gnu_pbds::priority_queue<int> p;
  vi act; vector<decltype(p)::point_iterator> v;
  int n = 1000000;
  F0R(i,n) { int r = rand(); act.pb(r), v.pb(p.push(r)); }
  F0R(i,n) { int r = rand(); act[i] = r, p.modify(v[i],r); }
  sort(rall(act));
  F0R(i,n) { assert(act[i] == p.top()); p.pop(); }
}
```

IndexedSet.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element.

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

<ext/pb_ds/tree_policy.hpp>, <ext/pb_ds/assoc_container.hpp> c5d6f2, 16 lines

```
using namespace __gnu_pbds;

template <class T> using Tree = tree<T, null_type, less<T>,
  rb_tree_tag, tree_order_statistics_node_update>;
// to get a map, change null_type

#define ook order_of_key
#define fbo find_by_order

void treeExample() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).f; assert(it == t.lb(9));
  assert(t.ook(10) == 1); assert(t.ook(11) == 2);
  assert(*t.fbo(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

Rope.h

Description: insert element at i-th position, cut a substring and re-insert somewhere else

Time: $\mathcal{O}(\log N)$ per operation? not well tested

<ext/rope> 4fea66, 17 lines

```
using namespace __gnu_cxx;

void ropeExample() {
  rope<int> v(5, 0); // initialize with 5 zeroes
  F0R(i,sz(v)) v.mutable_reference_at(i) = i+1;
  F0R(i,5) v.pb(i+1); // constant time pb
  rope<int> cur = v.substr(1,2);
```

```
  v.erase(1,3); // erase 3 elements starting from 1st element
  for (rope<int>::iterator it = v.mutable_begin();
    it != v.mutable_end(); ++it)
    cout << *it << " ";
  cout << "\n"; // 1 5 1 2 3 4 5
  v.insert(v.mutable_begin()+2,cur); // index or const_iterator
  v += cur;
  FOR(i,sz(v)) cout << v[i] << " ";
  cout << "\n"; // 1 5 2 3 1 2 3 4 5 2 3
}
```

LineContainer.h

Description: Given set of lines, computes greatest y-coordinate for any x.
Time: $\mathcal{O}(\log N)$

0888fa, 35 lines

```
struct Line {
  mutable ll k, m, p; // slope, y-intercept, last optimal x
  ll eval (ll x) { return k*x+m; }
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

// for doubles, use inf = 1/.0, div(a,b) = a/b
const ll inf = LLONG_MAX;
// floored division
ll divi(ll a, ll b) { return a/b-((a^b) < 0 && a%b); }
// last x such that first line is better
ll bet(const Line& x, const Line& y) {
  if (x.k == y.k) return x.m >= y.m ? inf : -inf;
  return divi(y.m-x.m,x.k-y.k);
}

struct LC : multiset<Line,less<>> {
  // updates x->p, determines if y is unneeded
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return 0; }
    x->p = bet(*x,*y); return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k,m,0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lb(x); return l.k*x+l.m;
  }
};
```

LCDeque.h

Description: same as LineContainer but linear time given assumptions

"LineContainer.h" edc7d3, 34 lines

```
struct LC0 : deque<Line> {
  void addBack(Line L) { // assume nonempty
    while (1) {
      auto a = back(); pop_back(); a.p = bet(a,L);
      if (size() && back().p >= a.p) continue;
      pb(a); break;
    }
    L.p = inf; pb(L);
  }
  void addFront(Line L) {
    while (1) {
      if (!size()) { L.p = inf; break; }
      if ((L.p = bet(L,front())) >= front().p) pop_front();
      else break;
    }
    push_front(L);
  }
  void add(ll k, ll m) { // line goes to one end of deque
    if (!size() || k <= front().k) addFront({k,m,0});
    else assert(k >= back().k), addBack({k,m,0});
  }
  int ord = 0; // 1 = increasing, -1 = decreasing
  ll query(ll x) {
    assert(ord);
    if (ord == 1) {
      while (front().p < x) pop_front();
      return front().eval(x);
    } else {
      while (size() > 1 && prev(prev(end()))->p >= x)
        pop_back();
      return back().eval(x);
    }
  }
};
```

3.2 1D Range Queries

RMQ.h

Description: 1D range minimum query
Time: $\mathcal{O}(N \log N)$ build, $\mathcal{O}(1)$ query

b1fe94, 21 lines

```
template<class T> struct RMQ {
  // floor(log_2(x))
  int level(int x) { return 31-__builtin_clz(x); }
  vector<T> v; vector<vi> jmp;
  int comb(int a, int b) {
    return v[a] == v[b] ? min(a,b) : (v[a] < v[b] ? a : b);
  } // index of minimum
  void init(const vector<T>& _v) {
    v = _v; jmp = {vi(sz(v))}; iota(all(jmp[0]),0);
    for (int j = 1; 1<<j <= sz(v); ++j) {
      jmp.pb(vi(sz(v)-(1<<j)+1));
      FOR(i,sz(jmp[j])) jmp[j][i] = comb(jmp[j-1][i],
                  jmp[j-1][i+(1<<(j-1))]);
    }
  }
  int index(int l, int r) { // get index of min element
    int d = level(r-l+1);
    return comb(jmp[d][l],jmp[d][r-(1<<d)+1]);
  }
  T query(int l, int r) { return v[index(l,r)]; }
};
```

BIT.h

Description: N-D range sum query with point update
Usage: {BIT<int,10,10>} gives a 2D BIT
Time: $\mathcal{O}\left((\log N)^D\right)$

e39d3e, 18 lines

```
template <class T, int ...Ns> struct BIT {
  T val = 0;
  void upd(T v) { val += v; }
  T query() { return val; }
};
template <class T, int N, int... Ns> struct BIT<T, N, Ns...> {
  BIT<T,Ns...> bit[N+1];
  template<typename... Args> void upd(int pos, Args... args) {
    for (; pos <= N; pos += (pos&-pos)) bit[pos].upd(args...);
  }
  template<typename... Args> T sum(int r, Args... args) {
    T res = 0; for (; r; r -= (r&-r))
      res += bit[r].query(args...);
    return res;
  }
  template<typename... Args> T query(int l, int r, Args...
    args) { return sum(r,args...)-sum(l-1,args...); }
};
```

BITrange.h

Description: 1D range increment and sum query
Time: $\mathcal{O}(\log N)$

"BIT.h" 77a935, 14 lines

```
template<class T, int SZ> struct BITrange {
  BIT<T,SZ> bit[2]; // piecewise linear functions
  // let cum[x] = sum_{i=1}^{x}a[i]
  void upd(int hi, T val) { // add val to a[1..hi]
    // if x <= hi, cum[x] += val*x
    bit[1].upd(1,val), bit[1].upd(hi+1,-val);
    // if x > hi, cum[x] += val*hi
    bit[0].upd(hi+1,hi*val);
  }
  void upd(int lo, int hi, T val) {
    upd(lo-1,-val), upd(hi,val); }
  T sum(int x) { return bit[1].sum(x)*x+bit[0].sum(x); }
  T query(int x, int y) { return sum(y)-sum(x-1); }
};
```

SegTree.h

Description: 1D point update, range query. Change comb to any associative (not necessarily commutative) operation
Time: $\mathcal{O}(\log N)$

bf15d6, 19 lines

```
template<class T> struct Seg {
  const T ID = 0; // comb(ID,b) must equal b
  T comb(T a, T b) { return a+b; }
  int n; vector<T> seg;
  void init(int _n) { n = _n; seg.rsz(2*n); }
  void pull(int p) { seg[p] = comb(seg[2*p],seg[2*p+1]); }
  void upd(int p, T value) {  // set value at position p
    seg[p += n] = value;
    for (p /= 2; p; p /= 2) pull(p);
  }
  T query(int l, int r) {  // sum on interval [l, r]
    T ra = ID, rb = ID;
    for (l += n, r += n+1; l < r; l /= 2, r /= 2) {
      if (l&1) ra = comb(ra,seg[l++]);
      if (r&1) rb = comb(seg[--r],rb);
    }
    return comb(ra,rb);
  }
};
```

SegTreeBeats.h

Description: supports modifications in the form ckmin(a_i,t) for all $l \le i \le r$, range max and sum queries
Time: $\mathcal{O}(\log N)$

f98405, 63 lines

```
template<int SZ> struct SegTreeBeats {
  int N;
  ll sum[2*SZ];
  int mx[2*SZ][2], maxCnt[2*SZ];
  void pull(int ind) {
    FOR(i,2) mx[ind][i] = max(mx[2*ind][i],mx[2*ind+1][i]);
    maxCnt[ind] = 0;
    FOR(i,2) {
      if (mx[2*ind+i][0] == mx[ind][0])
        maxCnt[ind] += maxCnt[2*ind+i];
      else ckmax(mx[ind][1],mx[2*ind+i][0]);
    }
    sum[ind] = sum[2*ind]+sum[2*ind+1];
  }
  void build(vi& a, int ind = 1, int L = 0, int R = -1) {
```

```
    if (R == -1) { R = (N = sz(a))-1; }
    if (L == R) {
      mx[ind][0] = sum[ind] = a[L];
      maxCnt[ind] = 1; mx[ind][1] = -1;
      return;
    }
    int M = (L+R)/2;
    build(a,2*ind,L,M); build(a,2*ind+1,M+1,R); pull(ind);
  }
  void push(int ind, int L, int R) {
    if (L == R) return;
    F0R(i,2)
      if (mx[2*ind^i][0] > mx[ind][0]) {
        sum[2*ind^i] -= (ll)maxCnt[2*ind^i]*
                (mx[2*ind^i][0]-mx[ind][0]);
        mx[2*ind^i][0] = mx[ind][0];
      }
  }
  void upd(int x, int y, int t, int ind = 1, int L = 0, int R =
    ↪ -1) {
    if (R == -1) R += N;
    if (R < x || y < L || mx[ind][0] <= t) return;
    push(ind,L,R);
    if (x <= L && R <= y && mx[ind][1] < t) {
      sum[ind] -= (ll)maxCnt[ind]*(mx[ind][0]-t);
      mx[ind][0] = t;
      return;
    }
    if (L == R) return;
    int M = (L+R)/2;
    upd(x,y,t,2*ind,L,M); upd(x,y,t,2*ind+1,M+1,R); pull(ind);
  }
  ll qsum(int x, int y, int ind = 1, int L = 0, int R = -1) {
    if (R == -1) R += N;
    if (R < x || y < L) return 0;
    push(ind,L,R);
    if (x <= L && R <= y) return sum[ind];
    int M = (L+R)/2;
    return qsum(x,y,2*ind,L,M)+qsum(x,y,2*ind+1,M+1,R);
  }
  int qmax(int x, int y, int ind = 1, int L = 0, int R = -1) {
    if (R == -1) R += N;
    if (R < x || y < L) return -1;
    push(ind,L,R);
    if (x <= L && R <= y) return mx[ind][0];
    int M = (L+R)/2;
    return max(qmax(x,y,2*ind,L,M), qmax(x,y,2*ind+1,M+1,R));
  }
};
```

PSeg.h

Description: Persistent min segtree with lazy updates. Unlike typical lazy segtree, assumes that `lazy[cur]` is included in `val[cur]` before propagating `cur`.
Time: $\mathcal{O}(\log N)$ ee77e6, 58 lines

```
template<class T, int SZ> struct pseg {
  static const int LIMIT = 10000000; // adjust
  int l[LIMIT], r[LIMIT], nex = 0;
  T val[LIMIT], lazy[LIMIT];
  int copy(int cur) {
    int x = nex++;
    val[x] = val[cur], l[x] = l[cur], r[x] = r[cur], lazy[x] =
      ↪lazy[cur];
    return x;
  }
  T comb(T a, T b) { return min(a,b); }
  void pull(int x) { val[x] = comb(val[l[x]],val[r[x]]); }
  void push(int cur, int L, int R) {
    if (!lazy[cur]) return;
    if (L != R) {
      l[cur] = copy(l[cur]);
      val[l[cur]] += lazy[cur], lazy[l[cur]] += lazy[cur];
      r[cur] = copy(r[cur]);
      val[r[cur]] += lazy[cur], lazy[r[cur]] += lazy[cur];
    }
    lazy[cur] = 0;
  }
  T query(int cur, int lo, int hi, int L, int R) {
    if (lo <= L && R <= hi) return val[cur];
    if (R < lo || hi < L) return INF;
    int M = (L+R)/2;
    return lazy[cur]+comb(query(l[cur],lo,hi,L,M),
              query(r[cur],lo,hi,M+1,R));
  }
  int upd(int cur, int lo, int hi, T v, int L, int R) {
    if (R < lo || hi < L) return cur;
    int x = copy(cur);
    if (lo <= L && R <= hi) {
      val[x] += v, lazy[x] += v;
      return x;
    }
    push(x,L,R);
    int M = (L+R)/2;
    l[x] = upd(l[x],lo,hi,v,L,M);
    r[x] = upd(r[x],lo,hi,v,M+1,R);
    pull(x); return x;
  }
  int build(vector<T>& arr, int L, int R) {
    int cur = nex++;
    if (L == R) {
      if (L < sz(arr)) val[cur] = arr[L];
      return cur;
    }
    int M = (L+R)/2;
    l[cur] = build(arr,L,M), r[cur] = build(arr,M+1,R);
    pull(cur); return cur;
  }
  vi loc;
  void upd(int lo, int hi, T v) {
    loc.pb(upd(loc.back(),lo,hi,v,0,SZ-1)); }
  T query(int ti, int lo, int hi) {
    return query(loc[ti],lo,hi,0,SZ-1); }
  void build(vector<T>& arr) { loc.pb(build(arr,0,SZ-1)); }
};
```

Treap.h

Description: easy BBST, use split and merge to implement insert and delete
Time: $\mathcal{O}(\log N)$ b45b6a, 72 lines

```
typedef struct tnode* pt;
struct tnode {
  int pri, val; pt c[2]; // essential
  int sz; ll sum; // for range queries
  bool flip; // lazy update
  tnode (int _val) {
    pri = rand()+(rand()<<15); val = _val; c[0] = c[1] = NULL;
    sz = 1; sum = val;
    flip = 0;
  }
};

int getsz(pt x) { return x?x->sz:0; }
ll getsum(pt x) { return x?x->sum:0; }
pt prop(pt x) {
  if (!x || !x->flip) return x;
  swap(x->c[0],x->c[1]);
  x->flip = 0;
  F0R(i,2) if (x->c[i]) x->c[i]->flip ^= 1;
  return x;
}
pt calc(pt x) {
  assert(!x->flip);
  prop(x->c[0]), prop(x->c[1]);
  x->sz = 1+getsz(x->c[0])+getsz(x->c[1]);
  x->sum = x->val+getsum(x->c[0])+getsum(x->c[1]);
  return x;
}
void tour(pt x, vi& v) {
  if (!x) return;
  prop(x);
  tour(x->c[0],v); v.pb(x->val); tour(x->c[1],v);
}

pair<pt,pt> split(pt t, int v) { // >= v goes to the right
  if (!t) return {t,t};
  prop(t);
  if (t->val >= v) {
    auto p = split(t->c[0], v); t->c[0] = p.s;
    return {p.f,calc(t)};
  } else {
    auto p = split(t->c[1], v); t->c[1] = p.f;
    return {calc(t),p.s};
  }
}
pair<pt,pt> splitsz(pt t, int sz) { // sz nodes go to left
  if (!t) return {t,t};
  prop(t);
  if (getsz(t->c[0]) >= sz) {
    auto p = splitsz(t->c[0],sz); t->c[0] = p.s;
    return {p.f,calc(t)};
  } else {
    auto p = splitsz(t->c[1],sz-getsz(t->c[0])-1); t->c[1] = p.
      ↪f;
    return {calc(t),p.s};
  }
}
pt merge(pt l, pt r) {
  if (!l || !r) return l ? l : r;
  prop(l), prop(r);
  pt t;
  if (l->pri > r->pri) l->c[1] = merge(l->c[1],r), t = l;
  else r->c[0] = merge(l,r->c[0]), t = r;
  return calc(t);
}
pt ins(pt x, int v) { // insert v
  auto a = split(x,v), b = split(a.s,v+1);
  return merge(a.f,merge(new tnode(v),b.s));
}
pt del(pt x, int v) { // delete v
  auto a = split(x,v), b = split(a.s,v+1);
  return merge(a.f,b.s);
}
```

3.3 2D Range Queries

OffBit2D.h

Description: offline 2D binary indexed tree, supports point update and rectangle sum queries
Memory: $\mathcal{O}(N \log N)$
Time: $\mathcal{O}(N \log^2 N)$ 4d90a6, 57 lines

```
template<class T, int SZ> struct OffBIT2D {
  bool mode = 0; // mode = 1 -> initialized
  vpi todo;
  int cnt[SZ], st[SZ];
  vi val, bit;
```

```
  void init() {
    assert(!mode); mode = 1;
    int lst[SZ]; FOR(i,SZ) lst[i] = cnt[i] = 0;
    sort(all(todo),[](const pi& a, const pi& b) {
      return a.s < b.s; });
    trav(t,todo) for (int X = t.f; X < SZ; X += X&-X)
      if (lst[X] != t.s) {
        lst[X] = t.s;
        cnt[X] ++;
      }
    int sum = 0;
    FOR(i,SZ) {
      st[i] = sum; lst[i] = 0; // stores start index for each x
      sum += cnt[i];
    }
    val.rsz(sum); bit.rsz(sum); // store BITs in single vector
    trav(t,todo) for (int X = t.f; X < SZ; X += X&-X)
      if (lst[X] != t.s) {
        lst[X] = t.s;
        val[st[X]++] = t.s;
      }
  }
  int rank(int y, int l, int r) {
    return ub(begin(val)+l,begin(val)+r,y)-begin(val)-1;
  }
  void UPD(int x, int y, int t) {
    int z = st[x]-cnt[x]; // BIT covers range from z to st[x]-1
    for (y = rank(y,z,st[x]); y <= cnt[x]; y += y&-y)
      bit[z+y-1] += t;
  }
  void upd(int x, int y, int t = 1) { // x-coordinate in [1,SZ)
    if (!mode) todo.pb({x,y});
    else {
      for (; x < SZ; x += x&-x) UPD(x,y,t);
    }
  }
  int QUERY(int x, int y) {
    int z = st[x]-cnt[x], ans = 0;
    for (y = rank(y,z,st[x]); y; y -= y&-y)
      ans += bit[z+y-1];
    return ans;
  }
  int query(int x, int y) {
    assert(mode);
    int t = 0; for (; x; x -= x&-x) t += QUERY(x,y);
    return t;
  }
  int query(int lox, int hix, int loy, int hiy) {
    return query(hix,hiy)-query(lox-1,hiy)
      -query(hix,loy-1)+query(lox-1,loy-1);
  }
};
```

Number Theory (4)

4.1 Modular Arithmetic

ModInt.h

Description: modular arithmetic operations

"CppIO.h" 6d49db, 48 lines

```
typedef decay<decltype(MOD)>::type T;
struct mi {
  T val;
  explicit operator T() const { return val; }
  mi() { val = 0; }
  mi(const ll& v) {
    val = (-MOD <= v && v <= MOD) ? v : v % MOD;
    if (val < 0) val += MOD;
  }
  // friend ostream& operator<<(ostream& os, const mi& a) {
    // return os << a.val; }
  friend void pr(const mi& a) { pr(a.val); }
  friend void re(mi& a) { ll x; re(x); a = mi(x); }

  friend bool operator==(const mi& a, const mi& b) {
    return a.val == b.val; }
  friend bool operator!=(const mi& a, const mi& b) {
    return !(a == b); }
  friend bool operator<(const mi& a, const mi& b) {
    return a.val < b.val; }

  mi operator-() const { return mi(-val); }
  mi& operator+=(const mi& m) {
    if ((val += m.val) >= MOD) val -= MOD;
    return *this; }
  mi& operator-=(const mi& m) {
    if ((val -= m.val) < 0) val += MOD;
    return *this; }
  mi& operator*=(const mi& m) {
    val = (ll)val*m.val%MOD; return *this; }
  friend mi pow(mi a, ll p) {
    mi ans = 1; assert(p >= 0);
    for (; p; p /= 2, a *= a) if (p&1) ans *= a;
    return ans;
  }
  friend mi inv(const mi& a) {
    assert(a != 0); return pow(a,MOD-2); }
  mi& operator/=(const mi& m) { return (*this) *= inv(m); }

  friend mi operator+(mi a, const mi& b) { return a += b; }
  friend mi operator-(mi a, const mi& b) { return a -= b; }
  friend mi operator*(mi a, const mi& b) { return a *= b; }
  friend mi operator/(mi a, const mi& b) { return a /= b; }
};

typedef pair<mi,mi> pmi;
typedef vector<mi> vmi;
typedef vector<pmi> vpmi;
```

ModFact.h

Description: pre-compute factorial mod inverses for MOD, assumes MOD is prime and $SZ < MOD$

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

caf808, 14 lines

```
vi invs, fac, ifac;
void genFac(int SZ) {
  invs.rsz(SZ), fac.rsz(SZ), ifac.rsz(SZ);
  invs[1] = fac[0] = ifac[0] = 1;
  FOR(i,2,SZ) invs[i] = MOD-(ll)MOD/i*invs[MOD%i]%MOD;
  FOR(i,1,SZ) {
    fac[i] = (ll)fac[i-1]*i%MOD;
    ifac[i] = (ll)ifac[i-1]*invs[i]%MOD;
  }
}
ll comb(int a, int b) {
  if (a < b || b < 0) return 0;
  return (ll)fac[a]*ifac[b]%MOD*ifac[a-b]%MOD;
}
```

ModMulLL.h

Description: Multiply two 64-bit integers mod another if 128-bit is not available. modMul is equivalent to `(ul)(__int128(a)*b%mod)`. Works for $0 \le a, b < mod < 2^{63}$.

cc0f9d, 12 lines

```
typedef unsigned long long ul;
ul modMul(ul a, ul b, const ul mod) {
  ll ret = a*b-mod*(ul)((ld)a*b/mod);
  return ret+((ret<0)-(ret>=(ll)mod))*mod;
}
ul modPow(ul a, ul b, const ul mod) {
  if (b == 0) return 1;
  ul res = modPow(a,b/2,mod);
  res = modMul(res,res,mod);
  if (b&1) return modMul(res,a,mod);
  return res;
}
```

ModSqrt.h

Description: square root of integer mod a prime

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

"ModInt.h" 4e4cb0, 15 lines

```
T sqrt(mi a) {
  mi p = pow(a,(MOD-1)/2);
  if (p != 1) return p == 0 ? 0 : -1; // check if 0 or no sqrt
  T s = MOD-1; int e = 0; while (s % 2 == 0) s /= 2, e ++;
  // find non-square residue
  mi n = 1; while (pow(n,(MOD-1)/2) == 1) n = (T)(n)+1;
  mi x = pow(a,(s+1)/2), b = pow(a,s), g = pow(n,s);
  int r = e;
  while (1) {
    mi B = b; int m = 0; while (B != 1) B *= B, m ++;
    if (m == 0) return min((T)x,MOD-(T)x);
    FOR(i,r-m-1) g *= g;
    x *= g; g *= g; b *= g; r = m;
  }
}
```

ModSum.h

Description: divsum computes $\sum_{i=0}^{to-1}\left\lfloor\frac{ki+c}{m}\right\rfloor$, modsum defined similarly

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

50ee96, 13 lines

```
typedef unsigned long long ul;

ul sumsq(ul to) { return (to-1)*to/2; } // sum of 0..to-1
ul divsum(ul to, ul c, ul k, ul m) {
  ul res = k/m*sumsq(to)+c/m*to;
  k %= m; c %= m; if (!k) return res;
  ul to2 = (to*k+c)/m;
  return res+(to-1)*to2-divsum(to2,m-1-c,m,k);
}
ll modsum(ul to, ll c, ll k, ll m) {
  c = (c%m+m)%m, k = (k%m+m)%m;
  return to*c+k*sumsq(to)-m*divsum(to,c,k,m);
}
```

4.2 Primality

4.2.1 Primes

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

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

4.2.2 Divisors

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

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 $n < 1e19$.

Let $s(x) = \sum_{i=1}^{x} \phi(i)$. Then

$$s(n) = \frac{n(n+1)}{2} - \sum_{i=2}^{n} s\left(\left\lfloor \frac{n}{i} \right\rfloor\right).$$

PrimeSieve.h
Description: tests primality up to SZ
Time: $\mathcal{O}(SZ \log\log SZ)$ or $\mathcal{O}(SZ)$
2627d0, 22 lines

```
template<int SZ> struct Sieve {
  bitset<SZ> pri;
  vi pr;
  Sieve() {
    pri.set(); pri[0] = pri[1] = 0;
    for (int i = 4; i < SZ; i += 2) pri[i] = 0;
    for (int i = 3; i*i < SZ; i += 2) if (pri[i])
      for (int j = i*i; j < SZ; j += i*2) pri[j] = 0;
    FOR(i,SZ) if (pri[i]) pr.pb(i);
  }
  int sp[SZ];
  void linear() { // above is faster
    memset(sp,0,sizeof sp);
    FOR(i,2,SZ) {
      if (sp[i] == 0) { sp[i] = i; pr.pb(i); }
      trav(p,pr) {
        if (p > sp[i] || i*p >= SZ) break;
        sp[i*p] = p;
      }
    }
  }
};
```

FactorFast.h
Description: Factors integers up to 2^{60}
Time: $\mathcal{O}\left(N^{1/4}\right)$ gcd calls, less for numbers with small factors
"PrimeSieve.h", "ModMulLL.h" 8c89cc, 46 lines

```
Sieve<1<<20> S; // primes up to N^{1/3}

bool millerRabin(ll p) { // test primality
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ll s = p-1; while (s % 2 == 0) s /= 2;
  FOR(i,30) { // strong liar with probability <= 1/4
    ll a = rand()%(p-1)+1, tmp = s;
    ll mod = modPow(a,tmp,p);
    while (tmp != p-1 && mod != 1 && mod != p-1) {
      mod = modMul(mod,mod,p);
      tmp *= 2;
    }
    if (mod != p-1 && tmp%2 == 0) return false;
  }
  return true;
}

ll f(ll a, ll n, ll &has) { return (modMul(a,a,n)+has)%n; }
vpl pollardsRho(ll d) {
  vpl res;
  auto& pr = S.pr;
  for (int i = 0; i < sz(pr) && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      int co = 0; while (d % pr[i] == 0) d /= pr[i], co ++;
      res.pb({pr[i],co});
    }
  if (d > 1) { // d is now a product of at most 2 primes.
    if (millerRabin(d)) res.pb({d,1});
    else while (1) {
      ll has = rand()%2321+47;
      ll x = 2, y = 2, c = 1;
      for (; c == 1; c = __gcd(abs(x-y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      } // should cycle in ~sqrt(smallest nontrivial divisor)
      if (c != d) {
        d /= c; if (d > c) swap(d,c);
        if (c == d) res.pb({c,2});
        else res.pb({c,1}), res.pb({d,1});
        break;
      }
    }
  }
  return res;
}
```

4.3 GCD

4.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

Euclid.h
Description: euclid finds $\{x, y\}$ such that $ax + by = \gcd(a,b)$ such that $|ax|, |by| \leq \frac{ab}{\gcd(a,b)}$, should work for $a, b < 2^{62}$
Time: $\mathcal{O}(\log ab)$
338527, 9 lines

```
pl euclid(ll a, ll b) {
  if (!b) return {1,0};
  pl p = euclid(b,a%b);
  return {p.s,p.f-a/b*p.s};
}
ll invGeneral(ll a, ll b) {
  pl p = euclid(a,b); assert(p.f*a+p.s*b == 1); // gcd is 1
  return p.f+(p.f<0)*b;
}
```

CRT.h
Description: Chinese Remainder Theorem, combine $a.f \pmod{a.s}$ and $b.f \pmod{b.s}$ into something $\pmod{\operatorname{lcm}(a.s, b.s)}$, should work for $ab < 2^{62}$
"Euclid.h" a7ebbe, 10 lines

```
pl solve(pl a, pl b) {
  if (a.s < b.s) swap(a,b);
  ll x,y; tie(x,y) = euclid(a.s,b.s);
  ll g = a.s*x+b.s*y, l = a.s/g*b.s;
  if ((b.f-a.f)%g) return {-1,-1}; // no solution
  // ?*a.s+a.f \equiv b.f \pmod{b.s}
  // ?=(b.f-a.f)/g*(a.s/g)^{-1} \pmod{b.s/g}
  x = (b.f-a.f)%b.s*x%b.s/g*a.s+a.f;
  return {x+(x<0)*l,l};
}
```

4.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

Combinatorial (5)

5.1 Permutations

5.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

n	11	12	13	14	15	16	17
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14

n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

IntPerm.h
Description: Unused. Convert permutation of $\{0, 1, \ldots, N-1\}$ to integer in $[0, N!)$ and back.
Usage: assert(encode(decode(5,37)) == 37);
Time: $\mathcal{O}(N)$
f295dd, 19 lines

```
vi decode(int n, int a) {
  vi el(n), b; iota(all(el),0);
  FOR(i,n) {
    int z = a%sz(el);
    b.pb(el[z]); a /= sz(el);
    swap(el[z],el.back()); el.pop_back();
  }
  return b;
}
int encode(vi b) {
  int n = sz(b), a = 0, mul = 1;
  vi pos(n); iota(all(pos),0); vi el = pos;
  FOR(i,n) {
    int z = pos[b[i]]; a += mul*z; mul *= sz(el);
    swap(pos[el[z]],pos[el.back()]);
    swap(el[z],el.back()); el.pop_back();
  }
  return a;
}
```

5.1.2 Cycles

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

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

5.1.3 Derangements

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

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

5.1.4 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

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

PermGroup.h

Description: Used only once. Schreier-Sims lets you add a permutation to a group, count number of permutations in a group, and test whether a permutation is a member of a group.

Time: ?

590e00, 50 lines

```
int n;
vi inv(vi v) { vi V(sz(v)); FOR(i,sz(v)) V[v[i]] = i; return V;
  ↪ }
vi id() { vi v(n); iota(all(v),0); return v; }
vi operator*(const vi& a, const vi& b) {
  vi c(sz(a)); FOR(i,sz(a)) c[i] = a[b[i]];
  return c;
}

const int N = 15;
struct Group {
  bool flag[N];
  vi sigma[N]; // sigma[t][k] = t, sigma[t][x] = x if x > k
  vector<vi> gen;
  void clear(int p) {
    memset(flag,0, sizeof flag);
    flag[p] = 1; sigma[p] = id();
    gen.clear();
  }
} g[N];

bool check(const vi& cur, int k) {
  if (!k) return 1;
  int t = cur[k];
  return g[k].flag[t] ? check(inv(g[k].sigma[t])*cur,k-1) : 0;
}
void updateX(const vi& cur, int k);
void ins(const vi& cur, int k) {
  if (check(cur,k)) return;
  g[k].gen.pb(cur);
  FOR(i,n) if (g[k].flag[i]) updateX(cur*g[k].sigma[i],k);
}
void updateX(const vi& cur, int k) {
  int t = cur[k]; // if flag, fixes k -> k
  if (g[k].flag[t]) ins(inv(g[k].sigma[t])*cur,k-1);
  else {
    g[k].flag[t] = 1, g[k].sigma[t] = cur;
    trav(x,g[k].gen) updateX(x*cur,k);
  }
}

ll order(vector<vi> gen) {
  assert(sz(gen)); n = sz(gen[0]); FOR(i,n) g[i].clear(i);
  trav(a,gen) ins(a,n-1); // insert perms into group one by one
  ll tot = 1;
  FOR(i,n) {
    int cnt = 0; FOR(j,i+1) cnt += g[i].flag[j];
    tot *= cnt;
  }
  return tot;
}
```

5.2 Partitions and subsets

5.2.1 Partition function

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

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

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

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

5.2.2 Lucas' Theorem

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

5.3 General purpose numbers

5.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

5.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

5.3.3 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. $\pi(j) > j$.

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

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

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

5.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

5.3.5 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,

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

5.3.6 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: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

5.3.7 Catalan numbers

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

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

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

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

5.4 Matroid

MatroidIntersect.h

Description: computes a set of maximum size which is independent in both graphic and colorful matroids, aka a spanning forest where no two edges are of the same color

Time: $\mathcal{O}\left(GI^{1.5}\right)$ calls to oracles, where G is the size of the ground set and I is the size of the independent set

"DSU.h" a78eca, 102 lines

```
int R;
map<int,int> m;

struct Element {
  pi ed;
  int col;
  bool in_indep_set = 0;
  int indep_set_pos;
  Element(int u, int v, int c) { ed = {u,v}; col = c; }
};
vi indep_set;
vector<Element> ground_set;
bool col_used[300];

struct GBasis {
  DSU D;
  void reset() { D.init(sz(m)); }
  void add(pi v) { assert(D.unite(v.f,v.s)); }
  bool indep_with(pi v) { return !D.sameSet(v.f,v.s); }
};
GBasis basis, basis_wo[300];

bool graph_oracle(int inserted) {
  return basis.indep_with(ground_set[inserted].ed);
}
bool graph_oracle(int inserted, int removed) {
  int wi = ground_set[removed].indep_set_pos;
  return basis_wo[wi].indep_with(ground_set[inserted].ed);
}
void prepare_graph_oracle() {
  basis.reset();
  FOR(i,sz(indep_set)) basis_wo[i].reset();
  FOR(i,sz(indep_set)) {
    pi v = ground_set[indep_set[i]].ed; basis.add(v);
    FOR(j,sz(indep_set)) if (i != j) basis_wo[j].add(v);
  }
}
bool colorful_oracle(int ins) {
  ins = ground_set[ins].col;
  return !col_used[ins];
}
bool colorful_oracle(int ins, int rem) {
  ins = ground_set[ins].col;
  rem = ground_set[rem].col;
  return !col_used[ins] || ins == rem;
}
void prepare_colorful_oracle() {
  FOR(i,R) col_used[i] = 0;
  trav(t,indep_set) col_used[ground_set[t].col] = 1;
}

bool augment() {
  prepare_graph_oracle();
  prepare_colorful_oracle();
  vi par(sz(ground_set),MOD);
  queue<int> q;
  FOR(i,sz(ground_set)) if (colorful_oracle(i)) {
    assert(!ground_set[i].in_indep_set);
    par[i] = -1; q.push(i);
  }
  int lst = -1;
  while (sz(q)) {
    int cur = q.front(); q.pop();
    if (ground_set[cur].in_indep_set) {
      FOR(to,sz(ground_set)) if (par[to] == MOD) {
        if (!colorful_oracle(to,cur)) continue;
        par[to] = cur; q.push(to);
      }
    } else {
      if (graph_oracle(cur)) { lst = cur; break;  }
      trav(to,indep_set) if (par[to] == MOD) {
        if (!graph_oracle(cur,to)) continue;
        par[to] = cur; q.push(to);
      }
    }
  }
  if (lst == -1) return 0;
  do {
    ground_set[lst].in_indep_set ^= 1;
    lst = par[lst];
  } while (lst != -1);
  indep_set.clear();
  FOR(i,sz(ground_set)) if (ground_set[i].in_indep_set) {
    ground_set[i].indep_set_pos = sz(indep_set);
    indep_set.pb(i);
  }
  return 1;
}
void solve() {
  cin >> R;
  m.clear(); ground_set.clear(); indep_set.clear();
  FOR(i,R) {
    int a,b,c,d; cin >> a >> b >> c >> d;
    ground_set.pb(Element(a,b,i));
    ground_set.pb(Element(c,d,i));
    m[a] = m[b] = m[c] = m[d] = 0;
  }
  int co = 0;
  trav(t,m) t.s = co++;
  trav(t,ground_set) t.ed.f = m[t.ed.f], t.ed.s = m[t.ed.s];
  while (augment()); // keep increasing size of indep set
}
```

Numerical (6)

6.1 Matrix

Matrix.h

Description: 2D matrix operations

33ea2d, 34 lines

```
template<class T> struct Mat {
  int r,c;
  vector<vector<T>> d;
  Mat(int _r, int _c) : r(_r), c(_c) {
    d.assign(r,vector<T>(c)); }
  Mat() : Mat(0,0) {}
  Mat(const vector<vector<T>>& _d) : r(sz(_d)), c(sz(_d[0])) {
     ↪d = _d; }
  friend void pr(const Mat& m) { pr(m.d); }
  Mat& operator+=(const Mat& m) {
    assert(r == m.r && c == m.c);
    FOR(i,r) FOR(j,c) d[i][j] += m.d[i][j];
    return *this;
  }
  Mat& operator-=(const Mat& m) {
    assert(r == m.r && c == m.c);
    FOR(i,r) FOR(j,c) d[i][j] -= m.d[i][j];
    return *this;
  }
  Mat operator*(const Mat& m) {
    assert(c == m.r); Mat x(r,m.c);
    FOR(i,r) FOR(j,c) FOR(k,m.c)
      x.d[i][k] += d[i][j]*m.d[j][k];
    return x;
  }
  Mat operator+(const Mat& m) { return Mat(*this)+=m; }
  Mat operator-(const Mat& m) { return Mat(*this)-=m; }
  Mat& operator*=(const Mat& m) { return *this = (*this)*m; }
  friend Mat pow(Mat m, ll p) {
    assert(m.r == m.c);
    Mat res(m.r,m.c); FOR(i,m.r) res.d[i][i] = 1;
    for (; p; p /= 2, m *= m) if (p&1) res *= m;
    return res;
  }
};
```

MatrixInv.h

Description: Uses gaussian elimination to convert into reduced row echelon form and calculates determinant. For determinant via arbitrary modulos, use a modified form of the Euclidean algorithm because modular inverse may not exist. If you have computed $A^{-1} \pmod{p^k}$, then the inverse $\pmod{p^{2k}}$ is $A^{-1}(2I - AA^{-1})$.

Time: $\mathcal{O}\left(N^3\right)$, determinant of 1000×1000 matrix of modular ints in 1 second if you reduce # of operations by half

"Matrix.h" 879b16, 40 lines

```
const ld EPS = 1e-12;
int getRow(Mat<ld>& m, int n, int i, int nex) {
    pair<ld,int> bes = {0,-1};
    FOR(j,nex,n) ckmax(bes,{abs(m.d[j][i]),j});
    return bes.f < EPS ? -1 : bes.s;
}
int getRow(Mat<mi>& m, int n, int i, int nex) {
    FOR(j,nex,n) if (m.d[j][i] != 0) return j;
    return -1;
}

template<class T> pair<T,int> gauss(Mat<T>& m) {
    int n = m.r, rank = 0, nex = 0;
    T prod = 1;
    FOR(i,n) {
        int row = getRow(m,n,i,nex);
        if (row == -1) { prod = 0; continue; }
        if (row != nex) prod *= -1, swap(m.d[row],m.d[nex]);
        prod *= m.d[nex][i]; rank ++;
        auto x = 1/m.d[nex][i]; FOR(k,i,m.c) m.d[nex][k] *= x;
        FOR(j,n) if (j != nex) {
            auto v = m.d[j][i]; if (v == 0) continue;
            FOR(k,i,m.c) m.d[j][k] -= v*m.d[nex][k];
        }
        nex ++;
```

```
    }
    return {prod,rank};
}
template<class T> Mat<T> inv(Mat<T> m) {
    assert(m.r == m.c);
    int n = m.r; Mat<T> x(n,2*n);
    FOR(i,n) {
        x.d[i][i+n] = 1;
        FOR(j,n) x.d[i][j] = m.d[i][j];
    }
    if (gauss(x).s != n) return Mat<T>();
    Mat<T> res(n,n);
    FOR(i,n) FOR(j,n) res.d[i][j] = x.d[i][j+n];
    return res;
}
```

MatrixTree.h

Description: Kirchhoff's Matrix Tree Theorem. Given adjacency matrix, calculates # of spanning trees.

"MatrixInv.h", "ModInt.h" 5b0a26, 12 lines

```
mi numSpan(Mat<mi> m) {
  int n = m.r;
  Mat<mi> res(n-1,n-1);
  FOR(i,n) FOR(j,i+1,n) {
    mi ed = m.d[i][j]; res.d[i][i] += ed;
    if (j != n-1) {
      res.d[j][j] += ed;
      res.d[i][j] -= ed, res.d[j][i] -= ed;
    }
  }
  return gauss(res).f;
}
```

6.2 Polynomials

VecOp.h

Description: polynomial operations using vectors

59e9d1, 71 lines

```
namespace VecOp {
  template<class T> vector<T> rev(vector<T> v) {
    reverse(all(v)); return v; }
  template<class T> vector<T> shift(vector<T> v, int x) {
    v.insert(begin(v),x,0); return v; }
  template<class T> vector<T>& remLead(vector<T>& v) {
    while (sz(v) && v.back() == 0) v.pop_back();
    return v; }
  template<class T> T eval(const vector<T>& v, const T& x) {
    T res = 0; ROF(i,sz(v)) res = x*res+v[i];
    return res; }
  template<class T> vector<T> dif(const vector<T>& v)  {
    if (!sz(v)) return v;
    vector<T> res(sz(v)-1); FOR(i,1,sz(v)) res[i-1] = i*v[i];
    return res;
  }
  template<class T> vector<T> integ(const vector<T>& v)  {
    vector<T> res(sz(v)+1);
    FOR(i,sz(v)) res[i+1] = v[i]/(i+1);
    return res;
  }

  template<class T> vector<T>& operator+=(vector<T>& l, const
    ↪vector<T>& r) {
    l.rsz(max(sz(l),sz(r))); FOR(i,sz(r)) l[i] += r[i];
    return l; }
  template<class T> vector<T>& operator-=(vector<T>& l, const
    ↪vector<T>& r) {
    l.rsz(max(sz(l),sz(r))); FOR(i,sz(r)) l[i] -= r[i];
    return l; }
  template<class T> vector<T>& operator*=(vector<T>& l, const T
    ↪& r) {
    trav(t,l) t *= r; return l;  }
  template<class T> vector<T>& operator/=(vector<T>& l, const T
    ↪& r) {
    trav(t,l) t /= r; return l; }

  template<class T> vector<T> operator+(vector<T> l, const
    ↪vector<T>& r) { return l += r; }
  template<class T> vector<T> operator-(vector<T> l, const
    ↪vector<T>& r) { return l -= r; }
  template<class T> vector<T> operator*(vector<T> l, const T& r
    ↪) { return l *= r; }
  template<class T> vector<T> operator*(const T& r, const
    ↪vector<T>& l) { return l*r; }
  template<class T> vector<T> operator/(vector<T> l, const T& r
    ↪) { return l /= r;  }
  template<class T> vector<T> operator*(const vector<T>& l,
    ↪const vector<T>& r) {
    if (min(sz(l),sz(r)) == 0) return {};
    vector<T> x(sz(l)+sz(r)-1);
    FOR(i,sz(l)) FOR(j,sz(r)) x[i+j] += l[i]*r[j];
    return x;
  }
  template<class T> vector<T>& operator*=(vector<T>& l, const
    ↪vector<T>& r) { return l = l*r; }

  template<class T> pair<vector<T>,vector<T>> qr(vector<T> a,
    ↪vector<T> b) { // quotient and remainder
    assert(sz(b)); auto B = b.back(); assert(B != 0);
    B = 1/B; trav(t,b) t *= B;
    remLead(a); vector<T> q(max(sz(a)-sz(b)+1,0));
    while (sz(a) >= sz(b)) {
      q[sz(a)-sz(b)] = a.back();
      a -= a.back()*shift(b,sz(a)-sz(b));
      remLead(a);
    }
    trav(t,q) t *= B;
    return {q,a};
  }
  template<class T> vector<T> quo(const vector<T>& a, const
    ↪vector<T>& b) { return qr(a,b).f; }
  template<class T> vector<T> rem(const vector<T>& a, const
    ↪vector<T>& b) { return qr(a,b).s; }
  template<class T> vector<T> interpolate(vector<pair<T,T>> v)
    ↪{
    vector<T> ret, prod = {1};
    FOR(i,sz(v)) prod *= vector<T>({-v[i].f,1});
    FOR(i,sz(v)) {
      T todiv = 1; FOR(j,sz(v)) if (i != j) todiv *= v[i].f-v[j
        ↪].f;
      ret += qr(prod,{-v[i].f,1}).f*(v[i].s/todiv);
    }
    return ret;
  }
}
using namespace VecOp;
```

PolyRoots.h

Description: Finds the real roots of a polynomial.

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

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

"VecOp.h" fbe593, 19 lines

```
vd polyRoots(vd p, ld xmin, ld xmax) {
  if (sz(p) == 2) { return {-p[0]/p[1]}; }
  auto dr = polyRoots(dif(p),xmin,xmax);
  dr.pb(xmin-1); dr.pb(xmax+1); sort(all(dr));
  vd ret;
  FOR(i,sz(dr)-1) {
    auto l = dr[i], h = dr[i+1];
    bool sign = eval(p,l) > 0;
    if (sign ^ (eval(p,h) > 0)) {
      FOR(it,60) { // while (h - l > 1e-8)
        auto m = (l+h)/2, f = eval(p,m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.pb((l+h)/2);
    }
  }
  return ret;
}
```

Karatsuba.h

Description: multiply two polynomials, FFT is usually fine

Time: $\mathcal{O}\left(N^{\log_2 3}\right)$

21f372, 24 lines

```
int size(int s) { return s > 1 ? 32-__builtin_clz(s-1) : 0; }
void karatsuba(ll *a, ll *b, ll *c, ll *t, int n) {
  int ca = 0, cb = 0; FOR(i,n) ca += !!a[i], cb += !!b[i];
  if (min(ca, cb) <= 1500/n) { // few numbers to multiply
    if (ca > cb) swap(a, b);
    FOR(i,n) if (a[i]) FOR(j,n) c[i+j] += a[i]*b[j];
  } else {
    int h = n >> 1;
    karatsuba(a, b, c, t, h); // a0*b0
    karatsuba(a+h, b+h, c+n, t, h); // a1*b1
    FOR(i,h) a[i] += a[i+h], b[i] += b[i+h];
    karatsuba(a, b, t, t+n, h); // (a0+a1)*(b0+b1)
    FOR(i,h) a[i] -= a[i+h], b[i] -= b[i+h];
    FOR(i,n) t[i] -= c[i]+c[i+n];
    FOR(i,n) c[i+h] += t[i], t[i] = 0;
  }
}
vl conv(vl a, vl b) {
  int sa = sz(a), sb = sz(b); if (!sa || !sb) return {};
  int n = 1<<size(max(sa,sb)); a.rsz(n), b.rsz(n);
  vl c(2*n), t(2*n); FOR(i,2*n) t[i] = 0;
  karatsuba(&a[0], &b[0], &c[0], &t[0], n);
  c.rsz(sa+sb-1); return c;
}
```

FFT.h

Description: Multiply two polynomials. For xor convolution don't multiply by roots[ind].

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

"ModInt.h" c2ec1b, 43 lines

```
typedef complex<db> cd;
typedef vector<cd> vcd;
const int MOD = (119 << 23) + 1, root = 3; // = 998244353
// NTT: For p < 2^30 there is also e.g. (5 << 25, 3),
// (7 << 26, 3), (479 << 21, 3) and (483 << 21, 5).
// The last two are > 10^9.

int size(int s) { return s > 1 ? 32-__builtin_clz(s-1) : 0; }
void genRoots(vcd& roots) { // primitive n-th roots of unity
  int n = sz(roots); double ang = 2*PI/n;
  // good way to compute these trig functions more quickly?
  FOR(i,n) roots[i] = cd(cos(ang*i),sin(ang*i));
}
void genRoots(vmi& roots) {
  int n = sz(roots); mi r = pow(mi(root),(MOD-1)/n);
  roots[0] = 1; FOR(i,1,n) roots[i] = roots[i-1]*r;
}
```

```
template<class T> void fft(vector<T>& a, const vector<T>& roots
    ↪, bool inv = 0) {
  int n = sz(a);
  // sort #s from 0 to n-1 by reverse bit representation
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n>>1;
    for (; j&bit; bit >>= 1) j ^= bit;
    j ^= bit; if (i < j) swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len <<= 1)
    for (int i = 0; i < n; i += len)
      FOR(j,len/2) {
        int ind = n/len*j; if (inv && ind) ind = n-ind;
        auto u = a[i+j], v = a[i+j+len/2]*roots[ind];
        a[i+j] = u+v, a[i+j+len/2] = u-v;
      }
  if (inv) { T i = T(1)/T(n); trav(x,a) x *= i; }
}
template<class T> vector<T> mult(vector<T> a, vector<T> b) {
  if (!min(sz(a),sz(b))) return {};
  int s = sz(a)+sz(b)-1, n = 1<<size(s);
  vector<T> roots(n); genRoots(roots);
  a.rsz(n), fft(a,roots); b.rsz(n), fft(b,roots);
  FOR(i,n) a[i] *= b[i];
  fft(a,roots,1); a.rsz(s); return a;
}
```

FFTmod.h

Description: multiply two polynomials with arbitrary MOD ensures precision by splitting in half

Time: ~0.8s when `sz(a)=sz(b)=1<<19`

"FFT.h" a8a6ed, 29 lines

```
vl multMod(const vl& a, const vl& b) {
  if (!min(sz(a),sz(b))) return {};
  int s = sz(a)+sz(b)-1, n = 1<<size(s), cut = sqrt(MOD);
  vcd roots(n); genRoots(roots);
  vcd ax(n), bx(n);
  // ax(x)=a1(x)+i*a0(x)
  FOR(i,sz(a)) ax[i] = cd((int)a[i]/cut, (int)a[i]%cut);
  // bx(x)=b1(x)+i*b0(x)
  FOR(i,sz(b)) bx[i] = cd((int)b[i]/cut, (int)b[i]%cut);
  fft(ax,roots), fft(bx,roots);
  vcd v1(n), v0(n);
  FOR(i,n) {
    int j = (i ? (n-i) : i);
    // v1 = a1*(b1+b0*cd(0,1));
    v1[i] = (ax[i]+conj(ax[j]))*cd(0.5,0)*bx[i];
    // v0 = a0*(b1+b0*cd(0,1));
    v0[i] = (ax[i]-conj(ax[j]))*cd(0,-0.5)*bx[i];
  }
  fft(v1,roots,1), fft(v0,roots,1);
  vl ret(n);
  FOR(i,n) {
    ll V2 = (ll)round(v1[i].real()); // a1*b1
    ll V1 = (ll)round(v1[i].imag())+(ll)round(v0[i].real());
    // a0*b1+a1*b0
    ll V0 = (ll)round(v0[i].imag()); // a0*b0
    ret[i] = ((V2%MOD*cut+V1)%MOD*cut+V0)%MOD;
  }
  ret.rsz(s); return ret;
}
```

PolyInv.h

Description: computes v^{-1} such that $vv^{-1} \equiv 1 \pmod{x^p}$

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

"FFT.h" e69e0c, 12 lines

```
template<class T> vector<T> inv(vector<T> v, int p) {
  v.rsz(p); vector<T> a = {T(1)/v[0]};
  for (int i = 1; i < p; i *= 2) {
    if (2*i > p) v.rsz(2*i);
    auto l = vector<T>(begin(v),begin(v)+i);
    auto r = vector<T>(begin(v)+i,begin(v)+2*i);
    auto c = mult(a,l); c = vector<T>(begin(c)+i,end(c));
    auto b = mult(a*T(-1),mult(a,r)+c); b.rsz(i);
    a.insert(end(a),all(b));
  }
  a.rsz(p); return a;
}
```

PolyDiv.h

Description: For two polys f, g computes q, r such that $f = qg + r$, $\deg(r) < \deg(g)$

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

"PolyInv.h" a70b14, 8 lines

```
template<class T> pair<vector<T>,vector<T>> divi(
  const vector<T>& f, const vector<T>& g) {
  if (sz(f) < sz(g)) return {{},f};
  auto q = mult(inv(rev(g),sz(f)-sz(g)+1),rev(f));
  q.rsz(sz(f)-sz(g)+1); q = rev(q);
  auto r = f-mult(q,g); r.rsz(sz(g)-1);
  return {q,r};
}
```

PolySqrt.h

Description: for p a power of 2, computes ans such that $ans^2 \equiv v \pmod{x^p}$

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

"PolyInv.h" 0063be, 7 lines

```
template<class T> vector<T> sqrt(vector<T> v, int p) {
  assert(v[0] == 1); if (p == 1) return {1};
  v.rsz(p); auto S = sqrt(v,p/2);
  auto ans = S+mult(v,inv(S,p));
  ans.rsz(p); ans *= T(1)/T(2);
  return ans;
}
```

6.3 Misc

LinRec.h

Description: Berlekamp-Massey, computes linear recurrence of order N for sequence of $2N$ terms

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

"VecOp.h", "ModInt.h" 32c214, 32 lines

```
struct LinRec {
  vmi x; // original sequence
  vmi C, rC;
  void init(const vmi& _x) {
    x = _x; int n = sz(x), m = 0;
    vmi B; B = C = {1}; // B is fail vector
    mi b = 1; // B gives 0,0,0,...,b
    FOR(i,n) {
      m ++;
      mi d = x[i]; FOR(j,1,sz(C)) d += C[j]*x[i-j];
      if (d == 0) continue; // recurrence still works
      auto _B = C; C.rsz(max(sz(C),m+sz(B)));
      // subtract recurrence that gives 0,0,0,...,d
      mi coef = d/b; FOR(j,m,m+sz(B)) C[j] -= coef*B[j-m];
      if (sz(_B) < m+sz(B)) { B = _B; b = d; m = 0; }
    }
    rC = C; reverse(all(rC)); // polynomial for getPo
    C.erase(begin(C)); trav(t,C) t *= -1;
    // x[i]=sum_{j=0}^{sz(C)-1}C[j]*x[i-j-1]
  }
  vmi getPo(int n) {
    if (n == 0) return {1};
    vmi x = getPo(n/2); x = rem(x*x,rC);
    if (n&1) { vmi v = {0,1}; x = rem(x*v,rC); }
    return x;
  }
  mi eval(int n) {
    vmi t = getPo(n);
    mi ans = 0; FOR(i,sz(t)) ans += t[i]*x[i];
    return ans;
  }
};
```

Integrate.h

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

693e87, 7 lines

```
// db f(db x) { return x*x+3*x+1; }
db quad(db (*f)(db), db a, db b) {
  const int n = 1000;
  db dif = (b-a)/2/n, tot = f(a)+f(b);
  FOR(i,1,2*n) tot += f(a+i*dif)*(i&1?4:2);
  return tot*dif/3;
}
```

IntegrateAdaptive.h

Description: Fast integration using adaptive Simpson's rule

b48168, 16 lines

```
// db f(db x) { return x*x+3*x+1; }
db simpson(db (*f)(db), db a, db b) {
  db c = (a+b)/2;
  return (f(a)+4*f(c)+f(b))*(b-a)/6;
}
db rec(db (*f)(db), db a, db b, db eps, db S) {
  db c = (a+b)/2;
  db S1 = simpson(f, a, c);
  db S2 = simpson(f, c, b), T = S1+S2;
  if (abs(T-S) <= 15*eps || b-a < 1e-10)
    return T+(T-S)/15;
  return rec(f, a, c, eps/2, S1)+rec(f, c, b, eps/2, S2);
}
db quad(db (*f)(db), db a, db b, db eps = 1e-8) {
  return rec(f,a,b,eps,simpson(f,a,b));
}
```

Simplex.h

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

Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}};`
`vd b = {1,1,-4}, c = {-1,-1}, x;`
`T val = LPSolver(A, b, c).solve(x);`

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

8a2587, 78 lines

```
typedef db T;
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;

#define ltj(X) if (s == -1 || mp(X[j],N[j]) < mp(X[s],N[s])) s=
    ↪j

struct LPSolver {
  int m, n; // # contraints, # variables
  vi N, B;
```

```
  vvd D;
  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      FOR(i,m) FOR(j,n) D[i][j] = A[i][j];
      FOR(i,m) {
        B[i] = n+i, D[i][n] = -1, D[i][n+1] = b[i];
        // B[i]: add basic variable for each constraint,
        // convert ineqs to eqs
        // D[i][n]: artificial variable for testing feasibility
      }
      FOR(j,n) {
        N[j] = j; // non-basic variables, all zero
        D[m][j] = -c[j]; // minimize -c^T x
      }
      N[n] = -1; D[m+1][n] = 1;
    }

  void pivot(int r, int s) { // r = row, c = column
    T *a = D[r].data(), inv = 1/a[s];
    FOR(i,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), binv = b[s]*inv;
      FOR(j,n+2) b[j] -= a[j]*binv;
      // make column corresponding to s all 0s
      b[s] = a[s]*binv; // swap N[s] with B[r]
    }
    // equation for r scaled so x_r coefficient equals 1
    FOR(j,n+2) if (j != s) D[r][j] *= inv;
    FOR(i,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv; swap(B[r], N[s]); // swap basic w/ non-basic
  }
  bool simplex(int phase) {
    int x = m+phase-1;
    while (1) {
      int s = -1; FOR(j,n+1) if (N[j] != -phase) ltj(D[x]);
      // find most negative col for nonbasic (nb) variable
      if (D[x][s] >= -eps) return true;
      // can't get better sol by increasing nb variable,
        ↪terminate
      int r = -1;
      FOR(i,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || mp(D[i][n+1] / D[i][s], B[i])
              < mp(D[r][n+1] / D[r][s], B[r])) r = i;
        // find smallest positive ratio, max increase in
          ↪nonbasic variable
      }
      if (r == -1) return false; // increase N[s] infinitely ->
        ↪ unbounded
      pivot(r,s);
    }
  }

  T solve(vd &x) {
    int r = 0; FOR(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) { // x=0 not feasible, run simplex to
        ↪ find feasible
      pivot(r, n); // N[n] = -1 is artificial variable
      // initially set to smth large
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      // D[m+1][n+1] is max possible value of the negation of
      // artificial variable, optimal value should be zero
      // if exists feasible solution
      FOR(i,m) if (B[i] == -1) { // ?
        int s = 0; FOR(j,1,n+1) ltj(D[i]);
        pivot(i,s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    FOR(i,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

Graphs (7)

7.1 DSU

DSU.h

Description: Disjoint Set Union with path compression. Add edges and test connectivity. Use for Kruskal's minimum spanning tree.
Time: $\mathcal{O}(\alpha(N))$

cc5aa3, 12 lines

```
struct DSU {
  vi e; void init(int n) { e = vi(n,-1); }
  int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }
  bool sameSet(int a, int b) { return get(a) == get(b); }
  int size(int x) { return -e[get(x)]; }
  bool unite(int x, int y) { // union-by-rank
    x = get(x), y = get(y); if (x == y) return 0;
    if (e[x] > e[y]) swap(x,y);
    e[x] += e[y]; e[y] = x;
    return 1;
  }
};
```

ManhattanMST.h

Description: Given N points, returns up to $4N$ edges which are guaranteed 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: $\mathcal{O}(N \log N)$

"DSU.h" 3aa99a, 24 lines

```
vector<pair<int,pi>> manhattanMst(vpi v) {
  vi id(sz(v)); iota(all(id),0);
  vector<pair<int,pi>> ed;
  FOR(k,4) {
    sort(all(id),[&](int i, int j) {
      return v[i].f+v[i].s < v[j].f+v[j].s; });
    map<int,int> sweep;
    trav(i,id) { // find neighbors for first octant
      for (auto it = sweep.lb(-v[i].s);
        it != end(sweep); sweep.erase(it++)) {
        int j = it->s;
        pi d = {v[i].f-v[j].f,v[i].s-v[j].s};
        if (d.s > d.f) break;
        ed.pb({d.f+d.s,{i,j}});
      }
      sweep[-v[i].s] = i;
    }
    trav(p,v) {
      if (k&1) p.f *= -1;
      else swap(p.f,p.s);
    }
  }
  return ed;
}
```

7.2 Trees

LCAjump.h

Description: Calculates least common ancestor in tree with binary jumping. Vertices labeled from 1 to N, R is the root.
Time: $\mathcal{O}(N \log N)$

a5a7dd, 33 lines

```
template<int SZ> struct LCA {
  static const int BITS = 32-__builtin_clz(SZ);
  int N, R = 1;
  vi adj[SZ];
  int par[BITS][SZ], depth[SZ];
  // INITIALIZE
  void addEdge(int u, int v) { adj[u].pb(v), adj[v].pb(u); }
  void dfs(int u, int prev){
    par[0][u] = prev;
    depth[u] = depth[prev]+1;
    trav(v,adj[u]) if (v != prev) dfs(v, u);
  }
  void init(int _N) {
    N = _N; dfs(R, 0);
    FOR(k,1,BITS) FOR(i,1,N+1)
      par[k][i] = par[k-1][par[k-1][i]];
  }
  // QUERY
  int getPar(int a, int b) {
    ROF(k,BITS) if (b&(1<<k)) a = par[k][a];
    return a;
  }
  int lca(int u, int v){
    if (depth[u] < depth[v]) swap(u,v);
    u = getPar(u,depth[u]-depth[v]);
    ROF(k,BITS) if (par[k][u] != par[k][v])
      u = par[k][u], v = par[k][v];
    return u == v ? u : par[0][u];
  }
  int dist(int u, int v) {
    return depth[u]+depth[v]-2*depth[lca(u,v)];
  }
};
```

Centroid.h

Description: The centroid of a tree of size N is a vertex such that after removing it, all resulting subtrees have size at most $\frac{N}{2}$. Can support tree path queries and updates
Time: $\mathcal{O}(N \log N)$

806d6b, 36 lines

```
template<int SZ> struct Centroid {
  vi adj[SZ];
  bool done[SZ];
  int sub[SZ], par[SZ]; // subtree size, current par
  pi cen[SZ]; // immediate centroid anc
  vi dist[SZ]; // dists to all centroid ancs
  void addEdge(int a, int b) { adj[a].pb(b), adj[b].pb(a); }
  void dfs(int x) {
    sub[x] = 1;
    trav(y,adj[x]) if (!done[y] && y != par[x]) {
      par[y] = x; dfs(y);
      sub[x] += sub[y];
    }
  }
  int centroid(int x) {
    par[x] = -1; dfs(x);
    for (int sz = sub[x];;) {
      pi mx = {0,0};
      trav(y,adj[x]) if (!done[y] && y != par[x])
        ckmax(mx,{sub[y],y});
      if (mx.f*2 <= sz) return x;
      x = mx.s;
    }
  }
  void genDist(int x, int p) {
    dist[x].pb(dist[p].back()+1);
    trav(y,adj[x]) if (!done[y] && y != p) genDist(y,x);
  }
  void gen(pi CEN, int x) {
    done[x = centroid(x)] = 1; cen[x] = CEN;
    dist[x].pb(0); int co = 0;
    trav(y,adj[x]) if (!done[y]) genDist(y,x);
```

```
    trav(y,adj[x]) if (!done[y]) gen({x,co++},y);
  }
  void init() { gen({-1,0},1); }
};
```

HLD.h

Description: Heavy-Light Decomposition, add val to verts and query sum in path/subtree

Time: any tree path is split into $\mathcal{O}(\log N)$ parts

"LazySeg.h" 0e5434, 48 lines

```
template<int SZ, bool VALS_IN_EDGES> struct HLD {
  int N; vi adj[SZ];
  int par[SZ], sz[SZ], depth[SZ];
  int root[SZ], pos[SZ]; vi rpos;
  void addEdge(int a, int b) { adj[a].pb(b), adj[b].pb(a); }
  void dfsSz(int v = 1) {
    if (par[v]) adj[v].erase(find(all(adj[v]),par[v]));
    sz[v] = 1;
    trav(u,adj[v]) {
      par[u] = v; depth[u] = depth[v]+1;
      dfsSz(u); sz[v] += sz[u];
      if (sz[u] > sz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }
  void dfsHld(int v = 1) {
    static int t = 0; pos[v] = t++; rpos.pb(v);
    trav(u,adj[v]) {
      root[u] = (u == adj[v][0] ? root[v] : u);
      dfsHld(u);
    }
  }
  void init(int _N) {
    N = _N; par[1] = depth[1] = 0; root[1] = 1;
    dfsSz(); dfsHld();
  }
  LazySeg<ll,SZ> tree;
  template <class BinaryOp>
  void processPath(int u, int v, BinaryOp op) {
    for (; root[u] != root[v]; v = par[root[v]]) {
      if (depth[root[u]] > depth[root[v]]) swap(u, v);
      op(pos[root[v]], pos[v]);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u]+VALS_IN_EDGES, pos[v]);
  }
  void modifyPath(int u, int v, int val) {
    processPath(u, v, [this, &val](int l, int r) {
      tree.upd(l, r, val); });
  }
  void modifySubtree(int v, int val) {
    tree.upd(pos[v]+VALS_IN_EDGES,pos[v]+sz[v]-1,val);
  }
  ll queryPath(int u, int v) {
    ll res = 0; processPath(u, v, [this, &res](int l, int r) {
      res += tree.qsum(l, r); });
    return res;
  }
```

7.2.1 SqrtDecompton

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

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

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

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

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

7.3 DFS Algorithms

SCC.h

Description: Kosaraju's Algorithm, DFS two times to generate SCCs in topological order

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

f53f41, 21 lines

```
template<int SZ> struct SCC {
  int N, comp[SZ];
  vi adj[SZ], radj[SZ], todo, allComp;
  bitset<SZ> visit;
  void addEdge(int a, int b) { adj[a].pb(b), radj[b].pb(a); }
  void dfs(int v) {
    visit[v] = 1;
    trav(w,adj[v]) if (!visit[w]) dfs(w);
    todo.pb(v);
  }
  void dfs2(int v, int val) {
    comp[v] = val;
    trav(w,radj[v]) if (comp[w] == -1) dfs2(w,val);
  }
  void init(int _N) { // fills allComp
    N = _N; FOR(i,N) comp[i] = -1, visit[i] = 0;
    FOR(i,N) if (!visit[i]) dfs(i);
    reverse(all(todo));
    trav(i,todo) if (comp[i] == -1) dfs2(i,i), allComp.pb(i);
  }
};
```

2SAT.h

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

Usage: TwoSat ts;

```
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setVal(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(N); // Returns true iff it is solvable
ts.ans[0..N-1] holds the assigned values to the vars
```

"SCC.h" 6c209d, 36 lines

```
template<int SZ> struct TwoSat {
  SCC<2*SZ> S;
  bitset<SZ> ans;
  int N = 0;
  int addVar() { return N++; }
  void either(int x, int y) {
    x = max(2*x,-1-2*x), y = max(2*y,-1-2*y);
    S.addEdge(x^1,y); S.addEdge(y^1,x);
  }
  void implies(int x, int y) { either(~x,y); }
  void setVal(int x) { either(x,x); }
  void atMostOne(const vi& li) {
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    FOR(i,2,sz(li)) {
      int next = addVar();
      either(cur,~li[i]);
      either(cur,next);
      either(~li[i],next);
      cur = ~next;
    }
    either(cur,~li[1]);
  }
  bool solve(int _N) {
    if (_N != -1) N = _N;
    S.init(2*N);
    for (int i = 0; i < 2*N; i += 2)
      if (S.comp[i] == S.comp[i^1]) return 0;
    reverse(all(S.allComp));
    vi tmp(2*N);
    trav(i,S.allComp) if (tmp[i] == 0)
      tmp[i] = 1, tmp[S.comp[i^1]] = -1;
    FOR(i,N) if (tmp[S.comp[2*i]] == 1) ans[i] = 1;
    return 1;
  }
};
```

EulerPath.h

Description: Eulerian Path for both directed and undirected graphs

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

fd7ad7, 29 lines

```
template<int SZ, bool directed> struct Euler {
  int N, M = 0;
  vpi adj[SZ];
  vpi::iterator its[SZ];
  vector<bool> used;
  void addEdge(int a, int b) {
    if (directed) adj[a].pb({b,M});
    else adj[a].pb({b,M}), adj[b].pb({a,M});
    used.pb(0); M ++;
  }
  vpi solve(int _N, int src = 1) {
    N = _N;
    FOR(i,1,N+1) its[i] = begin(adj[i]);
    vector<pair<pi,int>> ret, s = {{{src,-1},-1}};
    while (sz(s)) {
      int x = s.back().f.f;
      auto& it = its[x], end = adj[x].end();
      while (it != end && used[it->s]) it ++;
      if (it == end) {
        if (sz(ret) && ret.back().f.s != s.back().f.f)
          return {}; // path isn't valid
        ret.pb(s.back()), s.pop_back();
      } else { s.pb({{it->f,x},it->s}); used[it->s] = 1; }
    }
    if (sz(ret) != M+1) return {};
    vpi ans; trav(t,ret) ans.pb({t.f.f,t.s});
    reverse(all(ans)); return ans;
  }
};
```

BCC.h

Description: Biconnected components. To get block-cut tree, create a bipartite graph with the original vertices on the left and a vertex for each BCC on the right. Draw edge $u \leftrightarrow v$ if u is contained within the BCC for v.

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

fe86d5, 36 lines

```
template<int SZ> struct BCC {
  int N;
  vpi adj[SZ], ed;
  void addEdge(int u, int v) {
    adj[u].pb({v,sz(ed)}), adj[v].pb({u,sz(ed)});
    ed.pb({u,v});
  }
  int disc[SZ];
  vi st; vector<vi> bccs; // edges for each bcc
  int bcc(int u, int p = -1) { // return lowest disc
    static int ti = 0;
    disc[u] = ++ti; int low = disc[u];
    int child = 0;
    trav(i,adj[u]) if (i.s != p) {
      if (!disc[i.f]) {
        child ++; st.pb(i.s);
        int LOW = bcc(i.f,i.s); ckmin(low,LOW);
        // if (disc[u] < LOW) -> bridge
        if (disc[u] <= LOW) { // get edges in bcc
          // if (p != -1 || child > 1) -> u is articulation pt
          bccs.eb(); vi& tmp = bccs.back(); // new bcc
          for (bool done = 0; !done; tmp.pb(st.back()),
            st.pop_back()) done |= st.back() == i.s;
        }
      } else if (disc[i.f] < disc[u]) {
        ckmin(low,disc[i.f]); st.pb(i.s);
      }
    }
    return low;
  }
  void init(int _N) {
    N = _N; FOR(i,N) disc[i] = 0;
    FOR(i,N) if (!disc[i]) bcc(i);
    // st should be empty after each iteration
  }
};
```

7.4 Flows & Matching

Konig's Theorem: In a bipartite graph, max matching = min vertex cover.

Dilworth's Theorem: For any partially ordered set, the sizes of the largest antichain and of the smallest chain decomposition are equal. Equivalent to Konig's theorem on the bipartite graph (U, V, E) where $U = V = S$ and (u, v) is an edge when $u < v$.

Dinic.h

Description: Fast flow. After computing flow, edges $\{u, v\}$ such that $level[u] \neq -1$, $level[v] = -1$ are part of min cut.

Time: $\mathcal{O}\left(N^2M\right)$ flow, $\mathcal{O}\left(M\sqrt{N}\right)$ bipartite matching

b096a0, 44 lines

```
template<int SZ> struct Dinic {
  typedef ll F; // flow type
  struct Edge { int to, rev; F flow, cap; };
  int N,s,t;
  vector<Edge> adj[SZ];
  typename vector<Edge>::iterator cur[SZ];
  void addEdge(int u, int v, F cap) {
    assert(cap >= 0); // don't try smth dumb
    Edge a{v, sz(adj[v]), 0, cap}, b{u, sz(adj[u]), 0, 0};
    adj[u].pb(a), adj[v].pb(b);
  }
  int level[SZ];
  bool bfs() { // level = shortest distance from source
    FOR(i,N) level[i] = -1, cur[i] = begin(adj[i]);
    queue<int> q({s}); level[s] = 0;
    while (sz(q)) {
      int u = q.front(); q.pop();
      trav(e,adj[u]) if (level[e.to] < 0 && e.flow < e.cap)
        q.push(e.to), level[e.to] = level[u]+1;
    }
    return level[t] >= 0;
  }
  F sendFlow(int v, F flow) {
    if (v == t) return flow;
    for (; cur[v] != end(adj[v]); cur[v]++) {
      Edge& e = *cur[v];
      if (level[e.to] != level[v]+1 || e.flow == e.cap)
        continue;
      auto df = sendFlow(e.to,min(flow,e.cap-e.flow));
      if (df) { // saturated at least one edge
        e.flow += df; adj[e.to][e.rev].flow -= df;
        return df;
      }
    }
    return 0;
  }
  F maxFlow(int _N, int _s, int _t) {
    N = _N, s = _s, t = _t; if (s == t) return -1;
    F tot = 0;
    while (bfs()) while (auto df = sendFlow(s,
      numeric_limits<F>::max())) tot += df;
    return tot;
  }
};
```

MCMF.h

Description: Minimum-cost maximum flow, assumes no negative cycles. Edges may be negative only during first run of SPFA.

Time: $\mathcal{O}(FM \log M)$ if caps are integers and F is max flow

e87102, 51 lines

```
template<class T> using pqg = priority_queue<T,vector<T>,
  ↪greater<T>>;
template<class T> T poll(pqg<T>& x) {
  T y = x.top(); x.pop(); return y; }

template<int SZ> struct MCMF {
  typedef ll F; typedef ll C;
  struct Edge { int to, rev; F flow, cap; C cost; };
  vector<Edge> adj[SZ];
  void addEdge(int u, int v, F cap, C cost) {
    assert(cap >= 0);
    Edge a{v,sz(adj[v]),0,cap,cost}, b{u,sz(adj[u]),0,0,-cost};
    adj[u].pb(a), adj[v].pb(b);
  }
  int N, s, t;
  pi pre[SZ]; // previous vertex, edge label on path
  pair<C,F> cost[SZ]; // tot cost of path, amount of flow
  C totCost, curCost; F totFlow;
  bool spfa() { // find lowest cost path such that you can send
    ↪ flow through it
    FOR(i,N) cost[i] = {numeric_limits<C>::max(),0};
    cost[s] = {0,numeric_limits<F>::max()};
    pqg<pair<C,int>> todo; todo.push({0,s});
    while (sz(todo)) {
      auto x = poll(todo); if (x.f > cost[x.s].f) continue;
      trav(a,adj[x.s]) if (a.flow < a.cap
        && ckmin(cost[a.to].f,x.f+a.cost)) {
        // if costs are doubles, add some small constant so
        // you don't traverse some ~0-weight cycle repeatedly
        pre[a.to] = {x.s,a.rev};
        cost[a.to].s = min(a.cap-a.flow,cost[x.s].s);
        todo.push({cost[a.to].f,a.to});
      }
    }
    return cost[t].s;
  }
  void backtrack() {
    F df = cost[t].s; totFlow += df;
    curCost += cost[t].f; totCost += curCost*df;
    for (int x = t; x != s; x = pre[x].f) {
      adj[x][pre[x].s].flow -= df;
      adj[pre[x].f][adj[x][pre[x].s].rev].flow += df;
    }
    FOR(i,N) trav(p,adj[i]) p.cost += cost[i].f-cost[p.to].f;
    // all reduced costs non-negative
    // edges on shortest path become 0
  }
  pair<F,C> calc(int _N, int _s, int _t) {
    N = _N; s = _s, t = _t; totFlow = totCost = curCost = 0;
    while (spfa()) backtrack();
    return {totFlow,totCost};
  }
};
```

GomoryHu.h

Description: Returns edges of Gomory-Hu tree. Max flow between pair of vertices of undirected graph is given by min edge weight along tree path. Uses the lemma that for any i, j, k, $\lambda_{ik} \geq \min(\lambda_{ij}, \lambda_{jk})$, where λ_{ij} denotes the flow from i to j.

Time: $\mathcal{O}(N)$ calls to Dinic

"Dinic.h" fd9171, 20 lines

```
template<int SZ> struct GomoryHu {
  vector<pair<pi,int>> ed;
  void addEdge(int a, int b, int c) { ed.pb({{a,b},c}); }
  vector<pair<pi,int>> init(int N) {
    vpi ret(N+1,mp(1,0));
    FOR(i,2,N+1) {
      Dinic<SZ> D;
      trav(t,ed) {
        D.addEdge(t.f.f,t.f.s,t.s);
        D.addEdge(t.f.s,t.f.f,t.s);
      }
      ret[i].s = D.maxFlow(N+1,i,ret[i].f);
      FOR(j,i+1,N+1) if (ret[j].f == ret[i].f
        && D.level[j] != -1) ret[j].f = i;
    }
    vector<pair<pi,int>> res;
    FOR(i,2,N+1) res.pb({{i,ret[i].f},ret[i].s});
    return res;
  }
};
```

DFSmatch.h

Description: naive bipartite matching

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

37ad8b, 25 lines

```
template<int SZ> struct MaxMatch {
  int N, flow = 0, match[SZ], rmatch[SZ];
  bitset<SZ> vis;
  vi adj[SZ];
  MaxMatch() {
    memset(match,0,sizeof match);
    memset(rmatch,0,sizeof rmatch);
  }
  void connect(int a, int b, bool c = 1) {
    if (c) match[a] = b, rmatch[b] = a;
    else match[a] = rmatch[b] = 0;
  }
  bool dfs(int x) {
```

```
    if (!x) return 1;
    if (vis[x]) return 0;
    vis[x] = 1;
    trav(t,adj[x]) if (t != match[x] && dfs(rmatch[t]))
      return connect(x,t),1;
    return 0;
  }
  void tri(int x) { vis.reset(); flow += dfs(x); }
  void init(int _N) {
    N = _N; FOR(i,1,N+1) if (!match[i]) tri(i);
  }
};
```

Hungarian.h

Description: Given array of (possibly negative) costs to complete each of N jobs w/ each of M workers ($N \leq M$), finds min cost to complete all jobs such that each worker is assigned to at most one job. Basically just Dijkstra with potentials.

Time: $\mathcal{O}\left(N^2 M\right)$ d8824c, 34 lines

```
int hungarian(const vector<vi>& a) {
  int n = sz(a)-1, m = sz(a[0])-1; // jobs 1..n, workers 1..m
  vi u(n+1), v(m+1); // potentials
  vi p(m+1); // p[j] -> job picked by worker j
  FOR(i,1,n+1) { // find alternating path with job i
    p[0] = i; int j0 = 0; // add "dummy" worker 0
    vi dist(m+1,INT_MAX), pre(m+1,-1);
    vector<bool> done(m+1, false);
    do { // dijkstra
      done[j0] = true; // fix dist[j0], update dists from j0
      int i0 = p[j0], j1; int delta = INT_MAX;
      FOR(j,1,m+1) if (!done[j]) {
        auto cur = a[i0][j]-u[i0]-v[j];
        if (ckmin(dist[j],cur)) pre[j] = j0;
        if (ckmin(delta,dist[j])) j1 = j;
      }
      FOR(j,m+1) { // subtract constant from all edges going
        // from done -> not done vertices, lowers all
        // remaining dists by constant
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]); // Potentials adjusted so all edge weights
    // are non-negative. Perfect matching has zero weight and
    // costs of augmenting paths do not change.
    while (j0) { // update jobs picked by workers on
      ↪alternating path
      int j1 = pre[j0];
      p[j0] = p[j1];
      j0 = j1;
    }
  }
  return -v[0]; // min cost
}
```

UnweightedMatch.h

Description: Edmond's Blossom Algorithm. General unweighted matching with 1-based indexing.

Time: $\mathcal{O}\left(N^2 M\right)$ facb88, 66 lines

```
template<int SZ> struct UnweightedMatch {
  int match[SZ], N;
  vi adj[SZ];
  void addEdge(int u, int v) { adj[u].pb(v), adj[v].pb(u); }
  void init(int _N) {
    N = _N; FOR(i,1,N+1) adj[i].clear(), match[i] = 0;
  }
  queue<int> Q;
  int par[SZ], vis[SZ], orig[SZ], aux[SZ], t;
  void augment(int u, int v) {
    // flip states of edges on u-v path
    int pv = v, nv;
    do {
      pv = par[v]; nv = match[pv];
      match[v] = pv; match[pv] = v;
      v = nv;
    } while (u != pv);
  }
  int lca(int v, int w) { // find LCA in O(dist)
    ++t;
    while (1) {
      if (v) {
        if (aux[v] == t) return v;
        aux[v] = t; v = orig[par[match[v]]];
      }
      swap(v,w);
    }
  }
  void blossom(int v, int w, int a) {
    while (orig[v] != a) {
      par[v] = w; w = match[v]; // go other way around cycle
      if (vis[w] == 1) Q.push(w), vis[w] = 0;
      orig[v] = orig[w] = a; // merge into supernode
      v = par[w];
    }
  }
  bool bfs(int u) {
    FOR(i,N+1) par[i] = aux[i] = 0, vis[i] = -1, orig[i] = i;
    Q = queue<int>(); Q.push(u); vis[u] = t = 0;
    while (sz(Q)) {
      int v = Q.front(); Q.pop();
      trav(x,adj[v]) {
        if (vis[x] == -1) {
          par[x] = v; vis[x] = 1;
          if (!match[x]) return augment(u, x), true;
          Q.push(match[x]); vis[match[x]] = 0;
        } else if (vis[x] == 0 && orig[v] != orig[x]) { // odd
          ↪cycle
          int a = lca(orig[v], orig[x]);
          blossom(x,v,a); blossom(v,x,a);
        }
      }
    }
    return false;
  }
  int calc() {
    int ans = 0; // find random matching, constant improvement
    vi V(N-1); iota(all(V),1); shuffle(all(V),rng);
    trav(x,V) if (!match[x])
      trav(y,adj[x]) if (!match[y]) {
        match[x] = y, match[y] = x;
        ++ans; break;
      }
    FOR(i,1,N+1) if (!match[i] && bfs(i)) ++ans;
    return ans;
  }
};
```

7.5 Misc

MaximalCliques.h

Description: Used only once. Finds all maximal cliques.

Time: $\mathcal{O}\left(3^{N/3}\right)$ 28a533, 21 lines

```
typedef bitset<128> B;
int N;
B adj[128];
// possibly in clique, not in clique, in clique
void cliques(B P = ~B(), B X={}, B R={}) {
  if (!P.any()) {
    if (!X.any()) {
      // do smth with R
    }
    return;
  }
  int q = (P|X)._Find_first();
  // clique must contain q or non-neighbor of q
  B cands = P&~adj[q];
  FOR(i,N) if (cands[i]) {
    R[i] = 1;
    cliques(P&adj[i],X&adj[i],R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

LCT.h

Description: Link-Cut Tree, solves USACO "The Applicant." Given a function $f(1 \ldots N) \to 1 \ldots N$, evaluates $f^b(a)$ for any a, b. Modifications return false in case of failure. Can use `vir` for subtree size queries.

Time: $\mathcal{O}(\log N)$ c3cff3, 140 lines

```
typedef struct snode* sn;
struct snode {
  sn p, c[2]; // parent, children
  sn extra; // extra cycle node
  bool flip = 0; // subtree flipped or not
  int val, sz; // value in node, # nodes in subtree
  // int vir = 0; stores sum of virtual children
  snode(int v) {
    p = c[0] = c[1] = NULL;
    val = v; calc();
  }
  friend int getSz(sn x) { return x?x->sz:0; }
  void prop() { // lazy prop
    if (!flip) return;
    swap(c[0],c[1]); FOR(i,2) if (c[i]) c[i]->flip ^= 1;
    flip = 0;
  }
  void calc() { // recalc vals
    FOR(i,2) if (c[i]) c[i]->prop();
    sz = 1+getSz(c[0])+getSz(c[1]);
  }
  int dir() {
    if (!p) return -2;
    FOR(i,2) if (p->c[i] == this) return i;
    return -1; // p is path-parent pointer,
    // so not in current splay tree
  }
  bool isRoot() { return dir() < 0; }
  // test if root of current splay tree
  friend void setLink(sn x, sn y, int d) {
    if (y) y->p = x;
    if (d >= 0) x->c[d] = y;
  }
  void rot() { // assume p and p->p propagated
    assert(!isRoot()); int x = dir(); sn pa = p;
    setLink(pa->p, this, pa->dir());
    setLink(pa, c[x^1], x);
    setLink(this, pa, x^1);
    pa->calc(); calc();
  }
  void splay() {
    while (!isRoot() && !p->isRoot()) {
      p->p->prop(), p->prop(), prop();
      dir() == p->dir() ? p->rot() : rot();
```

```
      rot();
    }
    if (!isRoot()) p->prop(), prop(), rot();
    prop();
  }
  void access() { // bring this to top of tree
    for (sn v = this, pre = NULL; v; v = v->p) {
      v->splay();
      // if (pre) v->vir -= pre->sz;
      // if (v->c[1]) v->vir += v->c[1]->sz;
      v->c[1] = pre; v->calc();
      pre = v;
      // v->sz should remain the same if using vir
    }
    splay(); // left subtree of this is now path to root
    assert(!c[1]); // right subtree is empty
  }
  void makeRoot() { access(); flip ^= 1; }
  void set(int v) { splay(); val = v; calc(); }
  // change val in node,
  // splay suffices instead of access because
  // it doesn't affect values in nodes above it
  friend sn lca(sn x, sn y) {
    if (x == y) return x;
    x->access(), y->access(); if (!x->p) return NULL;
    // access at y did not affect x
    // so they must not be connected
    x->splay(); return x->p ? x->p : x;
  }
  friend bool connected(sn x, sn y) { return lca(x,y); }
  // LCA is null if not connected
  int distRoot() { access(); return getSz(c[0]); }
  // # nodes above
  sn getRoot() { // get root of LCT component
    access(); auto a = this;
    while (a->c[0]) a = a->c[0], a->prop();
    a->access(); return a;
  }
  sn dfs(int b) {
    int z = getSz(c[0]);
    if (b < z) return c[0]->dfs(b);
    if (b == z) { access(); return this; }
    return c[1]->dfs(b-z-1);
  }
  sn getPar(int b) { // get b-th parent
    access();
    b = getSz(c[0])-b; assert(b >= 0);
    auto a = this;
    while (1) {
      int z = getSz(a->c[0]);
      if (b == z) { a->access(); return a; }
      if (b < z) a = a->c[0];
      else a = a->c[1], b -= z+1;
      a->prop();
    }
  }
  friend bool link(sn x, sn y) { // make x parent of y
    if (connected(x,y)) return 0; // don't induce cycle
    y->access(); assert(!y->c[0]);
    // or y->makeRoot() if you want to ensure link succeeds
    y->p = x;
    // x->access(); x->sz += y->sz; x->vir += y->sz;
    return 1;
  }
  friend bool cut(sn y) { // cut y from its parent
    y->access(); if (!y->c[0]) return 0;
    y->c[0]->p = NULL; y->c[0] = NULL;
    y->calc(); return 1;
  }
  friend bool cut(sn x, sn y) { // if x, y adjacent in tree
    x->makeRoot(); y->access();
    if (y->c[0] != x || x->c[0] || x->c[1]) return 0;
    // splay tree with y should not contain anything besides x
    assert(cut(y)); return 1;
  }
};

sn LCT[MX];
void setNex(sn a, sn b) { // set f[a] = b
  if (connected(a,b)) a->extra = b;
  else assert(link(b,a));
}
void delNex(sn a) { // set f[a] = NULL
  auto t = a->getRoot();
  if (t == a) { t->extra = NULL; return; }
  assert(cut(a)); assert(t->extra);
  if (!connected(t,t->extra)) {
    assert(link(t->extra,t)); t->extra = NULL; }
}
sn getPar(sn a, int b) { // get f^b[a]
  int d = a->distRoot();
  if (b <= d) return a->getPar(b);
  b -= d+1; auto r = a->getRoot()->extra; assert(r);
  d = r->distRoot()+1;
  return r->getPar(b%d);
}
```

DirectedMST.h

Description: Chu-Liu-Edmonds algorithm. Computes minimum weight directed spanning tree rooted at r, edge from $inv[i] \to i$ for all $i \neq r$

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

"DSUrb.h" 314387, 67 lines

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

pair<ll,vi> dmst(int n, int r, const vector<Edge>& g) {
  DSUrb dsu; dsu.init(n);
  // DSU with rollback if need to return edges
  vector<Node*> heap(n); // store edges entering each vertex
  // in increasing order of weight
  trav(e,g) heap[e.b] = merge(heap[e.b], new Node{e});
  ll res = 0; vi seen(n,-1); seen[r] = r;
  vpi in(n,{-1,-1});
  vector<pair<int,vector<Edge>>> cycs;
  FOR(s,n) {
    int u = s, w;
    vector<pair<int,Edge>> path;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      seen[u] = s;
      Edge e = heap[u]->top(); path.pb({u,e});
      heap[u]->delta -= e.w, pop(heap[u]);
      res += e.w, u = dsu.get(e.a);
      if (seen[u] == s) { // compress verts in cycle
        Node* cyc = 0; cycs.pb({u,{}});
        do {
          cyc = merge(cyc, heap[w = path.back().f]);
          cycs.back().s.pb(path.back().s);
          path.pop_back();
        } while (dsu.unite(u,w));
        u = dsu.get(u); heap[u] = cyc, seen[u] = -1;
      }
    }
    trav(t,path) in[dsu.get(t.s.b)] = {t.s.a,t.s.b};
    // found path from root
  }
  while (sz(cycs)) { // expand cycs to restore sol
    auto c = cycs.back(); cycs.pop_back();
    pi inEdge = in[c.f];
    trav(t,c.s) dsu.rollback();
    trav(t,c.s) in[dsu.get(t.b)] = {t.a,t.b};
    in[dsu.get(inEdge.s)] = inEdge;
  }
  vi inv;
  FOR(i,n) {
    assert(i == r ? in[i].s == -1 : in[i].s == i);
    inv.pb(in[i].f);
  }
  return {res,inv};
}
```

DominatorTree.h

Description: Used only once. a dominates b iff every path from 1 to b passes through a

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

0a9941, 43 lines

```
template<int SZ> struct Dominator {
  vi adj[SZ], ans[SZ]; // input edges, edges of dominator tree
  vi radj[SZ], child[SZ], sdomChild[SZ];
  int label[SZ], rlabel[SZ], sdom[SZ], dom[SZ], co = 0;
  int par[SZ], bes[SZ];
  int get(int x) {
    // DSU with path compression
    // get vertex with smallest sdom on path to root
    if (par[x] != x) {
      int t = get(par[x]); par[x] = par[par[x]];
      if (sdom[t] < sdom[bes[x]]) bes[x] = t;
    }
    return bes[x];
  }
  void dfs(int x) { // create DFS tree
    label[x] = ++co; rlabel[co] = x;
    sdom[co] = par[co] = bes[co] = co;
    trav(y,adj[x]) {
      if (!label[y]) {
        dfs(y);
        child[label[x]].pb(label[y]);
      }
      radj[label[y]].pb(label[x]);
    }
  }
  void init(int root) {
    dfs(root);
    ROF(i,1,co+1) {
      trav(j,radj[i]) ckmin(sdom[i],sdom[get(j)]);
      if (i > 1) sdomChild[sdom[i]].pb(i);
      trav(j,sdomChild[i]) {
```

```
      int k = get(j);
      if (sdom[j] == sdom[k]) dom[j] = sdom[j];
      else dom[j] = k;
    }
    trav(j,child[i]) par[j] = i;
  }
  FOR(i,2,co+1) {
    if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
    ans[rlabel[dom[i]]].pb(rlabel[i]);
  }
 }
};
```

EdgeColor.h

Description: Used only once. Naive implementation of Misra & Gries edge coloring. By Vizing's Theorem, a simple graph with max degree d can be edge colored with at most $d+1$ colors

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

723f0a, 51 lines

```
template<int SZ> struct EdgeColor {
  int N = 0, maxDeg = 0, adj[SZ][SZ], deg[SZ];
  EdgeColor() {
    memset(adj,0,sizeof adj);
    memset(deg,0,sizeof deg);
  }
  void addEdge(int a, int b, int c) {
    adj[a][b] = adj[b][a] = c; }
  int delEdge(int a, int b) {
    int c = adj[a][b]; adj[a][b] = adj[b][a] = 0;
    return c;
  }
  vector<bool> genCol(int x) {
    vector<bool> col(N+1); F0R(i,N) col[adj[x][i]] = 1;
    return col;
  }
  int freeCol(int u) {
    auto col = genCol(u);
    int x = 1; while (col[x]) x ++; return x;
  }
  void invert(int x, int d, int c) {
    F0R(i,N) if (adj[x][i] == d)
      delEdge(x,i), invert(i,c,d), addEdge(x,i,c);
  }
  void addEdge(int u, int v) { // follows wikipedia steps
    // check if you can add edge w/o doing any work
    assert(N); ckmax(maxDeg,max(++deg[u],++deg[v]));
    auto a = genCol(u), b = genCol(v);
    FOR(i,1,maxDeg+2) if (!a[i] && !b[i])
      return addEdge(u,v,i);
    // 2. find maximal fan of u starting at v
    vector<bool> use(N); vi fan = {v}; use[v] = 1;
    while (1) {
      auto col = genCol(fan.back());
      if (sz(fan) > 1) col[adj[fan.back()][u]] = 0;
      int i = 0; while (i < N && (use[i] || col[adj[u][i]])) i
        ↪++;
      if (i < N) fan.pb(i), use[i] = 1;
      else break;
    }
    // 3/4. choose free cols for endpoints of fan, invert cd_u
      ↪path
    int c = freeCol(u), d = freeCol(fan.back()); invert(u,d,c);
    // 5. find i such that d is free on fan[i]
    int i = 0; while (i < sz(fan) && genCol(fan[i])[d]
      && adj[u][fan[i]] != d) i ++;
    assert (i != sz(fan));
    // 6. rotate fan from 0 to i
    F0R(j,i) addEdge(u,fan[j],delEdge(u,fan[j+1]));
    // 7. add new edge
    addEdge(u,fan[i],d);
  }
};
```

Geometry (8)

8.1 Primitives

Point.h

Description: use in place of `complex<T>`

a85380, 50 lines

```
typedef ld T;
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }

namespace Point {
  typedef pair<T,T> P;
  typedef vector<P> vP;

  T norm(P x) { return x.f*x.f+x.s*x.s; }
  T abs(P x) { return sqrt(norm(x)); }
  T angle(P x) { return atan2(x.s,x.f); }
  P conj(P x) { return P(x.f,-x.s); }
  P perp(P x) { return P(-x.s,x.f); }
  P dir(T ang) {
    auto c = exp(ang*complex<T>(0,1));
    return P(c.real(),c.imag());
  }

  P operator-(const P& l) { return P(-l.f,-l.s); }
  P operator+(const P& l, const P& r) {
    return P(l.f+r.f,l.s+r.s); }
  P operator-(const P& l, const P& r) {
    return P(l.f-r.f,l.s-r.s); }
  P operator*(const P& l, const T& r) {
    return P(l.f*r,l.s*r); }
  P operator*(const T& l, const P& r) { return r*l; }
  P operator/(const P& l, const T& r) {
    return P(l.f/r,l.s/r); }
  P operator*(const P& l, const P& r) {
    return P(l.f*r.f-l.s*r.s,l.s*r.f+l.f*r.s); }
  P operator/(const P& l, const P& r) {
    return l*conj(r)/norm(r); }
  P& operator+=(P& l, const P& r) { return l = l+r; }
  P& operator-=(P& l, const P& r) { return l = l-r; }
  P& operator*=(P& l, const T& r) { return l = l*r; }
  P& operator/=(P& l, const T& r) { return l = l/r; }
  P& operator*=(P& l, const P& r) { return l = l*r; }
  P& operator/=(P& l, const P& r) { return l = l/r; }

  P unit(P x) { return x/abs(x); }
  T dot(P a, P b) { return (conj(a)*b).f; }
  T cross(P a, P b) { return (conj(a)*b).s; }
  T cross(P p, P a, P b) { return cross(a-p,b-p); }
  P rotate(P a, T b) { return a*P(cos(b),sin(b)); }
  P reflect(P p, P a, P b) {
    return a+conj((p-a)/(b-a))*(b-a); }
  P foot(P p, P a, P b) { return (p+reflect(p,a,b))/(T)2; }
  bool onSeg(P p, P a, P b) {
    return cross(a,b,p) == 0 && dot(p-a,p-b) <= 0; }
};
using namespace Point;
```

AngleCmp.h

Description: sorts points in ccw order about origin, atan2 returns real in $(-\pi,\pi]$ so points on negative x-axis come last

Usage: vP v;
sort(all(v),[](P a, P b) { return
atan2(a.s,a.f) < atan2(b.s,b.f); });
sort(all(v),angleCmp); // should give same result

"Point.h" f43f90, 6 lines

```
template<class T> int half(pair<T,T> x) {
  return x.s == 0 ? x.f < 0 : x.s > 0; }
bool angleCmp(P a, P b) {
  int A = half(a), B = half(b);
  return A == B ? cross(a,b) > 0 : A < B;
}
```

SegDist.h

Description: computes distance between P and line (segment) AB

"Point.h" d105ae, 7 lines

```
T lineDist(P p, P a, P b) {
  return abs(cross(p,a,b))/abs(a-b); }
T segDist(P p, P a, P b) {
  if (dot(p-a,b-a) <= 0) return abs(p-a);
  if (dot(p-b,a-b) <= 0) return abs(p-b);
  return lineDist(p,a,b);
}
```

LineIntersect.h

Description: computes the intersection point(s) of lines AB, CD; returns $\{-1,\{0,0\}\}$ if infinitely many, $\{0,\{0,0\}\}$ if none, $\{1,x\}$ if x is the unique point

"Point.h" d86521, 9 lines

```
P extension(P a, P b, P c, P d) {
  T x = cross(a,b,c), y = cross(a,b,d);
  return (d*x-c*y)/(x-y);
}
pair<int,P> lineIntersect(P a, P b, P c, P d) {
  if (cross(b-a,d-c) == 0)
    return {-(cross(a,c,d) == 0),P(0,0)};
  return {1,extension(a,b,c,d)};
}
```

SegIntersect.h

Description: computes the intersection point(s) of line segments AB, CD

"Point.h" 993634, 12 lines

```
vP segIntersect(P a, P b, P c, P d) {
  T x = cross(a,b,c), y = cross(a,b,d);
  T X = cross(c,d,a), Y = cross(c,d,b);
  if (sgn(x)*sgn(y) < 0 && sgn(X)*sgn(Y) < 0)
    return {(d*x-c*y)/(x-y)};
  set<P> s;
  if (onSeg(a,c,d)) s.insert(a);
  if (onSeg(b,c,d)) s.insert(b);
  if (onSeg(c,a,b)) s.insert(c);
  if (onSeg(d,a,b)) s.insert(d);
  return {all(s)};
}
```

8.2 Polygons

Area.h

Description: area, center of mass of a polygon with constant mass per unit area

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

"Point.h" 11ed70, 16 lines

```
T area(const vP& v) {
  T area = 0;
  F0R(i,sz(v)) {
    int j = (i+1)%sz(v); T a = cross(v[i],v[j]);
    area += a;
```

```
  }
  return abs(area)/2;
}
P centroid(const vP& v) {
  P cen(0,0); T area = 0; // 2*signed area
  FOR(i,sz(v)) {
    int j = (i+1)%sz(v); T a = cross(v[i],v[j]);
    cen += a*(v[i]+v[j]); area += a;
  }
  return cen/area/(T)3;
}
```

InPoly.h

Description: tests whether a point is inside, on, or outside of the perimeter of a polygon
Time: $\mathcal{O}(N)$

"Point.h" 8f2d6a, 10 lines

```
string inPoly(const vP& p, P z) {
  int n = sz(p), ans = 0;
  FOR(i,n) {
    P x = p[i], y = p[(i+1)%n];
    if (onSeg(z,x,y)) return "on";
    if (x.s > y.s) swap(x,y);
    if (x.s <= z.s && y.s > z.s && cross(z,x,y) > 0) ans ^= 1;
  }
  return ans ? "in" : "out";
}
```

ConvexHull.h

Description: top-bottom convex hull
Time: $\mathcal{O}(N \log N)$

"Point.h" c39426, 22 lines

```
// typedef ll T;
pair<vi,vi> ulHull(const vP& P) {
  vi p(sz(P)), u, l; iota(all(p), 0);
  sort(all(p), [&P](int a, int b) { return P[a] < P[b]; });
  trav(i,p) {
    #define ADDP(C, cmp) while (sz(C) > 1 && cross(\
      P[C[sz(C)-2]],P[C.back()],P[i]) cmp 0) C.pop_back(); C.pb
        ↪(i);
    ADDP(u, >=); ADDP(l, <=);
  }
  return {u,l};
}
vi hullInd(const vP& P) {
  vi u,l; tie(u,l) = ulHull(P);
  if (sz(l) <= 1) return l;
  if (P[l[0]] == P[l[1]]) return {0};
  l.insert(end(l),u.rbegin()+1,u.rend()-1); return l;
}
vP hull(const vP& P) {
  vi v = hullInd(P);
  vP res; trav(t,v) res.pb(P[t]);
  return res;
}
```

PolyDiameter.h

Description: rotating caliphers, gives greatest distance between two points in P
Time: $\mathcal{O}(N)$ given convex hull

"ConvexHull.h" 38208a, 10 lines

```
ld diameter(vP P) {
  P = hull(P);
  int n = sz(P), ind = 1; ld ans = 0;
  FOR(i,n)
    for (int j = (i+1)%n;;ind = (ind+1)%n) {
      ckmax(ans,abs(P[i]-P[ind]));
      if (cross(P[j]-P[i],P[(ind+1)%n]-P[ind]) <= 0) break;
    }
  return ans;
}
```

HullTangents.h

Description: Given convex polygon with no three points collinear and a point strictly outside of it, computes the lower and upper tangents.
Time: $\mathcal{O}(\log N)$

968ad8, 37 lines

```
bool lower;
bool better(P a, P b, P c) {
  ll z = cross(a,b,c);
  return lower ? z < 0 : z > 0;
}
int tangent(const vP& a, P b) {
  if (sz(a) == 1) return 0;
  int lo, hi;
  if (better(b,a[0],a[1])) {
    lo = 0, hi = sz(a)-1;
    while (lo < hi) {
      int mid = (lo+hi+1)/2;
      if (better(b,a[0],a[mid])) lo = mid;
      else hi = mid-1;
    }
    lo = 0;
  } else {
    lo = 1, hi = sz(a);
    while (lo < hi) {
      int mid = (lo+hi)/2;
      if (!better(b,a[0],a[mid])) lo = mid+1;
      else hi = mid;
    }
    hi = sz(a);
  }
  while (lo < hi) {
    int mid = (lo+hi)/2;
    if (better(b,a[mid],a[(mid+1)%sz(a)])) lo = mid+1;
    else hi = mid;
  }
  return lo%sz(a);
}
pi tangents(const vP& a, P b) {
  lower = 1; int x = tangent(a,b);
  lower = 0; int y = tangent(a,b);
  return {x,y};
}
```

LineHull.h

Description: `lineHull` accepts line and ccw convex polygon. If all vertices in poly lie to one side of the line, returns a vector of closest vertices to line as well as orientation of poly with respect to line (± 1 for above/below). Otherwise, returns the range of vertices that lie on or below the line. `extrVertex` returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(\log N)$

34d6ab, 41 lines

```
typedef array<P,2> Line;
#define cmp(i,j) sgn(-dot(dir,poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i+1,i) >= 0 && cmp(i,i-1+n) < 0
int extrVertex(const vP& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo+1 < hi) {
    int m = (lo+hi)/2;
    if (extr(m)) return m;
    int ls = cmp(lo+1,lo), ms = cmp(m+1,m);
    (ls < ms || (ls == ms && ls == cmp(lo,m)) ? hi : lo) = m;
  }
  return lo;
}
vi same(Line line, const vP& poly, int a) {
  // points on same parallel as a
  int n = sz(poly); P dir = perp(line[0]-line[1]);
  if (cmp(a+n-1,a) == 0) return {(a+n-1)%n,a};
  if (cmp(a,a+1) == 0) return {a,(a+1)%n};
  return {a};
}
#define cmpL(i) sgn(cross(line[0],line[1],poly[i]))
pair<int,vi> lineHull(Line line, const vP& poly) {
  int n = sz(poly); assert(n>1);
  int endA = extrVertex(poly,perp(line[0]-line[1])); // lowest
  if (cmpL(endA) >= 0) return {1,same(line,poly,endA)};
  int endB = extrVertex(poly,perp(line[1]-line[0])); // highest
  if (cmpL(endB) <= 0) return {-1,same(line,poly,endB)};
  array<int,2> res;
  FOR(i,2) {
    int lo = endA, hi = endB; if (hi < lo) hi += n;
    while (lo < hi) {
      int m = (lo+hi+1)/2;
      if (cmpL(m%n) == cmpL(endA)) lo = m;
      else hi = m-1;
    }
    res[i] = lo%n; swap(endA,endB);
  }
  if (cmpL((res[0]+1)%n) == 0) res[0] = (res[0]+1)%n;
  return {0,{(res[1]+1)%n,res[0]}};
}
```

8.3 Circles

Circle.h

Description: represent circle as {center,radius}

"Point.h" eb86de, 7 lines

```
typedef pair<P,T> circ;
bool on(circ x, P y) { return abs(y-x.f) == x.s; }
bool in(circ x, P y) {  return abs(y-x.f) <= x.s; }
T arcLength(circ x, P a, P b) {
  P d = (a-x.f)/(b-x.f);
  return x.s*acos(d.f);
}
```

CircleIntersect.h

Description: circle intersection points and intersection area

"Circle.h" 410985, 15 lines

```
vP intersectPoint(circ x, circ y) {
  T d = abs(x.f-y.f), a = x.s, b = y.s;
  if (d == 0) { assert(a != b); return {}; }
  T C = (a*a+d*d-b*b)/(2*a*d); if (abs(C) > 1) return {};
  T S = sqrt(1-C*C); P tmp = (y.f-x.f)/d*x.s;
  return {x.f+tmp*P(C,S),x.f+tmp*P(C,-S)};
}
T intersectArea(circ x, circ y) { // not thoroughly tested
  T d = abs(x.f-y.f), a = x.s, b = y.s; if (a < b) swap(a,b);
  if (d >= a+b) return 0;
  if (d <= a-b) return PI*b*b;
  auto ca = (a*a+d*d-b*b)/(2*a*d), cb = (b*b+d*d-a*a)/(2*b*d);
  auto s = (a+b+d)/2, h = 2*sqrt(s*(s-a)*(s-b)*(s-d))/d;
  return a*a*acos(ca)+b*b*acos(cb)-d*h;
}
```

CircleTangents.h

Description: internal and external tangents between two circles

"Circle.h" bb7166, 22 lines

```
P tangent(P x, circ y, int t = 0) {
  y.s = abs(y.s); // abs needed because internal calls y.s < 0
  if (y.s == 0) return y.f;
```

```
  T d = abs(x-y.f);
  P a = pow(y.s/d,2)*(x-y.f)+y.f;
  P b = sqrt(d*d-y.s*y.s)/d*y.s*unit(x-y.f)*dir(PI/2);
  return t == 0 ? a+b : a-b;
}
vector<pair<P,P>> external(circ x, circ y) {
  vector<pair<P,P>> v;
  if (x.s == y.s) {
    P tmp = unit(x.f-y.f)*x.s*dir(PI/2);
    v.pb(mp(x.f+tmp,y.f+tmp));
    v.pb(mp(x.f-tmp,y.f-tmp));
  } else {
    P p = (y.s*x.f-x.s*y.f)/(y.s-x.s);
    FOR(i,2) v.pb({tangent(p,x,i),tangent(p,y,i)});
  }
  return v;
}
vector<pair<P,P>> internal(circ x, circ y) {
  x.s *= -1; return external(x,y); }
```

Circumcenter.h

Description: returns {circumcenter,circumradius}

"Circle.h" cfb851, 5 lines

```
circ ccCenter(P a, P b, P c) {
  b -= a; c -= a;
  P res = b*c*(conj(c)-conj(b))/(b*conj(c)-conj(b)*c);
  return {a+res,abs(res)};
}
```

MinEnclosingCircle.h

Description: minimum enclosing circle
Time: expected $\mathcal{O}(N)$

"Circumcenter.h" 53963d, 13 lines

```
circ mec(vP ps) {
  shuffle(all(ps), rng);
  P o = ps[0]; T r = 0, EPS = 1+1e-8;
  FOR(i,sz(ps)) if (abs(o-ps[i]) > r*EPS) {
    o = ps[i], r = 0;
    FOR(j,i) if (abs(o-ps[j]) > r*EPS) {
      o = (ps[i]+ps[j])/2, r = abs(o-ps[i]);
      FOR(k,j) if (abs(o-ps[k]) > r*EPS)
        tie(o,r) = ccCenter(ps[i],ps[j],ps[k]);
    }
  }
  return {o,r};
}
```

8.4 Misc

ClosestPair.h

Description: line sweep to find two closest points
Time: $\mathcal{O}(N \log N)$

"Point.h" 34bbb1, 17 lines

```
pair<P,P> solve(vP v) {
  pair<ld,pair<P,P>> bes; bes.f = INF;
  set<P> S; int ind = 0;
  sort(all(v));
  FOR(i,sz(v)) {
    if (i && v[i] == v[i-1]) return {v[i],v[i]};
    for (; v[i].f-v[ind].f >= bes.f; ++ind)
      S.erase({v[ind].s,v[ind].f});
    for (auto it = S.ub({v[i].s-bes.f,INF});
      it != end(S) && it->f < v[i].s+bes.f; ++it) {
      P t = {it->s,it->f};
      ckmin(bes,{abs(t-v[i]),{t,v[i]}});
    }
    S.insert({v[i].s,v[i].f});
  }
  return bes.s;
}
```

DelaunayFast.h

Description: Delaunay Triangulation, concyclic points are OK (but not all collinear)
Time: $\mathcal{O}(N \log N)$

"Point.h" 765ba9, 94 lines

```
typedef ll T;

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

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

// test if p is in the circumcircle
bool circ(P p, P a, P b, P c) {
  ll ar = cross(a,b,c); assert(ar); if (ar < 0) swap(a,b);
  lll p2 = norm(p), A = norm(a)-p2,
    B = norm(b)-p2, C = norm(c)-p2;
  return cross(p,a,b)*C+cross(p,b,c)*A+cross(p,c,a)*B > 0;
}

Q makeEdge(P orig, P dest) {
  Q q[] = {new Quad{0,0,0,orig}, new Quad{0,0,0,arb},
       new Quad{0,0,0,dest}, new Quad{0,0,0,arb}};
  FOR(i,4) q[i]->o = q[-i & 3], q[i]->rot = q[(i+1) & 3];
  return *q;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = cross(s[0], 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) (cross(e->F(),H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s)-half});
  tie(B, rb) = rec({sz(s)-half+all(s)});
  while ((cross(B->p,H(A)) < 0 && (A = A->next())) ||
       (cross(A->p,H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

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

vector<array<P,3>> triangulate(vector<P> pts) {
  sort(all(pts)); assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};

  Q e = rec(pts).f; vector<Q> q = {e};
  int qi = 0;
  while (cross(e->o->F(), e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;

  vector<array<P,3>> ret;
  FOR(i,sz(pts)/3) ret.pb({pts[3*i],pts[3*i+1],pts[3*i+2]});
  return ret;
}
```

8.5 3D

Point3D.h

Description: basic 3D geometry

a4d471, 44 lines

```
typedef ld T;
namespace Point3D {
  typedef array<T,3> P3;
  typedef vector<P3> vP3;
  T norm(const P3& x) {
    T sum = 0; FOR(i,sz(x)) sum += x[i]*x[i];
    return sum;
  }
  T abs(const P3& x) { return sqrt(norm(x)); }

  P3& operator+=(P3& l, const P3& r) {
    FOR(i,3) l[i] += r[i]; return l; }
  P3& operator-=(P3& l, const P3& r) {
    FOR(i,3) l[i] -= r[i]; return l; }
  P3& operator*=(P3& l, const T& r) {
    FOR(i,3) l[i] *= r; return l; }
  P3& operator/=(P3& l, const T& r) {
    FOR(i,3) l[i] /= r; return l; }
  P3 operator+(P3 l, const P3& r) { return l += r; }
  P3 operator-(P3 l, const P3& r) { return l -= r; }
  P3 operator*(P3 l, const T& r) { return l *= r; }
  P3 operator*(const T& r, const P3& l) { return l*r; }
  P3 operator/(P3 l, const T& r) { return l /= r; }

  T dot(const P3& a, const P3& b) {
    T sum = 0; FOR(i,3) sum += a[i]*b[i];
    return sum;
  }
```

```
  P3 cross(const P3& a, const P3& b) {
    return {a[1]*b[2]-a[2]*b[1],
        a[2]*b[0]-a[0]*b[2],
        a[0]*b[1]-a[1]*b[0]};
  }
  bool isMult(const P3& a, const P3& b) {
    auto c = cross(a,b);
    FOR(i,sz(c)) if (c[i] != 0) return 0;
    return 1;
  }
  bool collinear(const P3& a, const P3& b, const P3& c) {
    return isMult(b-a,c-a); }
  bool coplanar(const P3& a, const P3& b, const P3& c, const P3
    ↪& d) {
    return isMult(cross(b-a,c-a),cross(b-a,d-a)); }
}
using namespace Point3D;
```

Hull3D.h

Description: 3D convex hull where no four points coplanar, polyedron volume

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

"Point3D.h" 1158ee, 46 lines

```
struct ED {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1)+(b != -1); }
  int a, b;
};
struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vP3& A) {
  assert(sz(A) >= 4);
  vector<vector<ED>> E(sz(A), vector<ED>(sz(A), {-1, -1}));
  #define E(x,y) E[f.x][f.y]
  vector<F> FS; // faces
  auto mf = [&](int i, int j, int k, int l) { // make face
    P3 q = cross(A[j]-A[i],A[k]-A[i]);
    if (dot(q,A[l]) > dot(q,A[i])) q *= -1; // make sure q
      ↪points outward
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.pb(f);
  };
  FOR(i,4) FOR(j,i+1,4) FOR(k,j+1,4) mf(i, j, k, 6-i-j-k);
  FOR(i,4,sz(A)) {
    FOR(j,sz(FS)) {
      F f = FS[j];
      if (dot(f.q,A[i]) > dot(f.q,A[f.a])) { // face is visible
        ↪, remove edges
        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();
      }
    }
    FOR(j,sz(FS)) { // add faces with new point
      F f = FS[j];
      #define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i,
        ↪ f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  trav(it, FS) if (dot(cross(A[it.b]-A[it.a],A[it.c]-A[it.a]),
    ↪it.q) <= 0)
    swap(it.c, it.b);
  return FS;
}
```

```
T signedPolyVolume(const vP3& p, const vector<F>& trilist) {
  T v = 0;
  trav(i,trilist) v += dot(cross(p[i.a],p[i.b]),p[i.c]);
  return v/6;
}
```

Strings (9)

9.1 Light

KMP.h

Description: f[i] equals the length of the longest proper suffix of the i-th prefix of s that is a prefix of s

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

a3579b, 15 lines

```
vi kmp(str s) {
  int N = sz(s); vi f(N+1); f[0] = -1;
  FOR(i,1,N+1) {
    f[i] = f[i-1];
    while (f[i] != -1 && s[f[i]] != s[i-1]) f[i] = f[f[i]];
    f[i] ++;
  }
  return f;
}
vi getOc(str a, str b) { // find occurrences of a in b
  vi f = kmp(a+"@"+b), ret;
  FOR(i,sz(a),sz(b)+1) if (f[i+sz(a)+1] == sz(a))
    ret.pb(i-sz(a));
  return ret;
}
```

Z.h

Description: for each index i, computes the the maximum len such that s.substr(0,len) == s.substr(i,len)

Usage: pr(z("abcababcabcaba"), getPrefix("abcab","uwetrabcerabcab"));

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

75b3ce, 16 lines

```
vi z(str s) {
  int N = sz(s); s += '#';
  vi ans(N); ans[0] = N;
  int L = 1, R = 0;
  FOR(i,1,N) {
    if (i <= R) ans[i] = min(R-i+1,ans[i-L]);
    while (s[i+ans[i]] == s[ans[i]]) ans[i] ++;
    if (i+ans[i]-1 > R) L = i, R = i+ans[i]-1;
  }
  return ans;
}
vi getPrefix(str a, str b) { // find prefixes of a in b
  vi t = z(a+b), T(sz(b));
  FOR(i,sz(T)) T[i] = min(t[i+sz(a)],sz(a));
  return T;
}
```

Manacher.h

Description: Calculates length of largest palindrome centered at each character of string

Usage: ps(manacher("abacaba"))

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

d920c2, 14 lines

```
vi manacher(str s) {
  str s1 = "@"; trav(c,s) s1 += c, s1 += "#";
  s1.back() = '&';
  vi ans(sz(s1)-1);
  int lo = 0, hi = 0;
  FOR(i,1,sz(s1)-1) {
    if (i != 1) ans[i] = min(hi-i,ans[hi-i+lo]);
    while (s1[i-ans[i]-1] == s1[i+ans[i]+1]) ans[i] ++;
    if (i+ans[i] > hi) lo = i-ans[i], hi = i+ans[i];
  }
  ans.erase(begin(ans));
  FOR(i,sz(ans)) if ((i&1) == (ans[i]&1)) ans[i] ++;
  return ans;
}
```

MinRotation.h

Description: minimum rotation of string

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

57b7f2, 9 lines

```
int minRotation(str s) {
  int a = 0, N = sz(s); s += s;
  FOR(b,N) FOR(i,N) {
    // a is current best rotation found up to b-1
    if (a+i == b || s[a+i] < s[b+i]) { b += max(0,i-1); break;
      ↪} // b to b+i-1 can't be better than a to a+i-1
    if (s[a+i] > s[b+i]) { a = b; break; } // new best found
  }
  return a;
}
```

LyndonFactorization.h

Description: A string is "simple" if it is strictly smaller than any of its own nontrivial suffixes. The Lyndon factorization of the string s is a factorization $s = w_1 w_2 \dots w_k$ where all strings w_i are simple and $w_1 \geq w_2 \geq \cdots \geq w_k$. Min rotation gets min index i such that cyclic shift of s starting at i is minimum.

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

ff5520, 19 lines

```
vector<string> duval(const string& s) {
  int n = sz(s); vector<string> factors;
  for (int i = 0; i < n; ) {
    int j = i+1, k = i;
    for (; j < n && s[k] <= s[j]; j++) {
      if (s[k] < s[j]) k = i;
      else k ++;
    }
    for (; i <= k; i += j-k) factors.pb(s.substr(i, j-k));
  }
  return factors;
}
int minRotation(string s) {
  int n = sz(s); s += s;
  auto d = duval(s); int ind = 0, ans = 0;
  while (ans+sz(d[ind]) < n) ans += sz(d[ind++]);
  while (ind && d[ind] == d[ind-1]) ans -= sz(d[ind--]);
  return ans;
}
```

HashRange.h

Description: polynomial double-hash substrings

Usage: HashRange H; H.init("ababab"); FOR(i,6) FOR(j,i,6) ps(i,j,H.hash(i,j));

8972d7, 33 lines

```
typedef array<int,2> T; // pick bases not too close to ends
uniform_int_distribution<int> MULT_DIST(0.1*MOD,0.9*MOD);
const T base = {MULT_DIST(rng),MULT_DIST(rng)};

T operator+(const T& l, const T& r) { T x;
  FOR(i,2) x[i] = (l[i]+r[i])%MOD; return x; }
T operator-(const T& l, const T& r) { T x;
  FOR(i,2) x[i] = (l[i]-r[i]+MOD)%MOD; return x; }
T operator*(const T& l, const T& r) { T x;
  FOR(i,2) x[i] = (ll)l[i]*r[i]%MOD; return x; }

struct HashRange {
  str S;
```

```
  vector<T> pows, cum;
  void init(str _S) {
    S = _S; pows.rsz(sz(S)), cum.rsz(sz(S)+1);
    pows[0] = {1,1}; FOR(i,1,sz(S)) pows[i] = pows[i-1]*base;
    FOR(i,sz(S)) {
      int c = S[i]-'a'+1;
      cum[i+1] = base*cum[i]+T{c,c};
    }
  }
  T hash(int l, int r) { return cum[r+1]-pows[r+1-l]*cum[l]; }
  int lcp(HashRange& b) {
    int lo = 0, hi = min(sz(S),sz(b.S));
    while (lo < hi) {
      int mid = (lo+hi+1)/2;
      if (cum[mid] == b.cum[mid]) lo = mid;
      else hi = mid-1;
    }
    return lo;
  }
};
```

9.2 Heavy

ACfixed.h

Description: for each prefix, stores link to max length suffix which is also a prefix

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

6b3108, 34 lines

```
struct ACfixed { // fixed alphabet
  struct node {
    array<int,26> to;
    int link;
  };
  vector<node> d;
  ACfixed() { d.eb(); }
  int add(str s) { // add word
    int v = 0;
    trav(C,s) {
      int c = C-'a';
      if (!d[v].to[c]) {
        d[v].to[c] = sz(d);
        d.eb();
      }
      v = d[v].to[c];
    }
    return v;
  }
  void init() { // generate links
    d[0].link = -1;
    queue<int> q; q.push(0);
    while (sz(q)) {
      int v = q.front(); q.pop();
      FOR(c,26) {
        int u = d[v].to[c]; if (!u) continue;
        d[u].link = d[v].link == -1 ? 0 : d[d[v].link].to[c];
        q.push(u);
      }
      if (v) FOR(c,26) if (!d[v].to[c])
        d[v].to[c] = d[d[v].link].to[c];
    }
  }
};
```

PalTree.h

Description: Used infrequently. Palindromic tree computes number of occurrences of each palindrome within string. `ans[i][0]` stores min even x such that the prefix $s[1..i]$ can be split into exactly x palindromes, `ans[i][1]` does the same for odd x.

Time: $\mathcal{O}(N\sum)$ for addChar, $\mathcal{O}(N\log N)$ for updAns

98ef7b, 45 lines

```
template<int SZ> struct PalTree {
  static const int sigma = 26;
  int s[SZ], len[SZ], link[SZ], to[SZ][sigma], oc[SZ];
  int slink[SZ], diff[SZ];
  array<int,2> ans[SZ], seriesAns[SZ];
  int n, last, sz;
  PalTree() {
    s[n++] = -1; link[0] = 1; len[1] = -1; sz = 2;
    ans[0] = {0,MOD};
  }
  int getLink(int v) {
    while (s[n-len[v]-2] != s[n-1]) v = link[v];
    return v;
  }
  void updAns() { // serial path has O(log n) vertices
    ans[n-1] = {MOD,MOD};
    for (int v = last; len[v] > 0; v = slink[v]) {
      seriesAns[v] = ans[n-1-(len[slink[v]]+diff[v])];
      if (diff[v] == diff[link[v]])
        FOR(i,2) ckmin(seriesAns[v][i],seriesAns[link[v]][i]);
      // previous oc of link[v] = start of last oc of v
      FOR(i,2) ckmin(ans[n-1][i],seriesAns[v][i^1]+1);
    }
  }
  void addChar(int c) {
    s[n++] = c;
    last = getLink(last);
    if (!to[last][c]) {
      len[sz] = len[last]+2;
      link[sz] = to[getLink(link[last])][c];
      diff[sz] = len[sz]-len[link[sz]];
      if (diff[sz] == diff[link[sz]])
        slink[sz] = slink[link[sz]];
      else slink[sz] = link[sz];
      // slink[v] = max suffix u of v such that diff[v]\neq
        ↪diff[u]
      to[last][c] = sz++;
    }
    last = to[last][c]; oc[last] ++;
    updAns();
  }
  void numOc() { // # occurrences of each palindrome
    vpi v; FOR(i,2,sz) v.pb({len[i],i});
    sort(rall(v)); trav(a,v) oc[link[a.s]] += oc[a.s];
  }
};
```

SuffixArray.h

Description: `sa` contains indices of suffixes in sorted order, `isa` contains inverses. Can speed up on random test data by breaking when elements of `isa` are all distinct.

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

9295b6, 47 lines

```
struct SuffixArray {
  string S; int N;
  void init(const string& _S) {
    S = _S; N = sz(S);
    genSa(); genLcp(); // R.init(lcp);
  }
  vi sa, isa;
  void genSa() {
    sa.rsz(N), isa.rsz(N); FOR(i,N) sa[i]=N-1-i, isa[i]=S[i];
    stable_sort(all(sa), [this](int i, int j) {
      return S[i] < S[j]; });
    for (int len = 1; len < N; len *= 2) {
      // sufs currently sorted by first len chars
      // those of shorter length go later
      vi is(isa), s(sa), nex(N); iota(all(nex),0);
      FOR(i,N) { // compare first len characters of each suf
        // those with length <= len don't change pos
        bool same = i && sa[i-1]+len < N
              && is[sa[i]] == is[sa[i-1]]
              && is[sa[i]+len/2] == is[sa[i-1]+len/2];
        isa[sa[i]] = same ? isa[sa[i-1]] : i;
      }
      FOR(i,N) { // rearrange sufs with length > len
        int s1 = s[i]-len;
        if (s1 >= 0) sa[nex[isa[s1]]++] = s1;
      }
    }
  }
  vi lcp;
  void genLcp() { // Kasai's Algo
    lcp = vi(N-1); int h = 0;
    FOR(i,N) if (isa[i]) {
      for (int j=sa[isa[i]-1]; j+h<N && S[i+h]==S[j+h]; h++);
      lcp[isa[i]-1] = h; if (h) h--;
      // if we cut off first chars of two strings
      // with lcp h then remaining portions still have lcp h-1
    }
  }
  /*RMQ<int> R;
  int getLCP(int a, int b) { // lcp of suffixes starting at a,b
    if (max(a,b) >= N) return 0;
    if (a == b) return N-a;
    int t0 = isa[a], t1 = isa[b];
    if (t0 > t1) swap(t0,t1);
    return R.query(t0,t1-1);
  }*/
};
```

ReverseBW.h

Description: Used only once. The Burrows-Wheeler Transform appends # to a string, sorts the rotations of the string in increasing order, and constructs a new string that contains the last character of each rotation. This function reverses the transform.

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

339117, 9 lines

```
str reverseBW(str s) {
  vi nex(sz(s)); vi v(sz(s)); iota(all(v),0);
  stable_sort(all(v),[&s](int a, int b) {
    return s[a] < s[b]; });
  FOR(i,sz(v)) nex[i] = v[i];
  int cur = nex[0]; str ret;
  for (; cur; cur = nex[cur]) ret += s[v[cur]];
  return ret;
}
```

SuffixAutomaton.h

Description: Used infrequently. Constructs minimal DFA that recognizes all suffixes of a string

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

1cb9d7, 71 lines

```
struct SuffixAutomaton {
  struct state {
    int len = 0, firstPos = -1, link = -1;
    bool isClone = 0;
    map<char, int> next;
    vi invLink;
  };
  vector<state> st;
  int last = 0;
  void extend(char c) {
    int cur = sz(st); st.eb();
    st[cur].len = st[last].len+1, st[cur].firstPos = st[cur].
      ↪len-1;
    int p = last;
```

```
    while (p != -1 && !st[p].next.count(c)) {
      st[p].next[c] = cur;
      p = st[p].link;
    }
    if (p == -1) {
      st[cur].link = 0;
    } else {
      int q = st[p].next[c];
      if (st[p].len+1 == st[q].len) {
        st[cur].link = q;
      } else {
        int clone = sz(st); st.pb(st[q]);
        st[clone].len = st[p].len+1, st[clone].isClone = 1;
        while (p != -1 && st[p].next[c] == q) {
          st[p].next[c] = clone;
          p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
      }
    }
    last = cur;
  }
  void init(string s) {
    st.eb(); trav(x,s) extend(x);
    FOR(v,1,sz(st)) st[st[v].link].invLink.pb(v);
  }
  // APPLICATIONS
  void getAllOccur(vi& oc, int v) {
    if (!st[v].isClone) oc.pb(st[v].firstPos);
    trav(u,st[v].invLink) getAllOccur(oc,u);
  }
  vi allOccur(string s) {
    int cur = 0;
    trav(x,s) {
      if (!st[cur].next.count(x)) return {};
      cur = st[cur].next[x];
    }
    vi oc; getAllOccur(oc,cur); trav(t,oc) t += 1-sz(s);
    sort(all(oc)); return oc;
  }

  vl distinct;
  ll getDistinct(int x) {
    if (distinct[x]) return distinct[x];
    distinct[x] = 1;
    trav(y,st[x].next) distinct[x] += getDistinct(y.s);
    return distinct[x];
  }
  ll numDistinct() { // # of distinct substrings including
    ↪empty
    distinct.rsz(sz(st));
    return getDistinct(0);
  }
  ll numDistinct2() { // another way to do above
    ll ans = 1;
    FOR(i,1,sz(st)) ans += st[i].len-st[st[i].link].len;
    return ans;
  }
};
```

SuffixTree.h

Description: Used infrequently. Ukkonen's algorithm for suffix tree.
Time: $\mathcal{O}(N \log \sum)$

1df16c, 68 lines

```
struct SuffixTree {
  str s; int node, pos;
  struct state { // edge to state is s[fpos,fpos+len)
    int fpos, len, link = -1;
    map<char,int> to;
    state(int fpos, int len) : fpos(fpos), len(len) {}
  };
  vector<state> st;
  int makeNode(int pos, int len) {
    st.pb(state(pos,len)); return sz(st)-1;
  }
  void goEdge() {
    while (pos>1 && pos>st[st[node].to[s[sz(s)-pos]]].len) {
      node = st[node].to[s[sz(s)-pos]];
      pos -= st[node].len;
    }
  }
  void extend(char c) {
    s += c; pos ++; int last = 0;
    while (pos) {
      goEdge();
      char edge = s[sz(s)-pos];
      int& v = st[node].to[edge];
      char t = s[st[v].fpos+pos-1];
      if (v == 0) {
        v = makeNode(sz(s)-pos,MOD);
        st[last].link = node; last = 0;
      } else if (t == c) {
        st[last].link = node;
        return;
      } else {
        int u = makeNode(st[v].fpos,pos-1);
        st[u].to[c] = makeNode(sz(s)-1,MOD); st[u].to[t] = v;
        st[v].fpos += pos-1; st[v].len -= pos-1;
        v = u; st[last].link = u; last = u;
      }
      if (node == 0) pos --;
      else node = st[node].link;
    }
  }
  void init(str _s) {
    makeNode(-1,0); node = pos = 0;
    trav(c,_s) extend(c);
    extend('$'); s.pop_back(); // terminal char
  }
  int maxPre(str x) { // max prefix of x which is substring
    int node = 0, ind = 0;
    while (1) {
      if (ind == sz(x) || !st[node].to.count(x[ind])) return
        ↪ind;
      node = st[node].to[x[ind]];
      FOR(i,st[node].len) {
        if (ind == sz(x) || x[ind] != s[st[node].fpos+i])
          return ind;
        ind ++;
      }
    }
  }
  vi sa; // generate suffix array
  void genSa(int x = 0, int len = 0) {
    if (!sz(st[x].to)) { // terminal node
      sa.pb(st[x].fpos-len);
      if (sa.back() >= sz(s)) sa.pop_back();
    } else {
      len += st[x].len;
      trav(t,st[x].to) genSa(t.s,len);
    }
  }
};
```

TandemRepeats.h

Description: Used only once. Main-Lorentz algorithm finds all (x, y) such that `s.substr(x,y-1) == s.substr(x+y,y-1)`.
Time: $\mathcal{O}(N \log N)$

"Z.h" fe5c66, 46 lines

```
struct TandemRepeats {
  str S;
  vector<array<int,3>> al;
  // (t[0],t[1],t[2]) -> exists repeating substr starting
  // at x with length t[0]/2 for all t[1] <= x <= t[2]
  vector<array<int,3>> solveLeft(str s, int m) {
    vector<array<int,3>> v;
    vi v2 = getPrefix(str(begin(s)+m+1,end(s)),
            str(begin(s),begin(s)+m+1));
    str V = str(begin(s),begin(s)+m+2); reverse(all(V));
    vi v1 = z(V); reverse(all(v1));
    FOR(i,m+1) if (v1[i]+v2[i] >= m+2-i) {
      int lo = max(1,m+2-i-v2[i]), hi = min(v1[i],m+1-i);
      lo = i-lo+1, hi = i-hi+1; swap(lo,hi);
      v.pb({2*(m+1-i),lo,hi});
    }
    return v;
  }
  void divi(int l, int r) {
    if (l == r) return;
    int m = (l+r)/2; divi(l,m); divi(m+1,r);
    str t(begin(S)+l,begin(S)+r+1);
    m = (sz(t)-1)/2;
    auto a = solveLeft(t,m);
    reverse(all(t));
    auto b = solveLeft(t,sz(t)-2-m);
    trav(x,a) al.pb({x[0],x[1]+l,x[2]+l});
    trav(x,b) {
      int ad = r-x[0]+1;
      al.pb({x[0],ad-x[2],ad-x[1]});
    }
  }
  void init(str _S) { S = _S; divi(0,sz(S)-1); }
  vi genLen() {
    // min length of repeating substr starting at each index
    priority_queue<pi,vpi,greater<pi>> m; m.push({MOD,MOD});
    vpi ins[sz(S)]; trav(a,al) ins[a[1]].pb({a[0],a[2]});
    vi len(sz(S));
    FOR(i,sz(S)) {
      trav(j,ins[i]) m.push(j);
      while (m.top().s < i) m.pop();
      len[i] = m.top().f;
    }
    return len;
  }
};
```

Various (10)

10.1 Dynamic programming

When doing DP on intervals:
$a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i, j)$, where the (minimal) optimal k increases with both i and j,

- one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$.
- This is known as Knuth DP. Sufficient criteria for this are if $f(b, c) \le f(a, d)$ and $f(a, c) + f(b, d) \le f(a, d) + f(b, c)$ for all $a \le b \le c \le d$.

- Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

CircLCS.h

Description: For strings a, b calculates longest common subsequence of a with all rotations of b

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

574233, 47 lines

```
pi dp[2001][4001];
str A,B;

void init() {
  FOR(i,1,sz(A)+1) FOR(j,1,sz(B)+1) {
    // naive LCS, store where value came from
    pi& bes = dp[i][j]; bes = {-1,-1};
    ckmax(bes,{dp[i-1][j].f,0});
    ckmax(bes,{dp[i-1][j-1].f+(A[i-1] == B[j-1]),-1});
    ckmax(bes,{dp[i][j-1].f,-2});
    bes.s *= -1;
  }
}
void adjust(int col) { // remove col'th character of b, fix DP
  int x = 1; while (x <= sz(A) && dp[x][col].s == 0) x ++;
  if (x > sz(A)) return; // no adjustments to dp
  pi cur = {x,col}; dp[cur.f][cur.s].s = 0;
  while (cur.f <= sz(A) && cur.s <= sz(B)) {
    // every dp[cur.f][y >= cur.s].f decreased by 1
    if (cur.s < sz(B) && dp[cur.f][cur.s+1].s == 2) {
      cur.s ++;
      dp[cur.f][cur.s].s = 0;
    } else if (cur.f < sz(A) && cur.s < sz(B)
    && dp[cur.f+1][cur.s+1].s == 1) {
      cur.f ++, cur.s ++;
      dp[cur.f][cur.s].s = 0;
    } else cur.f ++;
  }
}
int getAns(pi x) {
  int lo = x.s-sz(B)/2, ret = 0;
  while (x.f && x.s > lo) {
    if (dp[x.f][x.s].s == 0) x.f --;
    else if (dp[x.f][x.s].s == 1) ret ++, x.f --, x.s --;
    else x.s --;
  }
  return ret;
}
int circLCS(str a, str b) {
  A = a, B = b+b; init();
  int ans = 0;
  FOR(i,sz(b)) {
    ckmax(ans,getAns({sz(a),i+sz(b)}));
    adjust(i+1);
  }
  return ans;
}
```

10.2 Debugging tricks

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

10.3 Optimization tricks

10.3.1 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.
- `FOR(b,k) FOR(i,1<<K) if (i&1<<b) D[i] += D[i^(1<<b)];` computes all sums of subsets.

10.3.2 Pragmas

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

10.4 Other languages

Main.java

Description: Basic template/info for Java

11488d, 14 lines

```
import java.util.*;
import java.math.*;
import java.io.*;
public class Main {
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new
      ↪InputStreamReader(System.in));
    PrintStream out = System.out;
    StringTokenizer st = new StringTokenizer(br.readLine());
    assert st.hasMoreTokens(); // enable with java -ea main
    out.println("v=" + Integer.parseInt(st.nextToken()));
    ArrayList<Integer> a = new ArrayList<>();
    a.add(1234); a.get(0); a.remove(a.size()-1); a.clear();
  }
}
```

Python3.py

Description: Python3 (not Pypy3) demo, solves CF Good Bye 2018 G - Factorisation Collaboration

41 lines

```
from math import *
import sys
import random
def nextInt():
  return int(input())
def nextStrs():
  return input().split()
def nextInts():
  return list(map(int,nextStrs()))

n = nextInt()
v = [n]
def process(x):
  global v
  x = abs(x)
  V = []
  for t in v: # print(type(t)) -> <class 'int'>
    g = gcd(t,x)
    if g != 1:
      V.append(g)
    if g != t:
      V.append(t//g)
  v = V
for i in range(50):
  x = random.randint(0,n-1)
  if gcd(x,n) != 1:
    process(x)
  else:
    sx = x*x%n # assert(gcd(sx,n) == 1)
    print(f"sqrt {sx}") # print value of var
    sys.stdout.flush()
    X = nextInt()
    process(x+X)
    process(x-X)
print(f'! {len(v)}',end='')
for i in v:
  print(f' {i}',end='')
print()
sys.stdout.flush() # sys.exit(0) -> exit
# sys.setrecursionlimit(int(1e9)) -> stack size
# print(f'{ans:=.6f}') -> print ans to 6 decimal places
```