BRAC University

BRACU_Crows

Arman Ferdous, Ruhan Habib, Md Mazed Hossain

ICPC World Finals 2024-25

September 4, 2025

Contest (1)

instructions.txt
19 lines

```
1. vi .bashrc: export PATH="$PATH:$HOME/cp"
2. mkdir -p cp/bits/ && cd cp && vi cf (Type)
3. chmod +x cf; restart terminal
4. Type stdc++.h, template.cpp, hash.sh
-------- Kate --------
1. Go to Settings->Configure Kate.
 1. Editing->Default input mode->Vim
 2. Vi input mode->Insert mode->jk = <esc>
 3. Appearance->Turn off dynamic w.w.
 4. Color Themes->Gruvbox
 5. Terminal->Turn off hide console
    (View->Tool Views->Show sidebars is on)
2. Hotkey: Focus Terminal Panel=F4->"Reassign"
-------- Windows --------
1. Using cmd: echo %PATH%. Using Powershell:
    echo $env:PATH
2. Add path using cmd: set PATH=%PATH%;C:\
    Program Files\CodeBlocks\MinGW\bin
   It should be the directory where g++ is.
3. If we're using g++ of CodeBlocks, fsanitize
    won't be available :(
4. Write cf.bat at some directory. Ensure that
    directory is in PATH.
```

cf.sh
3 lines

```
#!/bin/bash
code=$1
g++ ${code}.cpp -o $code -std=c++20 -g -DDeBuG
    -Wall -Wshadow -fsanitize=address,
    undefined && ./$code
```

hash.sh
1 lines

```
cpp -dD -P -fpreprocessed | tr -d '[:space:]'|
    md5sum |cut -c-6
```

stdc++.h
90f4a7, 29 lines

```
#include <bits/stdc++.h>
using namespace std;
#define TT template <typename T

TT,typename=void> struct cerrok:false_type {};
TT> struct cerrok <T, void_t<decltype(cerr <<
    declval<T>() )>> : true_type {};

TT> constexpr void p1 (const T &x);
TT, typename V> void p1(const pair<T, V> &x) {
  cerr << "{"; p1(x.first); cerr << ", ";
  p1(x.second); cerr << "}";
}
TT> constexpr void p1 (const T &x) {
  if constexpr (cerrok<T>::value) cerr << x;
  else { int f = 0; cerr << '{';
    for (auto &i: x)
      cerr << (f++ ? ", " : ""), p1(i);
    cerr << "}";
} }
void p2() { cerr << "]\n"; }
TT, typename... V> void p2(T t, V... v) {
  p1(t);
  if (sizeof...(v)) cerr << ", ";
  p2(v...);
}

#ifdef DeBuG
#define dbg(x...) {cerr << "\t\e[93m"<<
    __func__<<":"<<__LINE__<<" [" << #x << "]
    = ["; p2(x); cerr << "\e[0m";}
#endif
```

template.cpp
640c64, 19 lines

```
// BRACU_Crows
#include "bits/stdc++.h"
using namespace std;

#ifndef DeBuG
  #define dbg(...)
#endif

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

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

stress.sh
14 lines

```
#!/bin/bash
cf gen > in              # input generator
cf bf < in > exp         # bruteforce
cf code < in > out       # buggy code name

for ((i = 1; ; ++i)) do
   echo $i
   ./gen > in
   ./bf < in > exp
   ./code < in > out    # buggy code name
   diff -w exp out || break
done
# Shows expected first, then user
notify-send "bug found!!!!"
```

cf.bat
5 lines

```
@echo off
setlocal
set prog=%1
g++ %prog%.cpp -o %prog% -DDeBuG -std=c++17 -g
    -Wall -Wshadow && .\%prog%
endlocal
```

Mathematics (2)

2.1 Equations

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

Cramer: Given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A} \quad \text{[where } A_i' \text{ is } A \text{ with the } i\text{'th column replaced by } b.]$$

Vieta: Let $P(x) = a_n x^n + ... + a_0$, be a polynomial with complex coefficients and degree n, having complex roots $r_n, ..., r_1$. Then for any integer $0 \le k \le n$,

$$\sum_{1 \le i_1 < i_2 < ... < i_k \le n} r_{i_1} r_{i_2} ... r_{i_k} = (-1)^k \frac{a_{n-k}}{a_n}$$

Rational Root Theorem: If $\frac{p}{q}$ is a reduced rational root of a polynomial with **integer coeffs**, then $p \mid a_0$ and $q \mid a_n$

2.2 Ceils and Floors

For $x, y \in \mathbb{R}$, $m, n \in \mathbb{Z}$:

- $\lfloor x \rfloor \le x < \lfloor x \rfloor + 1$; $\lceil x \rceil - 1 < x \le \lceil x \rceil$
- $-\lfloor x \rfloor = \lceil -x \rceil$; $-\lceil x \rceil = \lfloor -x \rfloor$
- $\lfloor x + n \rfloor = \lfloor x \rfloor + n$, $\lceil x + n \rceil = \lceil x \rceil + n$
- $\lfloor x \rfloor = m \Leftrightarrow x - 1 < m \le x < m + 1$
- $\lceil x \rceil = n \Leftrightarrow n - 1 < x \le n < x + 1$
- If $n > 0$, $\lfloor \frac{\lfloor x \rfloor + m}{n} \rfloor = \lfloor \frac{x+m}{n} \rfloor$
- If $n > 0$, $\lceil \frac{\lceil x \rceil + m}{n} \rceil = \lceil \frac{x+m}{n} \rceil$
- If $n > 0$, $\lfloor \frac{\lfloor \frac{x}{m} \rfloor}{n} \rfloor = \lfloor \frac{x}{mn} \rfloor$
- If $n > 0$, $\lceil \frac{\lceil \frac{x}{m} \rceil}{n} \rceil = \lceil \frac{x}{mn} \rceil$
- For $m, n > 0$, $\sum_{k=1}^{n-1} \lfloor \frac{km}{n} \rfloor = \frac{(m-1)(n-1) + \gcd(m,n) - 1}{2}$

2.3 Recurrences

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

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

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

2.4 Trigonometry

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

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

V, W are sides opposite to angles v, w.
$a \cos x + b \sin x = r \cos(x - \phi)$
$a \sin x + b \cos x = r \sin(x + \phi)$
where $r = \sqrt{a^2 + b^2}$, $\phi = \operatorname{atan2}(b, a)$.

2.5 Geometry

2.5.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Law of sines, cosines & tangents:
$\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$.....(1)
$a^2 = b^2 + c^2 - 2bc \cos \alpha$.....(2)
$\frac{a+b}{a-b} = \frac{\tan((\alpha+\beta)/2)}{\tan((\alpha-\beta)/2)}$.....(3)

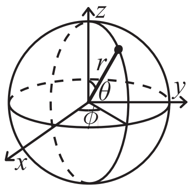
2.5.2 Quadrilaterals

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

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

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

2.5.3 Spherical coordinates



$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2+y^2+z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \mathrm{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \mathrm{atan2}(y,x)$$

2.6 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x = 1+\tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$
$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x) \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$
$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.7 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1}-c^a}{c-1}, c \neq 1$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = S_2 \times \frac{3n^2+3n-1}{5} = S_4$$
$$b\sum_{k=0}^{n-1}(a+kd)r^k = \frac{ab-(a+nd)br^n}{1-r} + \frac{dbr(1-r^n)}{(1-r)^2}$$

To compute $1^k + ... + n^k$ in $\mathcal{O}(k\lg k + k\lg MOD)$ compute first $t = k+2$ sums $y_1, ..., y_t$, then interpolate. Let $P = \prod_{i=1}^{t}(n-i)$. Then answer for n is

$$\sum_{i=1}^{t}\frac{P}{n-i}\cdot\frac{(-1)^{t-i}y_i}{(i-1)!(t-i)!}$$

Also $S_k = \frac{1}{k+1}\sum_{j=0}^{k}(-1)^j\binom{k+1}{j}B_j n^{k+1-j}$ where B_i are Bernoulli numbers.

2.8 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$
$$(1-x)^{-r} = \sum_{i=0}^{\infty}\binom{r+i-1}{i}x^i, (r \in \mathbb{R})$$

2.9 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.9.1 Discrete distributions

Binomial distribution

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

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

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

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

Poisson distribution

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

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

2.9.2 Continuous distributions

Uniform distribution

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

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$
$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

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

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

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

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

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

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

2.10 Markov chains

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

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

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

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

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

2.11 Trivia

Pythagorean triples: The Pythagorean triples are uniquely generated by $a = k\cdot(m^2-n^2)$, $b = k\cdot(2mn)$, $c = k\cdot(m^2+n^2)$ with $m > n > 0$, $k > 0$, $\gcd(m,n) = 1$, both m,n not odd.
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 modulo n exists iff $n = 1, 2, 4$ or, $n = p^k, 2p^k$ where p is an odd prime. Furthermore, the number of roots are $\phi(\phi(n))$.
To Find Generator g of M, factor $M-1$ and get the distinct primes p_i. If $g^{(M-1)/p_i} \neq 1(MODM)$ for each p_i then g is a valid root. Try all g until a hit is found (usually found very quick).
Esitmates: $\sum_{d|n} d = O(n\log\log n)$.
Prime count: 5133 upto 5e4. 9592 upto 1e5. 17984 upto 2e5. 78498 upto 1e6. 5761455 upto 1e8.
max NOD $\leq n$: 100 for $n = 5e4$. 500 for $n = 1e7$. 2000 for $n = 1e10$. 200 000 for $n = 1e19$.
max Unique Prime Factors: 6 upto 5e5. 7 upto 9e6. 8 upto 2e8. 9 upto 6e9. 11 upto 7e12. 15 upto 3e19.
Quadratic Residue: $(\frac{a}{p})$ is 0 if $p|a$, 1 if a is a quadratic residue, -1 otherwise. Euler: $(\frac{a}{p}) = a^{(p-1)/2}(\mod p)$ (prime). Jacobi: if $n = p_1^{e_1}\cdots p_k^{e_k}$ then $(\frac{a}{n}) = \prod(\frac{a}{p_i})^{e_i}$.
Chicken McNugget. If a,b coprime, there are $\frac{1}{2}(a-1)(b-1)$ numbers not of form $ax+by$ $(x,y \geq 0)$, the largest being $ab-a-b$.

Data structures (3)

OrderStatisticTree.h

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

782797, 14 lines

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

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

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
```

```
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

SegmentTree.h

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

9f8f73, 61 lines

```
template<class S> struct segtree {
  int n; V<S> t;
  void init(int _) { n = _; t.assign(n+n-1, S()); }
  void init(const V<S>& v) {
    n = sz(v); t.assign(n + n - 1, S());
    build(0,0,n-1,v);
  } template <typename... T>
  void upd(int l, int r, const T&... v) {
    assert(0 <= l && l <= r && r < n);
    upd(0, 0, n-1, l, r, v...);
  }
  S get(int l, int r) {
    assert(0 <= l && l <= r && r < n);
    return get(0, 0, n-1, l, r);
  }
private:
  inline void push(int u, int b, int e) {
    if (t[u].lazy == 0) return;
    int mid = (b+e)>>1, rc = u+((mid-b+1)<<1);
    t[u+1].upd(b, mid, t[u].lazy);
    t[rc].upd(mid+1, e, t[u].lazy);
    t[u].lazy = 0;
  }
  void build(int u,int b,int e,const V<S>&v) {
    if (b == e) return void(t[u] = v[b]);
    int mid = (b+e)>>1, rc = u+((mid-b+1)<<1);
    build(u+1, b,mid,v); build(rc, mid+1,e,v);
    t[u] = t[u+1] + t[rc];
  } template<typename... T>
  void upd(int u, int b, int e, int l, int r, const T&... v) {
    if (l <= b && e <= r) return t[u].upd(b, e, v...);
    push(u, b, e);
    int mid = (b+e)>>1, rc = u+((mid-b+1)<<1);
    if (l<=mid) upd(u+1, b, mid, l, r, v...);
    if (mid<r) upd(rc, mid+1, e, l, r, v...);
    t[u] = t[u+1] + t[rc];
  }
  S get(int u, int b, int e, int l, int r) {
    if (l <= b && e <= r) return t[u];
    push(u, b, e);
    S res; int mid = (b+e)>>1, rc = u+((mid-b+1)<<1);
    if (r<=mid) res = get(u+1, b, mid, l, r);
    else if (mid<l) res = get(rc,mid+1,e,l,r);
    else res = get(u+1, b, mid, l, r) + get(rc, mid+1, e, l, r);
    t[u] = t[u+1] + t[rc]; return res;
  }
}; // Hash upto here = 773c09
/* (1) Declaration:
Create a node class. Now, segtree<node> T;
T.init(10) creates everything as node()
Consider using V<node> leaves to build
(2) upd(l, r, ...v): update range [l, r]
order in ...v must be same as node.upd() fn */
struct node {
  ll sum = 0, lazy = 0;
  node () {} // write full constructor
  node operator+(const node &obj) {
    return {sum + obj.sum, 0};    }
  void upd(int b, int e, ll x) {
    sum += (e - b + 1) * x, lazy += x;
} };
```

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().

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

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

de4ad0, 21 lines

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

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").

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

8ec1c7, 30 lines

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

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

Lichao.h

Description: Add line segment, query minimum y at some x. Provide list of all query x points to constructor (offline solution). Use `add_segment(line, l, r)` to add a line segment $y = ax + b$ defined by $x \in [l, r)$. Use `query(x)` to get min at x.

Time: Both operations are $\mathcal{O}(\log \text{maxn})$.

566134, 43 lines

```
struct LiChaoTree {
  using Line = pair <ll, ll>;
  const ll linf = numeric_limits<ll>::max();
  int n; vector<ll> xl; vector<Line> dat;
  LiChaoTree(const vector<ll>& _xl):xl(_xl){
    n = 1; while(n < xl.size())n <<= 1;
    xl.resize(n,xl.back());
    dat = vector<Line>(2*n-1, Line(0,linf));
  }
  ll eval(Line f,ll x){return f.first * x + f.second;}
  void _add_line(Line f,int k,int l,int r){
    while (l != r) {
      int m = (l + r) / 2;
      ll lx = xl[l],mx = xl[m],rx = xl[r - 1];
      Line &g = dat[k];
      if(eval(f,lx) < eval(g,lx) && eval(f,rx) < eval(g,rx)) {
        g = f; return;
      }
      if(eval(f,lx) >= eval(g,lx) && eval(f,rx) >= eval(g,rx))
        return;
      if(eval(f,mx) < eval(g,mx))swap(f,g);
      if(eval(f,lx) < eval(g,lx)) k = k * 2 + 1, r = m;
      else k = k * 2 + 2, l = m;
    }
  }
  void add_line(Line f){_add_line(f,0,0,n);}
  void add_segment(Line f,ll lx,ll rx){
    int l = lower_bound(xl.begin(), xl.end(), lx) - xl.begin();
    int r = lower_bound(xl.begin(), xl.end(), rx) - xl.begin();
    int a0 = l, b0 = r, sz = 1; l += n;r += n;
    while(l < r){
      if(r & 1) r--, b0 -= sz, _add_line(f,r - 1,b0,b0 + sz);
      if(l & 1) _add_line(f,l - 1,a0,a0 + sz), l++, a0 += sz;
      l >>= 1, r >>= 1, sz <<= 1;
    }
  }
  ll query(ll x) {
    int i = lower_bound(xl.begin(), xl.end(),x) - xl.begin();
    i += n - 1; ll res = eval(dat[i],x);
    while (i) i = (i - 1) / 2, res = min(res, eval(dat[i], x));
    return res;
  }
};
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.

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

1754b4, 53 lines

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

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

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

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

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

```
    r->l = merge(l, r->l);
    return r->recalc(), r;
  }
}

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

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

FenwickTree.h

Description: `update(i,x): a[i] += x;` `query(i): sum in [0, i);` `lower_bound(sum): min pos st sum of [0, pos] >= sum, returns n if all < sum, or -1 if empty sum.`
Time: Both operations are $\mathcal{O}(\log N)$.

f74d01, 16 lines

```
struct FT {
  int n; V<ll> s;
  FT(int _n) : n(_n), s(_n) {}
  void update(int i, ll x) {
    for (; i < n; i |= i + 1) s[i] += x; }
  ll query(int i, ll r = 0) {
    for (; i > 0; i &= i - 1) r += s[i-1];
       return r; }
  int lower_bound(ll sum) {
    if (sum <= 0) return -1; int pos = 0;
    for (int pw = 1 << __lg(n); pw; pw >>= 1){
      if (pos+pw <= n && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
}; // Hash = d05c4f without lower_bound
```

FenwickTreeRange.h

Description: Range add Range sum with FT.
Time: Both operations are $\mathcal{O}(\log N)$.

8fc549, 11 lines

```
FT f1(n), f2(n);
// a[l...r] += v; 0 <= l <= r < n
auto upd = [&](int l, int r, ll v) {
  f1.update(l, v), f1.update(r + 1, -v);
  f2.update(l, v*(l-1)), f2.update(r+1, -v*r);
}; // a[l] + ... + a[r]; 0 <= l <= r < n
auto sum = [&](int l, int r) { ++r;
  ll sub = f1.query(l) * (l-1) - f2.query(l);
  ll add = f1.query(r) * (r-1) - f2.query(r);
  return add - sub;
};
```

FenwickTree2d.h

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

"FenwickTree.h" d53ef2, 20 lines

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

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);` `rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V| \log |V| + Q)$

7d2211, 15 lines

```
template<class T>
struct RMQ {
  V<V<T>> jmp;
  RMQ(const V<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= sz(V);
         pw *= 2, ++k) {
      jmp.emplace_back(sz(V) - pw * 2 + 1);
      rep(j,0,sz(jmp[k]))
        jmp[k][j] = min(jmp[k - 1][j], jmp[k -
             1][j + pw]);
    }
  }
  T query(int a, int b) {
    assert(a < b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 <<
         dep)]);
  }
};
```

MoTree.h

Description: Build Euler tour of $2N$ size - write node at first enter and last exit. Now, Path(u, v) with $in[u] < in[v]$ is a segment. If $\mathrm{lca}(u, v) = u$ then it is $[in[u], in[v]]$. Otherwise it is $[out[u], in[v]] +$ LCA node. Nodes that appear exactly once in each segment are relevant, ignore others, handle LCA separately.
Time: $\mathcal{O}\left(Q\sqrt{N}\right)$

MoUpdate.h

Description: Set block size $B = (2n^2)^{1/3}$. Sort queries by $(\lfloor \frac{L}{B} \rfloor, \lfloor \frac{R}{B} \rfloor, t)$, where t = number of updates before this query. Then process queries in sorted order, modify L, R and then apply/undo the updates to answer.
Time: $\mathcal{O}\left(Bq + qn^2/B^2\right)$ or $\mathcal{O}\left(qn^{2/3}\right)$ with that B.

Numerical (4)

4.1 Polynomials and recurrences

BerlekampMassey.h

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

"../number-theory/ModPow.h" 96548b, 20 lines

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;

  ll b = 1;
  rep(i,0,n) { ++m;
    ll d = s[i] % mod;
    rep(j,1,L+1) d = (d + C[j] * s[i - j]) %
         mod;
    if (!d) continue;
    T = C; ll coef = d * modpow(b, mod-2) %
         mod;
    rep(j,m,n) C[j] = (C[j] - coef * B[j - m])
         % mod;
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }

  C.resize(L + 1); C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i - j - 1]tr[j]$, given $S[0 \ldots \geq n - 1]$ and $tr[0 \ldots n - 1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2 \log k\right)$

f4e444, 26 lines

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

  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i,0,n+1) rep(j,0,n+1)
      res[i + j] = (res[i + j] + a[i] * b[j])
           % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i
           ] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };

  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;

  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }

  ll res = 0;
  rep(i,0,n) res = (res + pol[i + 1] * S[i]) %
       mod;
  return res;
}
```

Polynomial.h

c9b7b0, 17 lines

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

PolyRoots.h

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

"Polynomial.h" b00bfe, 23 lines

```
vector<double> polyRoots(Poly p, double xmin,
     double xmax) {
  if (sz(p.a)==2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$. For fast interpolation in $O(n \log^2 n)$ use Lagrange. $P(x) = \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$. To compute values $\prod_{j \neq i}(x_i - x_j)$ fast, compute $A(x) = \prod_{i=1}^{n}(x - x_i)$ with divide and conquer. The required values are $A'(x_i)$, (values at derivative), compute fast with multipoint evaluation.
Time: $\mathcal{O}(n^2)$

08bf48, 13 lines

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

4.2 Fourier transforms

FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x - i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (~1s for $N = 2^{22}$)

00ced6, 35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster
      if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] *
        x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) <<
      L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[
      i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j
        ,0,k) {
      C z = rt[j+k] * a[i+j+k]; // (25% faster
          if hand-rolled)
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1
      << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(
      in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4
      * n);
  return res;
}
```

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.
Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

"FastFourierTransform.h" b82773, 22 lines

```
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const
    vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut
      =int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (
      int)a[i] % cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (
      int)b[i] % cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] /
        (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] /
        (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag
        (outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(
        outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut +
        cv) % M;
  }
  return res;
}
```

NumberTheoreticTransform.h

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x] g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For arbitrary modulo, see FFTMod. `conv(a, b) = c`, where $c[x] = \sum a[i]b[x - i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in [0, mod).
Time: $\mathcal{O}(N \log N)$

"../number-theory/ModPow.h" ced03d, 35 lines

```
const ll mod = (119 << 23) + 1, root = 62; //
    = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7
    << 26, 479 << 21
// and 483 << 21 (same root). The last two are
    > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vl rt(2, 1);
  for (static int k = 2, s = 2; k < n; k *= 2,
      s++) {
    rt.resize(n);
    ll z[] = {1, modpow(root, mod >> s)};
    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1]
        % mod;
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) <<
      L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[
      i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j
        ,0,k) {
      ll z = rt[j + k] * a[i + j + k] % mod, &
          ai = a[i + j];
      a[i + j + k] = ai - z + (z > ai ? mod :
          0);
      ai += (ai + z >= mod ? z - mod : z);
    }
}
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = sz(a) + sz(b) - 1, B = 32 -
      __builtin_clz(s),
      n = 1 << B;
  int inv = modpow(n, mod - 2);
  vl L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  rep(i,0,n)
    out[-i & (n - 1)] = (ll)L[i] * R[i] % mod
        * inv % mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h

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

464cf3, 16 lines

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

FastSubsetConvolution.h

Description: $\text{ans}[i] = \sum_{j \subseteq i} f_j g_{i \oplus j}$
Time: $\mathcal{O}(n^2 2^n)$ or, $\mathcal{O}(N \log^2 N)$

7571e4, 28 lines

```
int f[N], g[N], fh[LG][N], gh[LG][N], h[LG][N
    ], ans[N];
void conv() {
for (int mask = 0; mask < 1 << n; ++mask) {
 fh[__builtin_popcount(mask)][mask]=f[mask];
 gh[__builtin_popcount(mask)][mask]=g[mask];
}
for (int i = 0; i <= n; ++i) {
 for (int j = 0; j < n; ++j)
  for (int mask = 0; mask < 1 << n; ++mask)
   if (mask & 1 << j) {
    fh[i][mask] += fh[i][mask ^ 1 << j];
    gh[i][mask] += gh[i][mask ^ 1 << j];
   }
}
for (int mask = 0; mask < 1 << n; ++mask) {
 for (int i = 0; i <= n; ++i)
  for (int j = 0; j <= i; ++j)
   h[i][mask]+=fh[j][mask] * gh[i-j][mask];
}
for (int i = 0; i <= n; ++i) {
 for (int j = 0; j < n; ++j)
  for (int mask = 0; mask < 1 << n; ++mask)
   if (mask & 1 << j)
    h[i][mask] -= h[i][mask ^ 1 << j];
}
for (int mask = 0; mask < 1 << n; ++mask)
 ans[mask]=h[__builtin_popcount(mask)][mask];
}
```

GCDconvolution.h

Description: Computes $c_1, ..., c_n$, where $c_k = \sum_{\gcd(i,j)=k} a_i b_j$. Generate all primes upto n into pr first using sieve.
Time: $\mathcal{O}(N \log \log N)$

bc0c7a, 21 lines

```
void fw_mul_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i = n/p; i>0; --i) a[i]+=a[i*p];
} } // A[i] = \sum_{j} a[i * j]

void bw_mul_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i=1; i*p <= n; ++i) a[i]-=a[i*p];
} } // From A get a

V<ll>gcd_conv (const V<ll>&a, const V<ll>&b){
  assert(sz(a) == sz(b)); int n = sz(a);
  auto A = a, B = b;
  fw_mul_transform(A); fw_mul_transform(B);
  for (int i = 1; i < n; ++i) A[i] *= B[i];
  bw_mul_transform(A); return A;
}
```

LCMconvolution.h

Description: Computes $c_1, ..., c_n$, where $c_k = \sum_{lcm(i,j)=k} a_i b_j$. Generate all primes upto n into pr first using sieve.

Time: $\mathcal{O}(N \log \log N)$ 1c5704, 21 lines

```
void fw_div_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i=1; i*p <= n; ++i) a[i*p]+=a[i];
} } // A[i] = \sum_{d | i} a[d]

void bw_div_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i=n/p; i>0; --i) a[i*p]-=a[i];
} } // From A get a

V<ll>lcm_conv (const V<ll>&a, const V<ll>&b){
  assert(sz(a) == sz(b)); int n = sz(a);
  auto A = a, B = b;
  fw_div_transform(A); fw_div_transform(B);
  for (int i = 1; i < n; ++i) A[i] *= B[i];
  bw_div_transform(A); return A;
}
```

4.3 Matrices

Matrix.h

Description: Basic operations on square matrices.

Usage:
```
Matrix<int, 3> A;
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
array<int, 3> vec = {1,2,3};
vec = (A^N) * vec;
```
6ab5db, 26 lines

```
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j
          ];
    return a;
  }
  array<T, N> operator*(const array<T, N>& vec
      ) const {
    array<T, N> ret{};
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] *
        vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

Determinant.h

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

Time: $\mathcal{O}(N^3)$ bd5cec, 15 lines

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

IntDeterminant.h

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

Time: $\mathcal{O}(N^3)$ 3313dc, 18 lines

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

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.

Time: $\mathcal{O}(n^2 m)$ 44c9ab, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return
          -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank
      < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h" 08e495, 7 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i
    +1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto
      fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.

Time: $\mathcal{O}(n^2 m)$ fa2d7a, 34 lines

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x,
    int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any())
        break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }

  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank
      < m)
}
```

XorBasis.h

Description: Maintain the basis of bit vectors.

Time: $\mathcal{O}\left(D^2/64\right)$ per insert 0daa2d, 19 lines

```
const int D = 1000; // use ll if < 64
struct Xor_Basis {
  V<int> who; V<bitset<D>> a;
  Xor_Basis () : who(D, -1) {}
  bool insert (bitset<D> x) {
    for (int i = 0; i < D; ++i)
      if (x[i] && who[i]!=-1) x^=a[who[i]];
    int pivot = -1;
    for (int i = 0; i < D; ++i)
      if (x[i]) { pivot = i; break; }
    if (pivot == -1) return false;
    // ^ null vector detected
    who[pivot] = sz(a);
    for (int i = 0; i < sz(a); ++i)
      if (a[i][pivot] == 1) a[i] ^= x;
    a.push_back(x);
    return true;
  }
};
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.

Time: $\mathcal{O}(n^3)$ ebfff6, 35 lines

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

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

```
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

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

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

Tridiagonal.h

Description: x = tridiagonal(d, p, q, b) solves the equation system

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

This is useful for solving problems on the type

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

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

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

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

115ed4, 25 lines

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

4.4 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}(\log((b-a)/\epsilon))$

31d45b, 14 lines

```
double gss(double a, double b, double (*f)(
    double)) {
  double r = (sqrt(5)-1)/2, eps = 1e-7;
  double x1 = b - r*(b-a), x2 = a + r*(b-a);
  double f1 = f(x1), f2 = f(x2);
  while (b-a > eps)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
```

HillClimbing.h

Description: Poor man's optimization for unimodal functions.

8eeeaf, 14 lines

```
typedef array<double, 2> P;

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

Integrate.h

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

4756fc, 7 lines

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

IntegrateAdaptive.h

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

92dd79, 15 lines

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

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

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. 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: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

aa8530, 68 lines

```
typedef double T; // long double, Rational,
    double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

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

struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;

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

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

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

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

Number theory (5)

5.1 Modular arithmetic

ModInverse.h

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

6f684f, 3 lines

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

ModPow.h

b83e45, 8 lines

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

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

ModLog.h

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

c040b8, 11 lines

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

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) $= \sum_{i=0}^{\mathrm{to}-1}(ki+c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant.

5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) |
    1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m
      -1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(
      to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

bbbd8f, 11 lines

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

ModSqrt.h

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

"ModPow.h" 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else
      no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p)
      ;
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works
      if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1)
      ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

LinearSieve.h

Description: Can be used to precompute multiplicative functions using $f(px) = f(p)f(x)$ when $p \nmid x$. We compute $f(px) = f(p^{e+1} \cdot x/p^e) = f(p^{e+1})f(x/p^e)$ by multiplicativity (bookkeeping e, the max power of p dividing x where p is the smallest prime dividing x). If $f(px)$ can be computed easily when $p \mid x$ then we can simplify the code.
Time: $\mathcal{O}(n)$

e696bd, 16 lines

```
int func[N],cnt[N]; bool isc[N]; V<int> prime;
void sieve (int n) {
  fill(isc, isc + n, false); func[1] = 1;
  for (int i = 2; i < n; ++i) {
    if (!isc[i]) {
      prime.push_back(i); func[i]=1; cnt[i]=1;
    }
    for (int j = 0; j < prime.size () && i *
        prime[j] < n; ++j) {
      isc[i * prime[j]] = true;
      if (i % prime[j] == 0) {
        func[i * prime[j]] = func[i] / cnt[i]
            * (cnt[i] + 1);
        cnt[i * prime[j]] = cnt[i] + 1; break;
      } else {
        func[i * prime[j]] = func[i] * func[
            prime[j]];
        cnt[i * prime[j]] = 1;
} } } }
```

phiFunction.h

Description: *Euler's* ϕ function is defined as $\phi(n) :=$ # of positive integers $\le n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1}p_2^{k_2}...p_r^{k_r}$ then $\phi(n) = (p_1-1)p_1^{k_1-1}...(p_r-1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1-1/p)$. $\sum_{d|n}\phi(d) = n$, $\sum_{1\le k\le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p \;\forall a$.

cf7d6d, 8 lines

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

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

FastEratosthenes.h

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

6b2912, 20 lines

```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
  const int S = (int)round(sqrt(LIM)), R = LIM
      / 2;
  vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/
      log(LIM)*1.1));
  vector<pii> cp;
  for (int i = 3; i <= S; i += 2) if (!sieve[i
      ]) {
    cp.push_back({i, i * i / 2});
    for (int j = i * i; j <= S; j += 2 * i)
        sieve[j] = 1;
  }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto &[p, idx] : cp)
      for (int i=idx; i < S+L; idx = (i+=p))
          block[i-L] = 1;
    rep(i,0,min(S, R - L))
      if (!block[i]) pr.push_back((L + i) * 2
          + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \bmod c$.

"ModMulLL.h" 60dcd1, 12 lines

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

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h" d8d98d, 18 lines

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n)
      + i; };
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y),
        n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

5.3 Divisibility

euclid.h

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

33ba8f, 5 lines

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

CRT.h

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

"euclid.h" 04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no
      solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

5.3.1 Bézout's identity

For $a \neq$, $b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to $ax + by = d$. If (x, y) is one solution, then all solutions are given by $(x + kb/d, y - ka/d)$, $k \in \mathbb{Z}$. Find one solution using egcd.

5.4 Fractions

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.

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

Time: $\mathcal{O}(\log(N))$ 27ab3e, 25 lines

```
struct Frac { ll p, q; };

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

5.5 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$, $\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$

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

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

If f multiplicative,
$\sum_{d|n} \mu(d) f(d) = \prod_{\text{prime } p|n} (1 - f(p))$ and
$\sum_{d|n} \mu^2(d) f(d) = \prod_{\text{prime } p|n} (1 + f(p))$.

If $s_f(n) = \sum_{i=1}^{n} f(i)$ is a prefix sum of mulitplicative f then $s_{f*g}(n) = \sum_{1 \leq xy \leq n} f(x) g(y)$.
Then $s_f(n) = \{s_{f*g}(n) - \sum_{d=2}^{n} s_f(\lfloor n/d \rfloor) g(d)\}/g(1)$ where $f * g(n) = \sum_{d|n} f(d) g(n/d)$ (Dirichlet).
Precompute (linear sieve) $O(n^{2/3})$ first values of s_f for complexity $O(n^{2/3})$.

Useful sums and convolutions: $\epsilon = \mu * \mathbf{1}$, id $= \phi * \mathbf{1}$, id $= g * \text{id}_2$, where $\epsilon(n) = [n = 1]$, $\mathbf{1}(n) = 1$, $\text{id}(n) = n$, $\text{id}_k(n) = n^k$,
$g(n) = \sum_{d|n} \mu(d) nd$.
coprime pairs in $[1, n]$ is $\sum_{d=1}^{n} \mu(d) \lfloor n/d \rfloor^2$. Sum of GCD pairs in $[1, n]$ is $\sum_{d=1}^{n} \phi(d) \lfloor n/d \rfloor^2$. Sum of LCM pairs in $[1, n]$ is $\sum_{d=1}^{n} (\frac{\lfloor n/d \rfloor (1 + \lfloor n/d \rfloor)}{2})^2 g(d)$, where g is defined above with $g(p^k) = p^k - p^{k+1}$.

Combinatorial (6)

6.1 Permutations

IntPerm.h

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

Time: $\mathcal{O}(n)$ 044568, 6 lines

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

multinomial.h

Description: Computes $\binom{v_0 + \cdots + v_{n-1}}{v_0, \ldots, v_{n-1}}$ a0a312, 6 lines

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

Cycles Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then $\sum_{n \geq 0} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$
Derangements Permutations of a set such that none of the elements appear in their original position. $D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$

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 $g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k) \phi(n/k)$.
Partition function Number of ways of writing n as a sum of positive integers, disregarding the order of the summands. $p(0) = 1$,
$p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$.
$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$
First few values: 1, 1, 2, 3, 5, 7, 11, 15, 22, 30.
$p(20) = 627, p(50) \approx 2e5, p(100) \approx 2e8$.
Lucas' Theorem: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \ldots + n_1 p + n_0$ and $m = m_k p^k + \ldots + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.
Bernoulli numbers EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able). $\sum \frac{B_i}{i!} x^i = \frac{x}{1 - e^{-x}}$.
$B[0, \ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$.
Sums of powers:

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

Stirling numbers of the first kind Number of permutations on n items with k cycles.
$c(n, k) = c(n-1, k-1) + (n-1) c(n-1, k)$,
$c(0, 0) = 1. \sum_{k=0}^{n} c(n, k) x^k = x(x+1) \ldots (x+n-1)$
$c(8, k) =$
$8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n, 2) =$
$0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$
Stirling numbers of the second kind Partitions of n distinct elements into exactly k non-empty subsets. $S(n, k) = S(n-1, k-1) + kS(n-1, k)$.
$S(n, 1) = S(n, n) = 1$.
$S(n, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$.
Eulerian numbers Number of n-permutations with exactly k rises (positions i with $p_i > p_{i-1}$).
$E(n, k) = (n-k) E(n-1, k-1) + (k+1) E(n-1, k)$.
$E(n, 0) = E(n, n-1) = 1$.
$E(n, k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$.
Bell numbers Total number of partitions of n distinct elements. $B(n) =$
$1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$.
$B(3) = 5 = \{a|b|c, a|bc, b|ac, c|ab, abc\}$. For p prime,
$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$.
Catalan numbers
$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$
$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2} C_n,\ C_{n+1} = \sum C_i C_{n-i}$
$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$
- UR path from $(0, 0)$ to (n, n) below $y = x$.
- 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.

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: $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$
Number of Spanning Trees Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--, mat[b][b]++` (and `mat[b][a]--, mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).
Erdős–Gallai theorem A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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