VNUHCM University of Science

HCMUS-HLD

Nguyen Anh Dung
Phan Binh Nguyen Lam
Le Huu Hoa

2025-10-11

1 Contest 1
.bashrc 1
hash.sh 1
note.txt 1

2 Mathematics 1
2.1 Equations 1
2.2 Recurrences 1
2.3 Trigonometry 1
2.4 Geometry 1
2.5 Derivatives/Integrals 1
2.6 Sums 1
2.7 Series 1
2.8 Probability theory 2
2.9 Markov chains 2
2.10 Bézout's identity 2
2.11 Mobius Function 2
InverseModulo.h 2
PolyInterpolate.h 2
BerlekampMassey.h 3
LinearRecurrence.h 3
GaussianElimination.h 3
ModInt.h 3
Lagrange.h 4
FFTMOD.h 4
XorBasis.h 4
BernoulliNumber.h 4
LinearSieve.h 5

3 Combinatorial 5
3.1 Permutations 5
3.2 Partitions and subsets 5
3.3 General purpose numbers 5

4 Data structure 6
LiChaoTreeArray.h 6
LiChaoTreePtr.h 6
DynamicConvexHull.h 6
SplayTree.h 6
OrderTree.h 7
HashMap.h 7
BitsetTree.h 7
ManhattanSpanningTree.h 7
PersistentIT.h 8
ConvexHullTrick.h 8

5 Graph 8
TwoSat.h 8
TwoSatPtr.h 8
HLD.h 9
Centroid.h 9
BlockCutTree.h 9
OnlineBridgeCounting.h 10
CheckBridgeArticulation.h 10

6 Various 10

7 MISC 10
yCombinator.h 10
BigNum.h 10
7.1 Debugging tricks 11
7.2 Optimization tricks 11
FastMod.h 11
FastInput.h 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 $x=-b/2a$.

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

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 $a_n=c_1a_{n-1}+\cdots+c_ka_{n-k}$, and $r_1,\dots,r_k$ are distinct roots of $x^k-c_1x^{k-1}-\cdots-c_k$, there are $d_1,\dots,d_k$ s.t.

$$a_n=d_1r_1^n+\cdots+d_kr_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n=(d_1n+d_2)r^n$.

2.3 Trigonometry

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

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

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a,b,c

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

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

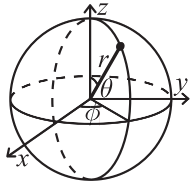
2.4.2 Quadrilaterals

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

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

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



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

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

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

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) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

First success distribution

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

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

Poisson distribution

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

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

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

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)

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

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

InverseModulo.h

d41d8c, 11 lines

```cpp
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: $\mathcal{O}(n^2)$

d41d8c, 13 lines

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

d41d8c, 20 lines

```cpp
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: $\mathcal{O}(n^2 \log k)$

d41d8c, 26 lines

```cpp
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: $\mathcal{O}(N^3)$

d41d8c, 44 lines

```cpp
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

```cpp
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;
    }
```

```cpp
    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

```cpp
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

```cpp
// 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 $n = n_k p^k + \dots + n_1 p + n_0$ and $m = m_k p^k + \dots + m_1 p + m_0$. 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 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

3.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

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

3.3.6 Labeled unrooted trees

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

3.3.7 Catalan numbers

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

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

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

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

Data structure (4)

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 (5)

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
}
```

Various (6)

MISC (7)

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 $\mathcal{O}((N/9)^2)$ 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;
    }
};
```

7.1 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`).

7.2 Optimization tricks

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

7.2.1 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.

7.2.2 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;
}
```