VNUHCM University of Science

HCMUS-HLD

Nguyen Anh Dung
Phan Binh Nguyen Lam
Le Huu Hoa

2025-10-11

Contest (1)

.bashrc

2 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
```

hash.sh

3 lines

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

note.txt

1 lines

```
#define rep(i, a, b) for(int i = a; i < (b); ++i)
```

Mathematics (2)

2.1 Equations

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

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

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

In general, given an equation $Ax=b$, the solution to a variable x_i is given by

$$x_i=\frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

2.2 Recurrences

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

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

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

2.3 Trigonometry

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

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

$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

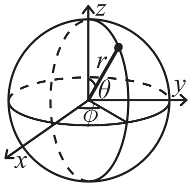
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

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

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

2.7 Series

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

2.8 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

$\operatorname{Bin}(n, p)$ is approximately $\operatorname{Po}(np)$ 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 $\operatorname{Fs}(p)$, $0 \leq p \leq 1$.

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

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

Poisson distribution

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

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

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

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

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

A Markov chain is an 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 $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$.

2.10 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

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

2.11 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:

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

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

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

InverseModulo.h

d41d8c, 11 lines

```
template <typename T>
T inverse_modulo(T a, T m) {
  T u = 0, v = 1;
  while (a > 0) {
    T t = m / a;
    m -= t * a; std::swap(a, m);
    u -= t * v; std::swap(u, v);
  }
  assert(m == 1);
  return u;
}
```

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

d41d8c, 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;
}
```

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

d41d8c, 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: $\mathcal{O}\left(n^2 \log k\right)$

d41d8c, 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;
}
```

GaussianElimination.h

Description: Gaussian Elimination for solving systems of linear equations.
Usage: gauss({{1, 2, 3}, {4, 5, 6}}); // returns {1, -2}
can be use for modulo arithmetic, but be careful with division
replace Z with the desired type
Time: $\mathcal{O}\left(N^3\right)$

d41d8c, 44 lines

```
using Z = double;

std::vector <Z>* gauss(std::vector <std::vector <Z>> a) {
  #define ABS(x) ((x) < 0 ? -(x) : (x))
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;

  std::vector <int> pivot(m, -1);

  for (int col = 0, row = 0; col < m and row < n; col++) {
    int cur = row;
    for (int i = row; i < n; i++)
      if (ABS(a[i][col]) > ABS(a[cur][col]))
        cur = i;
    if (a[cur][col] == 0)
      continue;
    for (int i = col; i <= m; i++)
      swap(a[cur][i], a[row][i]);
    pivot[col] = row;

    for (int i = 0; i < n; i++) if (i != row) {
      if (a[i][col] == 0)
        continue;
      Z c = a[i][col] / a[row][col];
      for (int j = col; j <= m; j++)
        a[i][j] -= a[row][j] * c;
    }
    row++;
  }

  std::vector <Z> *ans = new std::vector <Z> (m, 0);
  for (int i = 0; i < m; i++) if (pivot[i] != -1)
    (*ans)[i] = a[pivot[i]][m] / a[pivot[i]][i];
  for (int i = 0; i < n; i++) {
    Z s = a[i][m];
    for (int j = 0; j < m; j++)
      s -= (*ans)[j] * a[i][j];
    if (s)
      return nullptr;
  }

  return ans;
  #undef ABS
}
```

ModInt.h

Description: Operators for modular arithmetic.
Usage: using Z = Mint<MOD>;
Z inverse = CInv<42, MOD>;

d41d8c, 97 lines

```
using i64 = long long;

template<class T>
constexpr T power(T a, i64 b) {
  T res = 1;
  for (; b; b /= 2, a *= a) {
    if (b % 2) {
      res *= a;
    }
  }
  return res;
}

template<int P>
struct MInt {
  int x;
  constexpr MInt() : x{} {}
  constexpr MInt(i64 x) : x{norm(x % P)} {}

  constexpr int norm(int x) const {
    if (x < 0) {
      x += P;
    }
    if (x >= P) {
      x -= P;
    }
    return x;
  }
  constexpr int val() const {
    return x;
  }
  explicit constexpr operator int() const {
    return x;
  }
  constexpr MInt operator-() const {
    MInt res;
    res.x = norm(P - x);
    return res;
  }
  constexpr MInt inv() const {
    assert(x != 0);
    return power(*this, P - 2);
  }
  constexpr MInt &operator*=(MInt rhs) {
    x = 1LL * x * rhs.x % P;
    return *this;
  }
  constexpr MInt &operator+=(MInt rhs) {
    x = norm(x + rhs.x);
    return *this;
  }
  constexpr MInt &operator-=(MInt rhs) {
    x = norm(x - rhs.x);
    return *this;
  }
  constexpr MInt &operator/=(MInt rhs) {
    return *this *= rhs.inv();
  }
  friend constexpr MInt operator*(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res *= rhs;
    return res;
  }
  friend constexpr MInt operator+(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res += rhs;
    return res;
  }
  friend constexpr MInt operator-(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res -= rhs;
    return res;
  }
  friend constexpr MInt operator/(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res /= rhs;
    return res;
  }
  friend constexpr std::istream &operator>>(std::istream &is,
      MInt &a) {
    i64 v;
    is >> v;
    a = MInt(v);
    return is;
  }
  friend constexpr std::ostream &operator<<(std::ostream &os,
      const MInt &a) {
    return os << a.val();
  }
  friend constexpr bool operator==(MInt lhs, MInt rhs) {
    return lhs.val() == rhs.val();
  }
  friend constexpr bool operator!=(MInt lhs, MInt rhs) {
    return lhs.val() != rhs.val();
  }
};

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();
```

Lagrange.h

d41d8c, 22 lines

```
Z lagrange(const std::vector <Z> &p, int x) {
  if (x < (int) p.size())
    return p[x];
  Z ans = 0, prod = 1;

  for (int i = 1; i < (int) p.size(); i++) {
    prod *= x - i;
    prod /= -i;
  }

  for (int i = 0; i < (int) p.size(); i++) {
    ans += prod * p[i];
    if (i + 1 == (int) p.size())
      break;
    prod *= x - i;
    prod /= x - (i + 1);
    prod *= i - (int) p.size() + 1;
    prod /= i + 1;
  }

  return ans;
}
```

FFTMOD.h

d41d8c, 54 lines

```
typedef complex<double> C;
typedef long double ld;
void fft(vector<C> &a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<ld>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (int k = 2; k < n; k *= 2) {
    R.resize(n), rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    for (int i = k; i < 2 * k; ++i) rt[i] = R[i] = (i & 1) ? R[
         i / 2] * x : R[i / 2];
  }
  vector<int> rev(n);
  for (int i = 0; i < n; ++i) rev[i] = (rev[i / 2] | (i & 1) <<
       L) / 2;
  for (int i = 0; i < n; ++i) if(i < rev[i]) swap(a[i], a[rev[i
      ]]);
  for (int k = 1; k < n; k <<= 1) {
    for (int i = 0; i < n; i += k << 1) {
          for (int j = 0; j < k; ++j) {
        auto x = (double*) &rt[j + k], y = (double*) &a[i + j +
             k];
        C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y
             [0]);
        a[i + j + k] = a[i + j] - z; a[i + j] += z;
      }
    }
  }
}

template<ll MOD> vector<ll> convMod(const vector<ll> &a, const
     vector<ll> &b) {
  if (a.empty() || b.empty()) return {};
  vector<ll> res(sz(a) + sz(b) - 1, 0);
  int B = 32 - __builtin_clz(sz(res));
  int n = 1 << B, cut = ll(sqrt(MOD));
  vector<C> L(n), R(n), outs(n), outl(n);
  for (int i = 0; i < sz(a); ++i) L[i] = C(ll(a[i] / cut), ll(a
       [i] % cut));
  for (int i = 0; i < sz(b); ++i) R[i] = C(ll(b[i] / cut), ll(b
       [i] % cut));
  fft(L), fft(R);
  for (int i = 0; i < n; ++i) {
    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) / C(1.0i);
  }
  fft(outl), fft(outs);
  for (int i = 0; i < sz(res); ++i) {
    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 % MOD * cut % MOD + bv) % MOD * cut % MOD +
         cv) % MOD;
  }
  return res;
}

void mul(int a[], int b[], ll c[]) {
    vector<ll> pa, pb;
    for (int i = 0; i < k; ++i) pa.push_back(a[i]), pb.
         push_back(b[i]);
    vector<ll> res = convMod<MOD>(pa, pb);
    for (int i = 0; i < sz(res); ++i) c[i] = res[i];
}
```

XorBasis.h

d41d8c, 75 lines

```
struct Node {
    int basis[MAXLOG], szBasis, basisMask;

    Node() {
        szBasis = basisMask = 0;
        memset(basis, 0, sizeof(basis));
    }

    bool insertVector(int mask, int idx) {
        while(mask > 0) {
            int i = 31 - __builtin_clz(mask);
            if(!basis[i]) {
                ++szBasis, basis[i] = mask;
                basisMask |= (1 << i);
                return true;
            }
            mask ^= basis[i];
        }
        return false;
    }

    bool checkVector(int mask) const {
        while(mask > 0) {
            int i = 31 - __builtin_clz(mask);
            if(!basis[i]) return false;
            mask ^= basis[i];
        }
        return true;
    }

    Node mergeNode(const Node &A) {
        Node res(*this);
        int mask = A.basisMask;
        while(mask > 0) {
            int i = 31 - __builtin_clz(mask);
            res.insertVector(A.basis[i]);
            mask ^= (1 << i);
        }

        return res;
    }

    // find k-th element from small to big in Basis
    int query(int k) const { // 1-indexed
        int mask = 0, tot = 1 << szBasis, fMask = basisMask;
        while(fMask > 0) {
            int i = 31 - __builtin_clz(fMask), low(tot >> 1);
            if(low < k && !(mask >> i & 1) || low >= k && (mask
                 >> i & 1)) mask ^= basis[i];
            if(low < k) k -= low;
            fMask ^= (1 << i), tot >>= 1;
        }
        return mask;
    }

    int getValPos(int mask) const { // 1-indexed, if mask < 0,
         result = 0
        if(mask < 0) return 0;
        int firstMask = mask, fMask = basisMask, tot(1 <<
             szBasis), cnt(1);
        while(fMask > 0) {
            int i = 31 - __builtin_clz(fMask), low(tot >> 1);
            if(firstMask >> i & 1) cnt += low, mask ^= basis[i
                ];
            fMask ^= (1 << i), tot >>= 1;
        }
        return cnt;
    }

    int getMax(void) const {
        int res = 0, mask = basisMask;
        while(mask > 0) {
            int i = 31 - __builtin_clz(mask);
            if(!(res >> i & 1)) res ^= basis[i];
            mask ^= (1 << i);
        }
        return res;
    }
}
```

BernoulliNumber.h

d41d8c, 44 lines

```
inline ll C2(ll n) { return (n & 1) ? (n + 1) / 2 % MOD * (n %
     MOD) % MOD : n / 2 % MOD * ((n + 1) % MOD) % MOD; }

inline ll nCk(int n, int k) {
    return (k > n) ? 0 : frac[n] * finv[k] % MOD * finv[n - k]
         % MOD;
}

ll powermod(ll a, int exponent) {
    ll res(1);
    while(exponent > 0) {
        if(exponent & 1) res = res * a % MOD;
        exponent >>= 1;
        a = a * a % MOD;
    }
    return res;
}

ll tmp[MAXN];
ll bernoulli(int n) {
    for (int i = 0; i <= n; ++i) {
        tmp[i] = inv[i + 1];
        for (int j = i; j > 0; --j) tmp[j - 1] = 1LL * j * (tmp
             [j - 1] - tmp[j] + MOD) % MOD;
    }
    return tmp[0];
}

ll calc(int n) {
    ll res(0), invn = powermod(n, MOD - 2);
    n = powermod(n, expo + 1);
    for (int k = 0; k <= expo; ++k) {
        res = (res + n * nCk(expo + 1, k) % MOD * B[k] % MOD) %
             MOD;
        n = n * invn % MOD;
    }
    return res * powermod(expo + 1, MOD - 2) % MOD;
}

void init(void) {
    frac[0] = finv[0] = 1;
    for (int i = 1; i <= 21; ++i) {
        frac[i] = frac[i - 1] * i % MOD;
        finv[i] = powermod(frac[i], MOD - 2);
        inv[i] = powermod(i, MOD - 2);
    }
    for (int i = 0; i <= 20; ++i) B[i] = bernoulli(i);
}
```

LinearSieve.h

d41d8c, 26 lines

```
// calc mobius inversion and number divisor of x with all (x <=
     n) O(n)
void linearSieve(void) {
    u[1] = numd[1] = 1, phi[1] = 0;
    vector<int> primes;
    for (int i = 2; i < MAXN; ++i) {
        if(!isComp[i]) {
            primes.push_back(i);
            u[i] = -1, numd[i] = 2, phi[i] = i - 1, cnt[i] = 1;
        }
        for (int j = 0; j < sz(primes) && i * primes[j] < MAXN;
             ++j) {
```

```
            isComp[i * primes[j]] = 1;
            if(i % primes[j] == 0) {
                u[i * primes[j]] = 0;
                numd[i * primes[j]] = numd[i] / (cnt[i] + 1) *
                    (cnt[i] + 2);
                phi[i * primes[j]] = phi[i] * primes[j];
                cnt[i * primes[j]] = cnt[i] + 1;
                break;
            } else {
                u[i * primes[j]] = u[i] * u[primes[j]];
                numd[i * primes[j]] = numd[i] * numd[primes[j
                    ]];
                phi[i * primes[j]] = phi[i] * phi[primes[j]];
                cnt[i * primes[j]] = 1;
            }
        }
    }
}
```

Combinatorial (3)

3.1 Permutations

3.1.1 Cycles

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

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

3.1.2 Derangements

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

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

3.1.3 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

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

3.2 Partitions and subsets

3.2.1 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

3.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_kp^k + ... + n_1p + n_0$ and $m = m_kp^k + ... + m_1p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$.

3.3 General purpose numbers

3.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

3.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

3.3.3 Eulerian numbers

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

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

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

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

3.3.4 Motzkin numbers

- different ways of drawing non-intersecting chords between n points on a circle.
- **positive** integer sequences of length $n+1$ in which the opening and ending elements are 1, and the difference between any two consecutive elements is -1, 0 or 1.
- $1, 1, 2, 4, 9, 21, 51, 127, 323, 835, \ldots$

$$M_n = M_{n-1} + \sum_{i=0}^{n-2} M_iM_{n-2-i} = \frac{2n+1}{n+2}M_{n-1} + \frac{3n-3}{n+2}M_{n-2}$$

3.3.5 Delannoy numbers

- Number of ways to move from (0,0) to (m,n) with 3 kinds of steps: (1,0), (1,1), (0,1)
- $D(0,k) = 1, 1, 1, 1, \ldots$
- $D(3,k) = 1, 7, 25, 63, \ldots$

$$D(m,n) = \sum_{k=0}^{\min(m,n)}\binom{m+n-k}{m}\binom{m}{k} = \sum_{k=0}^{\min(m,n)}\binom{m}{k}\binom{n}{k}2^k$$

3.3.6 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

3.3.7 Bell numbers

Number of partitions of a set of n items into non-empty disjoint subsets. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

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

$$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$$
$$B_{p+n} \equiv B_n + B_{n+1} \pmod p$$
$$B_{p^m+n} \equiv mB_n + B_{n+1} \pmod p$$

3.3.8 Labeled unrooted trees

# on n vertices: n^{n-2}
# on k existing trees of size n_i: $n_1n_2\cdots n_kn^{k-2}$
# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

3.3.9 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.
- full 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.
- correct bracket sequence of length $2n$.

PartitionNumbers.h

d41d8c, 20 lines

```
// ways of writing n as a sum of positive integers O(N \sqrt(N)
    )
ll partitionNumber(int n) {
    if (n < 0) return 0;
    vector<ll> dp(n + 1, 0);
    dp[0] = 1; // base

    for (int i = 1; i <= n; ++i) {
        for (int k = 1; ; ++k) {
            int pent1 = k * (3 * k - 1) / 2; // g(k)
            if (pent1 > i) break;
            ll sign = (k % 2 == 1) ? 1 : -1;
            dp[i] += sign * dp[i - pent1];

            int pent2 = k * (3 * k + 1) / 2; // g(-k)
            if (pent2 > i) continue;
            dp[i] += sign * dp[i - pent2];
        }
    }
    return dp[n];
}
```

Flow - Graph Matching (4)

DinicFlow.h

Time: $\mathcal{O}\left(V^2 * E\right)$ for general case $\mathcal{O}\left(E * \sqrt(E)\right)$ for unit network

d41d8c, 58 lines

```
struct DinicFlow {
    vector<int> flow, capa;
    vector<int> point, next, head, work, dist;
    int numNode, numEdge;

    DinicFlow(int _n = 0) {
        numNode = _n, numEdge = 0;
        dist = work = vector<int>(_n + 7, 0);
        head = vector<int>(_n + 7, -1);
    }

    void addEdge(int u, int v, int c1, int c2 = 0) {
        point.push_back(v), capa.push_back(c1), flow.push_back
            (0);
        next.push_back(head[u]), head[u] = numEdge++;
        point.push_back(u), capa.push_back(c2), flow.push_back
            (0);
        next.push_back(head[v]), head[v] = numEdge++;
    }

    bool bfs(int s, int t) {
        queue<int> qu;
        for (int i = 1; i <= numNode; ++i) dist[i] = -1;
        dist[s] = 0; qu.push(s);
        while(!qu.empty()) {
            int u(qu.front()); qu.pop();
            for (int i = head[u]; i >= 0; i = next[i])
                if(flow[i] < capa[i] && dist[point[i]] < 0) {
                    dist[point[i]] = dist[u] + 1;
                    qu.push(point[i]);
                }
        }
        return (dist[t] >= 0);
    }

    int dfs(int s, int t, int fl) {
        if(s == t) return fl;
        for (int &i = work[s]; i >= 0; i = next[i])
            if(flow[i] < capa[i] && dist[point[i]] == dist[s] +
                1) {
                int d = dfs(point[i], t, min(fl, capa[i] - flow
                    [i]));
                if(!d) continue;
                flow[i] += d, flow[i ^ 1] -= d;
                return d;
            }
        return 0;
    }

    int maxFlow(int s, int t) {
        for (int i = 0; i < int(flow.size()); ++i) flow[i] = 0;
        int totFlow(0);
        while(bfs(s, t)) {
            for (int i = 1; i <= numNode; ++i) work[i] = head[i
                ];
            while(true) {
                int d = dfs(s, t, 1e9+7); totFlow += d;
                if(!d) break;
            }
        }
        return totFlow;
    }
};
```

MaxFlowMinCost.h

d41d8c, 62 lines

```
class MaxFlowMinCost {
    public:
        struct Edge {
            int from, to, capa, flow, cost;
            Edge(int _u = 0, int _v = 0, int _ca = 0, int _co =
                0) : from(_u), to(_v), capa(_ca), flow(0),
                cost(_co) {}
            inline int residual(void) const { return capa -
                flow; }
        };

        vector<vector<int>> adj;
        vector<Edge> E;
        vector<int> dist, tr;
        int n;

        MaxFlowMinCost(int _n = 0) {
            n = _n, E.clear();
            adj.assign(_n + 7, vector<int>());
            dist = vector<int>(_n + 7);
            tr = vector<int>(_n + 7);
        }

        void addEdge(int u, int v, int ca, int co) {
            adj[u].push_back(E.size());
            E.push_back(Edge(u, v, ca, co));
            adj[v].push_back(E.size());
            E.push_back(Edge(v, u, 0, -co));
        }

        bool FordBellman(int s, int t) {
            for (int i = 1; i <= n; ++i) dist[i] = 1e9+7, tr[i]
                = 1;
            queue<int> qu;
            vector<bool> inq = vector<bool>(n + 7, false);
            inq[s] = 1, dist[s] = 0; qu.push(s);
            while(!qu.empty()) {
                int u(qu.front()); qu.pop();
                inq[u] = 0;
                for (auto &it : adj[u]) {
                    if(E[it].residual() > 0) {
                        int v(E[it].to);
                        if(dist[v] > dist[u] + E[it].cost) {
                            dist[v] = dist[u] + E[it].cost; tr[
                                v] = it;
                            if(!inq[v]) { inq[v] = 1; qu.push(v
                                ); }
                        }
                    }
                }
            }
            return (dist[t] < 1e9+7);
        }

        ii getFlow(int s, int t) {
            for (int i = 0; i < int(E.size()); ++i) E[i].flow =
                0;
            int totFlow(0), totCost(0);
            while(FordBellman(s, t)) {
                int delta(1e9+7);
                for (int u = t; u != s; u = E[tr[u]].from)
                    delta = min(delta, E[tr[u]].residual());
                for (int u = t; u != s; u = E[tr[u]].from)
                    E[tr[u]].flow += delta, E[tr[u] ^ 1].flow
                        -= delta;
                totFlow += delta, totCost += delta * dist[t];
            }
            return ii(totFlow, totCost);
        }
};
```

GraphMatching.h

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

d41d8c, 39 lines

```
struct GraphMatching {
    vector<vector<int>> adj;
    vector<int> asi;
    vector<bool> dx;
    int mNode, nNode, cntMatching;

    GraphMatching(int _m, int _n) {
        mNode = _m, nNode = _n;
        adj = vector<vector<int>>(mNode + 1, vector<int>());
        asi = vector<int>(nNode + 1);
        dx = vector<bool>(mNode + 1);
    }
    inline void addEdge(int u, int v) { adj[u].push_back(v); }

    bool visit(int u) {
        if(dx[u]) return false;
        dx[u] = 1;
        for (int v : adj[u]) {
            if(!asi[v] || visit(asi[v])) {
                asi[v] = u; return true;
            }
        }
        return false;
    }

    void solve(void) {
        cntMatching = 0;
        for (int i = 1; i <= mNode; ++i) {
            for (int j = 1; j <= mNode; ++j) dx[j] = 0;
            cntMatching += visit(i);
        }
    }

    void show(void) {
        cout << cntMatching << '\n';
        for (int i = 1; i <= nNode; ++i) if(asi[i] > 0) cout <<
             asi[i] << ' ' << i << '\n';
    }
};
```

GraphFastMatching.h

Time: $\mathcal{O}\left(N\sqrt(N)\right)$ d41d8c, 54 lines

```
struct GraphMatching {
    vector<vector<int>> adj;
    vector<int> dist, matx, maty;
    int mNode, nNode, cntMatching;

    GraphMatching(int _m, int _n) {
        mNode = _m, nNode = _n;
        adj = vector<vector<int>>(mNode + 1, vector<int>());
        dist = matx = vector<int>(mNode + 1);
        maty = vector<int>(nNode + 1);
    }

    inline void addEdge(int u, int v) { adj[u].push_back(v); }

    bool bfs(void) {
        queue<int> qu;
        for (int i = 1; i <= mNode; ++i) {
            dist[i] = 1e9+7;
            if(!matx[i]) { dist[i] = 0; qu.push(i); }
        }
        dist[0] = 1e9+7;
        while(sz(qu)) {
            int u(qu.front()); qu.pop();
            if(dist[u] >= dist[0]) continue;
            for (int v : adj[u]) {
                if(dist[maty[v]] < 1e9+7) continue;
                dist[maty[v]] = dist[u] + 1;
                qu.push(maty[v]);
            }
        }
        return (dist[0] < 1e9+7);
    }

    bool dfs(int u) {
        if(!u) return true;
        for (int v : adj[u]) {
            if(dist[maty[v]] == dist[u] + 1 && dfs(maty[v])) {
                matx[u] = v, maty[v] = u; return true;
            }
        }
        dist[u] = 1e9+7;
        return false;
    }

    void solve(void) {
        cntMatching = 0;
        while(bfs()) for (int i = 1; i <= mNode; ++i) if(!matx[
             i]) cntMatching += dfs(i);
    }

    void show(void) {
        cout << cntMatching << '\n';
        for (int i = 1; i <= mNode; ++i) if(matx[i]) cout << i
             << ' ' << matx[i] << '\n';
    }
};
```

Hungarian.h

Time: $\mathcal{O}\left(N^3\right)$ d41d8c, 88 lines

```
struct Hungarian {
    vector<vector<int>> c; // matrix cost
    vector<int> fx, fy, matchX, matchY; // potentials |
         corresponding node
    vector<int> trace, dist, arg; // last vertex on the left
         side | distance from the tree | the corresponding node
    queue<int> qu; // used for bfs

    int numNode; // assume that |L| = |R| = n
    int start;   // current root of the tree
    int finish;  // leaf node of augmenting path

    Hungarian(int _n) {
        numNode = _n;
        c = vector<vector<int>>(numNode + 1, vector<int>(
             numNode + 1, 1e9+7));
        fx = fy = matchX = matchY = trace = dist = arg = vector
             <int>(numNode + 1);
    }

    inline void addEdge(int u, int v, int _cost) { c[u][v] =
         min(c[u][v], _cost); }
    inline int cost(int u, int v) const { return c[u][v] - fx[u
         ] - fy[v]; }

    void initBFS(int root) {
        start = root;
        qu = queue<int>(); qu.push(start);
        for (int i = 1; i <= numNode; ++i) {
            trace[i] = 0, arg[i] = start;
            dist[i] = cost(start, i);
        }
    }

    int findPath(void) {
        while(sz(qu)) {
            int u(qu.front()); qu.pop();
            for (int v = 1; v <= numNode; ++v) {
                if(trace[v]) continue;
                int w = cost(u, v);
                if(w == 0) {
                    trace[v] = u;
                    if(!matchY[v]) return v;
                    qu.push(matchY[v]);
                }
                if(dist[v] > w) dist[v] = w, arg[v] = u;
            }
        }
        return 0;
    }

    void enlarge(void) {
        for (int y = finish, next; y != 0; y = next) {
            int x = trace[y]; next = matchX[x];
            matchX[x] = y, matchY[y] = x;
        }
    }

    void update(void) {
        int delta = 1e9+7;
        for (int i = 1; i <= numNode; ++i) if(!trace[i]) delta
             = min(delta, dist[i]);
        fx[start] += delta;
        for (int i = 1; i <= numNode; ++i) {
            if(trace[i]) {
                fx[matchY[i]] += delta, fy[i] -= delta;
            } else {
                dist[i] -= delta;
                if(dist[i] == 0) {
                    trace[i] = arg[i];
                    if(matchY[i] == 0) { finish = i; }
                    else qu.push(matchY[i]);
                }
            }
        }
    }

    void hungarian(void) {
        for (int i = 1; i <= numNode; ++i) {
            initBFS(i);
            do {
                finish = findPath();
                if(!finish) update();
            } while(!finish);
            enlarge();
        }
    }

    void show() {
        int ans = 0;
        for (int i = 1; i <= numNode; ++i) if(matchX[i]) ans +=
             c[i][matchX[i]];
        cout << ans << '\n';
        for (int i = 1; i <= numNode; ++i) cout << i << ' ' <<
             matchX[i] << '\n';
    }
};
```

Data structure (5)

LiChaoTreeArray.h

d41d8c, 36 lines

```
template <int MAX>
class LiChao {
private:
  struct Line {
    lli A, B;
    Line(lli A = 0, lli B = 0): A(A), B(B) { }
    inline lli operator () (lli X) const { return A * X + B; }
  } st[4 * MAX + 7];

  void add_line(int id, int l, int r, const Line &vars) {
    if(l > r) return;
    Line cur = st[id], L = vars;
    if(cur(l) < L(l)) swap(cur, L);
    if(cur(r) >= L(r)) st[id] = L;
    else {
      int mid = (l + r) >> 1;
      if(cur(mid) > L(mid)) st[id] = L, add_line(id << 1 | 1,
          mid + 1, r, cur);
      else st[id] = cur, add_line(id << 1, l, mid, L);
    }
  }

  lli query(int id, int l, int r, lli X) const {
    if (l > r || X < l || X > r) return LINF;
    lli res = st[id] (X);
    if (l == r) return res;
    int mid = (l + r) >> 1;
    res = min(res, query(id << 1, l, mid, X));
    res = min(res, query(id << 1 | 1, mid + 1, r, X));

    return res;
  }
public:
  LiChao(void) {}
  void add_line(lli A, lli B) { add_line(1, 0, MAX, Line(A, B))
      ; }
  lli query(lli X) const { return query(1, 0, MAX, X); }
};
```

LiChaoTreePtr.h

d41d8c, 64 lines

```
const int64_t INF = 1e18 + 7;

struct Line {
  int64_t a, b;

  Line(int64_t a = 0, int64_t b = -INF) : a(a), b(b) {}

  inline int64_t operator () (int64_t x) const { return a * x +
        b; }
};

struct LiChao {
  Line value;
  LiChao* lef;
  LiChao* rig;

  LiChao(void) : value(Line()), lef(nullptr), rig(nullptr) {}

  void update(int l, int r, int u, int v, const Line& LINE) {
    if (l > r or u > v or u > r or l > v)
      return;

    if (u <= l and r <= v) {
      Line current = value, other = LINE;
      if (current(l) > other(l))
        swap(current, other);

      if (current(r) <= other(r))
        value = other;
      else {
        if (l == r)
          return;
        int m = (l + r) >> 1;
        if (current(m) > other(m)) {
          value = current;
          lef = lef ? lef : new LiChao();
          lef->update(l, m, u, v, other);
        } else {
          value = other;
          rig = rig ? rig : new LiChao();
          rig->update(m + 1, r, u, v, current);
        }
      }
      return;
    }

    int m = (l + r) >> 1;
    lef = lef ? lef : new LiChao();
    rig = rig ? rig : new LiChao();

    lef->update(l, m, u, v, LINE);
    rig->update(m + 1, r, u, v, LINE);
  }

  int64_t query(int l, int r, int x) {
    if (l > r or x > r or l > x)
      return -INF;
    int64_t ans = value(x);

    int m = (l + r) >> 1;
    ans = max(ans, lef ? lef->query(l, m, x) : -INF);
    ans = max(ans, rig ? rig->query(m + 1, r, x) : -INF);
    return ans;
  }
};
```

DynamicConvexHull.h

Description: Dynamic Convex Hull find Min
Usage: For doubles, use inf = 1/.0, div(a,b) = a/b

d41d8c, 32 lines

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

struct DynamicHull : multiset<Line, less<>> {
    const ll inf = LLONG_MAX;

    ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b);
          }

    bool bad(iterator x, iterator y) {
        if(y == end()) { x->p = inf; return false; }
        if(x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }

    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (bad(y, z)) z = erase(z);
        if(x != begin() && bad(--x, y)) bad(x, y = erase(y));
        while((y = x) != begin() && (--x)->p >= y->p) bad(x,
             erase(y));
    }

    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

SplayTree.h

Usage: can split into k parts and then reverse / get sum these parts
Time: $\mathcal{O}(logN)$ average

d41d8c, 83 lines

```
namespace SplayTree {
    struct TNode {
        int sz, p, L, R;
        TNode() { sz = p = L = R = 0; }
    } node[MAXN];

    int root, nArr;

    inline void update(const int &x) {
        node[x].sz = 1 + node[node[x].L].sz + node[node[x].R].
            sz;
    }

    int buildSplay(int l, int r, int pa = 0) {
        if(l > r) return 0;
        int mid = (l + r) >> 1;
        node[mid].p = pa;
        node[mid].L = buildSplay(l, mid - 1, mid);
        node[mid].R = buildSplay(mid + 1, r, mid);
        update(mid);
        return mid;
    }

    inline void setChild(int pa, int child, bool isRight) {
        node[child].p = pa;
        (isRight ? node[pa].R : node[pa].L) = child;
    }

    inline void upTree(int x) {
        int y = node[x].p, z = node[y].p;
        if(x == node[y].R) {
            int b = node[x].L;
            setChild(y, b, 1), setChild(x, y, 0);
        } else {
            int b = node[x].R;
            setChild(y, b, 0), setChild(x, y, 1);
        }
        setChild(z, x, (node[z].R == y));
        update(y), update(x);
    }

    void splay(int x) {
        while(1) {
            if(node[x].p == 0) break;
            int y = node[x].p, z = node[y].p;
            if(z > 0) {
                if((y == node[z].R) == (x == node[y].R)) {
                     upTree(y); }
                else upTree(x);
            }
            upTree(x);
        }
    }

    int locate(int root, int c) {
        int x(root);
        while(1) {
            int s = node[node[x].L].sz;
            if(s + 1 == c) return x;
            if(s + 1 > c) { x = node[x].L; }
            else { c -= s + 1; x = node[x].R; }
        }
    }

    void split(int T, int &A, int &B, int c) {
        if(!c) { A = 0, B = T; return; }
        int x = locate(T, c);
        splay(x); A = x, B = node[A].R;
        node[A].R = node[B].p = 0; update(A);
    }

    int join(int A, int B) {
        if(A == 0) return B;
        while(node[A].R) A = node[A].R;
        splay(A), setChild(A, B, 1), update(A);
        return A;
    }

    void preOrder(const int &id) {
        if(!id) return;
        preOrder(node[id].L);
        cout << id << ' ';
        preOrder(node[id].R);
    }
}
```

OrderTree.h

<bits/extc++.h> d41d8c, 13 lines

```
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

<bits/extc++.h> d41d8c, 6 lines

```
// 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});
```

BitsetTree.h

d41d8c, 78 lines

```
struct BitsetTree {
    static const int LAYER = 3; // number of layer is log64(n)
    static const int BLOCK = 64;

    ull a[MAXN + 100];
    int layer_start[LAYER];

    void update(int x) {
        int id(x >> 6), prev(layer_start[LAYER - 1] + id);
        a[prev] ^= (1ULL << (x & (BLOCK - 1)));
        for (int i = LAYER - 2; i >= 0; --i) {
            x >>= 6, id >>= 6;
            int digit(x & (BLOCK - 1)), cur(layer_start[i] + id
                );
            if((a[prev] > 0) != (a[cur] >> digit & 1)) a[cur] ^
                = 1ULL << digit;
            prev = cur;
        }
    }

    inline int findR(ull mask, int i) {
        ull x(-1); mask &= (x << i);
        if(i >= BLOCK || mask == 0) return -1;
        return __builtin_ctzll(mask);
    }

    int walk_forward(int x) {
        int id(x >> 6), pos = findR(a[layer_start[LAYER - 1] +
             id], x & (BLOCK - 1));
        if(pos != -1) return x - (x & (BLOCK - 1)) + pos;
        for (int i = LAYER - 2; i >= 0; --i) {
            id >>= 6, x >>= 6;
            int cur(layer_start[i] + id), digit = (x & (BLOCK -
                1)) + 1;
            int pos = findR(a[cur], digit);
            if(pos != -1) {
                id = (id << 6) + pos;
                for (int j = i + 1; j < LAYER; ++j) {
                    int digit = __builtin_ctzll(a[layer_start[j
                        ] + id]);
                    id = (id << 6) + digit;
                }
                return id;
            }
        }
        return -1;
    }

    int findL(ull mask, int i) {
        mask &= (1ULL << i) - 1;
        if(i >= BLOCK || mask == 0) return -1;
        return 63 - __builtin_clzll(mask);
    }

    int walk_backward(int x) {
        int id(x >> 6), pos = findL(a[layer_start[LAYER - 1] +
             id], x & (BLOCK - 1));
        if(pos != -1) return x - (x & (BLOCK - 1)) + pos;
        for (int i = LAYER - 2; i >= 0; --i) {
            id >>= 6, x >>= 6;
            int cur(layer_start[i] + id), digit = (x & (BLOCK -
                1));
            int pos = findL(a[cur], digit);
            if(pos != -1) {
                id = (id << 6) + pos;
                for (int j = i + 1; j < LAYER; ++j) {
                    int digit = 63 - __builtin_clzll(a[
                        layer_start[j] + id]);
                    id = (id << 6) + digit;
                }
                return id;
            }
        }
        return -1;
    }

    void init(void) {
        memset(a, 0, sizeof(a));
        int cur(0), sum(1);
        for (int i = 0; i < LAYER; ++i) {
            layer_start[i] = cur;
            cur += sum, sum *= BLOCK;
        }
    }

} bst;
```

ManhattanSpanningTree.h

d41d8c, 60 lines

```
struct Point {
    int x, y, xy, id;
} p[MAXN];

struct Edge {
    int u, v; ll w;
    Edge(int _u = 0, int _v = 0, ll _w = 0) : u(_u), v(_v), w(
        _w) {}
};

int nArr;

typedef pair<ll, int> pli;
namespace ManhattanSpanningTree {
    vector<int> idx;
    pli fen[MAXN];
    int nTree;

    void modify(int i, pli x) {
        for (; i > 0; i -= i & -i) fen[i] = min(fen[i], x);
    }

    pli get(int i) {
        pli res = {2e9+7, -1};
        for (; i <= nTree; i += i & -i) res = min(res, fen[i]);
        return res;
    }

    void buildSpan(vector<Edge> &edges) {
        sort(p + 1, p + nArr + 1, [](const Point &a, const
            Point &b) { return ii(a.x, a.y) < ii(b.x, b.y); })
            ;
        idx.clear();
        for (int i = 1; i <= nArr; ++i) {
            p[i].xy = p[i].y - p[i].x;
            idx.push_back(p[i].xy);
        }
        sort(idx.begin(), idx.end());
        idx.erase(unique(idx.begin(), idx.end()), idx.end());
        nTree = idx.size();
        for (int i = 1; i <= nTree; ++i) fen[i] = {2e9+7, -1};
        for (int i = 1; i <= nArr; ++i) p[i].xy = upper_bound(
            idx.begin(), idx.end(), p[i].xy) - idx.begin();
        for (int i = nArr; i > 0; --i) {
            ii v = get(p[i].xy);
            if(v.se != -1) edges.push_back(Edge(p[i].id, p[v.se
                ].id, abs(p[i].x - p[v.se].x) + abs(p[i].y - p
                [v.se].y)));
            modify(p[i].xy, ii(p[i].x + p[i].y, i));
        }
    }

    void buildEdge(vector<Edge> &edges) {
        edges.clear();
        for (int loop = 0; loop < 4; ++loop) {
            buildSpan(edges);
            for (int i = 1; i <= nArr; ++i) swap(p[i].x, p[i].y
                );
            buildSpan(edges);
            for (int i = 1; i <= nArr; ++i) {
                swap(p[i].x, p[i].y);
                if(loop & 1) { p[i].y = -p[i].y; }
                else p[i].x = -p[i].x;
            }
        }
    }
}
```

PersistentIT.h

d41d8c, 41 lines

```
namespace PersistentSeg {
    struct SegNode {
        int cnt, L, R;
    } seg[50 * MAXN];

    int nNode, nTree;

    void init(int _n) { nNode = _n, nTree = 0; }

    int update(int oldID, int l, int r, int pos, int val) {
        if(l == r) {
            seg[++nTree] = seg[oldID], ++seg[nTree].cnt;
            return nTree;
        }
        int cur(++nTree), mid = (l + r) >> 1;
        seg[cur] = seg[oldID];
        if(pos <= mid) {
            seg[cur].L = update(seg[oldID].L, l, mid, pos, val)
                 ;
        } else {
            seg[cur].R = update(seg[oldID].R, mid + 1, r, pos,
                 val);
        }
        seg[cur].cnt = seg[seg[cur].L].cnt + seg[seg[cur].R].
             cnt;
        return cur;
    }

    int update(int oldID, int pos, int val) {
        return update(oldID, 1, nNode, pos, val);
    }

    int queryCnt(int id, int l, int r, int u, int v) {
        if(u <= l && r <= v) return seg[id].cnt;
        int res = 0, mid = (l + r) >> 1;
        if(mid >= u) res += queryCnt(seg[id].L, l, mid, u, v);
        if(mid + 1 <= v) res += queryCnt(seg[id].R, mid + 1, r,
             u, v);
        return res;
    }

    int queryCnt(int id, int u, int v) {
        return queryCnt(id, 1, nNode, u, v);
    }
};
```

ConvexHullTrick.h

d41d8c, 46 lines

```
using i64 = int64_t;

const int inf = 1e9 + 7;

i64 ceil_div(i64 a, i64 b) {
  if (b < 0)
    return ceil_div(-a, -b);
  return a < 0 ? a / b : (a + b - 1) / b;
}

class ConvexHullMax {
private:
  struct Line {
    i64 x, a, b;

    Line(i64 _x = -inf, i64 _a = -inf, i64 _b = -inf) : x(_x),
         a(_a), b(_b) {}

    inline i64 operator () (i64 x) const {
      return a * x + b;
    }

    inline i64 operator ^ (const Line& other) const {
      return ceil_div(other.b - b, a - other.a);
    }

    inline bool operator < (const Line& other) const {
      return x < other.x;
    }
  };

  std::vector <Line> q;
public:
  void insert(i64 a, i64 b) {
    Line l(-inf, a, b);

    while (not q.empty() and (q.back() ^ l) < q.back().x)
      q.pop_back();

    l.x = q.empty() ? -inf : q.back() ^ l;
    q.push_back(l);
  }

  i64 query(i64 x) const {
    return (*std::prev(std::upper_bound(q.begin(), q.end(),
         Line(x)))) (x);
  }
};
```

Graph (6)

TwoSat.h

Usage: add_disjunction(u, nu, v, nv) to represent the or and the negate of variable

d41d8c, 58 lines

```
class TWO_SAT {
private:
  int n;
  vector <vector <int>> forward_edge, back_edge;
  vector <bool> used;
  vector <int> order, comp;

  void dfs_first(int u) {
    used[u] = true;
    for (auto v : forward_edge[u])
      if (not used[v]) dfs_first(v);
    order.push_back(u);
  }

  void dfs_second(int u, int turn) {
    comp[u] = turn;
    for (auto v : back_edge[u])
      if (comp[v] == -1) dfs_second(v, turn);
  }
public:
  TWO_SAT(int n = 0): n(n) {
    used.assign(2 * n + 7, false);
    comp.assign(2 * n + 7, -1);
    forward_edge.resize(2 * n + 7);
    back_edge.resize(2 * n + 7);
  }

  void add_disjunction(int a, bool na, int b, bool nb) {
    a = (a << 1) ^ na;
    b = (b << 1) ^ nb;
    int neg_a = a ^ 1;
    int neg_b = b ^ 1;
    forward_edge[neg_a].push_back(b);
    forward_edge[neg_b].push_back(a);
    back_edge[a].push_back(neg_b);
    back_edge[b].push_back(neg_a);
  }

  vector <bool>* find_solution(void) {
    vector <bool> *assignment = new vector <bool> (2 * n + 7);

    for (int i = 2; i <= 2 * n + 1; i++)
      if (not used[i]) dfs_first(i);

    for (int i = 1, turn = 0; i <= 2 * n; i++) {
      int u = order[2 * n - i];
      if (comp[u] == -1) dfs_second(u, ++turn);
    }

    for (int i = 2; i <= 2 * n; i += 2) {
      if (comp[i] == comp[i + 1])
        return nullptr;
      (*assignment)[i / 2] = comp[i] > comp[i + 1];
    }

    return assignment;
  }
};
```

TwoSatPtr.h

Usage: add_disjunction(u, v) mean (u or v)
-u is not u

d41d8c, 71 lines

```
class TwoSat {
private:
  int n, no;
  int* comp;
  bool* was;
  std::vector <int>* g;
  std::vector <int>* g_t;
  std::vector <int> topo;

  void add_edge(int u, int v) {
    g[u].push_back(v);
    g_t[v].push_back(u);
  }

  void dfs_topo(int u) {
    was[u] = 1;
    for (int v : g[u])
      if (not was[v])
        dfs_topo(v);
    topo.push_back(u);
  }

  void dfs_scc(int u) {
    for (int v : g_t[u]) if (not comp[v]) {
      comp[v] = comp[u];
      dfs_scc(v);
    }
  }
public:
  TwoSat(int _n = 0) : n(_n), no(0) {
    topo.reserve(2 * n);
    comp = new int [2 * n + 1];
    g = new std::vector <int> [2 * n + 1];
    g_t = new std::vector <int> [2 * n + 1];
    was = new bool [2 * n + 1];

    comp += n;
    g += n;
    g_t += n;
    was += n;
  }
  void add_disjunction(int u, int v) {
    add_edge(-u, v);
    add_edge(-v, u);
  }

  std::vector <int>* solve(void) {
    for (int i = 1; i <= n; i += 1) {
      if (not was[i])
        dfs_topo(i);
      if (not was[-i])
        dfs_topo(-i);
    }
    std::reverse(topo.begin(), topo.end());
    for (int u : topo) {
      if (not comp[u]) {
        comp[u] = ++no;
        dfs_scc(u);
      }
    }
    std::vector <int>* ans = new std::vector <int> (n + 1);
    for (int i = 1; i <= n; i += 1) {
      int x = comp[i], y = comp[-i];
      if (x == y)
        return nullptr;
      (*ans)[i] = x > y;
    }
    return ans;
  }
};
```

HLD.h

d41d8c, 116 lines

```
constexpr int max_log = 18;

struct Tree {
  int n, T;
  std::vector <int> heavy, head, st, en, lvl, sz, pv;
  mutable std::vector <int> visited;
  std::vector <std::vector <int>> g, up;

  Tree(int _n = 0) : n(_n) {
    heavy.assign(n, -1);
    head.assign(n, 0);
    st.assign(n, 0);
    en.assign(n, 0);
    lvl.assign(n, 0);
    sz.assign(n, 0);
    pv.assign(n, 0);
    g.assign(n, {});
    up.assign(max_log, {});
    visited.assign(n, 0);
  }

  void add_edge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
  }

  int dfs(int u) {
    sz[u] = 1;
    heavy[u] = -1;
    int max_size = 0;
    for (int v : g[u]) if (v ^ pv[u]) {
      pv[v] = u;
      lvl[v] = lvl[u] + 1;
      dfs(v);
      if (max_size < sz[v]) {
        max_size = sz[v];
        heavy[u] = v;
      }
      sz[u] += sz[v];
    }
    return sz[u];
  }

  void decompose(int u, int h) {
    st[u] = T++;
    head[u] = h;
    if (heavy[u] != -1)
      decompose(heavy[u], h);
    for (int v : g[u]) if (v != pv[u] and v != heavy[u])
      decompose(v, v);
    en[u] = T;
  }

  void work(int x = 0) {
    T = 0;
    pv[x] = x;
    lvl[x] = 0;
    dfs(x);
    decompose(x, x);
    up[0] = pv;
    for (int j = 1; j < max_log; j += 1) {
      up[j].resize(n);
      for (int i = 0; i < n; i += 1)
        up[j][i] = up[j - 1][up[j - 1][i]];
    }
  }

  bool IS_ANCESTOR(int u, int v) const {
    return st[u] <= st[v] and en[u] >= en[v];
  }

  int LCA(int u, int v) const {
    if (IS_ANCESTOR(u, v))
      return u;
    if (IS_ANCESTOR(v, u))
      return v;
    for (int j = max_log - 1; j >= 0; j -= 1) {
      if (not IS_ANCESTOR(up[j][u], v))
        u = up[j][u];
    }
    return pv[u];
  }

  void apply_on_path(int x, int y, const std::function <void (
      int, int, bool)>& f) const {
    int z = LCA(x, y);
    {
      int v = x;
      while (v != z) {
        if (lvl[head[v]] <= lvl[z]) {
          f(st[z] + 1, st[v], true);
          break;
        }
        f(st[head[v]], st[v], true);
        v = pv[head[v]];
      }
    }
    f(st[z], st[z], false);
    int cnt_visited = 0;
    {
      int v = y;
      int cnt_visited = 0;
      while (v != z) {
        if (lvl[head[v]] <= lvl[z]) {
          f(st[z] + 1, st[v], false);
          break;
        }
        visited[cnt_visited++] = v;
        v = pv[head[v]];
      }
      for (int at = cnt_visited - 1; at >= 0; at--) {
        v = visited[at];
        f(st[head[v]], st[v], false);
      }
    }
  }
};
```

Centroid.h

d41d8c, 6 lines

```
int centroid(int u, int parent, int n) {
    for (int v : adj[u])
        if (v != parent && child[v] > n/2 && !del[v])
            return centroid(v, u, n);
    return u;
}
```

BlockCutTree.h

d41d8c, 22 lines

```
void tarjan(int u) {
    low[u] = num[u] = ++num[0];
    for (int it = 0; it < int(adj[u].size()); ++it) {
        int v(adj[u][it]);
        if(!num[v]) {
            st.push(u); tarjan(v);
            low[u] = min(low[u], low[v]);
            if(low[v] == num[u]) {
                lastComp[u] = ++numBCC;
                adjp[u].push_back(numNode + numBCC);
                do {
                    v = st.top(); st.pop();
                    if(lastComp[v] != numBCC) {
                        lastComp[v] = numBCC;
                        adjp[numNode + numBCC].push_back(v);
                    }
                } while(v != u);
            }
        } else { low[u] = min(low[u], num[v]); }
    }
    st.push(u);
}
```

OnlineBridgeCounting.h

d41d8c, 109 lines

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;

void init(int n) {
    par.resize(n);
    dsu_2ecc.resize(n);
    dsu_cc.resize(n);
    dsu_cc_size.resize(n);
    lca_iteration = 0;
    last_visit.assign(n, 0);
    for (int i=0; i<n; ++i) {
        dsu_2ecc[i] = i;
        dsu_cc[i] = i;
        dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}
```

```
int find_2ecc(int v) {
    if (v == -1)
        return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(
        dsu_2ecc[v]);
}

int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
}

void make_root(int v) {
    int root = v;
    int child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]);
        par[v] = child;
        dsu_cc[v] = root;
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}

void merge_path (int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a);
            path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a;
                break;
                }
            last_visit[a] = lca_iteration;
            a = par[a];
        }
        if (b != -1) {
            b = find_2ecc(b);
            path_b.push_back(b);
            if (last_visit[b] == lca_iteration){
                lca = b;
                break;
                }
            last_visit[b] = lca_iteration;
            b = par[b];
        }

    }

    for (int v : path_a) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
    for (int v : path_b) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
}

void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b)
        return;

    int ca = find_cc(a);
    int cb = find_cc(b);

    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b);
            swap(ca, cb);
        }
        make_root(a);
        par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
}
```

CheckBridgeArticulation.h

d41d8c, 14 lines

```
void tarjan(int u, int p_id) {
    low[u] = num[u] = ++num[0];
    int numChild = 0;
    for (auto [v, id] : adj[u])
        if(!num[v]) {
            tarjan(v, id);
            low[u] = min(low[u], low[v]);
            ++numChild;
            if(low[v] >= num[u]) {} // (u, v) la cau
            if(p_id != -1 && low[v] >= num[u]) {} // u la khop
        } else if(id != p_id) low[u] = min(low[u], num[v]);
    }
    if(p == -1 && sz >= 2) {} // u la khop
}
```

MaximalClique.h

d41d8c, 32 lines

```
int g[N][N];
int res;
ll edges[N];

void BronKerbosch(int n, ll R, ll P, ll X) { // O(3 ^ (n / 3))
    // here we will find all possible maximal cliques (not
        maximum) i.e. there is no node which can be included
        in this set
    if (P == 0ll && X == 0ll) {
        int t = __builtin_popcountll(R);
        res = max(res, t);
        return;
    }

    int u = 0;
    while (!((1LL << u) & (P | X))) u++;
    for (int v = 0; v < n; v++) {
        if (((1LL << v) & P) && !((1LL << v) & edges[u])) {
            BronKerbosch(n, R | (1LL << v), P & edges[v], X &
                edges[v]);
            P -= (1LL << v);
            X |= (1LL << v);
        }
    }
}

int max_clique(int n) {
    res = 0;
    for (int i = 1; i <= n; i++) {
        edges[i - 1] = 0;
        for (int j = 1; j <= n; j++)  if (g[i][j]) edges[i - 1]
            |= (1ll << (j - 1));
    }
    BronKerbosch(n, 0, (1ll << n) - 1, 0);
    return res;
}
```

String (7)

KMP.h

d41d8c, 9 lines

```
void kmp(const string &str) {
    int strLen(sz(str) - 1);
    lps[1] = 0;
    for (int i = 2; i <= strLen; ++i) {
        int k = lps[i - 1];
        while(k > 0 && str[k + 1] != str[i]) k = lps[k];
        lps[i] = k + (str[k + 1] == str[i]);
    }
}
```

Manacher.h

d41d8c, 35 lines

```
int n; string S;
int pod[MaxN], pev[MaxN];

void calc_pod() {
    int L = 1, R = 0;
    for(int i = 1; i <= n; i++) {
        if(i > R) pod[i] = 0;
        else pod[i] = min(R - i, pod[L + (R - i)]);
        while(i - pod[i] - 1 > 0 && i + pod[i] + 1 <= n && S[i
            - pod[i] - 1] == S[i + pod[i] + 1]) pod[i]++;

        if(i + pod[i] > R) {
            R = i + pod[i];
            L = i - pod[i];
        }
    }
}

void calc_pev() {
    int L = 1, R = 0;
    for(int i = 1; i < n; i++) {
        int j = i + 1;
        if(j > R) pev[i] = 0;
        else pev[i] = min(R - j + 1, pev[L + (R - j)]);
        while(i - pev[i] > 0 && j + pev[i] <= n && S[i - pev[i
            ]] == S[j + pev[i]]) pev[i]++;
        if(i + pev[i] > R) {
            R = i + pev[i];
            L = j - pev[i];
        }
    }
}

// n = S.length();
// S = ' ' + S;
// calc_pod();
// calc_pev();
```

AhoCorasick.h

d41d8c, 45 lines

```
namespace AhoCorasick {
    struct TrieNode {
        int nxt[NUM_CHAR], link, term;
    };
    vector<TrieNode> Trie;

    void insertString(const string &str) {
        int pt(0);
        for (char ch : str) {
            int c(ch - ’0’);
            if(!Trie[pt].nxt[c]) {
                Trie[pt].nxt[c] = Trie.size();
                Trie.emplace_back();
            }
            pt = Trie[pt].nxt[c];
        }
        Trie[pt].term = pt;
    }

    void buildAutomaton(void) {
        Trie.emplace_back();
        for (int i = 1; i <= nArr; ++i) insertString(str[i]);
        queue<int> qu; qu.push(0);
        while(qu.size()) {
            int v(qu.front()), u(Trie[v].link); qu.pop();
            if(!Trie[v].term) Trie[v].term = Trie[u].term;
            for (int c = 0; c < NUM_CHAR; ++c) {
                if(Trie[v].nxt[c]) {
                    Trie[Trie[v].nxt[c]].link = (v) ? Trie[u].nxt[c] : 0;
                    qu.push(Trie[v].nxt[c]);
                } else Trie[v].nxt[c] = Trie[u].nxt[c];
            }
        }
    }

    void query(const string &str) {
        int pt(0);
        for (char ch : str) {
            int c(ch - ’0’);
            while(pt > 0 && !Trie[pt].nxt[c]) pt = Trie[pt].link;
            pt = Trie[pt].nxt[c];
        }
    }
}
```

Hash.h

d41d8c, 49 lines

```
const int MOD[] = {(int) 1e9 + 2277, (int) 1e9 + 5277, (int) 1e9 + 8277, (int) 1e9 + 9277};
const int BASE = 256;

struct Hash {
    ll value[NMOD];

    Hash(char c = 0) {
        for (int i = 0; i < NMOD; ++i) value[i] = c;
    }

    Hash operator + (const Hash &x) const {
        Hash res;
        for (int j = 0; j < NMOD; ++j) {
            res.value[j] = value[j] + x.value[j];
            if(res.value[j] >= MOD[j]) res.value[j] -= MOD[j];
        }
        return res;
    }

    Hash operator - (const Hash &x) const {
        Hash res;
        for (int j = 0; j < NMOD; ++j) {
            res.value[j] = value[j] - x.value[j];
            if(res.value[j] < 0) res.value[j] += MOD[j];
        }
        return res;
    }

    Hash operator * (int k) const {
        Hash res;
        for (int j = 0; j < NMOD; ++j) res.value[j] = value[j] * pw[j][k] % MOD[j];
        return res;
    }

    bool operator == (const Hash &x) const {
        for (int j = 0; j < NMOD; ++j) if(value[j] != x.value[j]) return false;
        return true;
    }

};

Hash getHash(int l, int r) { return (hashVal[r] - hashVal[l - 1]) * (n - r); }

void prepare() {
    for (int j = 0; j < NMOD; ++j) {
        pw[j][0] = 1;
        for (int i = 1; i <= n; ++i) pw[j][i] = pw[j][i - 1] * BASE % MOD[j];
    }
}
```

RabinKarp2D.h

d41d8c, 112 lines

```
namespace RabinKarp {
    const int NMOD = 3;

    const int MOD[] = {(int) 1e9 + 2277, (int) 1e9 + 5277, (int) 1e9 + 8277, (int) 1e9 + 9277};
    const int BASE = 256;

    ll pw[NMOD][MAXN];

    struct Hash {
        ll value[NMOD];

        Hash(char c = 0) {
            for (int i = 0; i < NMOD; ++i) value[i] = c;
        }

        Hash operator + (const Hash &x) const {
            Hash res;
            for (int j = 0; j < NMOD; ++j) {
                res.value[j] = value[j] + x.value[j];
                if(res.value[j] >= MOD[j]) res.value[j] -= MOD[j];
            }
            return res;
        }

        Hash operator - (const Hash &x) const {
            Hash res;
            for (int j = 0; j < NMOD; ++j) {
                res.value[j] = value[j] - x.value[j];
                if(res.value[j] < 0) res.value[j] += MOD[j];
            }
            return res;
        }

        Hash operator * (int k) const {
            Hash res;
            for (int j = 0; j < NMOD; ++j) res.value[j] = value[j] * pw[j][k] % MOD[j];
            return res;
        }

        Hash operator * (Hash h) const {
            Hash res;
            for (int j = 0; j < NMOD; ++j) res.value[j] = 1LL * value[j] * h.value[j] % MOD[j];
            return res;
        }

        bool operator == (const Hash &x) const {
            for (int j = 0; j < NMOD; ++j) if(value[j] != x.value[j]) return false;
            return true;
        }

    } hashVal[MAXN];

    void prepare() {
        for (int j = 0; j < NMOD; ++j) {
            pw[j][0] = 1;
            for (int i = 1; i < MAXN; ++i) pw[j][i] = pw[j][i - 1] * BASE % MOD[j];
        }
    }

    Hash dr = 1, dc = 1; // Highest power for row/col hashing

    // Checks if all values of pattern matches with the text
    bool check(vector<vector<char>> &txt, vector<vector<char>> &pat, int r, int c) {
        /*for (int i = 0; i < pat.size(); ++i) {
            for (int j = 0; j < pat[0].size(); ++j)
                if(pat[i][j] != txt[i + r][j + c]) return false;
        }*/
        return true;
    }

    // Finds the first hash of first n rows where n is no. of rows in pattern
    vector<Hash> findHash(vector<vector<char>> &mat, int row) {
        vector<Hash> hash;
        int col = mat[0].size();
        for (int i = 0; i < col; ++i) {
            Hash h(0);
            for (int j = 0; j < row; ++j) h = h * 1 + mat[j][i];
            hash.push_back(h);
        }
        return hash;
    }

    //rolling hash function for columns
    void colRollingHash(vector<vector<char>> &txt, vector<Hash> &t_hash, int row, int p_row) {
        for (int i = 0; i < sz(t_hash); ++i)
            t_hash[i] = (t_hash[i] - Hash(txt[row][i]) * dr) * 1 + Hash(txt[row + p_row][i]);
    }

    int solve(vector<vector<char>> &txt, vector<vector<char>> &pat) {
        int t_row = sz(txt), t_col = sz(txt[0]);
        int p_row = sz(pat), p_col = sz(pat[0]);
        dr = Hash(1) * (p_row - 1);
        dc = Hash(1) * (p_col - 1);
        vector<Hash> t_hash = findHash(txt, p_row); // column
             hash of p_row rows
        vector<Hash> p_hash = findHash(pat, p_row); // column
             hash of p_row rows
        Hash p_val = 0;  // hash of entire pattern matrix
        for (int i = 0; i < p_col; ++i) p_val = p_val * 1 +
             p_hash[i];
        int res = 0;
        for(int i = 0; i <= (t_row - p_row); ++i) {
            Hash t_val = 0;
            for(int i = 0; i < p_col; ++i) t_val = t_val * 1 +
                 t_hash[i];
            for(int j = 0; j <= (t_col - p_col); ++j) {
                res += (p_val == t_val && check(txt, pat, i, j)
                    );
                // calculating t_val for next set of columns
                t_val = (t_val - t_hash[j] * dc) * 1 + t_hash[j
                     + p_col];
            }
            // call this function for hashing form next row
            if(i < t_row - p_row) colRollingHash(txt, t_hash, i
                 , p_row);
        }
        return res;
    }
}
```

SuffixArray.h

Time: $\mathcal{O}(N \log^2(N))$ d41d8c, 29 lines

```
namespace SuffixArray {
    int tmp[MAXN], gap;
    bool sufCmp(int i, int j) {
        if(pos[i] != pos[j]) return (pos[i] < pos[j]);
        return (max(i, j) + gap < strLen) ? (pos[i + gap] < pos
             [j + gap]) : (i > j);
    }

    void buildLCP(void) {
        int k(0);
        for (int i = 0; i < strLen; ++i) {
            if(pos[i] != strLen - 1) {
                for (int j = sa[pos[i] + 1]; str[i + k] == str[
                     j + k]; ) ++k;
                lcp[pos[i]] = k, k -= bool(k);
            }
        }
    }

    void buildSA(void) {
        for (int i = 0; i < strLen; ++i) sa[i] = i, pos[i] =
             str[i];
        for (gap = 1; ; gap <<= 1) {
            sort(sa, sa + strLen, sufCmp);
            for (int i = 0; i + 1 < strLen; ++i)
                tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
            for (int i = 0; i < strLen; ++i) pos[sa[i]] = tmp[i
                 ];
            if(tmp[strLen - 1] == strLen - 1) break;
        }
        buildLCP();
    }
}
```

SuffixArrayCountSort.h

Time: $\mathcal{O}(N \log(N))$ d41d8c, 36 lines

```
namespace SuffixArray {
    int cnt[MAXN], tmp[MAXN], gap;
    bool sufCmp(int u, int v) {
        if(pos[u] != pos[v]) return pos[u] < pos[v];
        return (max(u, v) + gap <= strLen) ? pos[u + gap] < pos
             [v + gap] : u > v;
    }

    void countsort(int k) {
        for (int i = 0; i <= max(256, strLen); ++i) cnt[i] = 0;
        for (int i = 1; i <= strLen; ++i) ++cnt[(i + k <=
             strLen ? pos[i + k] : 0)];
        for (int i = 1; i <= max(256, strLen); ++i) cnt[i] +=
             cnt[i - 1];
        for (int i = strLen; i > 0; --i) tmp[cnt[(sa[i] + k <=
             strLen ? pos[sa[i] + k] : 0)]--] = sa[i];
        for (int i = 1; i <= strLen; ++i) sa[i] = tmp[i];
    }

    void buildLCP(void) {
        int k(0);
        for (int i = 1; i <= strLen; ++i) {
            if(pos[i] < strLen) {
                for (int j = sa[pos[i] + 1]; str[i + k] == str[
                     j + k]; ) ++k;
                lcp[pos[i]] = k, k -= (k > 0);
            }
        }
    }

    void buildSA(void) {
        for (int i = 1; i <= strLen; ++i) sa[i] = i, pos[i] =
             str[i];
        for (gap = 1; gap <= strLen; gap <<= 1) {
            countsort(gap), countsort(0);
            tmp[sa[1]] = 1;
            for (int i = 2; i <= strLen; ++i) tmp[sa[i]] = tmp[
                 sa[i - 1]] + sufCmp(sa[i - 1], sa[i]);
            for (int i = 1; i <= strLen; ++i) pos[i] = tmp[i];
        }
        buildLCP();
    }
}
```

SuffixArrayDC3.h

Time: $\mathcal{O}(N)$ d41d8c, 112 lines

```
#define MASK(i) (1LL<<(i))
#define BIT(x,i) (((x)>>(i))&1)
#define tget(i) BIT(t[(i) >> 3], (i) & 7)
#define tset(i, b) { if (b) t[(i) >> 3] |= MASK((i) & 7); else
     t[(i) >> 3] &= ~MASK((i) & 7); }
#define chr(i) (cs == sizeof(int) ? ((int *)s)[i] : ((unc *)s)[
     i])
#define isLMS(i) ((i) > 0 && tget(i) && !tget((i) - 1))

typedef unsigned char unc;
class SuffixArray {
    public:
        int *sa, *lcp, *pos, n;
        unc *s;

        void getBuckets(unc s[], vector<int> &bkt, int n, int k
             , int cs, int end) {
            for (int i = 0; i <= k; ++i) bkt[i] = 0;
            for (int i = 0; i < n; ++i) ++bkt[chr(i)];
            int sum = 0;
            for (int i = 0; i <= k; ++i) {
                sum += bkt[i];
                bkt[i] = (end) ? sum : sum - bkt[i];
            }
        }

        void inducesal(vector<unc> &t, int sa[], unc s[],
             vector<int> &bkt, int n, int k, int cs, int end) {
            getBuckets(s, bkt, n, k, cs, end);
            for (int i = 0; i < n; ++i) {
                int j = sa[i] - 1;
                if(j >= 0 && !tget(j)) sa[bkt[chr(j)]++] = j;
            }
        }

        void inducesas(vector<unc> &t, int sa[], unc s[],
             vector<int> &bkt, int n, int k, int cs, int end) {
            getBuckets(s, bkt, n, k, cs, end);
            for (int i = n - 1; i >= 0; --i) {
                int j = sa[i] - 1;
                if(j >= 0 && tget(j)) sa[--bkt[chr(j)]] = j;
            }
        }

        void buildSA(unc s[], int sa[], int n, int k, int cs) {
            vector<unc> t = vector<unc>(n / 8 + 1, 0);
            int j;
            tset(n - 2, 0); tset(n - 1, 1);
            for (int i = n - 3; i >= 0; --i) tset(i, (chr(i) <
                 chr(i + 1)) || (chr(i) == chr(i + 1) && tget(i
                  + 1)));
            vector<int> bkt = vector<int>(k + 1, 0);
            getBuckets(s, bkt, n, k, cs, true);
            for (int i = n - 1; i >= 0; --i) sa[i] = -1;
            for (int i = n - 1; i >= 0; --i) if(isLMS(i)) sa[--
                 bkt[chr(i)]] = i;
            inducesal(t, sa, s, bkt, n, k, cs, false);
            inducesas(t, sa, s, bkt, n, k, cs, true);
            bkt.clear();
            int n1 = 0;
            for (int i = 0; i < n; ++i) if(isLMS(sa[i])) sa[n1
                 ++] = sa[i];
            for (int i = n1; i < n; ++i) sa[i] = -1;
            int name = 0, prev = -1;
            for (int i = 0; i < n1; ++i) {
                int pos = sa[i];
                bool diff = false;
                for (int d = 0; d < n; ++d) {
                    if(prev < 0 || chr(prev + d) != chr(pos + d
                         ) || tget(prev + d) != tget(pos + d))
                         {
                        diff = true;
                        break;
                    } else if(d > 0 && (isLMS(prev + d) ||
                         isLMS(pos + d))) break;
                }
                if(diff) ++name, prev = pos;
                sa[n1 + pos / 2] = name - 1;
            }
            j = n - 1;
            for (int i = n - 1; i >= n1; --i) if(sa[i] >= 0) sa
                 [j--] = sa[i];
            int *sa1 = sa, *s1 = sa + n - n1;
            if(name < n1) buildSA((unc*)s1, sa1, n1, name - 1,
                 sizeof(int));
            else for (int i = 0; i < n1; ++i) sa1[s1[i]] = i;
            bkt.assign(k + 1, 0);
            getBuckets(s, bkt, n, k, cs, true);
            j = 0;
            for (int i = 0; i < n; ++i) if(isLMS(i)) s1[j++] =
                 i;
            for (int i = 0; i < n1; ++i) sa1[i] = s1[sa1[i]];
            for (int i = n1; i < n; ++i) sa[i] = -1;
            for (int i = n1 - 1; i >= 0; --i) j = sa[i], sa[i]
                 = -1, sa[--bkt[chr(j)]] = j;
            inducesal(t, sa, s, bkt, n, k, cs, false);
            inducesas(t, sa, s, bkt, n, k, cs, true);
            bkt.clear(), t.clear();
        }

        void buildLCP(void) {
            for (int i = 1; i <= n; ++i) pos[sa[i]] = i;
            int k = 0;
            lcp[n] = 0;
            for (int i = 1; i <= n; ++i) if(pos[i] < n) {
                for (int j = sa[pos[i] + 1]; s[i + k - 1] == s[
                     j + k - 1]; ) ++k;
                lcp[pos[i]] = k; k -= (k > 0);
            }
        }

        void show(void) {
            for (int i = 1; i <= n; ++i) cout << sa[i] << ' ';
                 cout << '\n';
            for (int i = 1; i <= n; ++i) cout << pos[i] << ' ';
                  cout << '\n';
            for (int i = 1; i <= n; ++i) cout << lcp[i] << ' ';
                  cout << '\n';
        }

        SuffixArray() : n(0), sa(NULL), lcp(NULL), pos(NULL), s
             (NULL) {}

        SuffixArray(string ss) : n(sz(ss)) {
            sa = new int[n + 7];
            lcp = new int[n + 7];
            pos = new int[n + 7];
            s = (unc*) ss.c_str();
            buildSA(s, sa, n + 1, 256, sizeof(char));
            for (int i = 1; i <= n; ++i) ++sa[i];
            buildLCP();
        }
};
```

Geometry (8)

8.1 Geometric primitives

Point.h

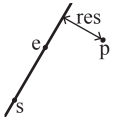
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

d41d8c, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

lineDistance.h

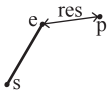
Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate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



"Point.h" d41d8c, 4 lines

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

SegmentDistance.h



Description:
Returns the shortest distance between point p and the line segment from point s to e.

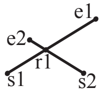
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" d41d8c, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



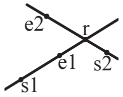
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

"Point.h", "OnSegment.h" d41d8c, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

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



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

"Point.h" d41d8c, 8 lines

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

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. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage: `bool left = sideOf(p1,p2,q)==1;`

"Point.h" d41d8c, 9 lines

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

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

OnSegment.h

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

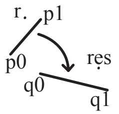
"Point.h" d41d8c, 3 lines

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

linearTransformation.h

Description:

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



"Point.h" d41d8c, 6 lines

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

Angle.h

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

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

d41d8c, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

8.2 Circles

CircleIntersection.h

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

"Point.h" d41d8c, 11 lines

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

CircleTangents.h

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

"Point.h" d41d8c, 13 lines

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

CirclePolygonIntersection.h

Description: Returns the area of the intersection of a circle with a ccw polygon.

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

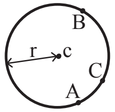
"../../content/geometry/Point.h" d41d8c, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = q + d * (t-1);
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

circumcircle.h

Description:

The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



"Point.h" d41d8c, 9 lines

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

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.

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

"circumcircle.h" d41d8c, 17 lines

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

8.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage:
```
vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);
```

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

"Point.h", "OnSegment.h", "SegmentDistance.h" d41d8c, 11 lines

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

PolygonArea.h

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

"Point.h" d41d8c, 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;
}
```

PolygonCenter.h

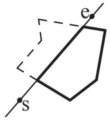
Description: Returns the center of mass for a polygon.

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

"Point.h" d41d8c, 9 lines

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

PolygonCut.h

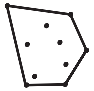


Description: Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h" d41d8c, 13 lines

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

ConvexHull.h



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

"Point.h" d41d8c, 13 lines

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

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h" d41d8c, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" d41d8c, 14 lines

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

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

"Point.h" d41d8c, 39 lines

```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" d41d8c, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

"Point.h" d41d8c, 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
```

```
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], . . . }, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h" d41d8c, 88 lines

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

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

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

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

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

8.5 3D

PolyhedronVolume.h

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

d41d8c, 6 lines

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

Point3D.h

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

d41d8c, 32 lines

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

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
Time: $\mathcal{O}(n^2)$

"Point3D.h" d41d8c, 49 lines

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

sphericalDistance.h

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

d41d8c, 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);
}
```

Various (9)

MISC (10)

yCombinator.h

d41d8c, 15 lines

```
template <class Fun> class y_combinator_result {
  Fun fun_;

  public:
  template <class T>
  explicit y_combinator_result(T &&fun) : fun_(std::forward<T>(
       fun)) {}

  template <class... Args> decltype(auto) operator()(Args &&...
       args) {
    return fun_(std::ref(*this), std::forward<Args>(args)...);
  }
};

template <class Fun> decltype(auto) y_combinator(Fun &&fun) {
  return y_combinator_result<std::decay_t<Fun>>(std::forward<
       Fun>(fun));
}
```

BigNum.h

Time: $\mathcal{O}(N/9)$ to $+/-$ or ($*/$ div / mod) a bignum with an int64 number $\mathcal{O}(N/9)$ to comparing 2 bignums or toString $\mathcal{O}\left((N/9)^2\right)$ to $*$ for 2 bignums

d41d8c, 132 lines

```
struct Bignum {
    static const int MAX_DIGIT = 1000;
    static const int BASE = (int) 1e9;
    int digits[MAX_DIGIT], numDigit;

    Bignum(ll x = 0) {
        numDigit = 0;
        memset(digits, 0, sizeof digits);
        if(!x) numDigit = 1;
        while(x > 0) digits[numDigit++] = x % BASE, x /= BASE;
    }

    Bignum(string s) {
        numDigit = 0;
        memset(digits, 0, sizeof digits);
        ll x(0);
        int i(s.length() - 1), l(i + 1);
        for (int i = l - 1; i >= 8; i -= 9) digits[numDigit++]
             = stoll(s.substr(i - 8, 9));
        if(l % 9) digits[numDigit++] = stoll(s.substr(0, l % 9)
             );
    }

    Bignum& operator += (const Bignum &x) {
        int carry(0);
        numDigit = max(numDigit, x.numDigit);
        for (int i = 0; i < numDigit; ++i) {
            digits[i] += x.digits[i] + carry;
            if(digits[i] >= BASE) { digits[i] -= BASE, carry =
                 1; }
            else carry = 0;
        }
        if(carry) digits[numDigit++] = carry;
        return *this;
    }

    Bignum operator + (const Bignum &x) const {
        Bignum res(*this);
        return res += x;
    }

    Bignum& operator -= (const Bignum &x) {
        int carry(0);
        for (int i = 0; i < numDigit; ++i) {
            digits[i] -= x.digits[i] + carry;
            if(digits[i] < 0) { digits[i] += BASE, carry = 1; }
            else carry = 0;
        }
        while(numDigit > 1 && !digits[numDigit - 1]) --numDigit
             ;
        return *this;
    }

    Bignum operator - (const Bignum &x) const {
        Bignum res(*this); res -= x;
        return res;
    }

    Bignum& operator *= (int x) {
        if (!x) { *this = Bignum(0); return *this; }
        ll remain = 0;
        for (int i = 0; i < numDigit; ++i) {
            remain += 1LL * digits[i] * x;
            digits[i] = remain % BASE, remain /= BASE;
        }
        while(remain > 0) digits[numDigit++] = remain % BASE,
             remain /= BASE;
        return *this;
    }

    Bignum operator * (int x) const {
        Bignum res(*this); res *= x;
        return res;
    }

    Bignum operator * (const Bignum &x) const {
        Bignum res(0);
        for (int i = 0; i < numDigit; ++i) {
            if (!digits[i]) continue;
            for (int j = 0; j < x.numDigit; ++j) {
                if(x.digits[j] > 0) {
                    ll tmp = 1LL * digits[i] * x.digits[j];
                    int pos(i + j);
                    while(tmp > 0) {
                        tmp += res.digits[pos];
                        res.digits[pos] = tmp % BASE;
                        tmp /= BASE, ++pos;
                    }
                }
            }
        }
        res.numDigit = MAX_DIGIT - 1;
        while (res.numDigit > 1 && !res.digits[res.numDigit -
             1]) --res.numDigit;
        return res;
    }

    ll operator % (ll x) const {
    ll res(0);
    for (int i = numDigit - 1; i >= 0; i--) res = (res * BASE +
          digits[i]) % x;
    return res;
  }

  Bignum operator / (ll x) const {
    Bignum res(0);
    ll rem(0);
    for (int i = numDigit - 1; i >= 0; i--) {
      res.digits[i] = (BASE * rem + digits[i]) / x;
      rem = (BASE * rem + digits[i]) % x;
    }
    res.numDigit = numDigit;
    while (res.numDigit > 1 && !res.digits[res.numDigit - 1])
         --res.numDigit;
    return res;
  }

    #define COMPARE(a, b) (((a) > (b)) - ((a) < (b)))
    int compare(const Bignum &x) const {
        if (numDigit != x.numDigit) return COMPARE(numDigit, x.
             numDigit);
        for (int i = numDigit - 1; i >= 0; --i)
            if(digits[i] != x.digits[i]) return COMPARE(digits[
                 i], x.digits[i]);
        return 0;
    }

    #define DEF_OPER(o) bool operator o (const Bignum &x) const
          { return compare(x) o 0; }
    DEF_OPER(<) DEF_OPER(>) DEF_OPER(>=) DEF_OPER(<=) DEF_OPER
         (==) DEF_OPER(!=)
    #undef DEF_OPER

    string toString(void) const {
        string res;
        for (int i = 0; i < numDigit; ++i) {
            int tmp = digits[i];
            for (int j = 0; j < 9; ++j) { res.push_back('0' +
                 tmp % 10); tmp /= 10; }
        }
        while (sz(res) > 1 && res.back() == '0') res.pop_back()
            ;
        reverse(res.begin(), res.end());
        return res;
    }
};
```

10.1 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).

- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

10.2 Optimization tricks

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

10.2.1 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.

10.2.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.

- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.

- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

FastMod.h

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

d41d8c, 8 lines

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

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: `./a.out < input.txt`
Time: About 5x as fast as cin/scanf.

d41d8c, 17 lines

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

int readInt() {
  int a, c;
  while ((a = gc()) < 40);
  if (a == '-') return -readInt();
  while ((c = gc()) >= 48) a = a * 10 + c - 480;
  return a - 48;
}
```