University of Wrocław

Pokor Fanclub

Olaf Surgut, Rafał Mańczyk, Krzysztof Olejnik

AMPPZ 2024

15-17 November

Contest (1)

template.cpp 50 lines

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
#define ll long long
#define ld long double
#define endl '\n'
#define st first
#define nd second
#define pb push_back
#define eb emplace_back
#define sz(x) (int)(x).size()
#define all(x) begin(x),end(x)
#define FOR(i,l,r) for(int i=(l);i<=(r);i++)
#define ROF(i,r,l) for(int i=(r);i>=(l);i--)
auto& operator<<(auto &o, pair<auto, auto> p) {
  return o << "(" << p.st << ", " << p.nd << ")";}
auto operator<<(auto &o, auto x)->decltype(end(x), o) {
  o << "{"; int i=0; for (auto e : x) o << ","+!i++ << e;
  return o << "}"; }
#ifdef LOCAL
#define debug(x...) cerr << "[" #x "]: ", [](auto...$) { \
  ((cerr << $ << "; "),...) << endl; }(x)
#else
#define debug(...)
#endif
#define rep(i,a,b) for(int i = a; i < (b); i++)
using pii = pair<int, int>;
using vi = vector<int>;
const int inf = 1e9+7;
void solve() {
}
signed main() {
  cin.tie(0)->sync_with_stdio(0);
  int tt = 1;
  cin >> tt;
  while (tt--)
    solve();
  return 0;
}
```

pokorrc 31 lines

```
# path to the bits/stdc++.h for fast compilation
# $ g++ a.cpp -H 2> 2>(head)
# Slow compilation
c() {
  g++ $1.cpp -o $1 -std=gnu20 -Wall -Wshadow -Wextra \
  -fsanitize=undefined,address -ggdb3 -DLOCAL -I$HOME/bits
}
# Quick compilation
nc() {
  g++ $1.cpp -o $1 -std=gnu20 -O2 -g -static
}
# Time and memory usage
r() { command time -f "%Us %M KB" ./$1; }
```

```
# Hash of a file, ignores all whitespaces and comments
haszuj() {
  cpp -dD -P -fprocessed $1.cpp | tr -d '[:space:]' | \
  md5sum | cut -c-6
}
# Tester script given brut and gen in the folder
testuj() {
  for ((i=0;;i++)); do
    ./gen > $1.test
    diff -bwq <(./$1 < $1.test) <(./brute < $1.test) || break
    echo "OK $i"
  done
}
```

.vimrc 2 lines

```
syntax on
set nu hls is ts=4 si sw=4
```

hash.sh 3 lines

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

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 $x = -b/2a$.

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

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 $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, 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

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two): $s_a = \sqrt{bc\left[1 - \left(\frac{a}{b + c}\right)^2\right]}$

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

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

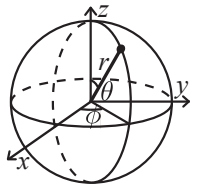
Law of tangents: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

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 $180°$,
$ac + bd = ef$, and $A = \sqrt{(p - a)(p - b)(p - c)(p - d)}$.

2f4.1ac Spherical coordinates



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

2.5 Derivatives/Integrals

$$\int \sqrt{a^2+x^2}dx = \frac{x}{2}\sqrt{a^2+x^2} + \frac{a^2}{2}\ln(x+\sqrt{a^2+x^2})$$

$$\int \sqrt{a^2-x^2}dx = \frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{|a|}$$

$$\int \frac{dx}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{|a|} = -\arccos\frac{x}{|a|}$$

$$\int \frac{dx}{\sqrt{a^2+x^2}} = \ln(x+\sqrt{a^2+x^2})$$

$$\text{Sub } s = \tan(x/2) \text{ to get: } dx = \frac{2\,ds}{1+s^2}$$

$$\sin x = \frac{2s}{1+s^2},\ \cos x = \frac{1-s^2}{1+s^2}$$

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

$$\text{(Integration by parts) } \int \tan ax = -\frac{\ln|\cos ax|}{a}$$

$$\int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\text{erf}(x),\quad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

$$\frac{d}{dx}\tan x = 1+\tan^2 x,\quad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\text{Curve length: } \int_a^b \sqrt{1+(f'(x))^2}dx$$

$$\text{When } X(t), Y(t): \int_a^b \sqrt{(X'(t))^2+(Y'(t))^2}dt$$

$$\text{Solid of revolution vol: } \pi\int_a^b (f(x))^2dx$$

$$\text{Surface area: } 2\pi\int_a^b |f(x)|\sqrt{1+(f'(x))^2}dx$$

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 $\text{Bin}(n,p)$, $n = 1,2,\ldots,\ 0 \leq p \leq 1$.

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

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

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

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

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

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

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 Other theorems

Dirichelt's approximation theorem: For any real r and N, with $1 \leq N$, there exist integers p and q such that $1 \leq q \leq N$ and

$$|q\alpha - p| \leq \frac{1}{\lfloor N \rfloor + 1} < \frac{1}{N}$$

$$0 < |\alpha - \frac{p}{q}| < \frac{1}{q^2}$$

Stirling approximation :
$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n e^{\frac{1}{12n}}$

Stirling Numbers(permutation $|P| = n$ with k cycles):
$S(n,k)$ = coefficient of x^k in $\Pi_{i=0}^{n-1}(x+i)$

Stirling Numbers(Partition n elements into k non-empty set):
$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$

Pick's Theorem : $A = i + b/2 - 1$
A: Area, i: grid number in the inner, b: grid number on the side

$C_n = \binom{2n}{n}/(n+1)$
$C_n^{n+m} - C_{n+1}^{n+m} = (m+n)!\frac{n-m+1}{n+1} \quad for \quad n \geq m$
$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$
$C_0 = 1 \quad and \quad C_{n+1} = 2(\frac{2n+1}{n+2})C_n$
$C_0 = 1 \quad and \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \quad for \quad n \geq 0$

Euler Characteristic:
planar graph: $V - E + F - C = 1$
convex polyhedron: $V - E + F = 2$
V, E, F, C: number of vertices, edges, faces(regions), and components

Kirchhoff's theorem:
$A_{ii} = deg(i), A_{ij} = (i,j) \in E \ ? -1 : 0$, Deleting any one row, one column, and cal the det(A)

Polya' theorem (c is number of color, m is the number of cycle size):
$(\sum_{i=1}^{m} c^{gcd(i,m)})/m$

Wilson's theorem :
$(p-1)! \equiv -1 (mod\ p)$

Euler's totient function:
$A^{B^C} mod\ p = pow(A, pow(B, C, p-1)) mod\ p$

Data structures (3)

PBDS.h

Description: Policy Based Data Structures 460200, 17 lines

```
// Order Statistics Tree: Caution: Not a multiset!
#include <bits/extc++.h>
using namespace __gnu_pbds;
template <class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
Tree<int> t, t2;
auto it = t.insert(10).first; // it == t.upper_bound(9);
t.order_of_key(10); // # of entries strictly smaller than key
t.join(t2); // fast only if max(T) < min(T2) or min(T) > max(T2)
// Hash Table: faster but can lead to MLE (1.5x worse performance),
    initial capacity must = 2^k
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); }
};
gp_hash_table<ll, int, chash> h({}, {}, {}, {}, {1 << 16}); //
    cc_hash_table also exists if needed
```

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided). d77092, 7 lines

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

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.
Usage: Node* tr = new Node(v, 0, sz(v));
Time: $\mathcal{O}(\log N)$.
"../various/BumpAllocator.h" 34ecf5, 50 lines

```
const int inf = 1e9;
struct Node {
  Node *l = 0, *r = 0;
  int lo, hi, mset = inf, madd = 0, val = -inf;
  Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
  Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
    if (lo + 1 < hi) {
      int mid = lo + (hi - lo)/2;
      l = new Node(v, lo, mid); r = new Node(v, mid, hi);
      val = max(l->val, r->val);
    }
    else val = v[lo];
  }
  int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
  }
  void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
      push(), l->set(L, R, x), r->set(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
      if (mset != inf) mset += x;
      else madd += x;
      val += x;
    }
    else {
      push(), l->add(L, R, x), r->add(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void push() {
    if (!l) {
      int mid = lo + (hi - lo)/2;
      l = new Node(lo, mid); r = new Node(mid, hi);
    }
    if (mset != inf)
      l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
      l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
  }
};
```

HilbertOrder.h

Description: Useful speed up for MoQueries.
Time: $\mathcal{O}(\log N)$ 8ce340, 14 lines

```
ll hilbertOrder(int x, int y, int pow = 20, int rotate = 0) {
  if(pow == 0) return 0;
  int hpow = 1 << (pow - 1);
  int seg = x < hpow ? (y < hpow ? 0:3) : (y < hpow ? 1:2);
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
  ll subSquareSize = 1ll << (pow * 2 - 2);
  ll ans = seg * subSquareSize;
  ll add = hilbertOrder(nx, ny, pow - 1, nrot);
  ans += seg == 1 || seg == 2 ? add : (subSquareSize - add-1);
  return ans;
}
```

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: int t = uf.time(); ...; uf.rollback(t);
Time: $\mathcal{O}(\log(N))$ de4ad0, 21 lines

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

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$ 8ec1c7, 30 lines

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

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$, increase works in $\mathcal{O}(\log N \log U)$ amortized 949abd, 113 lines

```
struct Treap {
  struct Node {
    int ch[2] = {0, 0}, size = 0;
    int val = 0, mini = 1e9, sum = 0; // Subtree aggregates
    bool flip = 0; int add = 0;      // Lazy tags
    Node() {}
    Node(int v) : size(1), val(v), mini(v), sum(v) {}
  };
  vector<Node> t;
  Treap() : t(1) {}
  void pull(int v) {
    auto [l, r] = t[v].ch;
    t[v].size = t[l].size + 1 + t[r].size;
    t[v].mini = min({t[l].mini, t[v].val, t[r].mini});
    t[v].sum = t[l].sum + t[v].val + t[r].sum;
  }
  int apply(int v, bool flip, int add) {
```

```
  if(!v) return 0;
  // t.pb(t[v]), v = SZ(t) - 1;       // <- persistency
  if(flip) t[v].flip ^= 1, swap(t[v].ch[0], t[v].ch[1]);
  t[v].val += add; t[v].mini += add;
  t[v].sum += add * t[v].size;
  t[v].add += add;
  return v;
}
void push(int v) {
  FOR(i, 0, 2)
    t[v].ch[i] = apply(t[v].ch[i], t[v].flip, t[v].add);
  t[v].add = t[v].flip = 0;
}
int rank(int v, int u) {
  static mt19937 gen(2137);
  return int(gen() % (t[v].size + t[u].size)) < t[v].size;
}
pii split(int v, int k) {
  if(!v) return {0, 0};
  push(v);
  auto [l, r] = t[v].ch;
  if(k <= t[l].size) {
  // if(k <= t[v].val) {          // <- by values
    auto [p, q] = split(l, k);
    t[v].ch[0] = q, pull(v);
    return {p, v};
  }
  else {
    auto [p, q] = split(r, k - t[l].size - 1);
    // auto [p, q] = split(r, k);   // <- by values
    t[v].ch[1] = p, pull(v);
    return {v, q};
  }
}
int merge(int v, int u) {
  if(!v || !u) return v ^ u;
  push(v), push(u);
  if(rank(v, u)) {
    t[v].ch[1] = merge(t[v].ch[1], u);
    return pull(v), v;
  }
  else {
    t[u].ch[0] = merge(v, t[u].ch[0]);
    return pull(u), u;
  }
}
void insert(int &v, int pos, int val) {
  // if(v) t.pb(t[v]), v = SZ(t) - 1;   // <- persistency
  auto [p, q] = split(v, pos);
  t.pb(Node(val)); int u = SZ(t) - 1;
  // t.pb(Node(pos)); int u = SZ(t) - 1;  // <- by values
  v = merge(merge(p, u), q);
}
void erase(int &v, int l, int r) {
  // if(v) t.pb(t[v]), v = SZ(t) - 1;   // <- persistency
  auto [p, q] = split(v, l);
  auto [u, s] = split(q, r - l + 1);
  // auto [u, s] = split(q, r + 1);   // <- by values
  v = merge(p, s);
}
void modify(int &v, int l, int r, bool flip, int add) {
  // if(v) t.pb(t[v]), v = SZ(t) - 1;   // <- persistency
  auto [p, q] = split(v, l);
  auto [u, s] = split(q, r - l + 1);
  // auto [u, s] = split(q, r + 1);   // <- by values
  u = apply(u, flip, add);
  v = merge(merge(p, u), s);
}
pii get(int &v, int l, int r) {
  // if(v) t.pb(t[v]), v = SZ(t) - 1;   // <- persistency
  auto [p, q] = split(v, l);
  auto [u, s] = split(q, r - l + 1);
  // auto [u, s] = split(q, r + 1);   // <- by values
  int mini = t[u].mini, sum = t[u].sum;
  v = merge(merge(p, u), s);
  return {mini, sum};
}
// only when by values
int join(int v, int u) {
  if(!v || !u) return v ^ u;
  if(!rank(v, u)) swap(v, u);
  auto [p, q] = split(u, t[v].val);
  push(v);
  t[v].ch[0] = join(t[v].ch[0], p);
  t[v].ch[1] = join(t[v].ch[1], q);
  return pull(v), v;
}
// only when by values, persistency destroys complexity
void increase(int &v, int l, int r, int increase) {
  // if(v) t.pb(t[v]), v = SZ(t) - 1; // <- persistency
  auto [p, q] = split(v, l);
  auto [u, s] = split(q, r + 1);
  u = apply(u, 0, increase);
  v = join(merge(p, s), u);
 }
};
```

LiChao.h

Description: Extended Li Chao tree (segment tree for functions). Let F be a family of functions closed under function addition, such that for every $f \neq g$ from the family F there exists x such that $f(z) \leq g(z)$ for $z \leq x$ else $f(z) \geq g(z)$ or the other way around (intersect at one point). Typically F is the family of linear functions. DS maintains a sequence $c_0, c_1 \ldots c_{n-1}$ under operations max, add. b88a40, 74 lines

```
struct LiChao {
  struct Func {
    ll a, b; // a*x + b
    // Evaluate function in point x
    ll operator()(ll x) const { return a*x+b; }
    Func operator+(Func r) const {
      return {a+r.a, b+r.b};
    } // Sum of two functions
  }; // ID_ADD/MAX neutral elements for add/max
  static constexpr Func ID_ADD{0, 0};
  static constexpr Func ID_MAX{0, ll(-1e9)};
  vector<Func> val, lazy;
  int len;
  // Initialize tree for n elements; time: O(n)
  LiChao(int n = 0) {
    for (len = 1; len < n; len *= 2);
    val.resize(len*2, ID_MAX);
    lazy.resize(len*2, ID_ADD);
  }
  void push(int i) {
    if (i < len) rep(j, 2) {
      lazy[i*2+j] = lazy[i*2+j] + lazy[i];
      val[i*2+j] = val[i*2+j] + lazy[i];
    }
    lazy[i] = ID_ADD;
  } // For each x in [vb;ve)
    // set c[x] = max(c[x], f(x));
    // time: O(log^2 n) in general case,
    //       O(log n) if [vb;ve) = [0;len)
  void max(int vb, int ve, Func f,
           int i = 1, int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve || i >= len*2)
      return;
    int m = (b+e) / 2;
    push(i);
    if (b >= vb && e <= ve) {
      auto& g = val[i];
      if (g(m) < f(m)) swap(g, f);
      if (g(b) < f(b))
        max(vb, ve, f, i*2, b, m);
      else
        max(vb, ve, f, i*2+1, m, e);
    } else {
      max(vb, ve, f, i*2, b, m);
      max(vb, ve, f, i*2+1, m, e);
    }
  } // For each x in [vb;ve)
    // set c[x] = c[x] + f(x);
    // time: O(log^2 n) in general case,
    //       O(1) if [vb;ve) = [0;len)
  void add(int vb, int ve, Func f,
           int i = 1, int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve) return;
    if (b >= vb && e <= ve) {
      lazy[i] = lazy[i] + f;
      val[i] = val[i] + f;
    } else {
      int m = (b+e) / 2;
      push(i);
      max(b, m, val[i], i*2, b, m);
      max(m, e, val[i], i*2+1, m, e);
      val[i] = ID_MAX;
      add(vb, ve, f, i*2, b, m);
      add(vb, ve, f, i*2+1, m, e);
    }
  } // Get value of c[x]; time: O(log n)
  auto query(int x) {
    int i = x+len;
    auto ret = val[i](x);
    while (i /= 2)
      ret = ::max(ret+lazy[i](x), val[i](x));
    return ret; } };
```

PersistentSegmentTree.h

Description: Persistent structures use a lot of memory in general, so remember to optimise the code whenever possible. Push function is responsible for creating copies of children, so whenever we enter a vertex (either add or query) it's already a fresh copy. When there is no lazy, it's better to modify this approach, and only create a vertex when we really visit it, so we can save memory.

Time: $\mathcal{O}(\log N)$ per operation c5c0ae, 58 lines

```
struct PersistentTree {
  using T = ll; // persistency needs a lot of memory
  struct Node { // so make sure to make it efficient
    int ch[2] = {-1, -1}; T sum, mini, add;
    Node() : sum(0), mini(0), add(0) {} // start / neutral
    Node(T val) : sum(val), mini(val), add(0) {}
    void apply(T val, int len) {
      sum += val * len; mini += val; add += val;
    }
    void pull(Node &l, Node &r) {
      sum = l.sum + r.sum; mini = min(l.mini, r.mini);
    }
  };
  vector<Node> t; int max_n;
  PersistentTree(int _max_n) : max_n(_max_n) {}
  PersistentTree(vector<T> vals) : max_n(SZ(vals) - 1) {
    function<int(int, int)> build = [&](int l, int r) {
      if(l == r) { t.pb(Node(vals[l])); return SZ(t)-1; }
      int mid = (l + r) / 2; Node a;
      a.ch[0] = build(l, mid), a.ch[1] = build(mid + 1, r);
      a.pull(t[a.ch[0]], t[a.ch[1]]);
      t.pb(a); return SZ(t) - 1;
    };
    build(0, max_n);
  }
  int copy(int v) { t.pb(~v ? t[v]:Node()); return SZ(t)-1; }
  void push(int v, int len) {
    FOR(i, 0, 2) {
      t[v].ch[i] = copy(t[v].ch[i]);
      t[t[v].ch[i]].apply(t[v].add, len / 2);
    }
    t[v].add = 0;
  }
  void add(int v, int l, int r, T val, int le, int re) {
    if(re < l || r < le) return;
    if(l <= le && re <= r) return t[v].apply(val, re-le + 1);
    push(v, re - le + 1); int mid = (le + re) / 2;
    add(t[v].ch[0], l, r, val, le, mid);
    add(t[v].ch[1], l, r, val, mid + 1, re);
    t[v].pull(t[t[v].ch[0]], t[t[v].ch[1]]);
  }
  Node query(int v, int l, int r, int le, int re) {
    if(re < l || r < le) return Node();
    if(l <= le && re <= r) return t[v];
    push(v, re - le + 1); int mid = (le + re) / 2;
    Node a = query(t[v].ch[0], l, r, le, mid);
    Node b = query(t[v].ch[1], l, r, mid + 1, re);
    Node res; res.pull(a, b); return res;
  }
  void add(int &v, int l, int r, T val) {
    v = copy(v); add(v, l, r, val, 0, max_n);
  }
  Node query(int v, int l, int r) {
    int sz = SZ(t);
    v = copy(v); Node res = query(v, l, r, 0, max_n);
    t.resize(sz); return res;
  }
};
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.

Time: Both operations are $\mathcal{O}(\log N)$. e62fac, 22 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
```

```
    }
    return pos;
  }
};
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).

Time: $\mathcal{O}\left(\log^2 N\right)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

"FenwickTree.h" 157f07, 22 lines

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

WaveletTree.h

Description: Wavelet tree. Supports fast kth order statistics on ranges (no updates).

Time: $\mathcal{O}(\log N)$ 587095, 35 lines

```
struct WaveletTree {
  vector<vi> seq, left;
  int len;
  WaveletTree() {}
  // time and space: O((n+maxVal) log maxVal)
  // Values are expected to be in [0;maxVal).
  WaveletTree(const vi& elems, int maxVal) {
    for (len = 1; len < maxVal; len *= 2);
    seq.resize(len*2); left.resize(len*2);
    seq[1] = elems; build(1, 0, len);
  }
  void build(int i, int b, int e) {
    if (i >= len) return;
    int m = (b+e) / 2;
    left[i].pb(0);
    for(auto &x : seq[i]) {
      left[i].pb(left[i].back() + (x < m));
      seq[i*2 + (x >= m)].pb(x);
    }
    build(i*2, b, m); build(i*2+1, m, e);
  } // Find k-th (0 indexed) smallest element in [begin;end)
  int kth(int begin, int end, int k, int i=1) {
    if (i >= len) return seq[i][0];
    int x = left[i][begin], y = left[i][end];
    if (k < y-x) return kth(x, y, k, i*2);
    return kth(begin-x, end-y, k-y+x, i*2+1);
  } // Count number of elements >= vb and < ve
  int count(int begin, int end, int vb, int ve, int i = 1, int b = 0, int
        e = -1) {
    if (e < 0) e = len;
    if (b >= ve || vb >= e) return 0;
    if (b >= vb && e <= ve) return end-begin;
    int m = (b+e) / 2; int x = left[i][begin], y = left[i][end];
    return count(x, y, vb, ve, i*2, b, m) + count(begin-x, end-y, vb, ve,
        i*2+1,m,e);
  }
};
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.

Usage: RMQ rmq(values);
rmq.query(inclusive, exclusive);

Time: $\mathcal{O}(|V| \log |V| + Q)$ 510c32, 16 lines

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

SlopeTrick.h

Description: Linear functions container. Not tested!

Time: $\mathcal{O}(\log N)$ per operation c81b01, 63 lines

```
struct SlopeTrick {
  const ll INF = 3e18;
  ll min_f, add_l, add_r;
  priority_queue<ll> L;
  priority_queue<ll, vector<ll>, greater<>> R;
  void push_R(ll a) { R.push(a - add_r); }
  ll top_R() const { return SZ(R) ? R.top() + add_r : INF; }
  ll pop_R() {
    ll val = top_R(); if(SZ(R)) R.pop();
    return val;
  }
  void push_L(ll a) { L.push(a - add_l); }
  ll top_L() const { return SZ(L) ? L.top() + add_l : -INF; }
  ll pop_L() {
    ll val = top_L(); if(SZ(L)) L.pop();
    return val;
  }
  int size() { return SZ(L) + SZ(R); }
  // use only functions below!
  SlopeTrick() : min_f(0), add_l(0), add_r(0) {}
  struct Query { ll lx, rx, min_f; };
  // return      min f(x)
  Query query() const { return Query(top_L(),top_R(),min_f); }
  //          f(x) += a
  void add_all(ll a) { min_f += a; }
  // add \_    f(x) += max(a - x, 0)
  void add_a_minus_x(ll a) {
    min_f += max(0ll, a - top_R()); push_R(a); push_L(pop_R());
  }
  // add _/    f(x) += max(x - a, 0)
  void add_x_minus_a(ll a) {
    min_f += max(0ll, top_L() - a); push_L(a); push_R(pop_L());
  }
  // add \/    f(x) += abs(x - a)
  void add_abs(ll a) { add_a_minus_x(a); add_x_minus_a(a); }
  // \/ -> \_    f_{new} (x) = min f(y) (y <= x)
  void clear_right() { while(SZ(R)) R.pop(); }
  // \/ -> _/    f_{new} (x) = min f(y) (y >= x)
  void clear_left() { while(SZ(L)) L.pop(); }
  // \/ -> \_/  f_{new} (x) = min f(y) (x-b <= y <= x-a)
  void shift(ll a, ll b) {
    assert(a <= b); add_l += a; add_r += b;
  }
  // \/. -> ./\ f_{new} (x) = f(x - a)
  void shift(ll a) { shift(a, a); }
  // L, R is destroyed
  ll get(ll x) {
    ll ret = min_f;
    while(SZ(L)) ret += max(0ll, pop_L() - x);
    while(SZ(R)) ret += max(0ll, x - pop_R());
    return ret;
  }
  void merge(SlopeTrick &o) {
    if(SZ(o) > size()) {
      swap(o.L, L); swap(o.R, R);
      swap(o.add_l, add_l); swap(o.add_r, add_r);
      swap(o.min_f, min_f);
    }
    while(SZ(o.R)) add_x_minus_a(o.pop_R());
    while(SZ(o.L)) add_a_minus_x(o.pop_L());
    min_f += o.min_f;
  }
};
```

MoQueries.h

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change step to add/remove the edge (a, c) and remove the initial add call (but keep in).

Time: $\mathcal{O}\left(N\sqrt{Q}\right)$ a12ef4, 49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer
vi mo(vector<pii> Q) {
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) {
    pii q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
  int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) -> void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) rep(end,0,2) {
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                  else { add(c, end); in[c] = 1; } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
  return res;
}
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

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

PolyRoots.h

Description: Finds the real roots to a polynomial.

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

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

"Polynomial.h" b00bfe, 23 lines

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

```
    rep(it,0,60) { // while (h - l > 1e-8)
      double m = (l + h) / 2, f = p(m);
      if ((f <= 0) ^ sign) l = m;
      else h = m;
    }
    ret.push_back((l + h) / 2);
  }
 }
 return ret;
}
```

PolyInterpolate.h

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

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

08bf48, 13 lines

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

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

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

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;
  ll b = 1;
  rep(i,0,n) { ++m;
    ll d = s[i] % mod;
    rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C; ll coef = d * modpow(b, mod-2) % mod;
    rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }
  C.resize(L + 1); C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots \geq n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.

Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`

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

f4e444, 26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);
  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i,0,n+1) rep(j,0,n+1)
      res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };
  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }
  ll res = 0;
  rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

4.2 Optimization

Integrate.h

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

4756fc, 7 lines

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

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.

Usage: `double sphereVolume = quad(-1, 1, [](double x) {`
`return quad(-1, 1, [&](double y) {`
`return quad(-1, 1, [&](double z) {`
`return x*x + y*y + z*z < 1; });});});`

92dd79, 15 lines

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
template <class F>
d rec(F& f, d a, d b, d eps, d S) {
  d c = (a + b) / 2;
  d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
  if (abs(T - S) <= 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
  return rec(f, a, b, eps, S(a, b));
}
```

Simplex.h

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

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

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

aa8530, 68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;
  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }
  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }
  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }
  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

4.3 Matrices

Determinant.h

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

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

bd5cec, 15 lines

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

IntDeterminant.h

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

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

3313dc, 18 lines

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

SolveLinear.h

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

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

44c9ab, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;
int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
```

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

SolveLinear2.h

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

"SolveLinear.h" 08e495, 7 lines

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

SolveLinearBinary.h

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

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

```
typedef bitset<1000> bs;
int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }
  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h

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

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

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

MatrixInverse-mod.h

Description: Invert matrix A modulo a prime. Returns rank; result is stored in A unless singular (rank < n). For prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1})$ (mod p^k) where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.

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

"../number-theory/ModPow.h" 0b7b13, 37 lines

```
int matInv(vector<vector<ll>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<ll>> tmp(n, vector<ll>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n) if (A[j][k]) {
      r = j; c = k; goto found;
    }
    return i;
found:
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    ll v = modpow(A[i][i], mod - 2);
    rep(j,i+1,n) {
      ll f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
      rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
  }
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    ll v = A[j][i];
    rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod;
  }
  rep(i,0,n) rep(j,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0)*mod;
  return n;
}
```

Tridiagonal.h

Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

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

This is useful for solving problems on the type

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

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

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

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

Time: $\mathcal{O}\left(N\right)$ 8f9fa8, 26 lines

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

4.4 Fourier transforms

FastFourierTransform.h

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

Time: $\mathcal{O}\left(N \log N\right)$ with $N = |A| + |B|$ ($\sim$1s for $N = 2^{22}$) 00ced6, 35 lines

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

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.

Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

"FastFourierTransform.h" b82773, 22 lines

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

NumberTheoreticTransform.h

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

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

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

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

FastSubsetTransform.h

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

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

464cf3, 16 lines

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

Number theory (5)

5.1 Modular arithmetic

ModInverse.h

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

6f684f, 3 lines

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

ModPow.h

b83e45, 8 lines

```
const ll mod = 1000000007; // faster if const
ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModLog.h

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

Time: $\mathcal{O}\left(\sqrt{m}\right)$

c040b8, 11 lines

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

ModSum.h

Description: Sums of mod'ed arithmetic progressions.

modsum(to, c, k, m) = $\sum_{i=0}^{\text{to}-1} (ki + c)\%m$. divsum is similar but for floored division.

Time: $\log(m)$, with a large constant.

5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}
ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$.

Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

bbbd8f, 11 lines

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

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).

Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModPow.h" 19a793, 24 lines

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

5.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.

Time: LIM=1e9 $\approx$ 1.5s

6b2912, 20 lines

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

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.

Time: 7 times the complexity of $a^b \mod c$.

"ModMulLL.h" 60dcd1, 12 lines

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

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

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

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n) + i; };
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
```

```
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

Pell.h

Description: $O(\log n)$, `pell(n)` oblicza rozwiązanie fundamentalne $x^2 - ny^2 = 1$, zwraca $(0, 0)$ jeżeli nie istnieje. 930a36, 27 lines

```
pair<LL, LL> pell(LL n) {
  LL s = LL(sqrtl(n));
  if (s * s == n) return {0, 0};
  LL m = 0, d = 1, a = s;
  __int128 num1 = 1, num2 = a, den1 = 0, den2 = 1;
  while (num2 * num2 - n * den2 * den2 != 1) {
    m = d * a - m;
    d = (n - m * m) / d;
    a = (s + m) / d;
    if (num2 > (1ll << 62) / a) return {0, 0};
    tie(num1, num2) = pair(num2, a * num2 + num1);
    tie(den1, den2) = pair(den2, a * den2 + den1);
  }
  return {num2, den2};
}
vector<pair<LL, LL>> all_pell(LL n, LL limit) {
  auto [x0, y0] = pell(n);
  if (!x0) return {};
  vector<pair<LL, LL>> ret;
  __int128 x = x0, y = y0;
  while (x <= limit) {
    ret.emplace_back(x, y);
    if (y0 * y > (1ll << 62) / n) break;
    tie(x, y) = pair(x0 * x + n * y0 * y, x0 * y + y0 * x);
  }
  return ret;
}
```

Pi.h

Description: $O(n^{\frac{3}{4}})$, liczba liczb pierwszych na przedziale $[1, n]$. `Pi pi(n); pi.query(d);` musi zachodzic d dzieli n 5af6fc, 28 lines

```
struct Pi {
  vector<LL> w, dp;
  int id(LL v) {
    if (v <= w.back() / v)
      return int(v - 1);
    return ssize(w) - int(w.back() / v);
  }
  Pi(LL n) {
    for (LL i = 1; i * i <= n; ++i) {
      w.push_back(i);
      if (n / i != i)
        w.emplace_back(n / i);
    }
    sort(w.begin(), w.end());
    for (LL i : w)
      dp.emplace_back(i - 1);
    for (LL i = 1; (i + 1) * (i + 1) <= n; ++i) {
      if (dp[i] == dp[i - 1])
        continue;
      for (int j = ssize(w) - 1; w[j] >= (i + 1) * (i + 1); --j)
        dp[j] -= dp[id(w[j] / (i + 1))] - dp[i - 1];
    }
  }
  LL query(LL v) {
    assert(w.back() % v == 0);
    return dp[id(v)];
  }
};
```

FloorSum.h

Description: $O(\log a)$, liczy $\sum_{i=0}^{n-1} \left\lfloor \frac{a \cdot i + b}{c} \right\rfloor$. Działa dla $0 \le a, b < c$ oraz $1 \le c, n \le 10^9$. 78c6f7, 15 lines

```
LL floor_sum(LL n, LL a, LL b, LL c) {
  LL ans = 0;
  if (a >= c) {
    ans += (n - 1) * n * (a / c) / 2;
    a %= c;
  }
  if (b >= c) {
    ans += n * (b / c);
    b %= c;
  }
  LL d = (a * (n - 1) + b) / c;
  if (d == 0) return ans;
  ans += d * (n - 1) - floor_sum(d, c, c - b - 1, a);
  return ans;
}
```

5.3 Divisibility

euclid.h

Description: Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in __gcd instead. If a and b are coprime, then x is the inverse of $a \pmod b$. 33ba8f, 5 lines

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

CRT.h

Description: Chinese Remainder Theorem.
`crt(a, m, b, n)` computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$
"euclid.h" 04d93a, 7 lines

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

Min25.h

Description: Calculates prefsums of multiplicative function at each floor(N/i). keys[id(N/i)]=N/i. Remember about overflows. See example below.
Time: $\mathcal{O}\left(\frac{n^{3/4}}{\log n}\right)$ f4fd1a, 50 lines

```
vector<ll> global_primes; // global_primes[-1]>sqrt(N)
template<typename T>
struct Min25 {
  ll N;
  vector<ll> keys, primes;
  Min25(ll N_) : N(N_) {
    for (ll l = 1; l <= N; ++l)
      keys.pb(l = N / (N / l));
    for (int i = 0; global_primes[i] * global_primes[i] <= N; ++i)
      primes.pb(global_primes[i]);
  }
  ll id(ll x) {
    ll id = x < N / x ? x - 1 : sz(keys) - N / x;
    assert(keys[id] == x);
    return id;
  }
// f has to be TOTALLY multiplicative
// pref(x) is regular prefix sum function of f
  vector<T> overPrimes(auto pref) {
    vector<T> dp(sz(keys));
    rep(i, sz(keys))
      dp[i] = pref(keys[i]) - T(1);
    for (ll p : primes) {
      auto fp = dp[p - 1] - dp[p - 2];
      for (int i = sz(keys) - 1; i >= 0 && p * p <= keys[i]; --i)
        dp[i] = dp[i] - (dp[id(keys[i] / p)] - dp[p - 2]) * fp;
    }
    return dp;
  }
// dp are prefix sums of f over primes
// f(p, k, p**k) calculates f on primes powers
  void fullSum(vector<T> &dp, auto f) {
    for (ll p : primes | views::reverse) {
      for (int i = sz(keys) - 1; i >= 0 && p * p <= keys[i]; --i) {
        for (ll k = 1, q = p; q * p <= keys[i]; ++k, q *= p)
          dp[i] = dp[i] + f(p, k + 1, q * p) + f(p, k, q) * (dp[id(keys[i]
                / q)] - dp[p - 1]);
      }
    }
    for (auto &v : dp) v = v + T(1);
  }
};
vector<ll> exampleUsage(Min25<ll> &m) { // OVERFLOWS!
  auto primeCnt = m.overPrimes([](ll x){return x; });
  auto primeSum = m.overPrimes([](ll x){return x*(x+1)/2; });
  vector<ll> phi; rep(i, sz(m.keys))
    phi.pb(primeSum[i] - primeCnt[i]);
  m.fullSum(phi, [](int p,int k,ll pk){return pk-pk/p; });
  return phi; }
```

LinearSieve.h

Description: Usefull for computing values of multiplicative function and its prefix sums.
Time: $\mathcal{O}(N)$ 308681, 29 lines

```
struct LinearSieve {
  vector<bool> isComposite; vi prime, cnt;
  vector<ll> phi, prefPhi;
  ll dPhi(ll x, int p, int a) { // x / phi(p^(a-1)) * phi(p^a)
    return x * (a == 1 ? p - 1 : p);
  }
  LinearSieve(int n) : isComposite(n), cnt(n), phi(n) {
    if(n > 1) phi[1] = 1;
    FOR(i, 2, n) {
      if(!isComposite[i]) {
        prime.pb(i), cnt[i] = 1, phi[i] = dPhi(1, i, 1);
      }
      FOR(j, 0, SZ(prime)) {
        if(i * prime[j] >= n) break;
        isComposite[i * prime[j]] = 1;
        if(i % prime[j] == 0) {
          cnt[i * prime[j]] = cnt[i] + 1;
          phi[i*prime[j]] = dPhi(phi[i], prime[j], cnt[i]+1);
          break;
        }
        else {
          cnt[i * prime[j]] = 1;
          phi[i * prime[j]] = phi[i] * phi[prime[j]];
        }
      }
    }
    partial_sum(all(phi), back_inserter(prefPhi));
  }
};
```

5.4 Pisano period

$\pi(n)$ is a period of Fibbonacci sequence modulo n.
$\pi(nm) = \pi(n)\pi(m)$ for $n \perp m$, $\pi(p^k) = p^{k-1}\pi(p)$.

$$\pi(p) \begin{cases} = 3 & p = 2 \\ = 20 & p = 5 \\ \mid p - 1 & p \equiv_{10} \pm 1 \\ \mid 2(p+1) & p \equiv_{10} \pm 3 \end{cases}$$

$F_i \equiv_p -F_{i+p+1}$ for $p \equiv_{10} \pm 3$. $\pi(n) \le 4n$ for $n \neq 2 \cdot 5^r$.

5.5 Morbius

Niech $M(n) = \sum_{i=1}^{n} \mu(i)$. Można policzyć $M(n)$ w $O\left(n^{2/3} \cdot log(\text{smth})\right)$. Dla $u = n^{1/3}$, wystarczy spreprocesować M do $n^{2/3}$ i obliczyć $M(n)$ wzorem:

$$M(n) = M(u) - \sum_{m=1}^{u} \mu(m) \sum_{i=\lfloor \frac{u}{m} \rfloor + 1}^{\lfloor \frac{n}{m} \rfloor} M\left(\left\lfloor \frac{n}{mi} \right\rfloor\right).$$

$$\mu(p^k) = [k = 0] - [k = 1]$$

$$\sum_{d \mid n} \mu(d) = [n = 1]$$

$$\sum_{d \mid n} \phi(d) = n$$

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

5.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.7 Pythagorean Tree

Primitive Pythagorean triples form infinite ternary tree, where each triple occurs exactly once. Node is a column vector $(a, b, c = \sqrt{a^2 + b^2})$, root is $(3, 4, 5)$, and each child is given by a product of a parent and one of the:

$$\begin{bmatrix} 1 & -2 & 2 \\ 2 & -1 & 2 \\ 2 & -2 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \end{bmatrix}, \begin{bmatrix} -1 & 2 & 2 \\ -2 & 1 & 2 \\ -2 & 2 & 3 \end{bmatrix}$$

5.8 Primes

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

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

5.9 Estimates

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

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

5.10 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)

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

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

Define Dirichlet convolution as $f * g(n) = \sum_{d|n} f(d) g(n/d)$. Let $s_f(n) = \sum_{i=1}^{n} f(i)$. Then $s_f(n) g(1) = s_{f*g}(n) - \sum_{d=2}^{n} s_f(\lfloor \frac{n}{d} \rfloor) g(d)$.

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11		12		13		14		15	
$n!$	4.0e7		4.8e8		6.2e9		8.7e10		1.3e12	
n	16		17		20		25		30	
$n!$	2.1e13		3.6e14		2e18		2e25		3e32	
n	40		50		100		150		171	
$n!$	8e47		3e64		9e157		6e262		>DBL_MAX	

IntPerm.h

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

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

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

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

6.2 Partitions and subsets

6.2.1 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

6.2.2 Lucas' Theorem

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

6.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$. a0a312, 6 lines

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

6.3 General purpose numbers

6.3.1 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:

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

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$

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

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$

$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

6.3.3 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. $\pi(j) \ge j$, k j:s s.t. $\pi(j) > j$.

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 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,

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

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

6.3.7 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.

6.4 Other

DeBruijn.h

Description: Recursive FKM, given alphabet $[0, k)$ constructs cyclic string of length k^n that contains every length n string as substr. a7faa5, 13 lines

```
vi dseq(int k, int n) {
  if (k == 1) return {0};
  vi res, aux(n+1);
  function<void(int,int)> gen = [&](int t, int p) {
    if (t > n) { // consider lyndon word of len p
      if (n%p == 0) FOR(i,1,p+1) res.pb(aux[i]);
    } else {
      aux[t] = aux[t-p]; gen(t+1,p);
      FOR(i,aux[t-p]+1,k) aux[t] = i, gen(t+1,t);
    }
  };
  gen(1,1); return res;
}
```

NimProduct.h

Description: Nim Product. 9bba25, 18 lines

```
using ull = uint64_t;
ull _nimProd2[64][64];
ull nimProd2(int i, int j) {
  if (_nimProd2[i][j]) return _nimProd2[i][j];
  if ((i & j) == 0) return _nimProd2[i][j] = 1ull << (i|j);
  int a = (i&j) & -(i&j);
  return _nimProd2[i][j] = nimProd2(i ^ a, j)
    ^ nimProd2((i ^ a) | (a-1), (j ^ a) | (i & (a-1)));
}
ull nimProd(ull x, ull y) {
  ull res = 0;
  for (int i = 0; (x >> i) && i < 64; i++)
    if ((x >> i) & 1)
      for (int j = 0; (y >> j) && j < 64; j++)
        if ((y >> j) & 1)
          res ^= nimProd2(i, j);
  return res;
}
```

PermGroup.h

Description: Schreier-Sims lets you add a permutation to a group, count number of permutations in a group, test whether a permutation is a member of a group. Works well for $n \leq 15$, maybe for larger too. Construct PermGroup() and run order() to get order of the group.

Time: $\mathcal{O}\left(n^6\right)$ d6edf4, 55 lines

```
vi inv(vi v) { vi V(sz(v)); rep(i,sz(v)) V[v[i]]=i; return V; }
vi id(int n) { vi v(n); iota(all(v),0); return v; }
vi operator*(const vi& a, const vi& b) {
  vi c(sz(a)); rep(i,sz(a)) c[i] = a[b[i]];
  return c;
}
struct PermGroup {
  struct Group {
    vi flag;
    vector<vi> gen, sigma;
    // sigma[t][k] = t, sigma[t][x] = x if x > k
    Group(int n, int p) : flag(n), sigma(n) {
      flag[p] = 1; sigma[p] = id(n);
    }
  };
  int n = 0; vector<Group> g;
  PermGroup() {}
  bool check(const vi& cur, int k) {
    if (!k) return 1;
    int t = cur[k];
    return g[k].flag[t] ? check(inv(g[k].sigma[t])*cur,k-1) : 0;
  }
  void updateX(const vi& cur, int k) {
    int t = cur[k]; // if flag, fixes k -> k
    if (g[k].flag[t]) ins(inv(g[k].sigma[t])*cur,k-1);
    else {
      g[k].flag[t] = 1, g[k].sigma[t] = cur;
      for(auto x: g[k].gen)
        updateX(x*cur,k);
    }
  }
  void ins(const vi& cur, int k) {
    if (check(cur,k)) return;
    g[k].gen.pb(cur);
    rep(i,n) if (g[k].flag[i]) updateX(cur*g[k].sigma[i],k);
  }
  ll order(vector<vi> gen) {
    if(sz(gen) == 0) return 1;
    n = sz(gen[0]);
    rep(i,n) g.pb(Group(n,i));
    for(auto a: gen)
      ins(a, n-1); // insert perms into group one by one
    ll tot = 1; // watch out for overflows, can be up to n!
    rep(i,n) {
      int cnt = 0;
      rep(j,i+1) cnt += g[i].flag[j];
      tot *= cnt;
    }
    return tot;
  }
};
```

GrayCode.h

Description: Gray code: $\text{gray}(0), \ldots, \text{gray}(2^n - 1)$ - permutation in which each two consecutive (cyclically) numbers differ in exactly one bit. b4cc82, 6 lines

```
using ull = unsigned long long;
ull gray(ull i) { return i^i>>1; }
ull invg(ull i) { // i=invg(gray(i))=gray(invg(i))
  i^=i>>1; i^=i>>2; i^=i>>4;
  i^=i>>8; i^=i>>16; i^=i>>32; return i;
}
```

MatroidIntersection.h

Description: Given two matroids, finds the largest common independent set. For the color and graph matroids, this would be the largest forest where no two edges are the same color. Pass the matroid with more expensive operations to M1.

Time: $\sim\mathcal{O}\left(N^3\right)$ oracle calls, in practice $\sim\mathcal{O}\left(N^2\right)$ 858b54, 43 lines

```
template <class M1, class M2> struct MatroidIsect {
  M1 m1; M2 m2; int n; vi iset;
  MatroidIsect(M1 _m1, M2 _m2, int _n) :
    m1(_m1), m2(_m2), n(_n), iset(n + 1) {}
  vi solve() {
    per(i, 0, n-1) if(m1.check(i) && m2.check(i)) {
      iset[i] = 1, m1.add(i), m2.add(i);
    }
    while(augment());
    vi ans;
    FOR(i, 0, n) if(iset[i]) ans.pb(i);
    return ans;
  }
  bool augment() {
    vi frm(n, -1); queue<int> q({n}); // starts at dummy node
    auto fwdE = [&](int a) {
      vi ans; m1.clear();
      FOR(v, 0, n) if(iset[v] && v != a) m1.add(v);
      FOR(b, 0, n) if(!iset[b] && frm[b] == -1 && m1.check(b))
        ans.pb(b), frm[b] = a;
      return ans;
    };
    auto backE = [&](int b) {
      m2.clear();
      FOR(cas, 0, 2) FOR(v, 0, n)
        if((v == b || iset[v]) && (frm[v] == -1) == cas) {
          if(!m2.check(v))
            return cas ? q.push(v), frm[v] = b, v : -1;
          m2.add(v);
        }
      return n;
    };
    while(!q.empty()) {
      int a = q.front(), c; q.pop();
      for(int b: fwdE(a))
        while((c = backE(b)) >= 0) if(c == n) {
          while(b != n) iset[b] ^= 1, b = frm[b];
          return 1;
        }
    }
    return 0;
  }
};
```

MatroidOracles.h

Description: An oracle has 3 functions: check(int x): returns if current matroid can add x without becoming dependent; add(int x): adds an element to the matroid (guaranteed to never make it dependent); clear(): sets the matroid to the empty matroid.

Time: $\mathcal{O}(\log N)$ 0959d1, 42 lines

```
struct ColorMat {
  vi clr, bound, cnt; int max_bound = 0, offset = 0;
  ColorMat(vi _clr, vi _bound) : clr(_clr), bound(_bound) {
    cnt.resize(SZ(bound));
    if(SZ(bound)) max_bound = *max_element(all(bound));
  }
  void fix(int x) { cnt[clr[x]] = max(cnt[clr[x]], offset); }
  bool check(int x) {
    fix(x); return cnt[clr[x]] - offset < bound[clr[x]];
  }
  void add(int x) { fix(x); cnt[clr[x]]++; }
  void clear() { offset += max_bound; }
};
struct GraphicMat { // use normal UF to reduce memory
  vector<pii> edg; UF uf; vi vis;
  GraphicMat(vector<pii> _edg) : edg(_edg), uf(0) {
    int max_n = -1;
    for(auto &[x, y]: edg) max_n = max({max_n, x, y});
    uf = UF(max_n + 1);
  }
  bool check(int x) {
    return uf.find(edg[x].st) != uf.find(edg[x].nd);
  }
  void add(int x) { uf.join(edg[x].st,edg[x].nd); vis.pb(x); }
  void clear() {
    for(int x: vis) uf.e[edg[x].st] = uf.e[edg[x].nd] = -1;
    vis.clear();
  }
};
struct LinearMat {
  vector<ll> vals; set<ll, greater<ll>> basis, start;
  LinearMat(vector<ll> _vals, vector<ll> _s) : vals(_vals) {
    for(ll x: _s) basis.insert(red(x));
    start = basis;
  }
  ll red(ll x) { for(ll y: basis) x = min(x, x^y); return x; }
  bool check(int x) { return red(vals[x]); }
  void add(int x) { basis.insert(red(vals[x])); }
  void clear() { basis = start; }
};
```

Graph (7)

7.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max|w_i| < \sim 2^{63}$.

Time: $\mathcal{O}(VE)$ 830a8f, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };
void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
  nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });
  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
  rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] = \texttt{inf}$ if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, inf if no path, or -inf if the path goes through a negative-weight cycle.

Time: $\mathcal{O}\left(N^3\right)$ 531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) {
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

Shapes.h

Description: Counts all subgraph shapes with at most 4 edges. No multiedges / loops allowed;

Time: $\mathcal{O}\left(m\sqrt{m}\right)$ f62c2b, 39 lines

```
struct Shapes {
  ll tri = 0, rect = 0, path3 = 0, path4 = 0, star3 = 0, p = 0;
  __int128_t y = 0, star4 = 0;
  Shapes(vector<vi> &g) {
    int n = sz(g);
    vector<vi> h(n);
    vector<ll> f(n), c(n), s(n);
    rep(v, n) f[v] = (s[v] = sz(g[v])) * n + v;
    rep(v, n) {
      ll x = 0;
      star3 += s[v] * (s[v] - 1) * (s[v] - 2);
      star4 += __int128_t(s[v] - 1) * s[v] * (s[v] - 2) * (s[v] - 3);
      for (auto u : g[v]) {
        path4 += s[u] * x - x; x += s[u] - 1;
        y += (s[v] - 1) * (s[u] - 1) * (s[u] - 2) / 2;
        if (f[u] < f[v]) h[v].pb(u);
      }
    }
    rep(v, n) {
      for (int u : h[v])
        for (int w : g[u]) if (f[v] > f[w])
          rect += c[w] ++;
      for(int u : h[v]) {
        tri += c[u]; c[u] *= -1;
        path3 += (s[v] - 1) * (s[u] - 1);
        for(int w : g[u])
          if (c[w] < 0)
            p += s[v] + s[u] + s[w] - 6, c[w] ++;
          else if (c[w] > 0)
            c[w] --;
      }
    }
    path3 -= 3 * tri;
    y -= 2 * p;
    path4 -= 4 * rect + 2 * p + 3 * tri;
    star3 /= 6;
    star4 /= 24;
  }
};
```

SPFA.h

Description: SPFA with subtree erasure heuristic. Returns array of distances or empty array if negative cycle is reachable from source. par[v] = parent in shortest path tree

Time: $\mathcal{O}(VE)$ but fast on random bf4a3f, 33 lines

```
using Edge = pair<int, ll>;
vector<ll> spfa(vector<vector<Edge>>& G,
                vi& par, int src) {
  int n = sz(G); vi que, prv(n+1);
  iota(all(prv), 0); vi nxt = prv;
  vector<ll> dist(n, INT64_MAX);
  par.assign(n, -1);
  auto add = [&](int v, int p, ll d) {
    par[v] = p; dist[v] = d;
    prv[n] = nxt[prv[v] = prv[nxt[v] = n]] = v;
  };
  auto del = [&](int v) {
    nxt[prv[nxt[v]] = prv[v]] = nxt[v];
    prv[v] = nxt[v] = v;
  };
  for (add(src, -2, 0); nxt[n] != n;) {
    int v = nxt[n]; del(v);
    for (auto e : G[v]) {
      ll alt = dist[v] + e.y;
      if (alt < dist[e.x]) {
        que = {e.x};
        rep(i, sz(que)) {
          int w = que[i]; par[w] = -1;
          del(w);
          for (auto f : G[w])
            if (par[f.x] == w) que.pb(f.x);
        }
        if (par[v] == -1) return {};
        add(e.x, v, alt);
      }
    }
  }
  return dist; }
```

SubgraphsCounting.h

Description: Given simple undirected graph, counts all 3- and 4-edges subgraphs.

Time: $\mathcal{O}\left(E\sqrt{E}\right)$ 4ca57a, 50 lines

```
struct Subgraphs {
  int triangles3 = 0;
  ll stars3 = 0, paths3 = 0;
  ll ps4 = 0, rectangles4 = 0, paths4 = 0;
  __int128 ys4 = 0, stars4 = 0;
  Subgraphs(vector<vi> &G) {
    int n = SZ(G); vector<pii> deg(n);
    FOR(i, 0, n) deg[i] = {SZ(G[i]), i};
    sort(all(deg)); vi id(n), cnt(n);
    FOR(i, 0, n) id[deg[i].nd] = i;
    FOR(v, 0, n) {
      for(int u: G[v]) if(id[v] > id[u]) cnt[u] = 1;
      for(int u: G[v]) if(id[v] > id[u])
        for(int w: G[u]) if(id[w] > id[u] && cnt[w]) {
          triangles3++;
          for(int x: {v, u, w}) ps4 += SZ(G[x]) - 2;
        }
      for(int u: G[v]) if(id[v] > id[u]) cnt[u] = 0;
      for(int u: G[v]) if(id[v] > id[u])
        for(int w: G[u]) if(id[v] > id[w]) {
          rectangles4 += cnt[w]++;
        }
      for(int u: G[v]) if(id[v] > id[u])
        for(int w: G[u]) cnt[w] = 0;
    }
    paths3 = -3 * triangles3;
    FOR(v, 0, n) for(int u: G[v])
      if(v < u) paths3 += ll(SZ(G[v]) - 1) * (SZ(G[u]) - 1);
    ys4 = -2 * ps4;
    auto choose2 = [&](int x) { return x * ll(x - 1) / 2; };
    FOR(v, 0, n) for(int u: G[v]) {
      ys4 += (SZ(G[v]) - 1) * choose2(SZ(G[u]) - 1);
    }
    paths4 = -(4 * rectangles4 + 2 * ps4 + 3 * triangles3);
    FOR(v, 0, n) {
      int x = 0;
      for(int u: G[v]) {
        x += SZ(G[u]) - 1;
        paths4 -= choose2(SZ(G[u]) - 1);
      }
      paths4 += choose2(x);
    }
    FOR(v, 0, n) {
      int s = SZ(G[v]);
      stars3 += s * ll(s - 1) * (s - 2);
      stars4 += s * (__int128)(s - 1) * (s - 2) * (s - 3);
    }
    stars3 /= 6; stars4 /= 24;
  }
};
```

7.2 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$ 0ae1d4, 48 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }
  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);
    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(FE\log(V)\right)$ where F is max flow. $\mathcal{O}(VE)$ for setpi. 58385b, 79 lines

```
#include <bits/extc++.h>
const ll INF = numeric_limits<ll>::max() / 4;
struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost, flow;
  };
  int N;
  vector<vector<edge>> ed;
  vi seen;
  vector<ll> dist, pi;
  vector<edge*> par;
  MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}
  void addEdge(int from, int to, ll cap, ll cost) {
    if (from == to) return;
    ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
    ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
  }
  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;
    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({ 0, s });
    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (edge& e : ed[s]) if (!seen[e.to]) {
        ll val = di - pi[e.to] + e.cost;
        if (e.cap - e.flow > 0 && val < dist[e.to]) {
          dist[e.to] = val;
          par[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({ -dist[e.to], e.to });
          else
            q.modify(its[e.to], { -dist[e.to], e.to });
        }
      }
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }
  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (edge* x = par[t]; x; x = par[x->from])
        fl = min(fl, x->cap - x->flow);
      totflow += fl;
      for (edge* x = par[t]; x; x = par[x->from]) {
        x->flow += fl;
```

```
        ed[x->to][x->rev].flow -= fl;
      }
    }
    rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
    return {totflow, totcost/2};
  }
  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (edge& e : ed[i]) if (e.cap)
          if ((v = pi[i] + e.cost) < pi[e.to])
            pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

EdmondsKarp.h

Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only. 1f2fed, 56 lines

```
template<class T> T edmondsKarp(vector<unordered_map<int, T>>&
    graph, int source, int sink) {
  assert(source != sink);
  T flow = 0;
  vi par(sz(graph)), q = par;
  for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;
    rep(i,0,ptr) {
      int x = q[i];
      for (auto e : graph[x]) {
        if (par[e.first] == -1 && e.second > 0) {
          par[e.first] = x;
          q[ptr++] = e.first;
          if (e.first == sink) goto out;
        }
      }
    }
    return flow;
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
      inc = min(inc, graph[par[y]][y]);
    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
      int p = par[y];
      if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
      graph[y][p] += inc;
    }
  }
}
```

Dinic.h

Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max|\text{cap}|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching. d7f0f1, 42 lines

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
  };
  vi lvl, ptr, q;
  vector<vector<Edge>> adj;
  Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  void addEdge(int a, int b, ll c, ll rcap = 0) {
    adj[a].push_back({b, sz(adj[b]), c, c});
    adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
  }
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < sz(adj[v]); i++) {
      Edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.c))) {
          e.c -= p, adj[e.to][e.rev].c += p;
          return p;
        }
    }
    return 0;
  }
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    rep(L,0,31) do { // 'int L=30' maybe faster for random data
      lvl = ptr = vi(sz(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 - L))
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

Time: $\mathcal{O}\left(V^3\right)$ 8b0e19, 21 lines

```
pair<int, vi> globalMinCut(vector<vi> mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i,0,n) co[i] = {i};
  rep(ph,1,n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i,0,n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i,0,n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

FlowDemands.h

Description: Flows with demands. e1c0d0, 52 lines

```
//#include "flow_edmonds_karp.h"
//#include "flow_push_relabel.h" // if you need
// Flow with demands; time: O(maxflow)
struct FlowDemands {
  MaxFlow net;
  vector<vector<flow_t>> demands;
  flow_t total = 0;
  // Initialize for k vertices
  FlowDemands(int k = 0) : net(2) {
    while (k--) addVert();
  }
  // Add new vertex
  int addVert() {
    int v = net.addVert();
    demands.pb({});
    net.addEdge(0, v, 0);
    net.addEdge(v, 1, 0);
    return v-2;
  }
  // Add edge from u to v with demand dem
  // and capacity cap (dem <= flow <= cap).
  // Returns edge index in adjacency list of u.
  int addEdge(int u, int v,
              flow_t dem, flow_t cap) {
    demands[u].pb(dem);
    demands[v].pb(0);
    total += dem;
    net.G[0][v].cap += dem;
    net.G[u+2][1].cap += dem;
    return net.addEdge(u+2, v+2, cap-dem) - 2;
  }
  // Check if there exists a flow with value f
  // for source src and destination dst.
  // For circulation, you can set args to 0.
  bool canFlow(int src, int dst, flow_t f) {
    net.addEdge(dst += 2, src += 2, f);
    f = net.maxFlow(0, 1);
    net.G[src].pop_back();
    net.G[dst].pop_back();
    return f == total;
  }
  // Get flow through e-th edge of vertex v
  flow_t getFlow(int v, int e) {
    return net.getFlow(v+2,e+2)+demands[v][e];
  }
};
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.

Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 0418b3, 13 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i,1,N) {
    PushRelabel D(N); // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j,i+1,N)
      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

7.3 Flow with demands

Say we want $d(e) \le f(e) \le c(e)$ for each edge. To find an arbitrary flow, add s', t' and the following edges:

- $\forall v \in V : c'((s', v)) = \sum_u d((u, v)), \quad c'((v, t')) = \sum_w d((v, w))$,
- $\forall (u, v) \in E : c'((u, v)) = c((u, v)) - d((u, v))$,
- $c'((t, s)) = \infty$.

For min flow, replace ∞ with L and find smallest L such that flow is saturated.

7.4 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. *btoa*[i] will be the match for vertex i on the right side, or -1 if it's not matched.

Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`

Time: $\mathcal{O}\left(\sqrt{V}E\right)$ f612e4, 42 lines

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a]) if (B[b] == L + 1) {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
  return 0;
}
int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
```

```
}
```

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); dfsMatching(g, btoa);`
Time: $\mathcal{O}(VE)$ 522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
  if (btoa[j] == -1) return 1;
  vis[j] = 1; int di = btoa[j];
  for (int e : g[di])
    if (!vis[e] && find(e, g, btoa, vis)) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
  vi vis;
  rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.
"DFSMatching.h" da4196, 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m);
  for (int it : match) if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    lfound[i] = 1;
    for (int e : g[i]) if (!seen[e] && match[e] != -1) {
      seen[e] = true;
      q.push_back(match[e]);
    }
  }
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}\left(N^2M\right)$ 1e0fe9, 31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) {
  if (a.empty()) return {0, {}};
  int n = sz(a) + 1, m = sz(a[0]) + 1;
  vi u(n), v(m), p(m), ans(n - 1);
  rep(i,1,n) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vi dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      rep(j,1,m) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      rep(j,0,m) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

GeneralMatching.h

Description: Unweighted matching for general graphs. If `white[v] = 0` at the end, v is part of every max matching.
Time: $\mathcal{O}(NM)$, faster in practice 28552a, 48 lines

```
struct MaxMatching {  // 1-indexed
  vector<vi> G; int n;
  vi mate, par, white; vector<pii> l;
  MaxMatching(vector<vi> _G) : G(_G), n(SZ(G)), mate(n),
    par(n), white(n), l(n) {}
  int group(int x) {
    return par[x] = (white[par[x]] ? group(par[x]) : par[x]);
  }
  void match(int p, int b) {
    swap(mate[p], b); if(mate[b] != p) return;
    if(!l[p].nd) mate[b] = l[p].st, match(l[p].st, b);
    else match(l[p].st, l[p].nd), match(l[p].nd, l[p].st);
  }
  bool augment(int a) {
    white[a] = 1; par[a] = 0; l[a] = {0, 0};
    queue<int> q; q.push(a);
    while(!q.empty()) {
      a = q.front(); q.pop();
      for(int b: G[a]) {
        if(white[b]) {
          int x = group(a), y = group(b), lca = 0;
          while(x || y) {
            if(y) swap(x, y);
            if(l[x] == mp(a, b)) { lca = x; break; }
            l[x] = {a, b}; x = group(l[mate[x]].st);
          }
          for(int v: {group(a), group(b)}) while(v != lca) {
            q.push(v); white[v] = 1; par[v] = lca;
            v = group(l[mate[v]].st);
          }
        } else if(!mate[b]) {
          mate[b] = a; match(a, b); fill(all(white), 0);
          return 1;
        } else if(!white[mate[b]]) {
          white[mate[b]] = 1; par[mate[b]] = b;
          l[b] = {0, 0}; l[mate[b]] = {a, 0};
          q.push(mate[b]);
        }
      }
    }
    return 0;
  }
  int max_matching() {
    int ans = 0;
    FOR(v, 1, n) if(!mate[v]) ans += augment(v);
    return ans;
  }
};
```

GeneralWeightedMatching.h

Description: Weighted matching for general graphs.
Time: $\mathcal{O}\left(N^3\right)$, fast in practice 8ba4a8, 146 lines

```
namespace WeightedBlossom { // 1-indexed
  #define d(x) (lab[x.u] + lab[x.v] - e[x.u][x.v].w * 2)
  const int N = 501 * 2;
  const ll INF = 1e18;
  struct Q { int u, v; ll w; } e[N][N]; vi p[N];
  int n, m = 0, id, h, t, lk[N], sl[N], sk[N], f[N], b[N][N];
  int s[N], ed[N], q[N]; ll lab[N];
  void upd(int u, int v) {
    if(!sl[v] || d(e[u][v]) < d(e[sl[v]][v])) sl[v] = u;
  }
  void ss(int v){
    sl[v] = 0;
    rep(u, 1, n)
      if(e[u][v].w > 0 && sk[u] != v && !s[sk[u]]) upd(u, v);
  }
  void ins(int u) {
    if(u <= n) q[++t] = u;
    else for(auto v: p[u]) ins(v);
  }
  void mdf(int u, int w) {
    sk[u] = w;
    if(u > n) for(auto v: p[u]) mdf(v, w);
  }
  int gr(int u,int v){
    if((v = find(all(p[u]), v) - p[u].begin()) & 1) {
      reverse(1 + all(p[u])); return SZ(p[u]) - v;
    }
    return v;
  }
  void stm(int u, int v){
    lk[u] = e[u][v].v;
    if(u <= n) return;
    Q w = e[u][v]; int x = b[u][w.u], y = gr(u,x);
    FOR(i, 0, y) stm(p[u][i], p[u][i ^ 1]);
    stm(x, v);
    rotate(p[u].begin(), p[u].begin() + y, p[u].end());
  }
  void aug(int u, int v){
    int w = sk[lk[u]]; stm(u, v); if(!w) return;
    stm(w, sk[f[w]]); aug(sk[f[w]], w);
  }
  int lca(int u, int v){
    for(++id; u|v; swap(u, v)){
      if(!u) continue;
      if(ed[u] == id) return u;
      ed[u] = id; if((u = sk[lk[u]])) u = sk[f[u]]; // not ==
    }
    return 0;
  }
  void add(int uu, int a, int vv){
    int x = n + 1; while(x <= m && sk[x]) x++;
    if(x > m) m++;
    lab[x] = s[x] = sk[x] = 0; lk[x] = lk[a];
    p[x].clear(); p[x].pb(a);
    for(auto i = uu, j = 0; i != a; i = sk[f[j]])
      p[x].pb(i), p[x].pb(j = sk[lk[i]]), ins(j);
    reverse(1 + all(p[x]));
    for(auto i = vv, j = 0; i != a; i = sk[f[j]])
      p[x].pb(i), p[x].pb(j = sk[lk[i]]), ins(j);
    mdf(x, x); rep(i, 1, m) e[x][i].w = e[i][x].w = 0;
    memset(b[x] + 1, 0, n * sizeof b[0][0]);
    for(auto u: p[x]){
      rep(v, 1, m) if(!e[x][v].w || d(e[u][v]) < d(e[x][v]))
        e[x][v] = e[u][v], e[v][x] = e[v][u];
      rep(v, 1, n) if(b[u][v]) b[x][v] = u;
    }
    ss(x);
  }
  void ex(int u) {  // s[u] == 1
    for(auto x: p[u]) mdf(x, x);
    int a = b[u][e[u][f[u]].u], r = gr(u, a);
    FOR(i, 0, r) {
      int x = p[u][i], y = p[u][i + 1];
      f[x] = e[y][x].u; s[x] = 1; s[y] = sl[x] = 0;
      ss(y); ins(y); i++;
    }
    s[a] = 1; f[a] = f[u];
    FOR(i, r + 1, SZ(p[u])) s[p[u][i]] = -1, ss(p[u][i]);
    sk[u] = 0;
  }
  bool on(const Q &ee) {
    int u = sk[ee.u], v = sk[ee.v], a;
    if(s[v] == -1) {
      f[v] = ee.u, s[v] = 1, a = sk[lk[v]];
      sl[v] = sl[a] = s[a] = 0, ins(a);
    }
    else if(!s[v]){
      a = lca(u, v);
      if(!a) return aug(u,v), aug(v,u), 1;
      else add(u,a,v);
    }
    return 0;
  }
  bool bfs() {
    memset(s + 1, -1, m * sizeof s[0]);
    memset(sl + 1, 0, m * sizeof sl[0]);
    h = 1; t = 0;
    rep(i, 1, m)
      if(sk[i] == i && !lk[i]) f[i] = s[i] = 0, ins(i);
    if(h > t) return 0;
    while(1){
      while(h <= t) {
        int u = q[h++];
        if(s[sk[u]] != 1) rep(v, 1, n) {
          if(e[u][v].w > 0 && sk[u] != sk[v]) {
            if(d(e[u][v])) upd(u, sk[v]);
            else if(on(e[u][v])) return 1;
          }
        }
      }
      ll x = INF;
```

```
    rep(i, n + 1, m) if(sk[i] == i && s[i] == 1)
      x = min(x, lab[i] >> 1);
    rep(i, 1, m) if(sk[i] == i && sl[i] && s[i] != 1)
      x = min(x, d(e[sl[i]][i]) >> (s[i] + 1));
    rep(i, 1, n) if(~s[sk[i]])
      if((lab[i] += (s[sk[i]] * 2 - 1) * x) <= 0) return 0;
    rep(i, n + 1, m) if(sk[i] == i && ~s[sk[i]])
      lab[i] += (2 - s[sk[i]] * 4) * x;
    h = 1; t = 0;
    rep(i, 1, m) if(sk[i] == i && sl[i] && sk[sl[i]] != i &&
      !d(e[sl[i]][i]) && on(e[sl[i]][i])) return 1;
    rep(i, n + 1, m) if(sk[i] == i && s[i] == 1 && !lab[i])
      ex(i);
  }
  return 0;
}
pair<ll, vector<pii>> run
(int _n, vector<tuple<int, int, ll>> edges) {
  memset(ed + 1, 0, m * sizeof ed[0]);
  memset(lk + 1, 0, m * sizeof lk[0]);
  n = m = _n; id = 0; iota(sk + 1, sk + n + 1, 1);
  ll wm = 0, weight = 0;
  rep(i, 1, n) rep(j, 1, n) e[i][j] = {i, j, 0};
  for(auto [u, v, w]: edges)
    wm = max(wm, e[v][u].w = e[u][v].w = max(e[u][v].w, w));
  rep(i, 1, n) p[i].clear();
  rep(i, 1, n) rep(j, 1, n) b[i][j] = i * (i == j);
  fill_n(lab + 1, n, wm); while(bfs());
  vector<pii> matching;
  rep(i, 1, n) if(i < lk[i])
    weight += e[i][lk[i]].w, matching.pb({i, lk[i]});
  return {weight, matching};
}
#undef d
}
```

MatroidIntersection.h

Description: Find largest subset S of [n] such that S is independent in both matroid A and B, given by their oracles, see example implementations below. Returns vector V such that V[i] = 1 iff i-th element is included in found set;

Time: $\mathcal{O}\left(r^2 \cdot (init + n \cdot add)\right)$, where r is max independent set. e3ed3b, 152 lines

```
template<class T, class U>
vector<bool> intersectMatroids(T& A, U& B, int n) {
  vector<bool> ans(n);
  bool ok = 1;
// NOTE: for weighted matroid intersection find
// shortest augmenting paths first by weight change,
// then by length using Bellman-Ford,
  // Speedup trick (only for unweighted):
  A.init(ans); B.init(ans);
  rep(i, n)
    if (A.canAdd(i) && B.canAdd(i))
      ans[i] = 1, A.init(ans), B.init(ans);
  //End of speedup
  while (ok) {
    vector<vi> G(n);
    vector<bool> good(n);
    queue<int> que;
    vi prev(n, -1);
    A.init(ans); B.init(ans); ok = 0;
    rep(i, n) if (!ans[i]) {
      if (A.canAdd(i)) que.push(i), prev[i]=-2;
      good[i] = B.canAdd(i);
    }
    rep(i, n) if (ans[i]) {
      ans[i] = 0;
      A.init(ans); B.init(ans);
      rep(j, n) if (i != j && !ans[j]) {
        if (A.canAdd(j)) G[i].pb(j); //-cost[j]
        if (B.canAdd(j)) G[j].pb(i); // cost[i]
      }
      ans[i] = 1;
    }
    while (!que.empty()) {
      int i = que.front();
      que.pop();
      if (good[i]) { // best found (unweighted = shortest path)
        ans[i] = 1;
        while (prev[i] >= 0) { // alternate matching
          ans[i = prev[i]] = 0;
          ans[i = prev[i]] = 1;
        }
        ok = 1; break;
      }
      for(auto j: G[i]) if (prev[j] == -1)
        que.push(j), prev[j] = i;
    }
  }
  return ans;
}
// Matroid where each element has color
// and set is independent iff for each color c
// #{elements of color c} <= maxAllowed[c].
struct LimOracle {
  vi color; // color[i] = color of i-th element
  vi maxAllowed; // Limits for colors
  vi tmp;
  // Init oracle for independent set S; O(n)
  void init(vector<bool>& S) {
    tmp = maxAllowed;
    rep(i, sz(S)) tmp[color[i]] -= S[i];
  }
  // Check if S+{k} is independent; time: O(1)
  bool canAdd(int k) { return tmp[color[k]] > 0;}
};
// Graphic matroid - each element is edge,
// set is independent iff subgraph is acyclic.
struct GraphOracle {
  vector<pii> elems; // Ground set: graph edges
  int n; // Number of vertices, indexed [0;n-1]
  vi par;
  int find(int i) {
    return par[i] == -1 ? i : par[i] = find(par[i]);
  }
  // Init oracle for independent set S; ~O(n)
  void init(vector<bool>& S) {
    par.assign(n, -1);
    rep(i, sz(S)) if (S[i])
      par[find(elems[i].st)] = find(elems[i].nd);
  }
  // Check if S+{k} is independent; time: ~O(1)
  bool canAdd(int k) {
    return find(elems[k].st) != find(elems[k].nd);
  }
};
// Co-graphic matroid - each element is edge,
// set is independent iff after removing edges
// from graph number of connected components
// doesn't change.
struct CographOracle {
  vector<pii> elems; // Ground set: graph edges
  int n; // Number of vertices, indexed [0;n-1]
  vector<vi> G;
  vi pre, low;
  int cnt;
  int dfs(int v, int p) {
    pre[v] = low[v] = ++cnt;
    for(auto e: G[v]) if (e != p)
      low[v] = min(low[v], pre[e] ?: dfs(e,v));
    return low[v];
  }
  // Init oracle for independent set S; O(n)
  void init(vector<bool>& S) {
    G.assign(n, {});
    pre.assign(n, 0);
    low.resize(n);
    cnt = 0;
    rep(i,sz(S)) if (!S[i]) {
      pii e = elems[i];
      G[e.st].pb(e.nd);
      G[e.nd].pb(e.st);
    }
    rep(v, n) if (!pre[v]) dfs(v, -1);
  }
  // Check if S+{k} is independent; time: O(1)
  bool canAdd(int k) {
    pii e = elems[k];
    return max(pre[e.st], pre[e.nd]) != max(low[e.st], low[e.nd]);
  }
};
// Matroid equivalent to linear space with XOR
struct XorOracle {
  vector<ll> elems; // Ground set: numbers
  vector<ll> base;
  // Init for independent set S; O(n*r^2)
  void init(vector<bool>& S) {
    base.assign(63, 0);
    rep(i, sz(S)) if (S[i]) {
      ll e = elems[i];
      rep(j, sz(base)) if ((e >> j) & 1) {
        if (!base[j]) {
          base[j] = e;
          break;
        }
        e ^= base[j];
      }
    }
  }
  // Check if S+{k} is independent; time: O(r)
  bool canAdd(int k) {
    ll e = elems[k];
    rep(i, sz(base)) if ((e >> i) & 1) {
      if (!base[i]) return 1;
      e ^= base[i];
    }
    return 0;
  }
};
```

Konig.h

Description: O(n + matching(n, m)) wyznaczanie w grafie dwudzielnym kolejno minimalnego pokrycia krawędziowego (PK), maksymalnego zbioru niezależnych wierzchołków (NW), minimalnego pokrycia wierzchołkowego (PW) korzystając z maksymalnego zbioru niezależnych krawędzi (NK) (tak zwany matching). Z tw. Koniga zachodzi |NK|=n-|PK|=n-|NW|=|PW|.

"../matching/main.cpp" d37a69, 41 lines

```
// BEGIN HASH
vector<pair<int, int>> get_min_edge_cover(vector<vector<int>> graph) {
  vector<int> match = Matching(graph)().second;
  vector<pair<int, int>> ret;
  REP(v, ssize(match))
    if(match[v] != -1 and v < match[v])
      ret.emplace_back(v, match[v]);
    else if(match[v] == -1 and not graph[v].empty())
      ret.emplace_back(v, graph[v].front());
  return ret;
} // END HASH
// BEGIN HASH
array<vector<int>, 2> get_coloring(vector<vector<int>> graph) {
  int n = ssize(graph);
  vector<int> match = Matching(graph)().second;
  vector<int> color(n, -1);
  function<void (int)> dfs = [&](int v) {
    color[v] = 0;
    for(int u : graph[v])
      if(color[u] == -1) {
        color[u] = true;
        dfs(match[u]);
      }
  };
  REP(v, n)
    if(match[v] == -1)
      dfs(v);
  REP(v, n)
    if(color[v] == -1)
      dfs(v);
  array<vector<int>, 2> groups;
  REP(v, n)
    groups[color[v]].emplace_back(v);
  return groups;
}
vector<int> get_max_independent_set(vector<vector<int>> graph) {
  return get_coloring(graph)[0];
}
vector<int> get_min_vertex_cover(vector<vector<int>> graph) {
  return get_coloring(graph)[1];
} // END HASH
```

7.5 DFS algorithms

kShortestWalks.h

Description: Given a non-negative weighted directed graph, computes lengths of K shortest walks (vertices can repeat). For graphs with negative weights, try your luck or change Dijkstra to SPFA.

Time: $\mathcal{O}\left((M + K) \log N\right)$ 5a47b8, 58 lines

```
struct PersistentHeap {
  struct Node {
    int id, l = -1, r = -1; ll cost;
    Node(int _id, ll _cost) : id(_id), cost(_cost) {}
  };
  vector<Node> ds;
  int add(int id, ll cost) {
    ds.pb(Node(id, cost)); return SZ(ds) - 1;
  }
  int ins(int v, int u) {
    if(v == -1) return u;
    ds.pb(ds[v]); v = SZ(ds) - 1; swap(ds[v].l, ds[v].r);
    if(ds[v].cost > ds[u].cost) {
      swap(ds[v].cost, ds[u].cost), swap(ds[v].id, ds[u].id);
    }
    ds[v].r = ins(ds[v].r, u); return v;
  }
  void insert(int &v, int u) { v = ins(v, u); }
};
vector<ll> kWalk(vector<vector<pii>> G, int s, int t, int k) {
  int n = SZ(G); vector<vector<pii>> GR(n);
  FOR(v, 0, n) for(auto &[u, d]: G[v]) GR[u].pb({v, d});
  const ll INF = 1e18; vector<ll> dist(n, INF); vi par(n, -1);
  using T = pair<ll, int>; dist[t] = 0;
```

```
  priority_queue<T, vector<T>, greater<T>> q; q.push({0, t});
  while(!q.empty()) {
    auto [dv, v] = q.top(); q.pop();
    if(dv != dist[v]) continue;
    for(auto &[u, d]: GR[v]) if(dv + d < dist[u]) {
      par[u] = v; dist[u] = dv + d; q.push({dist[u], u});
    }
  }
  vector<vi> tree(n);
  FOR(v, 0, n) if(~par[v]) tree[par[v]].pb(v);
  PersistentHeap heap; vi head(n, -1);
  function<void(int)> dfs = [&](int v) {
    bool skip = 0;
    for(auto &[u, d]: G[v]) if(dist[u] != INF) {
      if(dist[v] == dist[u] + d && par[v] == u && !skip)
        skip = 1;
      else
        heap.insert(head[v], heap.add(u, dist[u]-dist[v]+d));
    }
    for(int u: tree[v]) head[u] = head[v], dfs(u);
  }; dfs(t);
  vector<ll> ans(k, -1); q.push({dist[s], heap.add(s, 0)});
  FOR(i, 0, k) {
    if(q.empty() || dist[s] == INF) break;
    auto [dv, v] = q.top(); q.pop();
    ans[i] = dv; auto &node = heap.ds[v]; ll diff = 0;
    for(int u: {head[node.id], node.l, node.r}) {
      if(~u) q.push({dv + heap.ds[u].cost - diff, u});
      diff = node.cost;
    }
  }
  return ans;
}
```

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.

Time: $\mathcal{O}(E+V)$ 76b5c9, 24 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
  int low = val[j] = ++Time, x; z.push_back(j);
  for (auto e : g[j]) if (comp[e] < 0)
    low = min(low, val[e] ?: dfs(e,g,f));
  if (low == val[j]) {
    do {
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
  }
  return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
  int n = sz(g);
  val.assign(n, 0); comp.assign(n, -1);
  Time = ncomps = 0;
  rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.

Usage:
```
int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
```

Time: $\mathcal{O}(E+V)$ c6b7c7, 32 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, top = me;
  for (auto [y, e] : ed[at]) if (e != par) {
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me) st.push_back(e);
      else { /* e is a bridge */ }
    }
  }
  return top;
}
template<class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

2sat.h

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

Usage:
```
TwoSat ts(number of boolean variables);
ts.either(0, ∼3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,∼1,2}); // <= 1 of vars 0, ∼1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars
```

Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses. 5f9706, 56 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true
  TwoSat(int n = 0) : N(n), gr(2*n) {}
  int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }
  void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
  }
  void setValue(int x) { either(x, x); }
  void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }
  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = low;
      if (values[x>>1] == -1)
        values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
  }
  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

Dominators.h

Description: Tarjan's dominators in directed graph Returns tree (as array of parents) of immediate dominators idom. idom[root] = root, idom[v] = -1 if v is unreachable from root

Time: $\mathcal{O}(|E|log|V|)$ 2613e6, 32 lines

```
vi dominators(vector<vi>& G, int root) {
  int n = sz(G); vector<vi> in(n), bucket(n);
  vi pre(n, -1), anc(n, -1), par(n), best(n);
  vi ord, idom(n, -1), sdom(n, n), rdom(n);
  auto dfs = [&] (auto f, int v, int p) -> void {
    if (pre[v] == -1) {
      par[v] = p; pre[v] = sz(ord);
      ord.pb(v);
      for (auto e :  G[v])
        in[e].pb(v), f(f, e, v);
    }
  };
  auto find = [&](auto f, int v) -> pii {
    if (anc[v] == -1) return {best[v], v};
    int b; tie(b, anc[v]) = f(f, anc[v]);
    if (sdom[b] < sdom[best[v]]) best[v] = b;
    return {best[v], anc[v]};
  };
  rdom[root] = idom[root] = root;
  iota(all(best), 0); dfs(dfs, root, -1);
  rep(i, sz(ord)) {
    int v = ord[sz(ord)-i-1], b = pre[v];
    for (auto e : in[v])
      b = min(b, pre[e] < pre[v] ? pre[e] :
        sdom[find(find, e).st]);
    for (auto u : bucket[v]) rdom[u]=find(find,u).st;
    sdom[v] = b; anc[v] = par[v];
    bucket[ord[sdom[v]]].pb(v);
  }
  for (auto v : ord) idom[v] = (rdom[v] == v ?
    ord[sdom[v]] : idom[rdom[v]]);
  return idom; }
```

KthShortest.h

Description: Given directed weighted graph with non-negative edge weights gets K-th shortest walk (not necessarily simple) in O(log|E|). -1 if no next path (can only happen in DAG). WARNING: USES KLOGM memory and persistent heaps! 2d9393, 57 lines

```
constexpr ll INF = 1e18;
struct Eppstein {
  using T = ll; using Edge = pair<int, T>;
  struct Node { int E[2] = {}, s = 0; Edge x; };
  T shortest; // Shortest path length
  priority_queue<pair<T, int>> Q;
  vector<Node> P{1}; vi h;
  Eppstein(vector<vector<Edge>>& G, int s, int t) {
    int n = sz(G); vector<vector<Edge>> H(n);
    rep(i,n) for(auto &e : G[i])
      H[e.st].pb({i,e.nd});
    vi ord, par(n, -1); vector<T> d(n, -INF);
    Q.push({d[t] = 0, t});
    while (!Q.empty()) {
      auto v = Q.top(); Q.pop();
      if (d[v.nd] == v.st) {
        ord.pb(v.nd);
        for(auto &e : H[v.nd])
        if (v.st-e.nd > d[e.st]) {
          Q.push({d[e.st] = v.st-e.nd, e.st});
          par[e.st] = v.nd;
        }
      }
    }
    if ((shortest = -d[s]) >= INF) return;
    h.resize(n);
    for(auto &v : ord) {
      int p = par[v]; if (p+1) h[v] = h[p];
      for(auto &e : G[v]) if (d[e.st] > -INF) {
        T k = e.nd - d[e.st] + d[v];
        if (k || e.st != p)
          h[v] = push(h[v], {e.st, k});
        else p = -1;
      }
    }
    P[0].x.st = s; Q.push({0, 0});
  }
  int push(int t, Edge x) {
    P.pb(P[t]);
    if (!P[t = sz(P)-1].s || P[t].x.nd >= x.nd)
      swap(x, P[t].x);
    if (P[t].s) {
      int i = P[t].E[0], j = P[t].E[1];
      int d = P[i].s > P[j].s;
      int k = push(d ? j : i, x);
      P[t].E[d] = k; // Don't inline k!
    }
```

```
    P[t].s++; return t;
  }
  ll nextPath() { // next length, -1 if no next path
    if (Q.empty()) return -1;
    auto v = Q.top(); Q.pop();
    for (int i : P[v.nd].E) if (i)
      Q.push({ v.st-P[i].x.nd+P[v.nd].x.nd, i });
    int t = h[P[v.nd].x.st];
    if (t) Q.push({v.st - P[t].x.nd, t });
    return shortest - v.st; } };
```

PlanarFaces.h

Description: Read desc below. a391b4, 102 lines

```
/*
 * complexity mlogm, assumes that you are given an embedding
 * graph is drawn straightline non-intersecting
 * returns combinatorial embedding (inner face vertices clockwise, outer
      counter clockwise).
 * WAZNE czasem trzeba zlaczyc wszystkie sciany zewnetrzne (chodzi o kmine
      do konkretnego zadania)
 * (ktorych moze byc kilka, gdy jest wiele spojnych) w jedna sciane.
 * Zewnetrzne sciany moga wygladac jak kaktusy, a wewnetrzne zawsze sa
      niezdegenerowanym wielokatem.
 */
struct Edge {
  int e, from, to;
  // face is on the right of "from -> to"
};
ostream& operator<<(ostream &o, Edge e) {
  return o << vector{e.e, e.from, e.to};
}
struct Face {
  bool is_outside;
  vector<Edge> sorted_edges;
  // edges are sorted clockwise for inside and cc for outside faces
};
ostream& operator<<(ostream &o, Face f) {
  return o << pair(f.is_outside, f.sorted_edges);
}
vector<Face> split_planar_to_faces(vector<pii> coord, vector<pii> edges) {
  int n = sz(coord);
  int E = sz(edges);
  vector<vi> graph(n);
  rep(e, E) {
    auto [v, u] = edges[e];
    graph[v].eb(e);
    graph[u].eb(e);
  }
  vi lead(2 * E);
  iota(lead.begin(), lead.end(), 0);
  function<int (int)> find = [&](int v) {
    return lead[v] == v ? v : lead[v] = find(lead[v]);
  };
  auto side_of_edge = [&](int e, int v, bool outward) {
    return 2 * e + ((v != min(edges[e].first, edges[e].second)) ^ outward)
        ;
  };
  rep(v, n) {
    vector<pair<pii, int>> sorted;
    for(int e : graph[v]) {
      auto p = coord[edges[e].first ^ edges[e].second ^ v];
      auto center = coord[v];
      sorted.eb(pair(p.first - center.first, p.second - center.second), e)
          ;
    }
    sort(all(sorted), [&](pair<pii, int> l0, pair<pii, int> r0) {
      auto l = l0.first;
      auto r = r0.first;
      bool half_l = l > pair(0, 0);
      bool half_r = r > pair(0, 0);
      if(half_l != half_r)
        return half_l;
      return l.first * LL(r.second) - l.second * LL(r.first) > 0;
    });
    rep(i, sz(sorted)) {
      int e0 = sorted[i].second;
      int e1 = sorted[(i + 1) % sz(sorted)].second;
      int side_e0 = side_of_edge(e0, v, true);
      int side_e1 = side_of_edge(e1, v, false);
      lead[find(side_e0)] = find(side_e1);
    }
  }
  vector<vi> comps(2 * E);
  rep(i, 2 * E)
    comps[find(i)].eb(i);
  vector<Face> polygons;
  vector<vector<pii>> outgoing_for_face(n);
  rep(leader, 2 * E)
    if(sz(comps[leader])) {
      for(int id : comps[leader]) {
        int v = edges[id / 2].first;
        int u = edges[id / 2].second;
        if(v > u)
          swap(v, u);
        if(id % 2 == 1)
          swap(v, u);
        outgoing_for_face[v].eb(u, id / 2);
      }
      vector<Edge> sorted_edges;
      function<void (int)> dfs = [&](int v) {
        while(sz(outgoing_for_face[v])) {
          auto [u, e] = outgoing_for_face[v].back();
          outgoing_for_face[v].pop_back();
          dfs(u);
          sorted_edges.eb(e, v, u);
        }
      };
      dfs(edges[comps[leader].front() / 2].first);
      reverse(all(sorted_edges));
      LL area = 0;
      for(auto edge : sorted_edges) {
        auto l = coord[edge.from];
        auto r = coord[edge.to];
        area += l.first * LL(r.second) - l.second * LL(r.first);
      }
      polygons.eb(area >= 0, sorted_edges);
    }
  // Remember that there can be multiple outside faces.
  return polygons;
}
```

PlanarityCheck.h

Description: Read desc below. cc4508, 93 lines

```
/*
 * Opis: O(szybko) ale istnieja przyklady O(n2), przyjmuje graf
        nieskierowany bez petelek i multikrawedzi.
 */
bool is_planar(vector<vi> g) {
  int n = sz(g), m = 0;
  rep(v, n) m += sz(g[v]);
  m /= 2;
  if(n <= 3) return true;
  if(m > 3 * n - 6) return false;
  vector<vi> up(n), dn(n);
  vi low(n, -1), pre(n);
  rep(start, n)
    if(low[start] == -1) {
      vector<pii> e_up;
      int tm = 0;
      function<void (int, int)> dfs_low = [&](int v, int p) {
        low[v] = pre[v] = tm++;
        for(int u : g[v])
          if(u != p and low[u] == -1) {
            dn[v].eb(u);
            dfs_low(u, v);
            low[v] = min(low[v], low[u]);
          }
          else if(u != p and pre[u] < pre[v]) {
            up[v].eb(ssize(e_up));
            e_up.eb(v, u);
            low[v] = min(low[v], pre[u]);
          }
      };
      dfs_low(start, -1);
      vector<pair<int, bool>> dsu(sz(e_up));
      rep(v, sz(dsu)) dsu[v].first = v;
      function<pair<int, bool> (int)> find = [&](int v) {
        if(dsu[v].first == v)
          return pair(v, false);
        auto [u, ub] = find(dsu[v].first);
        return dsu[v] = pair(u, ub ^ dsu[v].second);
      };
      auto onion = [&](int x, int y, bool flip) {
        auto [v, vb] = find(x);
        auto [u, ub] = find(y);
        if(v == u)
          return not (vb ^ ub ^ flip);
        dsu[v] = {u, vb ^ ub ^ flip};
        return true;
      };
      auto interlace = [&](const vi &ids, int lo) {
        vi ans;
        for(int e : ids)
          if(pre[e_up[e].second] > lo)
            ans.eb(e);
        return ans;
      };
      auto add_fu = [&](const vi &a, const vi &b) {
        fwd(k, 1, sz(a))
          if(not onion(a[k - 1], a[k], 0))
            return false;
        fwd(k, 1, sz(b))
          if(not onion(b[k - 1], b[k], 0))
            return false;
        return a.empty() or b.empty() or onion(a[0], b[0], 1);
      };
      function<bool (int, int)> dfs_planar = [&](int v, int p) {
        for(int u : dn[v])
          if(not dfs_planar(u, v))
            return false;
        rep(i, sz(dn[v])) {
          fwd(j, i + 1, sz(dn[v]))
            if(not add_fu(interlace(up[dn[v][i]], low[dn[v][j]]),
                    interlace(up[dn[v][j]], low[dn[v][i]])))
              return false;
          for(int j : up[v]) {
            if(e_up[j].first != v)
              continue;
            if(not add_fu(interlace(up[dn[v][i]], pre[e_up[j].second]),
                    interlace({j}, low[dn[v][i]])))
              return false;
          }
        }
        for(int u : dn[v]) {
          for(int idx : up[u])
            if(pre[e_up[idx].second] < pre[p])
              up[v].eb(idx);
          exchange(up[u], {});
        }
        return true;
      };
      if(not dfs_planar(start, -1))
        return false;
    }
  return true;
}
```

7.6 Coloring

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D, computes a $(D + 1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

Time: $\mathcal{O}(NM)$ e210e2, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) {
  vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
  for (pii e : eds) ++cc[e.first], ++cc[e.second];
  int u, v, ncols = *max_element(all(cc)) + 1;
  vector<vi> adj(N, vi(ncols, -1));
  for (pii e : eds) {
    tie(u, v) = e;
    fan[0] = v;
    loc.assign(ncols, 0);
    int at = u, end = u, d, c = free[u], ind = 0, i = 0;
    while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
      loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
    cc[loc[d]] = c;
    for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
      swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
    while (adj[fan[i]][d] != -1) {
      int left = fan[i], right = fan[++i], e = cc[i];
      adj[u][e] = left;
      adj[left][e] = u;
      adj[right][e] = -1;
      free[right] = e;
    }
    adj[u][d] = fan[i];
    adj[fan[i]][d] = u;
    for (int y : {fan[0], u, end})
      for (int& z = free[y] = 0; adj[y][z] != -1; z++);
  }
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
  return ret;
}
```

ChordalGraph.h

Description: A graph is chordal if any cycle C >= 4 has a chord i.e. an edge (u, v) where u and v is in the cycle but (u, v) is not A perfect elimination ordering (PEO) in a graph is an ordering of the vertices of the graph such that, $\forall v$: v and its neighbors that occur after v in the order (later) form a clique. A graph is chordal if and only if it has a perfect elimination ordering. Optimal vertex coloring of the graph: first fit: col[i] = smallest color that is not used by any of the neighbours earlier in PEO. Max clique = Chromatic number = 1+max over number of later neighbours for all vertices. Chromatic polynomial = $(x - d_1)(x - d_2)\dots(x - d_n)$ where d_i = number of neighbors of i later in PEO.

Time: $\mathcal{O}(n + m)$ 34f560, 38 lines

```
vi perfectEliminationOrder(vector<vi>& g) { // 0-indexed, adj list
  int top = 0, n = sz(g);
  vi ord, vis(n), indeg(n);
  vector<vi> bucket(n);
  rep(i, n) bucket[0].pb(i);
  for(int i = 0; i < n; ) {
    while(bucket[top].empty()) --top;
    int u = bucket[top].back();
    bucket[top].pop_back();
    if(vis[u]) continue;
    ord.pb(u);
    vis[u] = 1;
    ++i;
    for(int v : g[u]) {
      if(vis[v]) continue;
      bucket[++indeg[v]].pb(v);
      top = max(top, indeg[v]);
    }
  }
  reverse(all(ord));
  return ord;
}
bool isChordal(vector<vi>& g, vi ord) {//ord = perfectEliminationOrder(g)
  int n = sz(g);
  set<pii> edg;
  rep(i, n) for(auto v:g[i]) edg.insert({i,v});
  vi pos(n); rep(i, n) pos[ord[i]] = i;
  rep(u, n){
    int mn = n;
    for(auto v : g[u]) if(pos[u] < pos[v]) mn = min(mn, pos[v]);
    if(mn != n) {
      int p = ord[mn];
      for(auto v : g[u]) if(pos[v] > pos[u] && v != p && !edg.count({v, p}
          )) return 0;
    }
  }
  return 1;
}
```

ChromaticNumber.h

Description: Calculates chromatic number of a graph represented by a vector of bitmasks. Self loops are not allowed.
Usage: chromaticNumber({6, 5, 3}) // 3-clique
Time: $\mathcal{O}(2^n n)$ 688cb2, 20 lines

```
const int MOD = 1000500103; // big prime
int chromaticNumber(vi g) {
  int n = sz(g);
  if (!n) return 0;
  vi ind(1 << n, 1), s(1 << n);
  rep(i, 1 << n) s[i] = __popcount(i) & 1 ? -1 : 1;
  fwd(i, 1, 1 << n) {
    int ctz = __builtin_ctz(i);
    ind[i] = ind[i - (1 << ctz)] + ind[(i - (1 << ctz)) & ~g[ctz]];
    if (ind[i] >= MOD) ind[i] -= MOD;
  }
  fwd(k, 1, n) {
    ll sum = 0;
    rep(i, 1 << n) {
      s[i] = int((ll)s[i] * ind[i] % MOD);
      sum += s[i];
    }
    if (sum % MOD) return k;
  }
  return n; }
```

7.7 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs b0d5b1, 12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
  if (!P.any()) { if (!X.any()) f(R); return; }
  auto q = (P | X)._Find_first();
  auto cands = P & ~eds[q];
  rep(i,0,sz(eds)) if (cands[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs. f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k,mnk,mxk + 1) for (int i : C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() { init(V), expand(V); return qmax; }
  Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

7.8 Trees

BinaryLifting.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$ bfce85, 25 lines

```
vector<vi> treeJump(vi& P){
  int on = 1, d = 1;
  while(on < sz(P)) on *= 2, d++;
  vector<vi> jmp(d, P);
  rep(i,1,d) rep(j,0,sz(P))
    jmp[i][j] = jmp[i-1][jmp[i-1][j]];
  return jmp;
}
int jmp(vector<vi>& tbl, int nod, int steps){
  rep(i,0,sz(tbl))
    if(steps&(1<<i)) nod = tbl[i][nod];
  return nod;
}
int lca(vector<vi>& tbl, vi& depth, int a, int b) {
  if (depth[a] < depth[b]) swap(a, b);
  a = jmp(tbl, a, depth[a] - depth[b]);
  if (a == b) return a;
  for (int i = sz(tbl); i--;) {
    int c = tbl[i][a], d = tbl[i][b];
    if (c != d) a = c, b = d;
  }
  return tbl[0][a];
}
```

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q)$
"../data-structures/RMQ.h" 0f62fb, 21 lines

```
struct LCA {
  int T = 0;
  vi time, path, ret;
  RMQ<int> rmq;
  LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
  void dfs(vector<vi>& C, int v, int par) {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
      path.push_back(v), ret.push_back(time[v]);
      dfs(C, y, v);
    }
  }
  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
  }
  //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};
```

LCA2.h

Description: Provides LCA, K-th ancestor and isAncestor queries in log(n) time with O(n) memory. 271bb1, 51 lines

```
struct LCA {
  vi par, jmp, depth, pre, post;
  int cnt = 0; LCA() {}
  LCA(vector<vi>& g, int v = 0) :
  par(sz(g), -1), jmp(sz(g), v),
  depth(sz(g)), pre(sz(g)), post(sz(g)) {
    dfs(g, v);
  }
  void dfs(vector<vi>& g, int v) {
    int j = jmp[v], k = jmp[j], x =
      depth[v]+depth[k] == depth[j]*2 ? k : v;
    pre[v] = ++cnt;
    for (auto e : g[v]) if (!pre[e]) {
      par[e] = v; jmp[e] = x;
      depth[e] = depth[v]+1;
      dfs(g, e);
    }
    post[v] = ++cnt;
  }
  int laq(int v, int d) {
    while (depth[v] > d)
      v = depth[jmp[v]] < d ? par[v] : jmp[v];
    return v;
  } // Lowest Common Ancestor; time: O(lg n)
  int operator()(int a, int b) {
    if (depth[a] > depth[b]) swap(a, b);
    b = laq(b, depth[a]);
    while (a != b) {
      if (jmp[a] == jmp[b])
        a = par[a], b = par[b];
      else
        a = jmp[a], b = jmp[b];
    }
    return a;
  } // Check if a is ancestor of b; time: O(1)
  bool isAncestor(int a, int b) {
    return pre[a] <= pre[b] &&
           post[b] <= post[a];
  } // Get distance from a to b; time: O(lg n)
  int distance(int a, int b) {
    return depth[a] + depth[b] -
           depth[operator()(a, b)]*2;
  } // Get k-th vertex on path from a to b,
    // a is 0, b is last; time: O(lg n)
    // Returns -1 if k > distance(a, b)
  int kthVertex(int a, int b, int k) {
    int c = operator()(a, b);
    if (depth[a]-k >= depth[c])
      return laq(a, depth[a]-k);
    k += depth[c]*2 - depth[a];
    return (k > depth[b] ? -1 : laq(b, k)); } };
```

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S|-1$) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.

Time: $\mathcal{O}(|S| \log |S|)$

"LCA.h" 9775a0, 21 lines

```
typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
  static vi rev; rev.resize(sz(lca.time));
  vi li = subset, &T = lca.time;
  auto cmp = [&](int a, int b) { return T[a] < T[b]; };
  sort(all(li), cmp);
  int m = sz(li)-1;
  rep(i,0,m) {
    int a = li[i], b = li[i+1];
    li.push_back(lca.lca(a, b));
  }
  sort(all(li), cmp);
  li.erase(unique(all(li)), li.end());
  rep(i,0,sz(li)) rev[li[i]] = i;
  vpi ret = {pii(0, li[0])};
  rep(i,0,sz(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.lca(a, b)], b);
  }
  return ret;
}
```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.

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

ab0cb2, 32 lines

```
struct HLD {  // 0-indexed
  int N, tim = 0, VALS_EDGES = 0; // change to 1 if needed
  vector<vi> adj; vi par, sz, depth, rt, pos;
  HLD(vector<vi> _adj) : N(SZ(_adj)), adj(_adj), par(N, -1),
    sz(N, 1), depth(N), rt(N), pos(N) { dfsSz(0); dfsHld(0); }
  void dfsSz(int v) {
    if(par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
    for(int &u: adj[v]) {
      par[u] = v, depth[u] = depth[v] + 1;
      dfsSz(u); sz[v] += sz[u];
      if(sz[u] > sz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }
  void dfsHld(int v) {
    pos[v] = tim++;
    for(int u: adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u); dfsHld(u); }
  }
  vector<pii> path(int u, int v) {
    vector<pii> paths;
    for(; rt[u] != rt[v]; v = par[rt[v]]) {
      if(depth[rt[u]] > depth[rt[v]]) swap(u, v);
      paths.pb({pos[rt[v]], pos[v]});
    }
    if(depth[u] > depth[v]) swap(u, v);
    paths.pb({pos[u] + VALS_EDGES, pos[v]});
    return paths;
  }
  pii subtree(int v) {
    return {pos[v] + VALS_EDGES, pos[v] + sz[v] - 1};
  }
};
```

LinkCutTree.h

Description: One-indexed. Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree. Also you can update a value in a vertex, and do standard path and subtree queries. Operation on subtrees must be invertible.

Time: All operations take amortized $\mathcal{O}(\log N)$.

42b461, 82 lines

```
struct SplayTree {
  struct Node {
    int p = 0, ch[2] = {0, 0};
    ll self = 0, path = 0;  // Path aggregates
    ll sub = 0, vir = 0;  // Subtree aggregates
    bool flip = 0;        // Lazy tags
  };
  vector<Node> t;
  SplayTree(int n) : t(n + 1) {}
  void push(int v) {
    if(!v || !t[v].flip) return;
    auto &[l, r] = t[v].ch;
    t[l].flip ^= 1, t[r].flip ^= 1;
    swap(l, r), t[v].flip = 0;
  }
  void pull(int v) {
    auto [l, r] = t[v].ch; push(l), push(r);
    t[v].path = t[l].path + t[v].self + t[r].path;
    t[v].sub = t[v].vir + t[l].sub + t[v].self + t[r].sub;
  }
  void set(int u, int d, int v) {
    t[u].ch[d] = v, t[v].p = u, pull(u);
  }
  void splay(int v) {
    auto dir = [&](int x) {
      int u = t[x].p;
      return t[u].ch[0] == x ? 0 : t[u].ch[1] == x ? 1 : -1;
    };
    auto rotate = [&](int x) {
      int y = t[x].p, z = t[y].p, dx = dir(x), dy = dir(y);
      set(y, dx, t[x].ch[!dx]), set(x, !dx, y);
      if(~dy) set(z, dy, x);
      t[x].p = z;
    };
    for(push(v); ~dir(v); ) {
      int y = t[v].p, z = t[y].p;
      push(z), push(y), push(v);
      int dv = dir(v), dy = dir(y);
      if(~dy) rotate(dv == dy ? y : v);
      rotate(v);
    }
  }
};
struct LinkCut : SplayTree {  // 1-indexed
  LinkCut(int n) : SplayTree(n) {}
  int access(int v) {
    int u = v, x = 0;
    for(; u; x = u, u = t[u].p) {
      splay(u);
      int &ox = t[u].ch[1];
      t[u].vir += t[ox].sub;
      t[u].vir -= t[x].sub;
      ox = x, pull(u);
    }
    return splay(v), x;
  }
  void reroot(int v) { access(v), t[v].flip ^= 1, push(v); }
  void link(int u, int v) {
    reroot(u), access(v);
    t[v].vir += t[u].sub; t[u].p = v, pull(v);
  }
  void cut(int u, int v) {
    reroot(u), access(v); t[v].ch[0] = t[u].p = 0, pull(v);
  }
  // Rooted tree LCA. Returns 0 if u and v are not connected.
  int lca(int u, int v) {
    if(u == v) return u;
    access(u); int ret = access(v); return t[u].p ? ret : 0;
  }
  // Query subtree of u where v is outside the subtree.
  ll getSub(int u, int v) {
    reroot(v), access(u); return t[u].vir + t[u].self;
  }
  ll getPath(int u, int v) {
    reroot(u), access(v); return t[v].path;
  }
  // Update vertex u with value val
  void update(int u, ll val) {
    access(u), t[u].self = val, pull(u);
  }
};
```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

Time: $\mathcal{O}(E \log V)$

"../data-structures/UnionFindRollback.h" 39e620, 60 lines

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
  ll res = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
  }
  for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  rep(i,0,n) par[i] = in[i].a;
  return {res, par};
}
```

GreenHackenbush.h

Description: Given a rooted graph computes the nimber of Green Hackenbush game. Rule one: for a tree, nimber of a root is a xorsum of (son's nimber + 1). Rule two: when u and v lies on a cycle, then they can be contracted.

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

17609f, 19 lines

```
struct GreenHack {
  vector<vi> G; vi pre, low; int T = 0;
  GreenHack(vector<vi> _G) : G(_G), pre(SZ(G)), low(SZ(G)) {}
  int dfs(int v, int p) {
    pre[v] = low[v] = ++T; int ans = 0;
    for(int u: G[v]) {
      if(u == p) { p += SZ(G) + 1; continue; }
      if(!pre[u]) {
        int son = dfs(u, v); low[v] = min(low[v], low[u]);
        if(pre[v] < low[u]) ans ^= (son + 1);
        else ans ^= son ^ 1;
      }
      else if(pre[v] <= pre[u]) ans ^= 1;
      else low[v] = min(low[v], pre[u]);
    }
    return ans;
  }
  int run() { return dfs(0, -1); }
};
```

RedBlueHackenbush.h

Description: Given a rooted tree computes the result of Red-Blue Hackenbush game. If result != 0 then {result} always wins, otherwise the second player to go wins.

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

72d802, 48 lines

```
struct Surreal {
  int value = 0, offset = 0; set<int> powers;
  int sign() {
    int tmp = 2 * value + !powers.empty();
    return tmp < 0 ? -1 : (tmp > 0);
  }
  int add_power(int power) {
    while(power) {
      if(!powers.count(power - offset)) {
        powers.insert(power - offset); break; }
      powers.erase(power - offset); power--;
    }
    return !power;
  }
  void operator+=(const Surreal &v) {
    value += v.value;
    for(int power: v.powers)
```

```
      value += add_power(power + v.offset);
  }
  void divide(int power) {
    offset += power; int to_add = 0;
    FOR(i, 0, power) {
      if(value & 1) to_add += add_power(power - i);
      value >>= 1;
    }
    value += to_add;
  }
  void get_next(int t) {
    int power = max(0, -t * value); value += t * (power + 1);
    if(value == -1 || (value == 1 && powers.empty())) {
      power++, value += t; }
    divide(power);
  }
};
struct RedBlueHack { /* Weights on edges should be -1 or 1 */
  vector<vector<pii>> G; vector<Surreal> ans;
  RedBlueHack(vector<vector<pii>> _G) : G(_G), ans(SZ(G)) {}
  void dfs(int u, int p) {
    for(auto &[v, w]: G[u]) if(v != p) {
      dfs(v, u); ans[v].get_next(w);
      if(SZ(ans[u].powers) < SZ(ans[v].powers)) {
        swap(ans[u], ans[v]); }
      ans[u] += ans[v];
    }
  }
  int run() { dfs(0, 0); return ans[0].sign(); }
};
```

Centroid.h

Description: Computes centroid tree for a given (0-indexed) tree, memory $O(n \log n)$ • child[v] = children of v in centroid tree • par[v] = parent of v in centroid tree (-1 for root) • depth[v] = depth of v in centroid tree (0 for root) = sz(ind[v])-1 • size[v] = size of centroid subtree of v • ind[v][i] = index of vertex v in i-th centroid subtree from root, preorder • subtree[v] = list of vertices in centroid subtree of v • dists[v] = distances from v to vertices in its centroid subtree (in the order of subtree[v]) • neigh[v] = neighbours of v in its centroid subtree • dir[v][i] = index of centroid neighbour that is first vertex on path from centroid v to i-th vertex of centroid subtree (-1 for centroid)

Time: $\mathcal{O}(n \log n)$ 5ba6c3, 47 lines

```
struct CentroidTree {
  vector<vi> child, ind, dists, subtree, neigh, dir;
  vi par, depth, size;
  int root; // Root centroid
  CentroidTree() {}
  CentroidTree(vector<vi>& G)
    : child(sz(G)), ind(sz(G)), dists(sz(G)), subtree(sz(G)), neigh(sz(G))
      , dir(sz(G)), par(sz(G), -2), depth(sz(G)), size(sz(G))
    { root = decomp(G, 0, 0); }
  void dfs(vector<vi>& G, int v, int p) {
    size[v] = 1;
    for(auto e: G[v]) if (e != p && par[e] == -2)
      dfs(G, e, v), size[v] += size[e];
  }
  void layer(vector<vi>& G, int v, int p, int c, int d) {
    ind[v].pb(sz(subtree[c]));
    subtree[c].pb(v); dists[c].pb(d);
    dir[c].pb(sz(neigh[c])-1); // possibly add extra functionalities here
    for(auto e: G[v]) if (e != p && par[e] == -2) {
      if (v == c) neigh[c].pb(e);
      layer(G, e, v, c, d+1);
    }
  }
  int decomp(vector<vi>& G, int v, int d) {
    dfs(G, v, -1);
    int p = -1, s = size[v];
  loop:
    for(auto e: G[v]) {
      if (e != p && par[e] == -2 && size[e] > s/2) {
        p = v; v = e; goto loop;
      }
    }
    par[v] = -1; size[v] = s; depth[v] = d;
    layer(G, v, -1, v, 0);
    for(auto e: G[v]) if (par[e] == -2) {
      int j = decomp(G, e, d+1);
      child[v].pb(j);
      par[j] = v;
    }
    return v;
  }
};
```

7.9 Math

7.9.1 Number of Spanning Trees

Create an $N \times N$ matrix mat, and for each edge $a \to b \in G$, do mat[a][b]--, mat[b][b]++ (and mat[b][a]--, mat[a][a]++ if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

7.9.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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

Geometry (8)

8.1 Geometric primitives

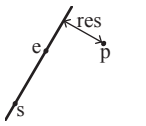
Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.) 47ec0a, 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.

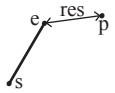


"Point.h" f6bf6b, 4 lines

```
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.

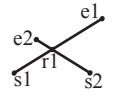


Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```
"Point.h" 5c88f4, 6 lines

```
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.

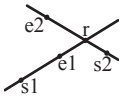


Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```
"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
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" a01f81, 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" 3af81c, 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);
}
```

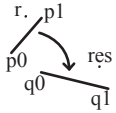
OnSegment.h

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" c597e8, 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" 03a306, 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();
}
```

LineProjectionReflection.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
  P v = b - a;
  return p - v.perp()*(1+refl)*v.cross(p-a)/v.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
```

0f0602, 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" 84d6d3, 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" b0153d, 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;
}
```

CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

"Point.h" e0cfba, 9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

CirclePolygonIntersection.h

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

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

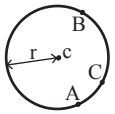
"../../content/geometry/Point.h" a1ee63, 19 lines

```
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 = p + d * t;
    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" 26b98f, 27 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;
}
Pt inCenter( Pt &A,  Pt &B,  Pt &C) {
  double a = norm(B-C), b = norm(C-A), c = norm(A-B);
  return (A * a + B * b + C * c) / (a + b + c);
}
Pt circumCenter( Pt &a,  Pt &b,  Pt &c) {
  Pt bb = b - a, cc = c - a;
  double db=norm2(bb), dc=norm2(cc), d=2*(bb ^ cc);
  return a-Pt(bb.Y*dc-cc.Y*db, cc.X*db-bb.X*dc) / d;
}
Pt othroCenter( Pt &a,  Pt &b,  Pt &c) {
  Pt ba = b - a, ca = c - a, bc = b - c;
  double Y = ba.Y * ca.Y * bc.Y,
    A = ca.X * ba.Y - ba.X * ca.Y,
    x0= (Y+ca.X*ba.Y*b.X-ba.X*ca.Y*c.X) / A,
    y0= -ba.X * (x0 - c.X) / ba.Y + ca.Y;
  return Pt(x0, y0);
}
```

MinimumEnclosingCircle.h

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

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

"circumcircle.h" 09dd0a, 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};
}
```

CirclesUnionArea.h

Description: Returns the area of the sum of circles.

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

"CircleIntersection.h" 8dfc1a, 34 lines

```
template<typename T> // double or long double
T circlesArea(vector<pair<P, T>> c) {
  const T PI = acos((T)-1);
  sort(all(c)); c.erase(unique(all(c)), c.end());
  T res = 0;
  for(auto &[p, r]: c) {
    int cnt = 0, cover = 0;
    vector<pair<T, int>> eve = {{-PI, 0}};
    for(auto &[q, s]: c) if(make_pair(p, r) != make_pair(q, s)) {
      T dst = (p - q).dist();
      if(r + dst <= s) { cover = 1; break; }
      pair<P, P> inters;
      if(!circleInter(p, q, r, s, &inters)) continue;
      T le = (inters.st - p).angle();
      T re = (inters.nd - p).angle();
      cnt += le > re;
      eve.pb({le, 1}), eve.pb({re, -1});
    }
    if(cover) continue;
    sort(eve.begin() + 1, eve.end());
    eve.pb({PI, 0});
    T loc = 0;
    fwd(i, 1, SZ(eve)) {
      if(!cnt) {
        T a = eve[i-1].st, b = eve[i].st;
        loc += r * (b - a) +
          p.cross(P(cos(b)-cos(a), sin(b)-sin(a)));
      }
      cnt += eve[i].nd;
    }
    res += r * loc;
  }
  return res / 2;
}
```

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" 2bf504, 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" f12300, 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" 9706dc, 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;
}
```

PolygonTangents.h

Description: Polygon tangents from a given point. The polygon must be ccw and have no collinear points. Returns a pair of indices of the given polygon. Should work for a point on border (for a point being polygon vertex returns previous and next one).

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

"Point.h" 4d1a00, 21 lines

```
#define pdir(i) (ph ? p - poly[(i)%n] : poly[(i)%n] - p)
#define cmp(i,j) sgn(pdir(i).cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P>
array<int, 2> polygonTangents(vector<P>& poly, P p) {
  auto bs = [&](int ph) {
    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;
  };
  array<int, 2> res = {bs(0), bs(1)};
  if(res[0] == res[1]) res[0] = (res[0] + 1) % SZ(poly);
  if(poly[res[0]] == p) res[0] = (res[0] + 1) % SZ(poly);
  return res;
}
```

Minkowski.h

Description: Computes Minkowski sum of two convex polygons in ccw order. Vertices are required to be in ccw order.

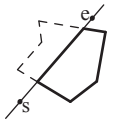
Time: $\mathcal{O}(n+m)$

"Point.h", "Angle.h" e0df19, 18 lines

```
P edgeSeq(vector<P> p, vector<P>& edges) {
  int i = 0, n = sz(p);
  rep(j, n) if (tie(p[i].y, p[i].x) > tie(p[j].y, p[j].x)) i = j;
  rep(j, n) edges.pb(p[(i+j+1)%n] - p[(i+j)%n]);
  return p[i];
}
vector<P> hullSum(vector<P> A, vector<P> B) {
  vector<P> sum, e1, e2, es(sz(A) + sz(B));
  P pivot = edgeSeq(A, e1) + edgeSeq(B, e2);
  merge(all(e1), all(e2), es.begin(), [&](P a, P b){
    return Angle(a.x, a.y) < Angle(b.x,b.y);
  });
  sum.pb(pivot);
  for(auto e: es) sum.pb(sum.back() + e);
  sum.pop_back();
  return sum; //can have collinear vertices!
}
```

PolygonCut.h

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



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

"Point.h", "lineIntersection.h" f2b7d4, 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();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

"Point.h", "sideOf.h" 3931c6, 33 lines

```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

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" 310954, 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])};
}
```

ConvexHullOnline.h

Description: Allows online point insertion. If exists, left vertical segment is included; right one is excluded. To get a lower hull add (-x, -y) instead of (x, y).

Time: amortized $\mathcal{O}(\log n)$ per add

"Point.h" 10c55b, 16 lines

```
using P = Point<ll>;
struct UpperHull : set<P> {
  bool rm(auto it) {
    if (it==begin() || it==end() || next(it)==end() ||
        it->cross(*prev(it), *next(it)) > 0)
      return false;
    erase(it); return true;
  }
  bool add(P p) { // true iff added
    auto [it, ok] = emplace(p);
    if (!ok || rm(it)) return false;
    while (rm(next(it)));
    while (it != begin() && rm(prev(it)));
    return true;
  }
};
```

ConvexHullOnline2.h

Description: Fully dynamic upper / lower convex hull, can be used for computing onion layers. All points should be known in advance. Points on the edges are included in the hull. Return indices are the same as in the input.

Time: $\mathcal{O}\left(\log^2 n\right)$, as fast as other $\mathcal{O}(\log n)$ hulls

"Point.h" 21e548, 70 lines

```
template<class T>
struct DynHull {
  using P = Point<T>;
  struct Node { int l, r; }; vector<P> ps;
  int n; vi in, id; int s; vector<Node> t; vector<T> m;
  DynHull(vector<P> _ps, bool lower = 0, int start = -1)
     : ps(_ps), n(sz(ps)), in(n), id(n) {
    if(start == -1) start = n;
    s = 1; while(s < n) s *= 2;
    t.resize(s * 2, {-1, -1}); m.resize(s);
    vector<pair<P, int>> pts;
    rep(i, n) pts.pb({ps[i] * (lower ? -1 : 1), i});
    sort(all(pts));
    rep(i, n) {
      tie(ps[i], id[i]) = pts[i]; in[id[i]] = i;
      int p = i + s; while((p & 1) ^ 1) p >>= 1;
      m[p >> 1] = ps[i].x;
    }
    rep(i, start) t[s + in[i]] = {in[i], in[i]};
    for (int i = s - 1; i >= 1; i --) pull(i);
  }
  int go(int v) {
    while(t[v].l < 0) v = v * 2 + t[v].l + 3;
    return v;
  }
  void pull(int v) {
    auto crossNegX = [](P a, P b, P c, P d, T x) {
      // change __int128 if using doubles!
      __int128 p = a.cross(b, c), q = b.cross(a, d);
      return p + q == 0 || (d.x - x) * p + (c.x - x) * q <= 0;
    };
    int p = v * 2, q = p + 1;
    if(t[p].l == -1 && t[q].l == -1) t[v] = {-1, -1};
    else if(t[p].l == -1) t[v] = {-2, -2};
    else if(t[q].l == -1) t[v] = {-3, -3};
    else {
      p = go(p), q = go(q);
      while(p < s || q < s) {
        auto [a, b] = t[p]; auto [c, d] = t[q];
        if(a != b && ps[a].cross(ps[b], ps[c]) > 0) {
          p = go(p * 2); }
        else if(c != d && ps[b].cross(ps[c], ps[d]) > 0) {
          q = go(q * 2 + 1); }
        else if(a == b) q = go(q * 2);
        else if(c == d ||
          crossNegX(ps[a], ps[b], ps[c], ps[d], m[v])) {
          p = go(p * 2 + 1); }
        else q = go(q * 2);
      }
      t[v] = {p - s, q - s};
    }
  }
  void add(int i) {
    i = in[i]; int v = i + s; t[v] = {i, i};
    while(v >>= 1) pull(v);
  }
  void del(int i) {
    i = in[i]; int v = i + s; t[v] = {-1, -1};
    while(v >>= 1) {
      if(t[v].l < 0 || t[v].l == i || t[v].r == i) pull(v); }
  }
  void dfs(int v, int l, int r, vi &h) {
    if(v >= s) return h.pb(id[t[v].l]);
    if(l <= t[v].l) dfs(go(v * 2), l, min(t[v].l, r), h);
```

```
    if(t[v].r <= r) dfs(go(v * 2 + 1), max(t[v].r, l), r,  h);
  }
  vi hull() {
    vi h; if(~t[1].l) dfs(go(1), 0, n - 1, h); return h;
  }
};
```

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" c571b8, 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" 71446b, 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: $\bullet$ $(-1,-1)$ if no collision, $\bullet$ $(i,-1)$ if touching the corner i, $\bullet$ (i,i) if along side $(i,i+1)$, $\bullet$ (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" 7cf45b, 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;
}
```

HalfplaneIntersection.h

Description: Online half plane intersection. Works both for ll and long double. Bounding box is optional, but needed for distinguishing bounded vs unbounded. Halfplanes are sorted ccw in HPI.s. Time: O(log n) per add.

"Point.h" f5db92, 98 lines

```
using T = ll; // has to fit 2*|pts|**2
using P = Point<T>; // only cross needed
using SuperT = __int128_t; // has to fit 6*|pts|**3
const SuperT EPS = 1e-12; // |pts| <= 10^6 (for T=dbl)
struct Line {
  T a,b,c;
  Line(T a_=0, T b_=0, T c_=0): a(a_), b(b_), c(c_) {} //ax + by + c >= 0
        (coords <= 10^9)
  Line(P p, P q): a(p.y-q.y), b(q.x-p.x), c(p.cross(q)) {} //p->q ccw (
        coords <= 10^6)
  Line operator- () const {return Line(-a, -b, -c); }
  bool up() const { return a?(a<0):(b>0);}
  P v() const {return P(a,b);}
  P vx() {return P(b,c);} P vy() {return P(a,c);}
  T wek(Line p) const {return v().cross(p.v());}
  bool operator<(Line b) const {
    if (up() != b.up()) return up() > b.up();
    return wek(b) > 0;
  }
};
bool parallel(Line a, Line b) {return !a.wek(b);}
bool same(Line a, Line b) {
  return parallel(a,b) && !a.vy().cross(b.vy()) && !a.vx().cross(b.vx());
}
T weaker (Line a, Line b) {
  if (abs(a.a) > abs(a.b)) return a.c*abs(b.a) - b.c*abs(a.a);
  return a.c*abs(b.b) - b.c*abs(a.b);
}
array<SuperT, 3> intersect(Line a, Line b) {
  SuperT det = a.wek(b);
  SuperT x = a.vx().cross(b.vx());
  SuperT y = a.vy().cross(b.vy());
  // if (T=dbl) return {x / det, -y / det, 1.0};
  if (det > 0) return {x, -y, det};
  return {-x, y, -det};
}
struct HPI {
  bool empty=0, pek=0;
  set<Line> s;
  typedef set<Line>::iterator iter;
  iter next(iter it){return ++it == s.end() ? s.begin() : it;}
  iter prev(iter it){return it == s.begin() ? --s.end() : --it;}
  bool hide(Line a, Line b, Line c) { // do a,b hide c?
    if (parallel(a,b)) {
      if (weaker(a, -b) < 0) empty = 1;
      return 0;
    }
    if (a.wek(b) < 0) swap(a,b);
    auto [rx, ry, rdet] = intersect(a,b);
    auto v = rx*c.a + ry*c.b + rdet*c.c;
    if (a.wek(c) >=0  && c.wek(b) >=0 && v >= -EPS) return 1;
    if (a.wek(c) < 0  && c.wek(b) < 0) {
      if (v < -EPS) empty = 1;
      else if (v <= EPS) pek = 1;
    }
    return 0;
  }
  void delAndMove(iter& i, int nxt) {
    iter j = i;
    if(nxt==1) i = next(i);
    else i = prev(i);
    s.erase(j);
  }
  void add(Line l) {
    if (empty) return;
    if (l.a == 0 && l.b == 0) {
      if (l.c < 0) empty = 1;
      return;
    }
    iter it = s.lower_bound(l);  //parallel
    if(it != s.end() && parallel(*it, l) && it->up() == l.up()) {
      if (weaker(l, *it)>=0) return;
      delAndMove(it,1);
    }
    if(it == s.end()) it = s.begin(); //*it>p
    while(sz(s) >= 2 && hide(l, *next(it), *it))
      delAndMove(it,1);
    if(sz(s)) it = prev(it); //*it<p
    while(sz(s) >= 2 && hide(l, *prev(it), *it))
      delAndMove(it,0);
    if(sz(s) < 2 || !hide(*it, *next(it), l)) s.insert(l);
  }
  int type() { // 0=empty, 1=point, 2=segment,
    if(empty) return 0; // 3=halfline, 4=line,
    if(sz(s) <= 4) { // 5=polygon or unbounded
      vector<Line> r(all(s));
      if(sz(r) == 2 && parallel(r[0], r[1]) && weaker(r[0],-r[1])<0)
        return 0;
      rep(i, sz(r)) rep(j, i) if(same(r[i], r[j])) {
        if(sz(r) == 2) return 4;
        if(sz(r) == 3) return 3;
        if(sz(r) == 4 && same(r[0], r[2]) && same(r[1], r[3])) return 1;
        return 2;
      }
      if(sz(r) == 3 && pek) return 1;
    }
    return 5;
  }
};
```

HalfplaneIntersection2.h

Description: Weszło na zadaniu gdzie trzeba bylo policzyc pole przeciecia

<bits/stdc++.h> b05643, 152 lines

```
using namespace std;
const int N = 100;
typedef long double LD;
const LD inf = 500 + 7;
struct Point {
  LD x, y;
  void read() {
    int xx, yy;
    cin >> xx >> yy;
    x = xx;
    y = yy;
  }
};
Point operator- (const Point& A, const Point& B) { return Point{A.x - B.x,
      A.y - B.y}; }
Point operator+ (const Point& A, const Point& B) { return Point{A.x + B.x,
      A.y + B.y}; }
Point operator* (const Point& A, const LD& value) { return Point{A.x *
     value, A.y * value}; }
int half(const Point& v) {
  if (v.x < 0 && v.y < 0)
    return 1;
  if (v.x >= 0 && v.y < 0)
    return 2;
  if (v.x >= 0 && v.y >= 0)
    return 3;
  if (v.x < 0 && v.y >= 0)
    return 4;
  assert(false);
  return -1;
}
LD dot(const Point& A, const Point& B) { return A.x * B.x + A.y * B.y; }
LD det(const Point& A, const Point& B) { return A.x * B.y - A.y * B.x; }
LD det(const Point& A, const Point& B, const Point& C) { return det(B - A,
      C - A); }
struct Halfplane {
  Point P, v;
  bool inside(const Point& Q) const {
    return det(v, Q - P) > -1e-9;
  }
  bool operator< (const Halfplane& other) const {
    if (half(v) != half(other.v))
      return half(v) < half(other.v);
    return det(v, other.v) > 0;
  }
};
Point inter(const Halfplane& k, const Halfplane& l) {
  return k.P + k.v * (det(l.P - k.P, l.v) / det(k.v, l.v));
}
int n;
Point pts[N][2];
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cout << fixed << setprecision(12);
  cin >> n;
  for (int i = 0; i < n; i++)
    for (int j : {0, 1})
      pts[i][j].read();
  vector<Halfplane> hps;
  for (int i = 0; i < n; i++) {
    for (int j : {0, 1}) {
      for (int k = 0; k < n; k++) if (i != k) {
        for (int l : {0, 1}) {
```

```
        bool ok = true;
        for (int m = 0; m < n; m++) if (m != i && m != k) {
          if (det(pts[i][j], pts[k][l], pts[m][0]) < 0 &&
              det(pts[i][j], pts[k][l], pts[m][1]) < 0) {
            ok = false;
            break;
          }
        }
        if (ok) {
          hps.push_back(Halfplane{pts[i][j], pts[k][l] - pts[i][j]});
        }
      }
    }
  }
  const Point box[4] = {
    Point{+inf, +inf},
    Point{+inf, -inf},
    Point{-inf, -inf},
    Point{-inf, +inf}
  };
  for (int i = 0, j = 3; i < 4; j = i++) {
    hps.push_back(Halfplane{box[i], box[j] - box[i]});
  }
  sort(hps.begin(), hps.end());
  deque<Halfplane> dq;
  for (const Halfplane& hp : hps) {
    while (dq.size() >= 2 && !hp.inside(inter(end(dq)[-1], end(dq)[-2])))
      dq.pop_back();
    while (dq.size() >= 2 && !hp.inside(inter(dq[0], dq[1])))
      dq.pop_front();
    if (dq.size() && fabsl(det(hp.v, dq.back().v)) < 1e-9L) {
      if (dot(hp.v, dq.back().v) < 0) {
        cout << 0.0L << '\n';
        return 0;
      }
      if (!hp.inside(dq.back().P)) {
        dq.pop_back();
        dq.push_back(hp);
      }
    }
    else {
      dq.push_back(hp);
    }
  }
  while (dq.size() >= 3 && !dq[0].inside(inter(end(dq)[-1], end(dq)[-2])))
    dq.pop_back();
  while (dq.size() >= 3 && !end(dq)[-1].inside(inter(dq[0], dq[1])))
    dq.pop_front();
  vector<Point> hull;
  for (int i = 0, j = (int) dq.size() - 1; i < (int) dq.size(); j = i++) {
    hull.push_back(inter(dq[j], dq[i]));
  }
  long double ans = 0;
  for (int i = 0, j = (int) hull.size() - 1; i < (int) hull.size(); j = i
       ++) {
    ans += det(hull[i], hull[j]);
  }
  ans /= 2;
  cout << fabsl(ans) << '\n';
  return 0;
}
```

PointLocation.h

Description: Computes (not necessarily convex) polygon tree structure. Also for each query point computes its location (including boundaries).
Time: $\mathcal{O}(n \log n)$

"directedSegment.h" 39f30f, 47 lines

```
template<class P>
pair<vi,vi> pointLoc(vector<vector<P>> polys, vector<P> pts) {
  vector<tuple<P, int, int>> eve; // {point, event_type, id}
  vector<pair<dirSeg<P>, int>> segs;  // {s, e, poly_id}
  rep(i, sz(polys)) rep(j, sz(polys[i])) {
    dirSeg<P> seg(polys[i][j], polys[i][(j+1)%sz(polys[i])]);
    eve.pb({seg.s,0,sz(segs)}), eve.pb({seg.e,2,sz(segs)});
    segs.pb({seg, i});
  }
  rep(i, sz(pts)) eve.pb({pts[i], 1, i});
  sort(all(eve));
  vi par(sz(polys), -2), ans(sz(pts), -1);
  auto cmp = [](auto a, auto b) {
    return make_pair(a.st.cmp(b.st), a.nd) < make_pair(0, b.nd);
  };
  set<pair<dirSeg<P>, int>, decltype(cmp)> s(cmp);
  for(auto &[_, eve_tp, id]: eve) {
    if(eve_tp == 1) { // point query
      P p = pts[id];
      auto it = s.lower_bound({dirSeg(p, p), 0});
      if(it != s.begin()) { // on vertical segment?
        auto prv = prev(it);
        if(!sgn(p.cross(prv->st.s, prv->st.e))) it--;
      }
      if(it == s.end()) ans[id] = -1;
      else {
        auto [seg, seg_id] = *it;
        int poly_id = segs[seg_id].nd; // strictness there!
        ans[id] = !seg.rev && sgn(p.cross(seg.s, seg.e))
          ? par[poly_id] : poly_id;
      }
    }
    if(eve_tp == 0) { // add segment
      auto it = next(s.insert({segs[id].st, id}).st);
      int poly_id = segs[id].nd;
      if(par[poly_id] == -2) {
        if(it == s.end()) par[poly_id] = -1;
        else {
          int up_rev = it->st.rev, up_id = segs[it->nd].nd;
          par[poly_id] = !up_rev ? par[up_id] : up_id;
        }
      }
    }
    if(eve_tp == 2) s.erase({segs[id].st, id}); // del segment
  }
  return {par, ans};
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" ac41a6, 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;
}
```

FurthestPair.h

Description: assumes n >= 2, O(n).
Time: $\mathcal{O}\left(n^2\right)$

"../convex-hull/main.cpp" d59d33, 12 lines

```
pair<P, P> furthest_pair(vector<P> in) {
  in = hull(in);
  int n = ssize(in), j = 1;
  pair<D, pair<P, P>> ret;
  REP(i, j)
    for(;; j = (j + 1) % n) {
      ret = max(ret, {dist(in[i], in[j]), {in[i], in[j]}});
      if (sign(cross(in[(j + 1) % n] - in[j], in[i + 1] - in[i])) <= 0)
        break;
    }
  return ret.second;
}
```

ManhattanMST.h

Description: Given N points, returns up to 4*N edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights w(p, q) = |p.x - q.x| + |p.y - q.y|. Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.
Time: $\mathcal{O}(N \log N)$

"Point.h" df6f59, 23 lines

```
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
  vi id(sz(ps));
  iota(all(id), 0);
  vector<array<int, 3>> edges;
  rep(k,0,4) {
    sort(all(id), [&](int i, int j) {
      return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
    map<int, int> sweep;
    for (int i : id) {
      for (auto it = sweep.lower_bound(-ps[i].y);
                it != sweep.end(); sweep.erase(it++)) {
        int j = it->second;
        P d = ps[i] - ps[j];
        if (d.y > d.x) break;
        edges.push_back({d.y + d.x, i, j});
      }
      sweep[-ps[i].y] = i;
    }
    for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
  }
  return edges;
}
```

kdTree.h

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

"Point.h" af33f9, 80 lines

```
const int MXN = 100005;
struct KDTree {
  struct Nd {
    int x,y,x1,y1,x2,y2;
    int id,f;
    Nd *L, *R;
  }tree[MXN];
  int n;
  Nd *root;
  LL dis2(int x1, int y1, int x2, int y2) {
    LL dx = x1-x2; LL dy = y1-y2;
    return dx*dx+dy*dy;
  }
  static bool cmpx(Nd& a, Nd& b){ return a.x<b.x; }
  static bool cmpy(Nd& a, Nd& b){ return a.y<b.y; }
  void init(vector<pair<int,int>> ip) {
    n = ip.size();
    for (int i=0; i<n; i++) {
      tree[i].id = i;
      tree[i].x = ip[i].first;
      tree[i].y = ip[i].second;
    }
    root = build_tree(0, n-1, 0);
  }
  Nd* build_tree(int L, int R, int dep) {
    if (L>R) return nullptr;
    int M = (L+R)/2;
    tree[M].f = dep%2;
    nth_element(tree+L, tree+M, tree+R+1,
                tree[M].f ? cmpy : cmpx);
    tree[M].x1 = tree[M].x2 = tree[M].x;
    tree[M].y1 = tree[M].y2 = tree[M].y;
    tree[M].L = build_tree(L, M-1, dep+1);
    if (tree[M].L) {
      tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
    }
    tree[M].R = build_tree(M+1, R, dep+1);
    if (tree[M].R) {
      tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
    }
    return tree+M;
  }
  int touch(Nd* r, int x, int y, LL d2){
    LL dis = sqrt(d2)+1;
    if (x<r->x1-dis || x>r->x2+dis ||
        y<r->y1-dis || y>r->y2+dis)
      return 0;
    return 1;
  }
  void nearest(Nd* r, int x, int y, int &mID, LL &md2){
    if (!r || !touch(r, x, y, md2)) return;
    LL d2 = dis2(r->x, r->y, x, y);
    if (d2 < md2 || (d2 == md2 && mID < r->id)) {
      mID = r->id; md2 = d2;
    }
    // search order depends on split dim
    if ((r->f == 0 && x < r->x) ||
        (r->f == 1 && y < r->y)) {
      nearest(r->L, x, y, mID, md2);
      nearest(r->R, x, y, mID, md2);
    } else {
      nearest(r->R, x, y, mID, md2);
      nearest(r->L, x, y, mID, md2);
    }
  }
  int query(int x, int y) {
    int id = 1029384756;
    LL d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
```

```
  }
}tree;
```

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" eefdf5, 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;
}
```

LiChaoST.h

Description: LiChaoTree (insert line, min on interval) 801fb6, 34 lines

```
struct LiChao_min{
  struct line{
    LL m, c;
    line(LL _m=0, LL _c=0) { m = _m; c = _c; }
    LL eval(LL x) { return m * x + c; }
  };
  struct node{
    node *l, *r; line f;
    node(line v) { f = v; l = r = NULL; }
  };
  typedef node* pnode;
  pnode root; int sz;
#define mid ((l+r)>>1)
  void insert(line &v, int l, int r, pnode &nd){
    if(!nd) { nd = new node(v); return; }
    LL trl = nd->f.eval(l), trr = nd->f.eval(r);
    LL vl = v.eval(l), vr = v.eval(r);
    if(trl <= vl && trr <= vr) return;
    if(trl > vl && trr > vr) { nd->f = v; return; }
    if(trl > vl) swap(nd->f, v);
    if(nd->f.eval(mid) < v.eval(mid)) insert(v, mid + 1, r, nd->r);
    else swap(nd->f, v), insert(v, l, mid, nd->l);
  }
  LL query(int x, int l, int r, pnode &nd){
    if(!nd) return LLONG_MAX;
    if(l == r) return nd->f.eval(x);
    if(mid >= x) return min(nd->f.eval(x), query(x, l, mid, nd->l));
    return min(nd->f.eval(x), query(x, mid + 1, r, nd->r));
  }
  /* -sz <= query_x <= sz */
  void init(int _sz){ sz = _sz + 1; root = NULL; }
  void add_line(LL m, LL c){ line v(m, c); insert(v, -sz, sz, root); }
  LL query(LL x) { return query(x, -sz, sz, root); }
};
```

RectUnion.h

Description: Rectangle union

<iostream>, <vector>, <map>, <cstdio>, <algorithm>, <functional> d5c0eb, 58 lines

```
using namespace std;
#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
struct rectangle { int xl, yl, xh, yh; };
long long rectangle_area(vector<rectangle> rs) {
  vector<int> ys; // coordinate compression
  for (int i = 0; i < rs.size(); ++i) {
    ys.push_back(rs[i].yl);
    ys.push_back(rs[i].yh);
  }
  sort(all(ys)); ys.erase(unique(all(ys)), ys.end());
  int n = ys.size(); // measure tree
  vector<int> C(8*n), A(8*n);
  function<void (int,int,int,int,int,int)> aux =
  [&](int a, int b, int c, int l, int r, int k) {
    if ((a = max(a,l)) >= (b = min(b,r))) return;
    if (a == l && b == r) C[k] += c;
    else {
      aux(a, b, c, l, (l+r)/2, 2*k+1);
      aux(a, b, c, (l+r)/2, r, 2*k+2);
    }
    if (C[k]) A[k] = ys[r] - ys[l];
    else      A[k] = A[2*k+1] + A[2*k+2];
  };
  struct event { int x, l, h, c; }; // plane sweep
  vector<event> es;
  for (auto r: rs) {
    int l = distance(ys.begin(), lower_bound(all(ys), r.yl));
    int h = distance(ys.begin(), lower_bound(all(ys), r.yh));
    es.push_back({r.xl, l, h, +1});
    es.push_back({r.xh, l, h, -1});
  }
  sort(all(es), [](event a, event b) { return a.x != b.x ? a.x < b.x : a.c
       > b.c; });
  long long area = 0, prev = 0;
  for (auto &e: es) {
    area += (e.x - prev) * A[0];
    prev = e.x;
    aux(e.l,e.h,e.c,0,n,0);
  }
  return area;
}
int main() {
  int ncase; scanf("%d", &ncase);
  for (int icase = 0; icase < ncase; ++icase) {
    int n; scanf("%d", &n);
    vector<rectangle> rs(n);
    for (int i = 0; i < n; ++i)
      scanf("%d %d %d %d", &rs[i].xl, &rs[i].yl, &rs[i].xh, &rs[i].yh);
    printf("Case %d: %lld\n", icase+1, rectangle_area(rs));
  }
}
```

SegmentInterDetection.h

Description: KD-tree (2d, can be extended to 3d) c58832, 114 lines

```
int cmp(int x, int y) {
    if (x == y) return 0;
    if (x < y) return -1;
    return 1;
}
struct Point {
    int x, y;
    Point() { x = y = 0; }
    Point(int x, int y) : x(x), y(y) {}
    Point operator - (const Point& a) const {
        return Point(x - a.x, y - a.y);
    }
    int operator % (const Point& a) const {
        return x*a.y - y*a.x;
    }
};
istream& operator >> (istream& cin, Point& p) {
    cin >> p.x >> p.y;
    return cin;
}
struct Segment {
    Point p, q;
    int id;
    double get_y(int x) const {
        if (p.x == q.x) return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (double) (q.x - p.x);
    }
};
istream& operator >> (istream& cin, Segment& s) {
    cin >> s.p >> s.q;
    return cin;
}
bool intersect1d(int l1, int r1, int l2, int r2) {
    if (l1 > r1) swap(l1, r1);
    if (l2 > r2) swap(l2, r2);
    return max(l1, l2) <= min(r1, r2);
}
int ccw(Point a, Point b, Point c) {
    return cmp((b - a) % (c - a), 0);
}
bool intersect(const Segment& a, const Segment& b) {
    return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x)
        && intersect1d(a.p.y, a.q.y, b.p.y, b.q.y)
        && ccw(a.p, a.q, b.p) * ccw(a.p, a.q, b.q) <= 0
        && ccw(b.p, b.q, a.p) * ccw(b.p, b.q, a.q) <= 0;
}
bool operator < (const Segment& a, const Segment& b) {
    int x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - 1e-9;
}
struct Event {
    int x;
    int tp, id;
    Event() {}
    Event(int x, int tp, int id) : x(x), tp(tp), id(id) {}
    bool operator < (const Event& e) const {
        if (x != e.x) return x < e.x;
        return tp > e.tp;
    }
};
set<Segment> s;
vector< set<Segment> :: iterator> where;
set<Segment> :: iterator get_prev(set<Segment>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<Segment> :: iterator get_next(set<Segment>::iterator it) {
    return ++it;
}
pair<int,int> solve(const vector<Segment>& a) {
    int n = SZ(a);
    vector<Event> e;
    REP(i,n) {
        e.push_back(Event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(Event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(ALL(e));
    s.clear();
    where.resize(SZ(a));
```

```
    REP(i,SZ(e)) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<Segment>::iterator next = s.lower_bound(a[id]), prev =
                 get_prev(next);
            if (next != s.end() && intersect(*next, a[id])) {
                return make_pair(next->id, id);
            }
            if (prev != s.end() && intersect(*prev, a[id])) {
                return make_pair(prev->id, id);
            }
            where[id] = s.insert(next, a[id]);
        } else {
            set<Segment>::iterator next = get_next(where[id]), prev =
                 get_prev(where[id]);
            if (next != s.end() && prev != s.end() && intersect(*next, *
                 prev)) {
                return make_pair(prev->id, next->id);
            }
            s.erase(where[id]);
        }
    }
    return make_pair(-1, -1);
}
```

8.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards. 3058c3, 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. 8058ae, 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: $\mathcal{O}\left(n^2\right)$

"Point3D.h" 5b45fc, 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. 611f07, 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);
}
```

Strings (9)

KMP.h

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.

Time: $\mathcal{O}(n)$ d4375c, 16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}
vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

Zfunc.h

Description: z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

Time: $\mathcal{O}(n)$ ee09e2, 12 lines

```
vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).

Time: $\mathcal{O}(N)$ e7ad79, 13 lines

```
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi,2> p = {vi(n+1), vi(n)};
  rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
    int t = r-i+!z;
    if (i<r) p[z][i] = min(t, p[z][l+t]);
    int L = i-p[z][i], R = i+p[z][i]-!z;
    while (L>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
  }
  return p;
}
```

MainLorentz.h

Description: Main-Lorentz algorithm for finding all squares in given word; Results are in compressed form: (b, e, l) means that for each b <= i < e there is square at position i of size 2l. Each square is present in only one interval.

Time: $\mathcal{O}(nlgn)$ 46fbbc, 46 lines

```
struct Sqr {
  int begin, end, len;
};
vector<Sqr> lorentz(const string &s) {
  vector<Sqr> ans;
  vi pos(sz(s) / 2 + 2, -1);
  fwd(mid, 1, sz(s)) {
    int part = mid & ~(mid - 1), off = mid - part;
    int end = min(mid + part, sz(s));
    auto a = s.substr(off, part);
    auto b = s.substr(mid, end - mid);
    string ra(a.rbegin(), a.rend());
    string rb(b.rbegin(), b.rend());
    rep(j, 2) {
      // Set # to some unused character!
      vi z1 = Z(ra, true);
      vi z2 = Z(b + "#" + a, true);
      z1.pb(0);
      z2.pb(0);
      rep(c, sz(a)) {
        int l = sz(a) - c;
        int x = c - min(l - 1, z1[l]);
        int y = c - max(l - z2[sz(b) + c + 1], j);
        if (x > y)
          continue;
        int sb = (j ? end - y - l * 2 : off + x);
        int se = (j ? end - x - l * 2 + 1 : off + y + 1);
        int &p = pos[l];
        if (p != -1 && ans[p].end == sb)
          ans[p].end = se;
        else
          p = sz(ans), ans.pb({sb, se, l});
      }
      a.swap(rb);
      b.swap(ra);
    }
  }
  return ans;
}
```

Lyndon.h

Description: Compute Lyndon factorization for s; Word is simple iff it's stricly smaller than any of it's nontrivial suffixes. Lyndon factorization is division of string into non-increasing simple words. It is unique.

Time: $\mathcal{O}(n)$ 688c1c, 12 lines

```
vector<string> duval(const string &s) {
  int n = sz(s), i = 0;
  vector<string> ret;
  while (i < n) {
    int j = i + 1, k = i;
    while (j < n && s[k] <= s[j])
      k = (s[k] < s[j] ? i : k + 1), j++;
    while (i <= k)
```

```
      ret.pb(s.substr(i, j - k)), i += j - k;
  }
  return ret;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$ d07a42, 8 lines

```
int minRotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(k,0,N) {
    if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
    if (s[a+k] > s[b+k]) { a = b; break; }
  }
  return a;
}
```

SuffixArray.h

Description: Builds suffix array for a string. `sa[i]` is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size $n+1$, and `sa[0] = n`. The `lcp` array contains longest common prefixes for neighbouring strings in the suffix array: `lcp[i] = lcp(sa[i], sa[i-1])`, `lcp[0] = 0`. The input string must not contain any zero bytes.
Time: $\mathcal{O}(n \log n)$ 769289, 23 lines

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim=256) { // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)), y(n), ws(max(n, lim)), rank(n);
    x.push_back(0), sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    rep(i,1,n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$ aae0b8, 50 lines

```
struct SuffixTree {
  enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
  int toi(char c) { return c - 'a'; }
  string a; // v = cur node, q = cur position
  int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;
  void ukkadd(int i, int c) { suff:
    if (r[v]<=q) {
      if (t[v][c]==-1) { t[v][c]=m;  l[m]=i;
        p[m++]=v; v=s[v]; q=r[v];  goto suff; }
      v=t[v][c]; q=l[v];
    }
    if (q==-1 || c==toi(a[q])) q++; else {
      l[m+1]=i;  p[m+1]=m;  l[m]=l[v];  r[m]=q;
      p[m]=p[v];  t[m][c]=m+1;  t[m][toi(a[q])]=v;
      l[v]=q;  p[v]=m;  t[p[m]][toi(a[l[m]])]=m;
      v=s[p[m]];  q=l[m];
      while (q<r[m]) { v=t[v][toi(a[q])];  q+=r[v]-l[v]; }
      if (q==r[m])  s[m]=v;  else s[m]=m+2;
      q=r[v]-(q-r[m]);  m+=2;  goto suff;
    }
  }
  SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
  }
  // example: find longest common substring (uses ALPHA = 28)
  pii best;
  int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
      mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
      best = max(best, {len, r[node] - len});
    return mask;
  }
  static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
  }
};
```

Runs.h

Description: Find all (i, p) such that `s.substr(i,p) == s.substr(i+p,p)`. No two intervals with the same period intersect or touch.
Usage: `solve("aaabababa") // {{0, 1, 1}, {2, 5, 2}}`
Time: $\mathcal{O}(N \log N)$ 3e559b, 14 lines

```
vector<array<int, 3>> solve(string s) {
  int N = SZ(s); SuffixArray A, B;
  A.init(s); reverse(all(s)); B.init(s);
  vector<array<int, 3>> runs;
  for(int p = 1; 2*p <= N; ++p) { // do in O(N/p) for period p
    for(int i = 0, lst = -1; i+p <= N; i += p) {
      int l = i-B.getLCP(N-i-p,N-i), r = i-p+A.getLCP(i,i+p);
      // getLCP(i,j) is lcp of suffixes starting at i and j
      if (l > r || l == lst) continue;
      runs.pb({lst = l,r,p}); // for each i in [l,r],
    } // s.substr(i,p) == s.substr(i+p,p)
  }
  return runs;
}
```

Eertree.h

Description: Famous Bohun's Eertree 26a87b, 24 lines

```
const int N = 1e6 + 5, A = 26;
int nxt[N][A], fail[N], last[N], len[N], cnt, par[N];
char s[N];
void prepare(int n) { // 0 <= i <= n + 1
  rep(i, 0, n + 1) FOR(j, 0, A) nxt[i][j] = 0;
  s[0] = '#'; // CUSTOM
  last[0] = cnt = fail[0] = fail[1] = 1;
  len[1] = -1;
}
void add(int n){
  int c = s[n] - 'a'; // CUSTOM
  int v = last[n - 1];
  while(s[n - len[v] - 1] != s[n]) v = fail[v];
  if(!nxt[v][c]){
    int now = ++cnt, k = fail[v];
    len[now] = len[v] + 2;
    while(s[n - len[k] - 1] != s[n]) k = fail[k];
    fail[now] = nxt[k][c]; nxt[v][c] = now;
    par[now] = v;
  }
  last[n] = nxt[v][c];
}
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$. f35677, 66 lines

```
struct AhoCorasick {
  enum {alpha = 26, first = 'A'}; // change this!
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(next)); }
  };
  vector<Node> N;
  vi backp;
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c : s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);
    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i,0,alpha) {
        int &ed = N[n].next[i], y = N[prev].next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
            = N[y].end;
          N[ed].nmatches += N[y].nmatches;
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    for (char c : word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
  }
  vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

ALCS.h

Description: All-substrings common sequences algorithm. Given strings A and B, algorithm computes: $C(i,j,k) = |LCS(A[:i), B[j:k))|$ in compressed form; To describe the compression, note that: 1. $C(i,j,k-1) <= C(i,j,k) <= C(i,j,k-1)+1$ 2. If $j < k$ and $C(i,j,k) = C(i,j,k-1)+1$, then $C(i,j+1,k) = C(i,j+1,k-1)+1$ 3. If $j >= k$, then $C(i,j,k) = 0$ This allows us to store just the following: $ih(i,k)$ = min j s.t. $C(i,j,k-1) < C(i,j,k)$
Time: $\mathcal{O}(nm)$ 8aadab, 58 lines

```
struct ALCS {
  string A, B;
  vector<vi> ih;
  // Precompute compressed matrix; time: O(nm)
  ALCS(string s, string t) : A(s), B(t) {
    int n = sz(A), m = sz(B);
    ih.resize(n + 1, vi(m + 1));
    iota(all(ih[0]), 0);
    fwd(l, 1, n + 1) {
      int iv = 0;
      fwd(j, 1, m + 1) {
        if (A[l - 1] != B[j - 1]) {
          ih[l][j] = max(ih[l - 1][j], iv);
          iv = min(ih[l - 1][j], iv);
        } else {
          ih[l][j] = iv;
          iv = ih[l - 1][j];
        }
      }
    }
  }
  // Compute |LCS(A[:i), B[j:k))|; time: O(k-j)
  // Note: You can precompute data structure
  // to answer these queries in O(log n)
```

```
  // or compute all answers for fixed 'i'.
  int operator()(int i, int j, int k) {
    int ret = 0;
    fwd(q, j, k) ret += (ih[i][q + 1] <= j);
    return ret;
  }
  // Compute subsequence LCS(A[:i), B[j:k));
  // time: O(k-j)
  string recover(int i, int j, int k) {
    string ret;
    while (i > 0 && j < k) {
      if (ih[i][k--] <= j) {
        ret.pb(B[k]);
        while (A[--i] != B[k])
          ;
      }
    }
    reverse(all(ret));
    return ret;
  }
  // Compute LCS'es of given prefix of A,
  // and all prefixes of given suffix of B.
  // Returns vector L of length |B|+1 s.t.
  // L[k] = |LCS(A[:i), B[j:k))|; time: O(|B|)
  vi row(int i, int j) {
    vi ret(sz(B) + 1);
    fwd(k, j + 1, sz(ret)) ret[k] = ret[k - 1] + (ih[i][k] <= j);
    return ret;
  }
};
```

AllSubstringLCS.h

Description: Computes LCS of s and t[i, j]
Time: $\mathcal{O}(NM)$ 23ce75, 17 lines

```
const int N = 2002;
int f[N][N], g[N][N], ans[N][N];
void all_substring_lcs(string s, string t) {
  int n = SZ(s), m = SZ(t);
  s = "#" + s; t = "#" + t;
  rep(i, 1, m) f[0][i] = i;
  rep(i, 1, n) rep(j, 1, m) {
    if(s[i] == t[j]) {
      f[i][j] = g[i][j - 1]; g[i][j] = f[i - 1][j];
    } else {
      f[i][j] = max(f[i - 1][j], g[i][j - 1]);
      g[i][j] = min(g[i][j - 1], f[i - 1][j]);
    }
  }
  rep(i, 1, m) for(int j = i, a = 0; j <= m; ++j) {
    a += i > f[n][j]; ans[i][j] = a; }
}
```

PalindromicTree.h

Description: Computes plaindromic tree: for each end position in the string we store longest palindrome ending in that position. `link` is the suffix palindrome links, eg ababa -> aba. Can be used to compute shortest decomposition of strings to palindromes in O(n log n) time - use [DP] lines.
Time: $\mathcal{O}(N)$ eb3607, 38 lines

```
constexpr int ALPHA = 26;
struct PalTree {
  vi txt; //;Node 0=empty pal (root of even), 1="-1" pal (of odd)
  vi len{0, -1}; // Lengths of palindromes
  vi link{1, 0}; // Suffix palindrome links, eg [ababa] -> [aba]
  vector<array<int, ALPHA>> to{{}, {}}; // egdes, ex: aba -c> cabac
  int last{0}; // Current node (max suffix pal)
  vi diff{0, 0};  //[DP] len[i]-len[link[i]]
  vi slink{0, 0}; //[DP] like link but to having different 'diff'
  vi series{0, 0};//[DP] dp for series (groups of pals with =diff)
  vi ans{0};      //[DP] ans for prefix
  int ext(int i) {
    while(len[i]+2>sz(txt) || txt[sz(txt)-len[i]-2]!=txt.back())
      i = link[i];
    return i;
  }
  void add(int x) {//x in [0,ALPHA), time O(1) or O(lg n) for DP
    txt.pb(x); last = ext(last);
    if(!to[last][x]) {
      len.pb(len[last] + 2);
      link.pb(to[ext(link[last])][x]);
      to[last][x] = sz(to);
      to.pb({});
      diff.pb(len.back() - len[link.back()]); //[DP]
      slink.pb(diff.back() == diff[link.back()] ? slink[link.back()] :
            link.back()); //[DP]
      series.pb(0); //[DP]
    }
    last = to[last][x];
    ans.pb(INT_MAX); //[DP]
    for(int i = last; len[i] > 0; i = slink[i]) { //[DP]
      series[i] = ans[sz(ans) - len[slink[i]] - diff[i] - 1]; //[DP]
      if(diff[i] == diff[link[i]]) //[DP]
        series[i] = min(series[i], series[link[i]]); //[DP]
      //For even only palindromes set ans only for even sz(txt) //[DP]
      ans.back() = min(ans.back(), series[i] + 1); //[DP]
    }
  }
};
```

Various (10)

10.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$ edce47, 23 lines

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
}
void removeInterval(set<pii>& is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->second;
  if (it->first == L) is.erase(it);
  else (int&)it->second = L;
  if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$ 9e9d8d, 19 lines

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

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$ 753a4c, 19 lines

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

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$ 9155b4, 11 lines

```
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) a = mid; // (A)
    else b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$ 2932a0, 17 lines

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

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$ b20ccc, 16 lines

```
int knapsack(vi w, int t) {
  int a = 0, b = 0, x;
  while (b < sz(w) && a + w[b] <= t) a += w[b++];
  if (b == sz(w)) return a;
  int m = *max_element(all(w));
  vi u, v(2*m, -1);
  v[a+m-t] = b;
  rep(i,b,sz(w)) {
    u = v;
    rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
    for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
      v[x-w[j]] = max(v[x-w[j]], j);
  }
  for (a = t; v[a+m-t] < 0; a--) ;
  return a;
}
```

HilbertMO.h

Description: Packing. 7646cf, 35 lines

```
// Modified MO's queries sorting algorithm,
// slightly better results than standard.
// Allows to process q queries in O(n*sqrt(q))
struct Query {
  int begin, end;
};
// Get point index on Hilbert curve
ll hilbert(int x, int y, int s, ll c = 0) {
  if (s <= 1) return c;
```

```
  s /= 2; c *= 4;
  if (y < s)
    return hilbert(x&(s-1), y, s, c+(x>=s)+1);
  if (x < s)
    return hilbert(2*s-y-1, s-x-1, s, c);
  return hilbert(y-s, x-s, s, c+3);
}
// Get good order of queries; time: O(n lg n)
vi moOrder(vector<Query>& queries, int maxN) {
  int s = 1;
  while (s < maxN) s *= 2;
  vector<ll> ord;
  for( auto &q : queries)
    ord.pb(hilbert(q.begin, q.end, s));
  vi ret(sz(ord));
  iota(all(ret), 0);
  sort(all(ret), [&](int l, int r) {
    return ord[l] < ord[r];
  });
  return ret;
}
```

Packing.h

Description: Packing. 03b70d, 50 lines

```
// Utilities for packing precomputed tables.
// Encodes 13 bits using two characters.
// Writer out; out.ints(-123, 8);
// out.done(); cout << out.buf;
struct Writer {
  string buf;
  int cur = 0, has = 0;
  void done() {
    buf.pb(char(cur%91 + 35));
    buf.pb(char(cur/91 + 35));
    cur = has = 0;
  } // Write unsigned b-bit integer.
  void intu(uint64_t v, int b) {
    assert(b == 64 || v < (1ull<<b));
    while (b--) {
      cur |= (v & 1) << has;
      if (++has == 13) done();
      v >>= 1;
    }
  } // Write signed b-bit integer (sign included)
  void ints(ll v, int b) {
    intu(v < 0 ? -v*2+1 : v*2, b);
  }
};
// Reader in("packed_data"); int first = in.ints(8);
struct Reader {
  const char *buf;
  ll cur = 0;
  Reader(const char *s) : buf(s) {}
  // Read unsigned b-bit integer.
  uint64_t intu(int b) {
    uint64_t n = 0;
    rep(i, b) {
      if (cur < 2) {
        cur = *buf++ + 4972;
        cur += *buf++ * 91;
      }
      n |= (cur & 1) << i;
      cur >>= 1;
    }
    return n;
  } // Read signed b-bit integer (sign included).
  ll ints(int b) {
    auto v = intu(b);
    return (v%2 ? -1 : 1) * ll(v/2);
  } };
```

Int128IO.h

Description: Packing. a481d3, 15 lines

```
istream& operator>>(istream& i, __int128& x) {
  char s[50], *p = s;
  for (i >> s, x = 0, p += *p < 48; *p;)
    x = x*10 + *p++ - 48;
  if (*s == 45) x = -x;
  return i;
}
ostream& operator<<(ostream& o, __int128 x) {
  if (x < 0) o << '-', x = -x;
  char s[50] = {}, *p = s+49;
  for (; x > 9; x /= 10) *--p = char(x%10+48);
  return o << ll(x) << p;
} // Note: Doesn't work for INT128_MIN!
```

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 $[0, 2b)$. 751a02, 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. 7b3c70, 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;
}
```

Code.py

Description: Python cheatsheet 24 lines

```
int(eval("1/3".replace("/","//"))) # parser string num
from fractions import Fraction
from decimal import *
getcontext().prec = 250 # set precision (MAX_PREC)
getcontext().Emax = 250 # set exponent limit (MAX_EMAX)
getcontext().rounding = ROUND_FLOOR # set round floor
itwo,two,N = Decimal(0.5),Decimal(2),200
def angle(cosT):
  for i in range(N):
    cosT = ((cosT + 1) / two) ** itwo
  sinT = (1 - cosT * cosT) ** itwo
  return sinT * (2 ** N)
pi = angle(Decimal(-1))
Fraction('3.1415926535897932').limit_denominator(1000)
format(10, '0.10f') # set precision
x = Fraction(1, 2)
y = Fraction(1)
print(x.as_integer_ratio()) # print (1, 2)
print(x.denominator == 1) # return true if x is integer
print(x.__round__())
print(float(x)) # print 0.5
```

TestDay.h

Description: testing

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

"../../data-structures/ordered-set/main.cpp" 2f76b1, 47 lines

```
LL llmul(LL a, LL b, LL m) {
  return (a * b - (LL)((long double) a * b / m) * m + m) % m;
}
void test_int128() {
  __int128 x = (1llu << 62);
  x *= x;
  string s;
  while(x) {
    s += char(x % 10 + '0');
    x /= 10;
  }
  assert(s == "6123155844692190646693568552397467621 2");
}
void test_float128() {
  __float128 x = 4.2;
  assert(abs(double(x * x) - double(4.2 * 4.2)) < 1e-9);
}
void test_clock() {
  long seeed = chrono::system_clock::now().time_since_epoch().count();
  (void) seeed;
  auto start = chrono::system_clock::now();
  while(true) {
    auto end = chrono::system_clock::now();
    int ms = int(chrono::duration_cast<chrono::milliseconds>(end - start).
        count());
    if(ms > 420)
      break;
  }
}
void test_rd() {
  // czy jest sens to testowac?
  mt19937_64 my_rng(0);
  auto rd = [&](int l, int r) {
    return uniform_int_distribution<int>(l, r)(my_rng);
  };
  assert(rd(0, 0) == 0);
}
void test_policy() {
  ordered_set<int> s;
  s.insert(1);
  s.insert(2);
  assert(s.order_of_key(1) == 0);
  assert(*s.find_by_order(1) == 2);
}
void test_math() {
  constexpr long double pi = acosl(-1);
  assert(3.14 < pi && pi < 3.15);
}
```

Mics.h

Description: random things af924a, 6 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int randint(int a, int b) { return uniform_int_distribution<int>(a, b)(rng
    ); }
__builtin_popcountll // ile jedynek w zapisie bitowym
__builtin_clzll // ile zer przed pierwsza jedynka __clz(2) = 1
__builtin_parityll() // liczba jedynek mod 2
__builtin_mul_overflow(a,b,&h) // mnozenie, ale jak sie wywali to daje
    true
```

10.3 Dynamic programming

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

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

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.

Time: $\mathcal{O}\left((N + (hi - lo))\log N\right)$ d38d2b, 18 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }
  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

AliensTrick.h

Description: Optimize dp where you want "k things with minimal cost". The slope of f(k) must be non increasing. Provide a function g(lambda) that computes the best answer for any k with costs increased by lambda. 71bca3, 8 lines

```
ll aliens(ll k, auto g) { // returns f(k)
  ll l = 0, r = 1e11; // make sure lambda range [l, r) is ok (r > max
      slope etc)
  while (l + 1 < r) {
    ll m = (l + r) / 2;
    (g(m - 1) + k <= g(m) ? l : r) = m;
  }
  return g(l) - l * k; // return l if you want the optimal lambda
}
```

10.3.1 Small 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).
- `for(ll l = 1; l <= n; l++) { ll r = n / (n / l); ... l = r; }` loops over all intervals having the same value of `n / i`.
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `FOR(b, 0, K) FOR(i, 0, 1 << K) if(i & 1 << b) D[i] += D[i ^ (1 << b)];` computes all sums of subsets.
- To jest wroclawski West Coast
  Serce mi pierdolnie caly czas se zwiekszam tempo
  Ludzie robia wow, mowia, ze rozpierdol
  Chcesz mi dawac rady, typie wez sie pierdol