Moscow Institute of Physics and Technology

My Pity

Fedor Alekseev, Dmitry Ivaschenko, Daria Kolodzey

NEERC

2 December 2018

Contest (1)

Makefile

7 lines

```
CXXFLAGS = -Wall -Wextra -Wformat=2 -Dmypity
CXXFLAGS += -fsanitize=address,undefined -g
# CXXFLAGS += -ggdb
# CXXFLAGS += -O3 -march=native

%.exe: %.cpp
  g++ $< -o $@ ${CXXFLAGS}
```

.vimrc

11 lines

```
set nocp si aw ai is ts=2 sw=2 et tm=100 nu bg=dark
sy on
im jj <esc>

nmap <F6> :make %:r.exe<CR>
nmap <F10> :make %:r.exe<CR>:!for i in tests/??; do ./%:r.exe <$i
     >$i.out 2>$i.err; done

nmap <F8> :copen<CR>
nmap <S-F8> :cclose<CR>
nmap <F9> :cn<CR>
nmap <S-F9> :cp<CR>
```

template.cpp

21 lines

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

#ifdef mypity
#define debug(...) fprintf(stderr, __VA_ARGS__)
#define cdebug(...) cerr << __VA_ARGS__
#else
#define debug(...) do {} while (false)
#define cdebug(...) do {} while (false)
#endif

#define WHOLE(v) v.begin(),v.end()
#define sz(v) static_cast<int>(v.size())

using i64 = int64_t;

int main() {
  cin.sync_with_stdio(false);
  cin.tie(nullptr);
  cin.exceptions(cin.failbit);
}
```

makeprob.sh

3 lines

```
#!/usr/bin/bash
prob = $1
mkdir $prob && cp template/Makefile $1 && cp template/template.cpp
      $1/$1.cpp && mkdir $1/tests
```

troubleshoot.txt

52 lines

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

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

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

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

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

Mathematics (2)

2.1 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. $a_n = (d_1 n + d_2) r^n$.

2.2 Trigonometry

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

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

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

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

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

2.3 Geometry

2.3.1 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):

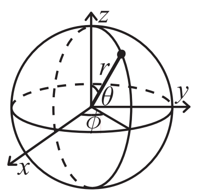
$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b+c}\right)^2\right]}$$

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

2.3.2 Spherical coordinates



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

2.4 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:

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

2.5 Sums

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

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

2.6 Series

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

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

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

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

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

Numerical (3)

PolyRoots.h

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

"Polynomial.h" 23 lines

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

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

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;
}
```

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

40 lines

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

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

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

FastFourierTransform.h

Description: Computes $\hat{f}(k) = \sum_x f(x) \exp(-2\pi ikx/N)$ for all k. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. a and b should be of roughly equal size. For convolutions of integers, consider using a number-theoretic transform instead, to avoid rounding issues.
Time: $\mathcal{O}(N \log N)$

<valarray> 29 lines

```
typedef valarray<complex<double> > carray;
void fft(carray& x, carray& roots) {
  int N = sz(x);
  if (N <= 1) return;
  carray even = x[slice(0, N/2, 2)];
  carray odd = x[slice(1, N/2, 2)];
  carray rs = roots[slice(0, N/2, 2)];
  fft(even, rs);
  fft(odd, rs);
  rep(k,0,N/2) {
    auto t = roots[k] * odd[k];
    x[k    ] = even[k] + t;
    x[k+N/2] = even[k] - t;
  }
}

typedef vector<double> vd;
vd conv(const vd& a, const vd& b) {
  int s = sz(a) + sz(b) - 1, L = 32-__builtin_clz(s), n = 1<<L;
  if (s <= 0) return {};
  carray av(n), bv(n), roots(n);
  rep(i,0,n) roots[i] = polar(1.0, -2 * M_PI * i / n);
  copy(all(a), begin(av)); fft(av, roots);
  copy(all(b), begin(bv)); fft(bv, roots);
  roots = roots.apply(conj);
  carray cv = av * bv; fft(cv, roots);
  vd c(s); rep(i,0,s) c[i] = cv[i].real() / n;
  return c;
}
```

NumberTheoreticTransform.h

Description: Can be used for convolutions modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For other primes/integers, use two different primes and combine with CRT. May return negative values.
Time: $\mathcal{O}(N \log N)$

"ModPow.h" 38 lines

```
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// 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.

typedef vector<ll> vl;
void ntt(ll* x, ll* temp, ll* roots, int N, int skip) {
  if (N == 1) return;
  int n2 = N/2;
  ntt(x     , temp, roots, n2, skip*2);
  ntt(x+skip, temp, roots, n2, skip*2);
  rep(i,0,N) temp[i] = x[i*skip];
  rep(i,0,n2) {
    ll s = temp[2*i], t = temp[2*i+1] * roots[skip*i];
    x[skip*i] = (s + t) % mod; x[skip*(i+n2)] = (s - t) % mod;
  }
}
void ntt(vl& x, bool inv = false) {
  ll e = modpow(root, (mod-1) / sz(x));
  if (inv) e = modpow(e, mod-2);
  vl roots(sz(x), 1), temp = roots;
  rep(i,1,sz(x)) roots[i] = roots[i-1] * e % mod;
  ntt(&x[0], &temp[0], &roots[0], sz(x), 1);
}
vl conv(vl a, vl b) {
  int s = sz(a) + sz(b) - 1; if (s <= 0) return {};
  int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1 << L;
  if (s <= 200) { // (factor 10 optimization for |a|,|b| = 10)
    vl c(s);
    rep(i,0,sz(a)) rep(j,0,sz(b))
      c[i + j] = (c[i + j] + a[i] * b[j]) % mod;
    return c;
  }
  a.resize(n); ntt(a);
  b.resize(n); ntt(b);
  vl c(n); ll d = modpow(n, mod-2);
  rep(i,0,n) c[i] = a[i] * b[i] % mod * d % mod;
  ntt(c, true); c.resize(s); return c;
}
```

Number theory (4)

4.1 Modular arithmetic

ModInverse.h

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

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;
```

ModSum.h

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

19 lines

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

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

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 large c.
Time: $\mathcal{O}(64/bits \cdot \log b)$, where $bits = 64 - k$, if we want to deal with k-bit numbers.

19 lines

```
typedef unsigned long long ull;
const int bits = 10;
// if all numbers are less than 2^k, set bits = 64-k
const ull po = 1 << bits;
ull mod_mul(ull a, ull b, ull &c) {
  ull x = a * (b & (po - 1)) % c;
  while ((b >>= bits) > 0) {
    a = (a << bits) % c;
    x += (a * (b & (po - 1))) % c;
  }
  return x % c;
}
ull mod_pow(ull a, ull b, ull mod) {
  if (b == 0) return 1;
  ull res = mod_pow(a, b / 2, mod);
  res = mod_mul(res, res, mod);
  if (b & 1) return mod_mul(res, a, mod);
  return res;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.
Time: $\mathcal{O}(\log^2 p)$ worst case, often $\mathcal{O}(\log p)$

"ModPow.h" 30 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);
  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;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
}
```

4.2 Primality

eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit. isprime$[i]$ is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

11 lines

```
const int MAX_PR = 5000000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;
  rep(i,2,lim) if (isprime[i]) pr.push_back(i);
  return pr;
}
```

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.
Time: 15 times the complexity of $a^b \mod c$.

"ModMulLL.h" 16 lines

```
bool prime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  rep(i,0,15) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = mod_pow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = mod_mul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run init(bits), where bits is the length of the numbers you use. Returns factors of the input without duplicates.
Time: Expected running time should be good enough for 50-bit numbers.

"ModMulLL.h", "MillerRabin.h", "eratosthenes.h" 35 lines

```
vector<ull> pr;
ull f(ull a, ull n, ull &has) {
  return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
  vector<ull> res;
  for (int i = 0; i < sz(pr) && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) d /= pr[i];
      res.push_back(pr[i]);
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
      for (; c==1; c = __gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c) res.push_back(d);
        break;
      }
    }
  }
  return res;
}
void init(int bits) {//how many bits do we use?
  vi p = eratosthenes_sieve(1 << ((bits + 2) / 3));
  pr.assign(all(p));
}
```

4.3 Divisibility

euclid.h

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

7 lines

```
ll gcd(ll a, ll b) { return __gcd(a, b); }

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

```
}
```

Euclid.java

Description: Finds {x, y, d} s.t. ax + by = d = gcd(a, b).

11 lines

```
static BigInteger[] euclid(BigInteger a, BigInteger b) {
  BigInteger x = BigInteger.ONE, yy = x;
  BigInteger y = BigInteger.ZERO, xx = y;
  while (b.signum() != 0) {
    BigInteger q = a.divide(b), t = b;
    b = a.mod(b); a = t;
    t = xx; xx = x.subtract(q.multiply(xx)); x = t;
    t = yy; yy = y.subtract(q.multiply(yy)); y = t;
  }
  return new BigInteger[]{x, y, a};
}
```

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

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

phiFunction.h

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

10 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] /= i) *= i-1;
}
```

4.4 Fractions

ContinuedFractions.h

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

21 lines

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
  for (;;) {
    ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
       a = (ll)floor(y), b = min(a, lim),
       NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that gives us a
      // better approximation; if b = a/2, we *may* have one.
      // Return {P, Q} here for a more canonical approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. 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: $\mathcal{O}(\log(N))$

24 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]
  assert(!f(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;
}
```

4.5 Chinese remainder theorem

chinese.h

Description: Chinese Remainder Theorem.
`chinese(a, m, b, n)` returns a number x, such that $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$. For not coprime n, m, use `chinese_common`. Note that all numbers must be less than 2^{31} if you have Z = unsigned long long.
Time: $\log(m + n)$

"euclid.h" 13 lines

```
template<class Z> Z chinese(Z a, Z m, Z b, Z n) {
  Z x, y; euclid(m, n, x, y);
  Z ret = a * (y + m) % m * n + b * (x + n) % n * m;
  if (ret >= m * n) ret -= m * n;
  return ret;
}

template<class Z> Z chinese_common(Z a, Z m, Z b, Z n) {
  Z d = gcd(m, n);
  if (((b -= a) %= n) < 0) b += n;
  if (b % d) return -1; // No solution
  return d * chinese(Z(0), m/d, b/d, n/d) + a;
}
```

4.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

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

4.8 Estimates

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

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

Combinatorial (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	20
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14	2e18

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

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

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 Binomials

binomialModPrime.h

Description: Lucas' thm: 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 $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.

Time: $\mathcal{O}(\log_p n)$ 10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

multinomial.h

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

5.3 General purpose numbers

5.3.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

5.3.2 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \ge j$, k j:s s.t. $\pi(j) > j$.

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

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

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

5.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

5.3.4 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,

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

5.3.5 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.

Geometry (6)

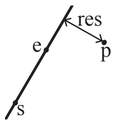
Point.h

35 lines

```
template<class T>
struct PointT {
  using P = PointT;
  T x, y;
  PointT() = default;
  PointT(T x, T y): x(x), y(y) {}
  explicit PointT(P a, P b): PointT(b - a) {}

  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }

  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }

  T operator*(P p) const { return x*p.x + y*p.y; }
  T operator%(P p) const { return x*p.y - y*p.x; }

  T hypot2() const { return x*x + y*y; }
  double hypot() const { return hypot(x, y); }

  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }

  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }

  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(
        x * cos(a) - y * sin(a),
        x * sin(a) + y * cos(a)
    );
  }
};
```

lineDistance.h

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



"Point.h" 4 lines

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

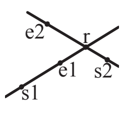
SegmentDistance.h

"Point.h" 5 lines

```
double segment_point_distance(Point a, Point b, Point p) {
  if (a==b) return (p-a).hypot();
  auto d = (b-a).hypot2(), t = min(d, max(.0, (p-a) * (b-a)));
  return ((p-a) * d - (b-a) * t).hypot() / d;
}
```

lineIntersection.h

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



"Point.h" 9 lines

```
template<class P>
int lineIntersection(const P& s1, const P& e1, const P& s2,
    const P& e2, P& r) {
  if ((e1-s1).cross(e2-s2)) { //if not parallell
    r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
    return 1;
  } else
    return -((e1-s1).cross(s2-s1)==0 || s2==e2);
}
```

onSegment.h

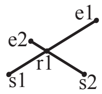
Description: Returns true iff p lies on the line segment from s to e. Intended for use with e.g. Point<long long> where overflow is an issue. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" 5 lines

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

SegmentIntersection.h

Description:
If a unique intersetion point between the line segments going from s1 to e1 and from s2 to e2 exists r1 is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists 2 is returned and r1 and r2 are set to the two ends of the common line. The wrong position will be returned if P is Point<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long. Use segmentIntersectionQ to get just a true/false answer.



"Point.h" 27 lines

```
template<class P>
int segmentIntersection(const P& s1, const P& e1,
    const P& s2, const P& e2, P& r1, P& r2) {
  if (e1==s1) {
    if (e2==s2) {
      if (e1==e2) { r1 = e1; return 1; } //all equal
      else return 0; //different point segments
    } else return segmentIntersection(s2,e2,s1,e1,r1,r2);//swap
  }
  //segment directions and separation
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = v1.cross(d), a2 = v2.cross(d);
  if (a == 0) { //if parallel
    auto b1=s1.dot(v1), c1=e1.dot(v1),
         b2=s2.dot(v1), c2=e2.dot(v1);
    if (a1 || a2 || max(b1,min(b2,c2))>min(c1,max(b2,c2)))
      return 0;
    r1 = min(b2,c2)<b1 ? s1 : (b2<c2 ? s2 : e2);
    r2 = max(b2,c2)>c1 ? e1 : (b2>c2 ? s2 : e2);
    return 2-(r1==r2);
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  if (0<a1 || a<-a1 || 0<a2 || a<-a2)
    return 0;
  r1 = s1-v1*a2/a;
  return 1;
}
```

SegmentIntersectionQ.h

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

"Point.h" 16 lines

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

CircleIntersection.h

Description: Computes a pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 14 lines

```
typedef Point<double> P;
bool circleIntersection(P a, P b, double r1, double r2,
    pair<P, P>* out) {
  P delta = b - a;
  assert(delta.x || delta.y || r1 != r2);
  if (!delta.x && !delta.y) return false;
  double r = r1 + r2, d2 = delta.dist2();
  double p = (d2 + r1*r1 - r2*r2) / (2.0 * d2);
  double h2 = r1*r1 - p*p*d2;
  if (d2 > r*r || h2 < 0) return false;
  P mid = a + delta*p, per = delta.perp() * sqrt(h2 / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

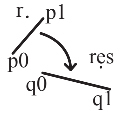
PolygonCenter.h

Description: Returns the center of mass for a polygon.

"Point.h" 10 lines

```
typedef Point<double> P;
Point<double> polygonCenter(vector<P>& v) {
  auto i = v.begin(), end = v.end(), j = end-1;
  Point<double> res{0,0}; double A = 0;
  for (; i != end; j=i++) {
    res = res + (*i + *j) * j->cross(*i);
    A += j->cross(*i);
  }
  return res / A / 3;
}
```

linearTransformation.h



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

"Point.h" 6 lines

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

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line.

"Point.h" 11 lines

```
template<class P>
int sideOf(const P& s, const P& e, const P& p) {
  auto a = (e-s).cross(p-s);
  return (a > 0) - (a < 0);
}
template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

insidePolygon.h

Description: Returns true if p lies within the polygon described by the points between iterators begin and end. If strict false is returned when p is on the edge of the polygon. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within epsilon from an edge should be considered as on the edge replace the line "if (onSegment..." with the comment bellow it (this will cause overflow for int and long long).

Usage: `typedef Point<int> pi;`
`vector<pi> v; v.push_back(pi(4,4));`
`v.push_back(pi(1,2)); v.push_back(pi(2,1));`
`bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);`

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

"Point.h", "onSegment.h", "SegmentDistance.h" 14 lines

```
template<class It, class P>
bool insidePolygon(It begin, It end, const P& p,
    bool strict = true) {
  int n = 0; //number of isects with line from p to (inf,p.y)
  for (It i = begin, j = end-1; i != end; j = i++) {
    //if p is on edge of polygon
    if (onSegment(*i, *j, p)) return !strict;
    //or: if (segDist(*i, *j, p) <= epsilon) return !strict;
    //increment n if segment intersects line from p
    n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y &&
        ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
  }
  return n&1; //inside if odd number of intersections
}
```

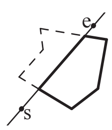
PolygonArea.h

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

"Point.h" 6 lines

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

PolygonCut.h



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

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

"Point.h", "lineIntersection.h" 15 lines

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

ConvexHull.h

28 lines

```
vector<Point> hull(vector<Point> pts) {
  sort(WHOLE(pts));
  pts.erase(unique(WHOLE(pts)), pts.end());
  sort(pts.begin()+1, pts.end(), [pivot=pts[0]](Point a, Point b){
    auto cross = (a - pivot) % (b - pivot);
    return cross > 0 || ( // Warning: consider using epsilon!
      cross == 0 && (pivot - a) * (b - a) < 0);
  });

  {  // Iff non strictly convex
    auto rit = pts.rbegin();
    while (rit != pts.rend()
        && 0 == (pts.back() - pts[0]) % (*rit - pts[0]))
      ++rit;
    reverse(pts.rbegin(), rit);
  }

  vector<Point> ret;
  for (auto p : pts) // Warning: consider using epsilon!
    while (sz(ret) > 1 && 0 >= // > 0 non-strict convex
        (ret.back() - ret[sz(ret) - 2]) %
        (p - ret.back()))
      ret.pop_back();
    ret.push_back(pts[i]);
  }

  return ret;
}
```

PolygonDiameter.h

Description: Calculates the max squared distance of a set of points.

"ConvexHull.h" 19 lines

```
vector<pii> antipodal(const vector<P>& S, vi& U, vi& L) {
  vector<pii> ret;
  int i = 0, j = sz(L) - 1;
  while (i < sz(U) - 1 || j > 0) {
    ret.emplace_back(U[i], L[j]);
    if (j == 0 || (i != sz(U)-1 && (S[L[j]] - S[L[j-1]])
          .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
    else --j;
  }
  return ret;
}

pii polygonDiameter(const vector<P>& S) {
  vi U, L; tie(U, L) = ulHull(S);
  pair<ll, pii> ans;
  trav(x, antipodal(S, U, L))
    ans = max(ans, {(S[x.first] - S[x.second]).dist2(), x});
  return ans.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a given polygon (counter-clockwise order). The polygon must be such that every point on the circumference is visible from the first point in the vector. It returns 0 for points outside, 1 for points on the circumference, and 2 for points inside.

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

"Point.h", "sideOf.h", "onSegment.h" 22 lines

```
typedef Point<ll> P;
int insideHull2(const vector<P>& H, int L, int R, const P& p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return 0;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == sz(H)))
      return 1;
    return 2;
  }
  int mid = L + len / 2;
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}

int insideHull(const vector<P>& hull, const P& p) {
  if (sz(hull) < 3) return onSegment(hull[0], hull.back(), p);
  else return insideHull2(hull, 1, sz(hull), p);
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear points. isct(a, b) returns a pair describing the intersection of a line with the polygon: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon.

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

"Point.h" 63 lines

```
ll sgn(ll a) { return (a > 0) - (a < 0); }
typedef Point<ll> P;
struct HullIntersection {
  int N;
  vector<P> p;
```

```
  vector<pair<P, int>> a;

  HullIntersection(const vector<P>& ps) : N(sz(ps)), p(ps) {
    p.insert(p.end(), all(ps));
    int b = 0;
    rep(i,1,N) if (P{p[i].y,p[i].x} < P{p[b].y, p[b].x}) b = i;
    rep(i,0,N) {
      int f = (i + b) % N;
      a.emplace_back(p[f+1] - p[f], f);
    }
  }

  int qd(P p) {
    return (p.y < 0) ? (p.x >= 0) + 2
         : (p.x <= 0) * (1 + (p.y <= 0));
  }

  int bs(P dir) {
    int lo = -1, hi = N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (make_pair(qd(dir), dir.y * a[mid].first.x) <
        make_pair(qd(a[mid].first), dir.x * a[mid].first.y))
        hi = mid;
      else lo = mid;
    }
    return a[hi%N].second;
  }

  bool isign(P a, P b, int x, int y, int s) {
    return sgn(a.cross(p[x], b)) * sgn(a.cross(p[y], b)) == s;
  }

  int bs2(int lo, int hi, P a, P b) {
    int L = lo;
    if (hi < lo) hi += N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (isign(a, b, mid, L, -1)) hi = mid;
      else lo = mid;
    }
    return lo;
  }

  pii isct(P a, P b) {
    int f = bs(a - b), j = bs(b - a);
    if (isign(a, b, f, j, 1)) return {-1, -1};
    int x = bs2(f, j, a, b)%N,
        y = bs2(j, f, a, b)%N;
    if (a.cross(p[x], b) == 0 &&
        a.cross(p[x+1], b) == 0) return {x, x};
    if (a.cross(p[y], b) == 0 &&
        a.cross(p[y+1], b) == 0) return {y, y};
    if (a.cross(p[f], b) == 0) return {f, -1};
    if (a.cross(p[j], b) == 0) return {j, -1};
    return {x, y};
  }
};
```

Point3D.h

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

32 lines

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

PolyhedronVolume.h

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

6 lines

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

sphericalDistance.h

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

8 lines

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

Data structures (7)

FenwickTree.h

26 lines

```
struct Fenwick {
  vector<i64> s;
  explicit Fenwick(int size): s(size, 0) {}

  void add(int at, i64 delta) {
    for (; at < sz(s); at |= at + 1)
      s[at] += delta;
  }

  i64 get_prefix(int end) {
    i64 sum = 0;
    for (; end > 0; end &= end - 1)
      sum += s[end - 1];
    return sum;
  }

  int lower_bound(i64 sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1)
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    return pos;
  }
};
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J. Requires that the elements to be updated are known in advance.

"FenwickTree.h" 34 lines

```
struct Fenwick2D {
  vector<vector<int>> ys;
  vector<Fenwick> ft;

  explicit Fenwick2D(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 (auto& v : ys) {
      sort(WHOLE(v));
      ft.emplace_back(sz(v));
    }
  }

  int ind(int x, int y) {
    return (int)(lower_bound(WHOLE(ys[x]), y) - ys[x].begin());
  }

  void update(int x, int y, i64 delta) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), delta);
  }

  i64 query(int x, int y) {
    i64 sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

SparseTable.h

27 lines

```
template<class T, class Better = std::less<T>>
struct SparseTable {
  explicit SparseTable(vector<T> vals) {
    log2.push_back(0);
    for (int i = 1; i <= sz(vals); ++i) {
      log2.push_back(log2.back() + (2 << log2.back() < i));
    }

    table.push_back(std::move(vals));
    for (int p = 1; log2.back() >= sz(table); ++p) {
      auto& row = table.emplace_back();
      for (int i = 0; i + (1<<p) <= sz(table[0]); ++i) {
        row.push_back(get(i, i + (1<<p)));
      }
    }
  }

  T get(int begin, int end) const {
    int p = log2[end - begin];
    return min(table[p][begin], table[p][end - (1<<p)], better);
  }

private:
  vector<vector<T>> table;
  vector<int> log2;
  Better better;
};
```

UnionFind.h

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

24 lines

```
struct UnionFind {
  vector<int> e;
  explicit UF(int n) : e(n, -1) {}

  int find(int x) {
    return e[x] < 0 ? x : e[x] = find(e[x]);
  }

  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);
    e[a] += e[b];
    e[b] = a;
    return true;
  }

  int size(int x) {
    return -e[find(x)];
  }
};
```

LineEnvelope.h

36 lines

```
const i64 is_query = -(1LL<<62);
struct Line {
    i64 m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        i64 x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};

struct HullDynamic : public multiset<Line> { // will maintain
     upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m
             - x->m);
    }
    void insert_line(i64 m, i64 b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y);
             };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    i64 eval(i64 x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

GnuExtensions.h

<bits/extc++.h> 45 lines

```
using namespace __gnu_pbds;

template<typename Key>
using ordered_set = tree<
  Key, null_type, std::less<Key>,
  rb_tree_tag,
  tree_order_statistics_node_update
>;
// gp_hash_table implements unordered_map
using __gnu_cxx::rope;

int main() {
  ordered_set<int> X;
  for (auto i : {1, 2, 4, 8, 16})
      X.insert(i);

  for (auto i : {1, 2, 4})
    std::cout << *X.find_by_order(i) << '\n'; // 2 4 16
  std::cout << (X.end()==X.find_by_order(10)) << '\n'; // 1

  for (auto key : {-5, 1, 3, 4, 400})
    std::cout << X.order_of_key(key) << '\n'; // 0 0 2 2 5

  rope<int> rp;
  rp.push_back(23);
  rp += rope<int>(5, 42);
  for (auto x : rp)
    std::cout << x << ' ';
  std::cout << '\n';  // 23 42 42 42 42 42

  rp.erase(3, 2);
  rp.mutable_reference_at(1) = 24;  // 2 substrs + 2 concats
  for (auto x : rp)
    std::cout << x << ' ';
  std::cout << '\n';  // 23 24 42 42

  rope<int> rp2 = rp;  // said to be fast
  std::iota(rp.mutable_begin(), rp.mutable_end(), 0); // slow
  rp.replace(2, 1, rp2);  // said to be fast
  for (auto x : rp)
    std::cout << x << ' ';
  std::cout << '\n';  // 0 1 23 24 42 42 3
  std::cout << rp.substr(2).size() << '\n';  // 1!
  std::cout << rope<char>(5, '!') + '\n';  // !!!!!
}
```

Strings (8)

Hashes.h

29 lines

```
using Hash = array<ui64, 3>;
#define HOP(op) \
  inline Hash operator op (Hash a, Hash b) { \
    return {a[0] op b[0], a[1] op b[1], a[2] op b[2]}; \
  }
HOP(+)HOP(-)HOP(*)HOP(%)
inline Hash makeHash(ui64 val) { return {val, val, val}; }

const Hash Multiplier{{228227, 227223, 22823}};
const Hash Modulus{{424242429, 2922827, 22322347}};

vector<Hash> pows(1);
struct Hashes {
  explicit Hashes(const string& s) {
    pows.front().fill(1);
    while (pows.size() <= s.size())
      pows.push_back(pows.back() * Multiplier % Modulus);
    prefs.push_back(makeHash(0));
    for (auto c : s)
      prefs.push_back((prefs.back() * Multiplier + makeHash(c))
          % Modulus);
  }
  Hash get(size_t begin, size_t end) const {
    return (prefs[end] - prefs[begin] * pows[end - begin]
        % Modulus + Modulus) % Modulus;
  }
private:
  vector<Hash> prefs;
};
```

AhoCorasick.h

Description: on-line tracking of the set of suffixes of a text that are prefixes of some words from a dictionary.

44 lines

```
struct AhoCorasick {
  AhoCorasick(): n(1) {
    n.reserve(TrieSize);
  }

  void addWord(const string& word, int id) {
    int v = 0;
    for (int ch : word) {
      ch -= 'a';
      auto& u = n[v].trans[ch];
      if (!u) {
        u = int(n.size());
        n.emplace_back();
      }
      v = u;
    }
    n[v].termId = id;
  }

  void build() {
    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      auto v = q.front();
      for (Char ch = 0; ch < Alph; ++ch) {
        auto& u = n[v].trans[ch];
        if (!u) {
          u = n[n[v].link].trans[ch];
          continue;
        }
        q.push(u);
        auto i = n[u].link = (v ? n[n[v].link].trans[ch] : 0);
        n[u].nextTerm = (n[i].termId >= 0 ? i : n[i].nextTerm);
      }
    }
  }

private:
  struct Node {
    int trans[Alph]{};
    int nextTerm = -1, termId = -1, link = 0;
  };

  vector<Node> n;
};
```

PrefixFunction.h

Description: pi[x] is the length of the longest prefix of s that ends at x, other than s[0..x] itself

10 lines

```
vector<size_t> pi(const string& s) {
  vector<size_t> p(s.size(), 0);
  for (size_t i = 1; i < s.size(); ++i) {
    auto px = p[i - 1];
    while (px && s[i] != s[px])
      px = p[px - 1];
    p[i] = px + (s[i] == s[g]);
  }
  return p;
}
```

ZFunction.h

Description: z[x] is max L: s[x:x+L] == s[:L]

11 lines

```
vector<size_t> zFun(const string& s) {
  vector<size_t> z(s.size(), 0);
  for (size_t left = 0, right = 0, i = 1; i < s.size(); ++i) {
    z[i] = (i < right ? min(right - i, z[i - left]) : 0);
    while (i + z[i] < s.size() && s[i + z[i]] == s[z[i]])
      ++z[i];
    if (i + z[i] > right)
      tie(left, right) = {i, i + z[i]};
  }
  return z;
}
```

Manacher.h

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

17 lines

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

SuffixArray.h

57 lines

```
struct SuffixArray {
  string s;
  vector<int> order, rank, lcp;

  SuffixArray(const string& _s): s(_s + '$') {
    int n = sz(s);
    std::vector<int> count(n + 130), nextPos(count.size() + 1);
    std::vector<int> nextOrder(n), nextColor(n);
    std::vector<int> color(WHOLE(s));

    auto norm = [n](int i) {
      return i < 0 ? i + n : i >= n ? i - n : i;
    };

    order.resize(n);
    std::iota(WHOLE(order), 0);
    std::sort(WHOLE(order),
        [&](int aa, int bb) { return s[aa] < s[bb]; });

    for (int half = 1; half < n; half *= 2) {
      count.assign(count.size(), 0);
      for (auto col : color)
        ++count[col];

      nextPos[0] = 0;
      partial_sum(WHOLE(count), nextPos.begin() + 1);

      for (auto pos : order) {
        auto shifted = norm(pos - half);
        nextOrder[nextPos[color[shifted]]++] = shifted;
      }
      order.swap(nextOrder);

      nextColor[order[0]] = 0;
      for (int i = 1; i < n; ++i) {
        auto pos = order[i], prev = order[i - 1];
        nextColor[pos] = nextColor[prev] + (
            tie(color[pos], color[norm(pos + half)]) !=
            tie(color[prev], color[norm(prev + half)])
        );
      }
      color.swap(nextColor);
    }

    rank.resize(n);
    for (int i = 0; i < n; ++i)
      rank[order[i]] = i;

    lcp.resize(n);
    for (int i = 0; i < n; ++i) if (rank[i]) {
      for (int p0 = order[rank[i] - 1]; s[i + h] == s[p0 + h];)
        h++;
      lcp[rank[i]] = h;
      h -= h > 0;
    }
  }
};
```

SuffixAutomaton.h

Description: Each node is a set of substrings with same set of end positions. The lengths of substrings in each node lie on segment $[maxlen(link(v)); maxlen(v)]$. Each path covers disjoint sets of substrings. Each path from 0 is a unique substring.

47 lines

```
struct Node {
  int maxLen;
  int sufLink;
  int trans[26];
} nodes[200005];

class Automaton {
public:
  void append(const int ch) {
    auto p = last;
    last = newNode(nodes[last].maxLen + 1, 0);

    for (; p > -1; p = nodes[p].sufLink) {
      const auto q = nodes[p].trans[ch];
      if (q == -1) {
        nodes[p].trans[ch] = last;
      } else {
        auto clone = q;
        if (nodes[p].maxLen + 1 != nodes[q].maxLen) {
          clone = copyNode(q);
          nodes[clone].maxLen = nodes[p].maxLen + 1;
          nodes[q].sufLink = clone;
          for (; p > -1 && nodes[p].trans[ch] == q; p = nodes[p].
              sufLink)
            nodes[p].trans[ch] = clone;
        }
        nodes[last].sufLink = clone;
        break;
      }
    }
  }

private:
  int nsz = 0;
  int last = newNode(0);

  int newNode(int maxLen, int sufLink = -1) {
    memset(nodes[nsz], -1, sizeof nodes[nsz]);
    nodes[nsz].maxLen = maxLen;
    nodes[nsz].sufLink = sufLink;
    return nsz++;
  };

  int copyNode(int orig) {
    memcpy(nodes + nsz, nodes + orig, sizeof(Node));
    return nsz++;
  }
};
```

SuffixAutomatonDiffSubstrs.h

Description: An excerpt from a problem with map-based implementation of automaton. Still probably gives a taste.

"SuffixAutomaton.h" 22 lines

```
std::int64_t diffSubstrs() {
  std::int64_t total = -1;

  std::vector<int> perm(nodes.size());
  std::iota(perm.begin(), perm.end(), 0);
  std::sort(
    perm.begin(), perm.end(),
    [&](int fi, int se) {
        return nodes[fi].maxLen < nodes[se].maxLen;
    }
  );

  nodes[0].dp = 1;
  for (auto i : perm) {
    total += nodes[i].dp;
    for (auto pa : nodes[i].trans) {
        pa.second->dp += nodes[i].dp;
    }
  }

  return total;
}
```

Graph (9)

Kuhn.h

28 lines

```
vector<int> vis, match;

int qq = 0;
bool try_kuhn(int v) {
  if (vis[v] == qq)
    return false;
  vis[v] = qq;
  for (auto u : e[v]) {
      if (match[u] == -1) {
          match[u] = v;
          return true;
      }
  }
  for (auto u : e[v]) {
      if (dfs(match[u])) {
          match[u] = v;
          return true;
      }
  }
  return false;
}

void kuhn() {
  fill(WHOLE(vis), -1);
  for (int qq = 0; qq < n; ++qq) {
    try_kuhn(qq);
  }
}
```

2sat.h

67 lines

```
struct TwoSAT {
  int n;
  vector<vector<int>> g;
  vector<vector<int>> gr;

  explicit TwoSAT(int n = 0): n(n), g(n * 2), gr(n * 2) {}

  void addImplication(int a, int b) {
    debug("%d -> %d", a, b);
    g[a].push_back(b);
    gr[b].push_back(a);

    debug("\t%d -> %d\n", b ^ 1, a ^ 1);
    g[b ^ 1].push_back(a ^ 1);
    gr[a ^ 1].push_back(b ^ 1);
  }

  void addOr(int a, int b) {
    debug("%d v %d\t", a, b);
    addImplication(a ^ 1, b);
  }

  vector<bool> used;
  vector<int> topOrder;
  void topTraverse(int v) {
    used[v] = true;
    for (int u : g[v])
      if (!used[u])
        topTraverse(u);
    topOrder.push_back(v);
  }

  int comps;
  vector<int> comp;
  void colorize(int v, int color) {
    comp[v] = color;
    for (int u : gr[v])
      if (comp[u] == -1)
        colorize(u, color);
  }

  vector<bool> solution;
  bool run() {
    used.assign(n * 2, false);
    for (int i = 0; i < 2 * n; ++i) {
      if (!used[i])
        topTraverse(i);
    reverse(WHOLE(topOrder));

    comps = 0;
    comp.assign(2 * n, -1);
    for (auto v : topOrder)
      if (comp[v] == -1)
        colorize(v, comps++);

    solution.resize(n * 2);
    for (int v = 0; v < n * 2; v += 2) {
      if (comp[v] == comp[v + 1]) {
        debug("No solution, as %d <-> %d\n", v, v + 1);
        return false;
      }
      solution[v] = comp[v] > comp[v + 1];
      solution[v + 1] = !solution[v];
    }
    return true;
  }
};
```

Dinic.h

75 lines

```
namespace Dinic {
const int maxn = 100100;
struct Edge {
  int to;
  i64 cap;
  i64 flow = 0;
};

vector<Edge> es;
vector<int> g[maxn];
int layer[maxn], pos[maxn];
int S, T;

void addEdge(int v, int u, ll c) {
  g[v].push_back(sz(es));
  es.push_back({u, c});
  g[u].push_back(sz(es));
  es.push_back({v, 0});
}

i64 dfs(int v, i64 curf) {
  if (v == T)
    return curf;
  i64 ret = 0;
  for (auto& i = pos[v]; curf && i < sz(g[v]); ++i) {
    auto& e = es[g[v][i]];
    if (layer[e.to] != layer[v])
      continue;
    if (i64 delta = dfs(e.to, min(curf, e.cap - e.flow))) {
      curf -= delta;
      ret += delta;
      e.flow += delta;
      es[g[v][i] ^ 1].flow -= delta;
    }
  }
  return ret;
}

bool bfs() {
  memset(layer, -1, sizeof layer);
  layer[S] = 0, q[0] = S;
  static queue<int> q;
  for (q.push(S); !q.empty(); q.pop) {
    int v = q.front();
    for (int id: g[v]) {
      const auto& e = es[id];
      if (e.cap > e.flow && layer[e.to] == -1) {
        layer[e.to] = layer[v] + 1;
        q.push(e.to);
      }
    }
  }
  return layer[T] != -1;
}

i64 dinic(int s, int t) {
  S = s; T = t;
  i64 res = 0;
  while (bfs()) {
    memset(pos, 0, sizeof pos);
    while (i64 cur = dfs(S, 1LL << 60))
      res += cur;
  }
  return res;
}
} // namespace Dinic

void test() {
  Dinic::addEdge(0, 1, 1);
  Dinic::addEdge(0, 2, 2);
  Dinic::addEdge(2, 1, 1);
  Dinic::addEdge(1, 3, 2);
  Dinic::addEdge(2, 3, 1);
  cout << Dinic::dinic(0, 3) << endl; // 3
}
```

EulerCycle.h

26 lines

```
struct Edge {
  int to, id;
};

bool usedEdge[maxm];
vector<Edge> g[maxn];
int ptr[maxn];

vector<int> cycle;
void eulerCycle(int u) {
  while (ptr[u] < sz(g[u]) && usedEdge[g[u][ptr[u]].id])
    ++ptr[u];
  if (ptr[u] == sz(g[u]))
    return;
  const Edge &e = g[u][ptr[u]];
  usedEdge[e.id] = true;
  eulerCycle(e.to);
```

```
  cycle.push_back(e.id);
  eulerCycle(u);
}

int edges = 0;
void addEdge(int u, int v) {
  g[u].push_back(Edge{v, edges});
  g[v].push_back(Edge{u, edges++});
}
```

MinCostMaxFlow.h

180 lines

```
namespace MinCost {
const ll infc = 1e12;

struct Edge {
    int to;
    ll c, f, cost;

    Edge(int to, ll c, ll cost): to(to), c(c), f(0), cost(cost)
    {  }
};

int N, S, T;
int totalFlow;
ll totalCost;
const int maxn = 505;
vector<Edge> edge;
vector<int> g[maxn];

void addEdge(int u, int v, ll c, ll cost) {
    g[u].push_back(edge.size());
    edge.emplace_back(v, c, cost);
    g[v].push_back(edge.size());
    edge.emplace_back(u, 0, -cost);
}

ll dist[maxn];
int fromEdge[maxn];

bool inQueue[maxn];
bool fordBellman() {
    forn (i, N)
        dist[i] = infc;
    dist[S] = 0;
    inQueue[S] = true;
    vector<int> q;
    q.push_back(S);
    for (int ii = 0; ii < int(q.size()); ++ii) {
        int u = q[ii];
        inQueue[u] = false;
        for (int e: g[u]) {
            if (edge[e].f == edge[e].c)
                continue;
            int v = edge[e].to;
            ll nw = edge[e].cost + dist[u];
            if (nw >= dist[v])
                continue;
            dist[v] = nw;
            fromEdge[v] = e;
            if (!inQueue[v]) {
                inQueue[v] = true;
                q.push_back(v);
            }
        }
    }
    return dist[T] != infc;
}

ll pot[maxn];
bool dikstra() {
    typedef pair<ll, int> Pair;
    priority_queue<Pair, vector<Pair>, greater<Pair>> q;
    forn (i, N)
        dist[i] = infc;
    dist[S] = 0;
    q.emplace(dist[S], S);
    while (!q.empty()) {
        int u = q.top().second;
        ll cdist = q.top().first;
        q.pop();
        if (cdist != dist[u])
            continue;
        for (int e: g[u]) {
            int v = edge[e].to;
            if (edge[e].c == edge[e].f)
                continue;
            ll w = edge[e].cost + pot[u] - pot[v];
            assert(w >= 0);
            ll ndist = w + dist[u];
            if (ndist >= dist[v])
                continue;
            dist[v] = ndist;
            fromEdge[v] = e;
            q.emplace(dist[v], v);
        }
    }
    if (dist[T] == infc)
        return false;
    forn (i, N) {
        if (dist[i] == infc)
            continue;
        pot[i] += dist[i];
    }
    return true;
}

bool push() {
    //2 variants
    //if (!fordBellman())
    if (!dikstra())
        return false;
    ++totalFlow;
    int u = T;
    while (u != S) {
        int e = fromEdge[u];
        totalCost += edge[e].cost;
        edge[e].f++;
        edge[e ^ 1].f--;
        u = edge[e ^ 1].to;
    }
    return true;
}

//min-cost-circulation
ll d[maxn][maxn];
int dfrom[maxn][maxn];
int level[maxn];
void circulation() {
    while (true) {
        int q = 0;
        fill(d[0], d[0] + N, 0);
        forn (iter, N) {
            fill(d[iter + 1], d[iter + 1] + N, infc);
            forn (u, N)
                for (int e: g[u]) {
                    if (edge[e].c == edge[e].f)
                        continue;
                    int v = edge[e].to;
                    ll ndist = d[iter][u] + edge[e].cost;
                    if (ndist >= d[iter + 1][v])
                        continue;
                    d[iter + 1][v] = ndist;
                    dfrom[iter + 1][v] = e;
                }
            q ^= 1;
        }
        int w = -1;
        ld mindmax = 1e18;
        forn (u, N) {
            ld dmax = -1e18;
            forn (iter, N)
                dmax = max(dmax,
                    (d[N][u] - d[iter][u]) / ld(N - iter));
            if (mindmax > dmax)
                mindmax = dmax, w = u;
        }
        if (mindmax >= 0)
            break;
        fill(level, level + N, -1);
        int k = N;
        while (level[w] == -1) {
            level[w] = k;
            w = edge[dfrom[k--][w] ^ 1].to;
        }
        int k2 = level[w];
        ll delta = infc;
        while (k2 > k) {
            int e = dfrom[k2--][w];
            delta = min(delta, edge[e].c - edge[e].f);
            w = edge[e ^ 1].to;
        }
        k2 = level[w];
        while (k2 > k) {
            int e = dfrom[k2--][w];
            totalCost += edge[e].cost * delta;
            edge[e].f += delta;
            edge[e ^ 1].f -= delta;
            w = edge[e ^ 1].to;
        }
    }
}
} // namespace MinCost

int main() {
    MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
    MinCost::addEdge(1, 0, 3, 5);
    MinCost::addEdge(0, 2, 4, 6);
    while (MinCost::push());
    cout << MinCost::totalFlow << ' '
        << MinCost::totalCost << '\n'; //3 33
}
```

Various (10)

10.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

23 lines

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

void removeInterval(set<pii>& is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->second;
  if (it->first == L) is.erase(it);
  else (int&)it->second = L;
  if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

19 lines

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

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

19 lines

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

10.2 Misc. algorithms

LIS.h

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

17 lines

```
template<class I> vi lis(vector<I> S) {
  vi prev(sz(S));
  typedef pair<I, int> p;
  vector<p> res;
  rep(i,0,sz(S)) {
    p el { S[i], i };
    //S[i]+1 for non-decreasing
    auto it = lower_bound(all(res), p { S[i], 0 });
    if (it == res.end()) res.push_back(el), it = --res.end();
    *it = el;
    prev[i] = it==res.begin() ?0:(it-1)->second;
  }
  int L = sz(res), cur = res.back().second;
  vi ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  return ans;
}
```

LCS.h

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

14 lines

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

10.3 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`).

10.4 Optimization tricks

10.4.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `rep(b,0,K) rep(i,0,(1 << K)) if (i & 1 << b) D[i] += D[i^(1 << b)];` computes all sums of subsets.

10.4.2 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).

BumpAllocator.h

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

8 lines

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

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory.

"BumpAllocator.h" 10 lines

```
template<class T> struct ptr {
  unsigned ind;
  ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
    assert(ind < sizeof buf);
  }
  T& operator*() const { return *(T*)(buf + ind); }
  T* operator->() const { return &**this; }
  T& operator[](int a) const { return (&**this)[a]; }
  explicit operator bool() const { return ind; }
};
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.

Usage: `vector<vector<int, small<int>>> ed(N);`

14 lines

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

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