"Alexandru Ioan Cuza" University of Iasi

Endgame

Cristian Vintur, Denis Banu, Matei Chiriac

ACM-ICPC World Finals 2021

November 2022

Contest (1)

templateMatei.txt

126 lines

```
Ordered set

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
     tree_order_statistics_node_update

using namespace __gnu_pbds;

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;

Random

#include <algorithm>
#include <chrono>
#include <iostream>
#include <random>
#include <vector>

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());

Turn stack DS into queue DS

When provided a B update (add B), we just push it to the top of
      S.
When provided an A
update: If S already had A on top, we just pop. Otherwise, we
     begin the following process (which I'll call "fixing"):
pop from S and save all the elements we popped, until we popped
      an equal amount of A's and B's,
or until no A's remain in the stack; empty stack
or only B's remain (we can keep an index of this position in
     the stack, which will only increase).
Then, push back all the elements we popped, where we first push
      all B's,
then all A's (we use commutativity here).
Since the top of S had a B, and we were asked to pop an
     existing A, after fixing,
the topmost element will be an A which we pop.

struct queue_dsu
{
  int bottom;
  vector<pos> act;
  stack_dsu s;

  void init(int n)
  {
    s.init(n);
  }
  void fmerge(int x, int y)
  {
    act.push_back(pos(x, y));
    s.fmerge(x, y);
  }

  void move_bottom()
  {
    while(bottom<act.size() && act[bottom].t=='B')
      bottom++;
  }
  void reverse_upd()
  {
    for(int i=0;i<act.size();i++)
    {
      s.pop();
      act[i].t='A';
    }

    reverse(act.begin(), act.end());

    for(int i=0;i<act.size();i++)
          s.fmerge(act[i].x, act[i].y);

        bottom=0;
    }
    void fix()
    {
        if(act.empty() || act.back().t=='A')
            return;

        move_bottom();
        vector<pos> va, vb;
        vb.push_back(act.back());
        act.pop_back();
        s.pop();

        while(va.size()!=vb.size() && act.size()>bottom)
        {
            if(act.back().t=='A')
                va.push_back(act.back());
            else
                vb.push_back(act.back());

            act.pop_back();
            s.pop();
        }

        reverse(va.begin(), va.end());
        reverse(vb.begin(), vb.end());

        for(auto it : vb)
        {
            act.push_back(it);
            s.fmerge(it.x, it.y);
        }
        for(auto it : va)
        {
            act.push_back(it);
            s.fmerge(it.x, it.y);
        }
        move_bottom();
    }
    void pop()
    {
        move_bottom();
        if(bottom==act.size())
            reverse_upd();

        fix();
        act.pop_back();
        s.pop();
    }
};
```

gen.py

21 lines

```
import sys
from random import randint, choice, shuffle, uniform, sample
from math import gcd, sqrt
from string import ascii_lowercase, ascii_uppercase

chars = ['0', '1']

def random_string(n, chars = ascii_lowercase):
  return "".join([choice(chars) for _ in range(n)])

def print_list(l, sep = " "):
  print(sep.join([str(item) for item in l]))

def gen_graph_edges(n, m):
  edges = []
  for i in range(1, n):
    edges.append((randint(0, i - 1), i))
  for nr in range(n, m + 1):
    edges.append((randint(0, n - 1), randint(0, n - 1)))
  return edges
```

stresstest.sh

31 lines

```
#/bin/bash

i=0
while true
do
  #python3 gen.py >in
  #./gen >in
  ./generators/graph >in
  ./c <in >out
  ./d <in >ok
  #python3 verif.py

  #if [ $? -eq 1 ]; then
  #   echo $?
  #   exit 1
  #fi

  if ! diff out ok; then
    echo $?
    exit 1
  fi

  #if ((i == 1000)); then
  # exit 0
  #fi

  let i=i+1
  if ((i % 1 == 0)); then
    echo $i
  fi
done
```

Mathematics (2)

2.1 Equations

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

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

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

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

2.2 Recurrences

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

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

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

2.3 Trigonometry

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

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

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

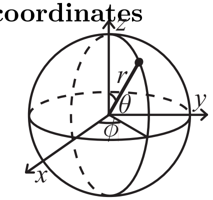
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Linear algebra

2.5.1 Matrix inverse

The inverse of a 2x2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

In general:

$$A^{-1} = \frac{1}{det(A)} A^*$$

where $A^*_{i,j} = (-1)^{i+j} * \Delta_{i,j}$ and $\Delta_{i,j}$ is the determinant of matrix A crossing out line i and column j.

2.6 Derivatives/Integrals

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

Integration by parts:

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

2.7 Sums

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

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

2.8 Series

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

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

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

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

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

2.9 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.9.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

Poisson distribution

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

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

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

2.9.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.10 Markov chains

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

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

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

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

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

Data structures (3)

binomialHeap.cpp
Description: wtf
200 lines

```
const int INF = 2000000001;
const int NMAX = 101;

struct Node {
    int key, degree;
    Node *child, *sibling, *parent;
};

Node* newNode( int val ){
    Node* temp = new Node;
    temp -> key = val;
    temp -> degree = 0;
    temp -> child = temp -> sibling = temp -> parent = NULL;
    return temp;
}

class BinomialHeap{

    list < Node* > H;

    list < Node* > :: iterator get_root(){

        list < Node* > :: iterator it, it_max;
        Node* vmax = newNode( -INF );

        for( it = H.begin(); it != H.end(); ++it )
            if( (*it) -> key > vmax -> key ){
                vmax = *it;
                it_max = it;
            }

        return it_max;
    }

    void delete_root( Node* tree, BinomialHeap& heap ){

        if( tree -> degree == 0 ){
            delete tree;
            return;
        }
```

```
        Node* temp = tree;

        tree -> child -> parent = NULL;
        heap.H.push_front( tree -> child );

        tree = tree -> child;
        while( tree -> sibling ){
            tree -> sibling -> parent = NULL;
            heap.H.push_front( tree -> sibling );
            tree = tree -> sibling;
        }
        delete temp;
    }

    void merge_tree( Node* tree1, Node* tree2 ){

        if( tree1 -> key < tree2 -> key )
            swap ( *tree1, *tree2 );

        tree2 -> sibling = tree1 -> child;
        tree2 -> parent = tree1;
        tree1 -> child = tree2;
        tree1 -> degree++;

    }

    void adjust(){

        if( H.size() <= 1 ) return;

        list < Node* > :: iterator prev;
        list < Node* > :: iterator curr;
        list < Node* > :: iterator next;
        list < Node* > :: iterator temp;

        prev = curr = H.begin();
        curr++;
        next = curr;
        next++;

        while( curr != H.end() ){

            while ( ( next == H.end() || (*next) -> degree > (*
                 curr) -> degree ) && curr != H.end() && (*prev
                 ) -> degree == (*curr) -> degree ){

                merge_tree( *curr, *prev );

                temp = prev;

                if( prev == H.begin() ){
                    prev++;
                    curr++;
                    if( next != H.end() )
                        next++;
                }
                else prev--;

                H.erase( temp );

            }

            prev++;
            if( curr != H.end() ) curr++;
            if( next != H.end() ) next++;
        }
    }
public:

    int top(){
        return (*get_root()) -> key;
    }

    void push( int val ){

        Node *tree = newNode( val );
        H.push_front( tree );
        adjust();
    }

    void heap_union( BinomialHeap& heap2){

        list < Node* > :: iterator it1 = H.begin();
        list < Node* > :: iterator it2 = heap2.H.begin();

        list < Node* > new_heap;

        while( it1 != H.end() && it2 != heap2.H.end() ){
            if( (*it1) -> degree <= (*it2) -> degree ){
                new_heap.push_back( *it1 );
                it1++;
            }
            else{
                new_heap.push_back( *it2 );
                it2++;
            }
        }

        while( it1 != H.end() ){
            new_heap.push_back( *it1 );
            it1++;
        }

        while( it2 != heap2.H.end() ){
            new_heap.push_back( *it2 );
            it2++;
        }

        heap2.H.clear();

        H = new_heap;
        adjust();
    }

    void pop(){

        list < Node* > :: iterator root = get_root();

        BinomialHeap new_heap;
        delete_root( (*root), new_heap );

        H.erase( root );

        heap_union( new_heap );

    }
};

int N, M;
BinomialHeap Heap[NMAX];

int main()
{
    fin >> N >> M;

    int task, h, x, h1, h2;
    for( int i = 1; i <= M; ++i ){
        fin >> task;

        if( task == 1 ){

            fin >> h >> x;
            Heap[h].push( x );

        }
        if( task == 2 ){

            fin >> h;
            fout << Heap[h].top() << '\n';
            Heap[h].pop();
        }
        if( task == 3 ){

            fin >> h1 >> h2;

            Heap[h1].heap_union( Heap[h2] );
        }
    }

    return 0;
}
```

convexhulltrick.cpp

Description: Add lines of the form ax+b and query maximum. Lines should be sorted in increasing order of slope

Time: $\mathcal{O}(\log N)$. 39 lines

```
template<class T = pll, class U = ll>
struct hull {
  struct frac {
    ll x, y;
    frac(ll _x, ll _y) : x(_x), y(_y) {
      if(y < 0) x = -x, y = -y;
    }
    bool operator <(const frac &other) const {
      return 1.0 * x * other.y < 1.0 * other.x * y;
    }
  };
  frac inter(T l1, T l2) { return { l2.se - l1.se, l1.fi - l2.
      fi }; }

  int nr = 0;
  vector<T> v;
  void add(T line) {
    // change signs for min
    if(!v.empty() && v.back().fi == line.fi) {
      if(v.back().se < line.se) v.back() = line;
      return;
    }
    while(nr >= 2 && inter(line, v[nr - 2]) < inter(v[nr - 1],
        v[nr - 2])) --nr, v.pop_back();
    v.push_back(line);
    ++nr;
  }
  U query(ll x) {
    int l, r, mid;
    for(l = 0, r = nr - 1; l < r; ) {
      mid = (l + r) / 2;
      if(inter(v[mid], v[mid + 1]) < frac(x, 1)) l = mid + 1;
      else r = mid;
    }
    // while(p + 1 < nr && eval(v[p + 1], x) < eval(v[p], x))
        ++p;
    return v[l];
  }
  ll eval(T line, ll x) {
```

```
    return line.fi * x + line.se;
  }
};
```

ConvexTree.h

Description: Container where you can add lines of the form a * x + b, and query maximum values at points x. Useful for dynamic programming. To change to minimum, either change the sign of all comparisons, the initialization of T and max to min, or just add lines of form (-a)*X + (-b) instead and negate the result.
Time: $\mathcal{O}(\log(kMax - kMin))$

<bits/stdc++.h> 50 lines

```
using int64 = int64_t;

struct Line {
  int a; int64 b;
  int64 Eval(int x) { return 1LL * a * x + b; }
};
const int64 kInf = 2e18; // Maximum abs(A * x + B)
const int kMin = -1e9, kMax = 1e9; // Bounds of query (x)

struct ConvexTree {
  struct Node { int l, r; Line line; };
  vector<Node> T = { Node{0, 0, {0, -kInf}} };
  int root = 0;

  int update(int node, int b, int e, Line upd) {
    if (node == 0) {
      T.push_back(Node{0, 0, upd});
      return T.size() - 1;
    }

    auto& cur = T[node].line;
    if (cur.Eval(b)>=upd.Eval(b) && cur.Eval(e)>=upd.Eval(e))
      return node;
    if (cur.Eval(b)<=upd.Eval(b) && cur.Eval(e)<=upd.Eval(e))
      return cur = upd, node;

    int m = (b + e) / 2;
    if (cur.Eval(b) < upd.Eval(b)) swap(cur, upd);
    if (cur.Eval(m) >= upd.Eval(m)) {
      int res = update(T[node].r, m + 1, e, upd);
      T[node].r = res; // DO NOT ATTEMPT TO OPTIMIZE
    } else {
      swap(cur, upd);
      int res = update(T[node].l, b, m, upd);
      T[node].l = res; // DO NOT ATTEMPT TO OPTIMIZE
    }
    return node;
  }
  void AddLine(Line l) { root = update(root, kMin, kMax, l); }

  int64 query(int node, int b, int e, int x) {
    int64 ans = T[node].line.Eval(x);
    if (node == 0) return ans;
    int m = (b + e) / 2;
    if (x <= m) ans = max(ans, query(T[node].l, b, m, x));
    if (x > m) ans = max(ans, query(T[node].r, m + 1, e, x));
    return ans;
  }
  int64 QueryMax(int x) { return query(root, kMin, kMax, x); }
};
```

dynamicCHT.cpp

Description: wtf

43 lines

```
const ll is_query = -(1LL << 62);
struct Line {
    ll m, b;
    mutable function<const Line *()> succ;

    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};

struct DynamicHull : public multiset<Line> {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) *
             (y->m - x->m);
    }

    void insert_line(ll m, ll b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y)
            ; };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y))
            ;
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }

    ll eval(ll x) {
        auto l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
};
```

FenwickTree2d.h

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

"FenwickTree.h" 32 lines

```
struct Fenwick2D {
  vector<vector<int>> ys;
  vector<vector<int>> T;
  Fenwick2D(int n) : ys(n + 1) {}

  void FakeUpdate(int x, int y) {
    for (++x; x < (int)ys.size(); x += (x & -x))
      ys[x].push_back(y);
  }
  void Init() {
    for (auto& v : ys) {
      sort(v.begin(), v.end());
      T.emplace_back(v.size());
    }
  }
  int ind(int x, int y) {
    auto it = lower_bound(ys[x].begin(), ys[x].end(), y);
    return distance(ys[x].begin(), it);
  }
  void Update(int x, int y, int val) {
    for (++x; x < (int)ys.size(); x += (x & -x))
    for (int i = ind(x,y); i < (int)T[x].size(); i += (i & -i))
      trees[x][i] = trees[x][i] + val;
  }
  int Query(int x, int y) {
    int sum = 0;
    for (; x > 0; x -= (x & -x))
    for (int i = ind(x,y); i > 0; i -= (i & -i))
      sum = sum + T[x][i];
    return sum;
  }
};
```

implicitTreapsMaxValeriu.cpp

Description: None
Usage: ask Djok

<bits/stdc++.h> 140 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 200005;

int i, n, q, a[N], x, y, z;

struct node;
typedef node* ln;

struct node
{
    int pr;

    int v;
    int dp;

    int id,sz;

    ln l, r;

    node (int v=0) : pr(rand() * rand() * rand()),v(v),l(0),r
         (0) { upd(); }

    void upd()
    {
        dp = v;
        if (l) dp=max(dp,l->dp);
        if (r) dp=max(dp,r->dp);

        sz = 1;
        if (l) sz+=l->sz;
        id = sz;
        if (r) sz+=r->sz;
    }
};

ln root;

void split ( ln t, int x, ln &l, ln &r)
{
    l=r=0;
    if (!t) return;
    if (t->id <= x)
    {
        split(t->r, x - t->id, t->r, r);
        l = t;
```

```
    } else
    {
        split(t->l, x, l, t->l);
        r = t;
    }

    t->upd();
}

ln merge(ln l, ln r)
{
    if (!l || !r) return (l?l:r);

    if (l->pr > r->pr)
    {
        l->r = merge(l->r, r);
        l->upd();
        return l;
    } else
    {
        r->l = merge(l,r->l);
        r->upd();
        return r;
    }
}

void insert(int x, int p)
{
    ln l,r;
    split(root,p,l,r);
    root = merge(merge(l,new node(x)),r);
}

void erase(int p)
{
    ln l,r,t;
    split(root,p,l,r);
    split(r,1,r,t);
    root = merge(l,t);
}

int query(int x, int y)
{
    ln l,t,r;
    split(root, x, l, t);
    split(t, y-x+1, t, r);

    int m = t->dp;

    root = merge(merge(l,t),r);
    return m;
}

void show(ln t)
{
    if (!t) return;
    show(t->l);
    cout<<' '<<t->v;
    show(t->r);
}

int getPoz(int p)
{
    ln l,r,t;
    split(root,p,l,r);
    split(r,1,r,t);
    int ans = r->v;
    r = merge(r,t);
    root = merge(l, r);
    return ans;
}

int main() {
  srand(time(0));
  root = 0;
  scanf("%d %d", &n, &q);
  for(i = 0; i < n; ++i) scanf("%d", a + i), insert(a[i], i);
  while(q--) {
    scanf("%d %d %d", &x, &y, &z);
    if(x == 1) {
      printf("%d\n", query(y - 1, z - 1));
      continue;
    }

    --z; x = getPoz(z);
    erase(z);
    if(y == 1) {
      insert(x, n - 1);
    } else {
      insert(x, 0);
    }
  }
  return 0;
}
```

LazySegmentTree.h

Description: wtf

38 lines

```
struct ST {
  int n;
  vector<int> st, lazy;

  ST(int n) : n(n), st(4 * n), lazy(4 * n) {}

  void push(int node) {
    st[2 * node] += lazy[node];
    lazy[2 * node] += lazy[node];
    st[2 * node + 1] += lazy[node];
    lazy[2 * node + 1] += lazy[node];
    lazy[node] = 0;
  }

  void update(int node, int l, int r, int a, int b, int val) {
    if(a <= l && r <= b) { st[node] += val; lazy[node] += val;
        return; }
    push(node);

    int mid = (l + r) / 2;
    if(a <= mid) update(2 * node, l, mid, a, b, val);
    if(mid + 1 <= b) update(2 * node + 1, mid + 1, r, a, b, val
        );

    st[node] = min(st[2 * node], st[2 * node + 1]);
  }

  int query(int node, int l, int r, int a, int b) {
    if(a <= l && r <= b) return st[node];
    push(node);

    int mid = (l + r) / 2;
    int v1 = (a <= mid ? query(2 * node, l, mid, a, b) : INF);
    int v2 = (mid + 1 <= b ? query(2 * node + 1, mid + 1, r, a,
         b) : INF);
    return min(v1, v2);
  }

  void update(int a, int b, int val) { update(1, 1, n, a, b,
      val); }
  int query(int a, int b) { return query(1, 1, n, a, b); }
};
```

LineContainer.h

Description: Container where you can add lines of the form ax+b, and query maximum values at points x. For each line, also keeps a value p, which is the last (maximum) point for which the current line is dominant. (obviously, for the last line, p is infinity) Useful for dynamic programming.

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

<bits/stdc++.h> 35 lines

```
using T = long long;

bool QUERY;
struct Line {
  mutable T a, b, p;
    T Eval(T x) const { return a * x + b; }
  bool operator<(const Line& o) const {
    return QUERY ? p < o.p : a < o.a;
  }
};

struct LineContainer : multiset<Line> {
  // for doubles, use kInf = 1/.0, div(a, b) = a / b
  const T kInf = numeric_limits<T>::max();
  T div(T a, T b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = kInf; return false; }
    if (x->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
    else x->p = div(y->b - x->b, x->a - y->a);
    return x->p >= y->p;
  }
  void InsertLine(T a, T b) {
    auto nx = insert({a, b, 0}), it = nx++, pv = it;
    while (isect(it, nx)) nx = erase(nx);
    if (pv != begin() && isect(--pv, it)) isect(pv, it = erase(
        it));
    while ((it = pv) != begin() && (--pv)->p >= it->p)
      isect(pv, erase(it));
  }
  T EvalMax(T x) {
    assert(!empty());
    QUERY = 1; auto it = lower_bound({0,0,x}); QUERY = 0;
    return it->Eval(x);
  }
};
```

pairingHeap.cpp

Description: wtf

131 lines

```
const int NMAX = 101;
const int INF = 2000000001;

ifstream fin("mergeheap.in");
ofstream fout("mergeheap.out");

struct Node{
    int key;
    Node *child, *sibling;

    Node( int x ) : key( x ), child( NULL ), sibling( NULL ) {}
};

class PairingHeap{

    Node *root;

    Node* merge_heap( Node* H1, Node* H2 ){
```

```
        if( H1 == NULL ){
            H1 = H2;
            return H1;
        }
        if( H2 == NULL ) return H1;

        if( H1 -> key < H2 -> key )
            swap( H1, H2 );

        H2 -> sibling = H1 -> child;
        H1 -> child = H2;

        return H1;
    }

    Node* two_pass_merge( Node *_Node ){

        if( _Node == NULL || _Node -> sibling == NULL )
            return _Node;

        Node *heap_1, *heap_2, *next_pair;

        heap_1 = _Node;
        heap_2 = _Node -> sibling;
        next_pair = _Node -> sibling -> sibling;

        heap_1 -> sibling = heap_2 -> sibling = NULL;

        return merge_heap( merge_heap( heap_1, heap_2 ),
             two_pass_merge( next_pair ) );
    }
public:

    PairingHeap() : root( NULL ) {}

    PairingHeap( int _key ){
        root = new Node( _key );
    }

    PairingHeap( Node* _Node ) : root( _Node ) {}

    int top(){
        return root -> key;
    }

    void merge_heap( PairingHeap H ){

        if( root == NULL ){
            root = H.root;
            return;
        }
        if( H.root == NULL ) return;

        if( root -> key < H.root -> key )
            swap( root, H.root );

        H.root -> sibling = root -> child;
        root -> child = H.root;
        H.root = NULL;
    }

    void push( int _key ){
        merge_heap( PairingHeap( _key ) );
    }

    void pop(){

        Node* temp = root;
        root = two_pass_merge( root -> child );

        delete temp;
    }

    void heap_union( PairingHeap &H ){
        merge_heap( H );
        H.root = NULL;
    }
};

int N, M;

PairingHeap Heap[NMAX];

int main()
{

    fin >> N >> M;

    int task, h, x, h1, h2;
    for( int i = 1; i <= M; ++i ){

        fin >> task;

        if( task == 1 ){
            fin >> h >> x;

            Heap[h].push( x );
        }
        if( task == 2 ){
            fin >> h;

            fout << Heap[h].top() << '\n';
            Heap[h].pop();
        }
        if( task == 3 ){
            fin >> h1 >> h2;

            Heap[h1].heap_union( Heap[h2] );
        }
    }

    return 0;
}
```

RMQ.h

Description: wtf

18 lines

```
struct RMQ {
  vector<vector<int>> rmq;

  void build(const vector<int> &vec) {
    rmq.push_back(vec);

    for(int i = 1; (1 << i) <= vec.size(); ++i) {
      rmq.push_back(vector<int>(vec.size()));
      for(int j = 0; j + (1 << i) - 1 < vec.size(); ++j)
        rmq[i][j] = gcd(rmq[i - 1][j], rmq[i - 1][j + (1 << (i
             - 1))]);
    }
  }

  int query(int l, int r) {
    int d = 31 - __builtin_clz(r - l + 1);
    return gcd(rmq[d][l], rmq[d][r - (1 << d) + 1]);
  }
};
```

slopeTrick.cpp

Description: Given an array a, on operation means increase or decrease an element by one What is the minimum number of operations to make it strictly increasing? Remove line "a -= i" for non-decreasing

28 lines

```
int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int n, a;

  cin >> n;

  priority_queue<int> q;

  ll ans = 0;
  for(int i = 0; i < n; ++i) {
    cin >> a;
    a -= i;

    q.push(a);
    q.push(a);

    ans += q.top() - a;

    q.pop();
  }

  cout << ans << '\n';

  return 0;
}
```

Treap.h

Description: wtf

69 lines

```
struct Treap {
  int key, pri, cnt, mn, mx, s;
  Treap *l, *r;

  Treap(int key) : key(key), pri(rand()) {
    cnt = s = 1;
    mn = mx = key;
    l = r = nullptr;
  }
};
using PTreap = Treap*;

void update(PTreap node) {
  // TODO: update node considering children are correct
}

void split(PTreap node, int key, PTreap &l, PTreap &r) {
  if(!node) return void(l = r = nullptr);

  if(key < node->key) split(node->l, key, l, node->l), r = node
      ;
  else split(node->r, key, node->r, r), l = node;

  update(node);
}

void merge(PTreap &node, PTreap l, PTreap r) {
  if(!l || !r) return void(node = (l ? l : r));

  if(l->pri < r->pri) merge(r->l, l, r->l), node = r;
  else merge(l->r, l->r, r), node = l;
```

```
  update(node);
}

bool addIfExists(PTreap node, int key) {
  if(!node) return false;
  if(node->key == key) return ++node->cnt, update(node), true;

  auto res = addIfExists(key < node->key ? node->l : node->r,
       key);
  update(node);
  return res;
}

void add(PTreap &node, PTreap item) {
  if(!node) return void(node = item);

  if(item->pri > node->pri) split(node, item->key, item->l,
       item->r), node = item;
  else add(item->key < node->key ? node->l : node->r, item);

  update(node);
}

void erase(PTreap &node, int key) {
  if(!node) return;

  if(node->key == key) {
    --node->cnt;
    if(!node->cnt) merge(node, node->l, node->r);
  } else erase(key < node->key ? node->l : node->r, key);

  if(node) update(node);
}

void print(PTreap node, string indent = "") {
  if(!node) return;
  cout << indent << ' ' << node->key << ' ' << node->cnt << '\n
       ';
  print(node->l, indent + " ");
  print(node->r, indent + " ");
}
```

UnionFind.h

Description: wtf

20 lines

```
struct UnionFind {
  vector<int> fth, sz;

  UnionFind(int n) {
    fth.assign(n, -1);
    sz.assign(n, 1);
  }

  int root(int x) { return fth[x] == -1 ? x : fth[x] = root(fth
       [x]); }

  bool join(int a, int b) {
    a = root(a);
    b = root(b);
    if(a == b) return false;

    if(sz[a] < sz[b]) fth[a] = b, sz[b] += sz[a];
    else fth[b] = a, sz[a] += sz[b];
    return true;
  }
};
```

queueDSU.cpp

Description: ???

173 lines

```
struct stack_upd
{
    int x, y;

    stack_upd(int x, int y)
    {
        this->x = x;
        this->y = y;
    }
};

struct stack_dsu
{
    stack<stack_upd> upd;

    vector<int> par;
    vector<int> sz;

    void init(int n)
    {
        par.resize(n+1);
        sz.resize(n+1);

        for(int i=1;i<=n;i++)
        {
            par[i] = i;
            sz[i] = 1;
        }
    }

    int anc(int x)
    {
        if(par[x] == x)
            return x;
        return anc(par[x]);
    }

    void fmerge(int x, int y)
    {
        x = anc(x);
        y = anc(y);

        if(sz[x] < sz[y])
            swap(x, y);

        upd.push( stack_upd(x, y) );

        if(x != y)
        {
            par[y] = x;
            sz[x] += sz[y];
        }
    }

    void pop()
    {
        int x = upd.top().x;
        int y = upd.top().y;
        upd.pop();

        if(x != y)
        {
            par[y] = y;
            sz[x] -= sz[y];
        }
    }
};

struct queue_upd
{
    char type;
    int x, y;

    queue_upd(int x, int y, char type = 'B')
    {
        this->type = type;
        this->x = x;
        this->y = y;
    }
};

struct queue_dsu
{
    int nrA, nrB;
    vector<queue_upd> upd;
    stack_dsu ds;

    void init(int n)
    {
        nrA = nrB = 0;
        ds.init(n);
    }

    void fmerge(int x, int y)
    {
        nrB++;

        upd.push_back(queue_upd(x, y));
        ds.fmerge(x, y);
    }

    void reverse_updates()
    {
        for(int i=0; i<(int)upd.size(); i++)
            ds.pop();

        reverse(upd.begin(), upd.end());

        for(auto &it : upd)
        {
            it.type = 'A';
            ds.fmerge(it.x, it.y);
        }

        nrA = (int)upd.size();
        nrB = 0;
    }

    void fix()
    {
        vector< queue_upd > auxA;
        vector< queue_upd > auxB;

        while( !upd.empty() )
        {
            queue_upd it = upd.back();

            ds.pop();
            upd.pop_back();

            if( it.type == 'A' )
                auxA.push_back(it);
            else
                auxB.push_back(it);

            if(!auxA.empty() && auxA.size() == auxB.size() )
```

```
            break;

        if( (int)auxA.size() == nrA )
            break;
    }

    reverse(auxA.begin(), auxA.end());
    reverse(auxB.begin(), auxB.end());

    for(auto it : auxB)
    {
        ds.fmerge(it.x, it.y);
        upd.push_back(it);
    }

    for(auto it : auxA)
    {
        ds.fmerge(it.x, it.y);
        upd.push_back(it);
    }
}

void pop()
{
    if(upd.back().type != 'A')
    {
        if(nrA)
            fix();
        else
            reverse_updates();
    }

    ds.pop();
    upd.pop_back();
    nrA --;
}
};
```

dynamicConnectivity.cpp

Description: ???

<bits/stdc++.h> 141 lines

```
struct stack_dsu
{
    int par[100005];
    int sz[100005];

    stack< pair<int, int> > upd;

    void init(int n)
    {
        for(int i=0;i<n;i++)
        {
            par[i] = i;
            sz[i] = 1;
        }
    }

    int anc(int p)
    {
        if(par[p] == p)
            return p;
        return anc(par[p]);
    }

    void fmerge(int x, int y)
    {
        x = anc(x);
        y = anc(y);

        if(sz[x] < sz[y])
            swap(x, y);

        upd.push({x, y});

        if(x != y)
        {
            par[y] = x;
            sz[x] += sz[y];
        }
    }

    void pop()
    {
        int x = upd.top().first;
        int y = upd.top().second;
        upd.pop();

        if(x == y)
            return;

        par[y] = y;
        sz[x] -= sz[y];
    }
};

struct edge
{
    int x, y, val;

    edge(int x, int y, int val)
    {
        this->x = x;
        this->y = y;
        this->val = val;
    }
};

int n, q;

map< pair<int, int>, pair<int, int> > m;

stack_dsu dsu[12];
vector< vector<edge> > seg;

pair<int, int> qv[100005];
int ans[100005];

void upd(int stt, int drt, int st, int dr, int p, edge val)
{
    //cout<<st<<' '<<dr<<'\n';

    if(stt == st && drt == dr)
    {
        seg[p].push_back(val);
        return;
    }

    int mij = (st+dr)/2;

    if(drt <= mij)
        upd(stt, drt, st, mij, 2*p, val);

    else if(stt > mij)
        upd(stt, drt, mij+1, dr, 2*p+1, val);

    else
    {
        upd(stt, mij, st, mij, 2*p, val);
        upd(mij+1, drt, mij+1, dr, 2*p+1, val);
    }
}

void solve(int st, int dr, int p)
{
    for(auto it : seg[p])
    {
        for(int i=it.val; i<10; i++)
            dsu[i].fmerge(it.x, it.y);
    }

    if(st == dr)
    {
        if(qv[st] != make_pair(0, 0))
        {
            int x = qv[st].first;
            int y = qv[st].second;

            for(int i=0; i<10; i++)
            {
                if(dsu[i].anc(x) == dsu[i].anc(y))
                {
                    ans[st] = i;
                    break;
                }
            }
        }
    }
    else
    {
        int mij = (st+dr)/2;

        solve(st, mij, 2*p);
        solve(mij+1, dr, 2*p+1);
    }

    for(auto it : seg[p])
    {
        for(int i=it.val; i<10; i++)
            dsu[i].pop();
    }
}
```

Numerical (4)

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first 2*n terms of the recurrence. Very useful for guessing linear recurrences after brute-force / backtracking the first terms. Should work on any field. Numerical stability for floating-point calculations is not guaranteed.
Usage: `BerlekampMassey({0, 1, 1, 3, 5, 11}) => {1, 2}`

<bits/stdc++.h>, "ModOps.h" 29 lines

```
vector<ModInt> BerlekampMassey(vector<ModInt> s) {
  int n = s.size();
  vector<ModInt> C(n, 0), B(n, 0);
  C[0] = B[0] = 1;

  ModInt b = 1; int L = 0;
  for (int i = 0, m = 1; i < n; ++i) {

    ModInt d = s[i];
    for (int j = 1; j <= L; ++j)
      d = d + C[j] * s[i - j];

    if (d.get() == 0) { ++m; continue; }
```

```
    auto T = C; ModInt coef = d * inv(b);
    for (int j = m; j < n; ++j)
      C[j] = C[j] - coef * B[j - m];

    if (2 * L > i) { ++m; continue; }

    L = i + 1 - L; B = T; b = d; m = 1;
  }

  C.resize(L + 1); C.erase(C.begin());
  for (auto& x : C) x = ModInt(0) - x;

  return C;
}
```

Polynomial.h

Description: Different operations on polynomials. Should work on any field.

<bits/stdc++.h> 114 lines

```
using TElem = double;
using Poly = vector<TElem>;

TElem Eval(const Poly& P, TElem x) {
  TElem val = 0;
  for (int i = (int)P.size() - 1; i >= 0; --i)
    val = val * x + P[i];
  return val;
}

// Differentiation
Poly Diff(Poly P) {
  for (int i = 1; i < (int)P.size(); ++i)
    P[i - 1] = i * P[i];
  P.pop_back();
  return P;
}

// Integration
Poly Integrate(Poly p) {
  P.push_back(0);
  for (int i = (int)P.size() - 2; i >= 0; --i)
    P[i + 1] = P[i] / (i + 1);
  P[0] = 0;
  return P;
}

// Division by (X - x0)
Poly DivRoot(Poly P, TElem x0) {
  int n = P.size();
  TElem a = P.back(), b; P.back() = 0;
  for (int i = n--; i--; )
    b = P[i], P[i] = P[i + 1] * x0 + a, a = b;
  P.pop_back();
  return P;
}

// Multiplication modulo X^sz
Poly Multiply(Poly A, Poly B, int sz) {
  static FFTSolver fft;
  A.resize(sz, 0); B.resize(sz, 0);
  auto R = fft.Multiply(A, B);
  R.resize(sz, 0);
  return r;
}

// Scalar multiplication
Poly Scale(Poly P, TElem s) {
  for (auto& x : P)
    x = x * s;
  return P;
}

// Addition modulo X^sz
Poly Add(Poly A, Poly B, int sz) {
  A.resize(sz, 0); B.resize(sz, 0);
  for (int i = 0; i < sz; ++i)
    A[i] = A[i] + B[i];
  return A;
}

// ************************************************
// For Invert, Sqrt, size of argument should be 2^k
// ************************************************

Poly inv_step(Poly res, Poly P, int n) {
  auto res_sq = Multiply(res, res, n);
  auto sub = Multiply(res_sq, P, n);
  res = Add(Scale(res, 2), Scale(sub, -1), n);
  return res;
}

// Inverse modulo X^sz
// EXISTS ONLY WHEN P[0] IS INVERTIBLE
Poly Invert(Poly P) {
  assert(P[0].Get() == 1);
  Poly res(1, 1);          // i.e., P[0]^(-1)

  int n = P.size();
  for (int step = 2; step <= n; step *= 2) {
    res = inv_step(res, P, step);
  }

  // Optional, but highly encouraged
  auto check = Multiply(res, P, n);
  for (int i = 0; i < n; ++i) {
    assert(check[i].Get() == (i == 0));
  }
  return res;
};

// Square root modulo X^sz
// EXISTS ONLY WHEN P[0] HAS SQUARE ROOT
Poly Sqrt(Poly P) {
  assert(P[0].Get() == 1);
  Poly res(1, 1);          // i.e., P[0]^(-1)
  Poly inv(1, 1);          // i.e., P[0]^(1/2)

  int n = P.size();
  for (int step = 2; step <= n; step *= 2) {
    auto now = inv_step(inv, res, step);
    now = Multiply(P, move(now), step);
    res = Add(res, now, step);
    res = Scale(res, (kMod + 1) / 2);
    inv = inv_step(inv, res, step);
  }

  // Optional, but highly encouraged
  auto check = Multiply(res, res, n);
  for (int i = 0; i < n; ++i) {
    assert(check[i].Get() == P[i].Get());
  }
  return res;
}
```

PolyRoots.h

Description: Finds the real roots to a polynomial.

Usage: `Poly p = {2, -3, 1} // x^2 - 3x + 2 = 0`
`auto roots = GetRoots(p, -1e18, 1e18); // {1, 2}`

<bits/stdc++.h>, "Polynomial.h" 26 lines

```
vector<double> GetRoots(Poly p, double xmin, double xmax) {
  if (p.size() == 2) { return {-p.front() / p.back()}; }
  else {
    Poly d = Diff(p);
    vector<double> dr = GetRoots(d, xmin, xmax);
    dr.push_back(xmin - 1);
    dr.push_back(xmax + 1);
    sort(dr.begin(), dr.end());

    vector<double> roots;
    for (auto i = dr.begin(), j = i++; i != dr.end(); j = i++){
      double lo = *j, hi = *i, mid, f;
      bool sign = Eval(p, lo) > 0;
      if (sign ^ (Eval(p, hi) > 0)) {
        // for (int it = 0; it < 60; ++it) {
        while (hi - lo > 1e-8) {
          mid = (lo + hi) / 2, f = Eval(p, mid);
          if ((f <= 0) ^ sign) lo = mid;
          else hi = mid;
        }
        roots.push_back((lo + hi) / 2);
      }
    }
    return roots;
  }
}
```

PolyInterpolate.h

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

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

<bits/stdc++.h>, "Polynomial.h" 15 lines

```
Poly Interpolate(vector<TElem> x, vector<TElem> y) {
  int n = x.size();
  Poly res(n), temp(n);
  for (int k = 0; k < n; ++k)
    for (int i = k + 1; i < n; ++i)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  TElem last = 0; temp[0] = 1;
  for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i) {
    res[i] = res[i] + y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] = temp[i] - last * x[k];
  }
  return res;
}
```

LinearRecurrence.h

Description: Generates the k-th term of a n-th order linear recurrence given the first n terms and the recurrence relation. Faster than matrix multiplication. Useful to use along with Berlekamp Massey.

Usage: `LinearRec<double>({0, 1}, {1, 1}).Get(k)` gives k-th Fibonacci number (0-indexed)

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

<bits/stdc++.h> 43 lines

```
template<typename T>
struct LinearRec {
  using Poly = vector<T>;
  int n; Poly first, trans;

  // Recurrence is S[i] = sum(S[i-j-1] * trans[j])
```

```
  // with S[0..(n-1)] = first
  LinearRec(const Poly &first, const Poly &trans) :
    n(first.size()), first(first), trans(trans) {}

  Poly combine(Poly a, Poly b) {
    Poly res(n * 2 + 1, 0);
    // You can apply constant optimization here to get a
    // ~10x speedup
    for (int i = 0; i <= n; ++i)
      for (int j = 0; j <= n; ++j)
        res[i + j] = res[i + j] + a[i] * b[j];

    for (int i = 2 * n; i > n; --i)
      for (int j = 0; j < n; ++j)
        res[i - 1 - j] = res[i - 1 - j] + res[i] * trans[j];

    res.resize(n + 1);
    return res;
  }

  // Consider caching the powers for multiple queries
  T Get(int k) {
    Poly r(n + 1, 0), b(r);
    r[0] = 1; b[1] = 1;

    for (++k; k; k /= 2) {
      if (k % 2)
        r = combine(r, b);
      b = combine(b, b);
    }

    T res = 0;
    for (int i = 0; i < n; ++i)
      res = res + r[i + 1] * first[i];
    return res;
  }
};
```

FFT.h

Description: Fast Fourier transform. Also includes a function for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. a and b should be of roughly equal size. Does about 1.2s for 10^6 elements. Rounding the results of conv works if $(|a|+|b|)\max(a,b) < \sim 10^9$ (in theory maybe 10^6); you may want to use an NTT from the Number Theory chapter instead.
Time: $\mathcal{O}(N \log N)$

<bits/stdc++.h> 76 lines

```
struct FFTSolver {
  using Complex = complex<double>;
  const double kPi = 4.0 * atan(1.0);
  vector<int> rev;

  int __lg(int n) { return n == 1 ? 0 : 1 + __lg(n / 2); }

  void compute_rev(int n, int lg) {
    rev.resize(n); rev[0] = 0;
    for (int i = 1; i < n; ++i) {
      rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (lg - 1));
    }
  }

  vector<Complex> fft(vector<Complex> V, bool invert) {
    int n = V.size(), lg = __lg(n);
    if ((int)rev.size() != n) compute_rev(n, lg);

    for (int i = 0; i < n; ++i) {
      if (i < rev[i])
        swap(V[i], V[rev[i]]);
    }

    for (int step = 2; step <= n; step *= 2) {
      const double ang = 2 * kPi / step;
      Complex eps(cos(ang), sin(ang));
      if (invert) eps = conj(eps);

      for (int i = 0; i < n; i += step) {
        Complex w = 1;
        for (int a = i, b = i+step/2; b < i+step; ++a, ++b) {
          Complex aux = w * V[b];
          V[b] = V[a] - aux;
          V[a] = V[a] + aux;
          w *= eps;
        }
      }
    }

    return V;
  }

  vector<Complex> transform(vector<Complex> V) {
    int n = V.size();
    vector<Complex> ret(n);
    Complex div_x = Complex(0, 1) * (4.0 * n);

    for (int i = 0; i < n; ++i) {
      int j = (n - i) % n;
      ret[i] = (V[i] + conj(V[j]))
        * (V[i] - conj(V[j])) / div_x;
    }

    return ret;
  }

  vector<int> Multiply(vector<int> A, vector<int> B) {
    int n = A.size() + B.size() - 1;
    vector<int> ret(n);
    while (n != (n & -n)) ++n;

    A.resize(n); B.resize(n);
    vector<Complex> V(n);
    for (int i = 0; i < n; ++i) {
      V[i] = Complex(A[i], B[i]);
    }

    V = fft(move(V), false);
    V = transform(move(V));
    V = fft(move(V), true);

    for (int i = 0; i < (int)ret.size(); ++i)
      ret[i] = round(real(V[i]));
    return ret;
  }
};
```

FST.h

Description: Fast Subset transform. Useful for performing the following convolution: R[a op b] += A[a] * B[b], where op is either of AND, OR, XOR. P has to have size $N = 2^n$, for some n.
Time: $\mathcal{O}(N \log N)$

<bits/stdc++.h> 16 lines

```
vector<int> Transform(vector<int> P, bool inv) {
  int n = P.size();
  for (int step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) {
      for (int j = i; j < i + step; ++j) {
        int u = P[j], v = P[j + step];
        tie(P[j], P[j + step]) =
          inv ? make_pair(v - u, u) : make_pair(v, u + v); // AND
          inv ? make_pair(v, u - v) : make_pair(u + v, u); // OR
          make_pair(u + v, u - v);                          // XOR
      }
    }
  }
  // if (inv) for (auto& x : P) x /= n; // XOR only
  return P;
}
```

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.

9 lines

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

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$

50 lines

```
namespace Gauss {
  // Transforms a matrix into its row echelon form
  // Returns a vector of pivots (for each variable)
  // or -1 if free variable
  vector<int> ToRowEchelon(vector<vector<double>> &M) {
    int cons = M.size(), vars = M[0].size() - 1;
    vector<int> pivot(vars, -1);

    int cur = 0;
    for (int var = 0; var < vars; ++var) {
      if (cur >= cons) continue;

      for (int con = cur + 1; con < cons; ++con)
        if(M[con][var] > M[cur][var])
          swap(M[con], M[cur]);

      if (abs(M[cur][var]) > kEps) {
        pivot[var] = cur;
        double aux = M[cur][var];

        for (int i = 0; i <= vars; ++i)
          M[cur][i] /= aux;

        for (int con = 0; con < cons; ++con) {
          if (con != cur) {
            double mul = M[con][var];
            for (int i = 0; i <= vars; ++i)
              M[con][i] -= mul * M[cur][i];

          }
        }
        ++cur;
      }
    }

    return pivot;
  }

  // Returns the solution of a system
```

```
  // Will not check if feasible
  // Will change matrix
  vector<double> SolveSystem(vector<vector<double>> &M) {
    int vars = M[0].size() - 1;
    auto pivs = ToRowEchelon(M);

    vector<double> solution(pivs.size());
    for(int i = 0; i < solution.size(); ++i)
      solution[i] = (pivs[i] == -1) ? 0.0 : M[pivs[i]][vars];
  }
};
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}\left(N^3\right)$

20 lines

```
using int64 = int64_t;
const int64 kMod = 12345;

int64 IntDeterminant(vector<vector<int64>>& M) {
  int n = M.size(); int64 ans = 1;
  for (int i = 0; i < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
      while (M[j][i] != 0) { // gcd step
        int64 t = M[i][i] / M[j][i];
        if (t) for (int k = i; k < n; ++k)
          M[i][k] = (M[i][k] - M[j][k] * t) % kMod;
        swap(M[i], M[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + kMod) % kMod;
}
```

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

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 $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}\left(n^3\right)$

36 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 $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ 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" 36 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
        : 0);
  return n;
}
```

Tridiagonal.h

Description: Solves a linear equation system with a tridiagonal matrix with diagonal diag, subdiagonal sub and superdiagonal super, i.e., $x = \text{Tridiagonal}(d, p, q, b)$ solves the equation system

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

The size of diag and b should be the same and super and sub should be one element shorter. T is intended to be double.
This is useful for solving problems on the type

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

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

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

Usage:
```
int n = 1000000;
vector<double> diag(n,-1), sup(n-1,.5), sub(n-1,.5), b(n,1);
vector<double> x = tridiagonal(diag, super, sub, b);
```
Time: $\mathcal{O}(N)$

14 lines

```
template <typename T>
vector<T> Tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
  for (int i = 0; i < b.size() - 1; ++i) {
    diag[i + 1] -= super[i] * sub[i] / diag[i];
    b[i + 1] -= b[i] * sub[i] / diag[i];
  }
  for (int i = b.size(); --i > 0;) {
```

```
    b[i] /= diag[i];
    b[i - 1] -= b[i] * super[i - 1];
  }
  b[0] /= diag[0];
  return b;
}
```

Number theory (5)

5.1 General

mathValeriu.h

Description: None
Usage: ask Djok

118 lines

```
bool isPrime(int x) {
  if(x < 2) return 0;
  if(x == 2) return 1;
  if(x % 2 == 0) return 0;
  for(int i = 3; i * i <= x; i += 2)
    if(x % i == 0) return 0;
  return 1;
}

int mul(int a, int b) {
  return (long long)a * b % MOD;
}

int add(int a, int b) {
  a += b;
  if(a >= MOD) return a - MOD;
  return a;
}

int getPw(int a, int b) {
  int ans = 1;
  for(; b > 0; b /= 2) {
    if(b & 1) ans = mul(ans, a);
    a = mul(a, a);
  }
  return ans;
}

long long modInv(long long a, long long m) {
  if(a == 1) return 1;
  return (1 - modInv(m % a, a) * m) / a + m;
}

long long CRT(vector<long long> &r, vector<long long> &p) {
  long long ans = r[0] % p[0], prod = p[0];
  for(int i = 1; i < r.size(); ++i) {
    long long coef = ((r[i] - (ans % p[i]) + p[i]) % p[i]) *
        modInv(prod % p[i], p[i]) % p[i];
    ans += coef * prod;
    prod *= p[i];
  }
  return ans;
}

long long getPhi(long long n) {
  long long ans = n - 1;
  for(int i = 2; i * i <= n; ++i) {
    if(n % i) continue;
    while(n % i == 0) n /= i;
    ans -= ans / i;
  }
  if(n > 1) ans -= ans / n;
  return ans;
}

// fact is a vector with prime divisors of N - 1 (N here is
    modulo) and N is prime
// the idea is that if N is prime, then N-1 is phi(N), which
    means the cycle has length N - 1
// now, lets try to see if X is a generator
// we know that if x ^ phi(N) == 1 then x ^ 2*phi(N) is also ==
    1, and here we get the idea
// if for some divisor of phi(N), x ^ div == 1, then obviously
    X is not a generator
// because the cycle is not of length N
// good luck to understand this after one year :)
bool isGenerator(int x, int n) {
  if(cmmdc(x, n) != 1) return 0;

  for(auto it : fact)
    if(Pow(x, (n - 1) / it, n) == 1)
      return 0;
  return 1;
}

// Lucas Theorem
// calc COMB(N, R) if N and R is VERY VERY BIG and MOD is PRIME
r -= 2; n += m - 2;
while(r > 0 || n > 0) {
  ans = (1LL * ans * comb(n % MOD, r % MOD)) % MOD;
  n /= MOD; r /= MOD;
}

// GAUSS FOR F2 space
// SZ is the size of basis
void gauss(int mask) {
  for(int i = 0; i < n; ++i) {
    if(!(mask & (1 << i))) continue;
    if(!basis[i]) {
      basis[i] = mask;
      ++sz;
      break;
    }
    mask ^= basis[i];
  }
}

// if A is a permutation of B, then A == B mod 9

bool isSquare(int x) {
  int a = sqrt(x) + 0.5;
  return a * a == x;
}

int getDiscreteLog(int a, int b, int m) {
  if(b == 1) return 0;
  int n = sqrt(m) + 1;
  int an = 1;
  for(int i = 0; i < n; ++i) an = (an * a) % m;
  unordered_map<int, int> vals;
  for(int i = 1, cur = an; i <= n; ++i) {
    if(!vals.count(cur)) vals[cur] = i;
    cur = (cur * an) % m;
  }
  for(int i = 0, cur = b; i <= n; ++i) {
    if(vals.count(cur)) {
      int ans = vals[cur] * n - i;
      return ans;
    }
    cur = (cur * a) % m;
  }
  return -1;
}
```

sieve.cpp

Description: wtf

14 lines

```
bool isPrime[VMAX];
vector<int> primes;

void linearSieve(int n) {
  for(int i = 2; i <= n; ++i) isPrime[i] = true;
  for(int i = 2; i <= n; ++i) {
    if(isPrime[i]) primes.push_back(i);
    for(auto p : primes) {
      if(i * p > n) break;
      isPrime[p * i] = false;
      if(i % p == 0) break;
    }
  }
}
```

5.2 Modular arithmetic

ModMulLL.h

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

19 lines

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

ModSqrt.h

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

"ModPow.h" 30 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1);
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
```

```
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
}
```

5.3 Number theoretic transform

NTT.h

Description: wtf 78 lines

```
template<int P>
struct NTT {
  int root, maxBase;
  std::vector<int> rev, roots{0, 1};

  int power(int base, int e) {
    int res;
    for (res = 1; e > 0; e >>= 1) {
      if (e % 2 == 1) res = 1LL * res * base % P;
      base = 1LL * base * base % P;
    }
    return res;
  }

  void init() {
    for(maxBase = 0; !((P - 1) >> maxBase); ++maxBase);

    for(int root = 3; ; ++root)
      if(power(x, (P - 1) / 2) != 1) {
        return;
      }
  }

  void fft(std::vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
      int k = __builtin_ctz(n) - 1;
      rev.resize(n);
      for (int i = 0; i < n; ++i)
        rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
    }
    for (int i = 0; i < n; ++i)
      if (rev[i] < i) std::swap(a[i], a[rev[i]]);

    if (int(roots.size()) < n) {
      int k = __builtin_ctz(roots.size());
      roots.resize(n);
      while ((1 << k) < n) {
        int e = power(root, (P - 1) >> (k + 1));
        for (int i = 1 << (k - 1); i < (1 << k); ++i) {
          roots[2 * i] = roots[i];
          roots[2 * i + 1] = 1LL * roots[i] * e % P;
        }
        ++k;
      }
    }
    for (int k = 1; k < n; k *= 2) {
      for (int i = 0; i < n; i += 2 * k) {
        for (int j = 0; j < k; ++j) {
          int num = 1LL * a[i + j + k] * roots[k + j] % P;
          a[i + j + k] = (a[i + j] - num + P) % P;
          a[i + j] = (a[i + j] + num) % P;
        }
      }
    }
  }
  void ifft(std::vector<int> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    fft(a);
    int inv = power(n, P - 2);
    for (int i = 0; i < n; ++i)
      a[i] = 1LL * a[i] * inv % P;
  }

  std::vector<int> multiply(std::vector<int> a, std::vector<int
      > b) {
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot) sz *= 2;
    a.resize(sz);
    b.resize(sz);
    fft(a);
    fft(b);
    for (int i = 0; i < sz; ++i) a[i] = 1LL * a[i] * b[i] % P;
    ifft(a);
    a.resize(tot);
    return a;
  }
};
```

5.4 Fast Fourier Transform

fftValeriu.h

Description: wtf 248 lines

```
struct ftype {
  double a, b;

  ftype(double a = 0, double b = 0) : a(a), b(b) {}
  ftype conj() { return ftype(a, -b); }
  friend ftype operator +(const ftype &x, const ftype &y) {
      return ftype(x.a + y.a, x.b + y.b); }
  friend ftype operator -(const ftype &x, const ftype &y) {
      return ftype(x.a - y.a, x.b - y.b); }
  friend ftype operator *(const ftype &x, const ftype &y) {
      return ftype(x.a * y.a - x.b * y.b, x.a * y.b + x.b * y.
      a); }
  friend ftype operator /(const ftype &x, int y) { return ftype
      (x.a / y, x.b / y); }
};

const double PI = acos(-1);

ftype polar(double ang) { return ftype(cos(ang), sin(ang)); }

int rv(int x, int sz) {
  int ans = 0;
  for(int i = 0; i < sz; ++i)
    if(x & (1 << i)) ans |= (1 << (sz - 1 - i));

  return ans;
}

vector<ftype> fft(vector<ftype> p, bool rev = false) {
  int i, sz, n = p.size();
  for(sz = 0; (1 << sz) < n; ++sz);

  for(int i = 0; i < n; ++i)
    if(i < rv(i, sz)) swap(p[i], p[rv(i, sz)]);

  for(int len = 2; len <= n; len <<= 1) {
    ftype wlen = polar((rev ? -1 : 1) * 2 * PI / len);

    for(int i = 0; i < n; i += len) {
      ftype w = 1;
      for(int j = 0; j < len / 2; ++j) {
        ftype u = p[i + j] + w * p[i + j + len / 2];
        ftype v = p[i + j] - w * p[i + j + len / 2];
        p[i + j] = u;
        p[i + j + len / 2] = v;
        w = w * wlen;
      }
    }
  }

  if(rev) {
    for(auto &val : p) val.a /= p.size(), val.b /= p.size();
  }

  return p;
}

vector<int> multiply(const vector<int> &a, const vector<int> &b
    ) {
  int sz = 2 * max(a.size(), b.size());
  while(__builtin_popcount(sz) != 1) ++sz;

  vector<ftype> na, nc;

  na.resize(sz);
  for(int i = 0; i < a.size(); ++i) na[i].a = a[i];
  for(int i = 0; i < b.size(); ++i) na[i].b = b[i];

  auto r = fft(na);

  for(int i = 0; i < r.size(); ++i) {
    ftype x = r[i];
    ftype y = r[i == 0 ? i : r.size() - i].conj();
    ftype ai = (x + y) / 2;
    ftype bi = (x - y) / 2 * ftype(0, -1);
    nc.push_back(ai * bi);
  }

  auto vc = fft(nc, true);

  vector<int> c;
  for(auto val : vc) c.push_back(round(val.a));

  return c;
}

// Tourist FFT
namespace fft {
  typedef double dbl;

  struct num {
    dbl x, y;
    num() { x = y = 0; }
    num(dbl x, dbl y) : x(x), y(y) { }
  };

  inline num operator+(num a, num b) { return num(a.x + b.x, a.
      y + b.y); }
  inline num operator-(num a, num b) { return num(a.x - b.x, a.
      y - b.y); }
  inline num operator*(num a, num b) { return num(a.x * b.x - a
      .y * b.y, a.x * b.y + a.y * b.x); }
  inline num conj(num a) { return num(a.x, -a.y); }
```

```
  int base = 1;
  vector<num> roots = {{0, 0}, {1, 0}};
  vector<int> rev = {0, 1};

  const dbl PI = acosl(-1.0);

  void ensure_base(int nbase) {
    if (nbase <= base) {
      return;
    }
    rev.resize(1 << nbase);
    for (int i = 0; i < (1 << nbase); i++) {
      rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
    }
    roots.resize(1 << nbase);
    while (base < nbase) {
      dbl angle = 2 * PI / (1 << (base + 1));
//      num z(cos(angle), sin(angle));
      for (int i = 1 << (base - 1); i < (1 << base); i++) {
        roots[i << 1] = roots[i];
//        roots[(i << 1) + 1] = roots[i] * z;
        dbl angle_i = angle * (2 * i + 1 - (1 << base));
        roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
      }
      base++;
    }
  }

  void fft(vector<num> &a, int n = -1) {
    if (n == -1) {
      n = a.size();
    }
    assert((n & (n - 1)) == 0);
    int zeros = __builtin_ctz(n);
    ensure_base(zeros);
    int shift = base - zeros;
    for (int i = 0; i < n; i++) {
      if (i < (rev[i] >> shift)) {
        swap(a[i], a[rev[i] >> shift]);
      }
    }
    for (int k = 1; k < n; k <<= 1) {
      for (int i = 0; i < n; i += 2 * k) {
        for (int j = 0; j < k; j++) {
          num z = a[i + j + k] * roots[j + k];
          a[i + j + k] = a[i + j] - z;
          a[i + j] = a[i + j] + z;
        }
      }
    }
  }

  vector<num> fa, fb;

  vector<int> multiply(vector<int> &a, vector<int> &b) {
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for (int i = 0; i < sz; i++) {
      int x = (i < (int) a.size() ? a[i] : 0);
      int y = (i < (int) b.size() ? b[i] : 0);
      fa[i] = num(x, y);
    }
    fft(fa, sz);
    num r(0, -0.25 / sz);
    for (int i = 0; i <= (sz >> 1); i++) {
      int j = (sz - i) & (sz - 1);
      num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
      if (i != j) {
        fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
      }
      fa[i] = z;
    }
    fft(fa, sz);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
      res[i] = fa[i].x + 0.5;
    }
    return res;
  }

  vector<int> multiply_mod(vector<int> &a, vector<int> &b, int
      m, int eq = 0) {
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for (int i = 0; i < (int) a.size(); i++) {
      int x = (a[i] % m + m) % m;
      fa[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fa.begin() + a.size(), fa.begin() + sz, num {0, 0});
    fft(fa, sz);
    if (sz > (int) fb.size()) {
      fb.resize(sz);
    }
    if (eq) {
      copy(fa.begin(), fa.begin() + sz, fb.begin());
    } else {
      for (int i = 0; i < (int) b.size(); i++) {
        int x = (b[i] % m + m) % m;
        fb[i] = num(x & ((1 << 15) - 1), x >> 15);
      }
      fill(fb.begin() + b.size(), fb.begin() + sz, num {0, 0});
      fft(fb, sz);
    }
    dbl ratio = 0.25 / sz;
    num r2(0, -1);
    num r3(ratio, 0);
    num r4(0, -ratio);
    num r5(0, 1);
    for (int i = 0; i <= (sz >> 1); i++) {
      int j = (sz - i) & (sz - 1);
      num a1 = (fa[i] + conj(fa[j]));
      num a2 = (fa[i] - conj(fa[j])) * r2;
      num b1 = (fb[i] + conj(fb[j])) * r3;
      num b2 = (fb[i] - conj(fb[j])) * r4;
      if (i != j) {
        num c1 = (fa[j] + conj(fa[i]));
        num c2 = (fa[j] - conj(fa[i])) * r2;
        num d1 = (fb[j] + conj(fb[i])) * r3;
        num d2 = (fb[j] - conj(fb[i])) * r4;
        fa[i] = c1 * d1 + c2 * d2 * r5;
        fb[i] = c1 * d2 + c2 * d1;
      }
      fa[j] = a1 * b1 + a2 * b2 * r5;
      fb[j] = a1 * b2 + a2 * b1;
    }
    fft(fa, sz);
    fft(fb, sz);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
      long long aa = fa[i].x + 0.5;
      long long bb = fb[i].x + 0.5;
      long long cc = fa[i].y + 0.5;
      res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
    }
    return res;
  }

  vector<int> square_mod(vector<int> &a, int m) {
    return multiply_mod(a, a, m, 1);
  }
};
```

5.5 Primality

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.

Time: 15 times the complexity of $a^b \mod c$.

"ModMulLL.h" 18 lines

```
using ull = unsigned long long;

bool IsPrime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  for (int i = 0; i < 15; ++i) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = ModPow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = ModMul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `Init(bits)`, where bits is the length of the numbers you use.

Time: Expected running time should be good enough for 50-bit numbers.

"MullerRabin.h", "Eratosthenes.h", "Euclid.h" 39 lines

```
using ull = unsigned long long;

vector<ull> pr;

ull f(ull a, ull n, ull &has) {
  return (ModMul(a, a, n) + has) % n;
}

vector<ull> Factorize(ull d) {
  vector<ull> res;
  for (size_t i = 0; i < pr.size() && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) d /= pr[i];
      res.push_back(pr[i]);
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
```

```
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
      for (; c==1; c = gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c) res.push_back(d);
        break;
      }
    }
  }
  return res;
}

void Init(int bits) {//how many bits do we use?
  pr = Sieve(1 << ((bits + 2) / 3));
}
```

5.6 Matrix

gauss.cpp
Description: ceva

29 lines

```
const ld EPS = 1e-9;
vector<ld> solve(vector<vector<ld>> &eqs) {
  int m = eqs.size(), n = eqs[0].size();

  for(int i = 0, j = 0; i < m && j < n - 1; ++i, ++j) {
    for(int k = i + 1; k < m; ++k) if(eqs[k][j] > eqs[i][j])
        eqs[i].swap(eqs[k]);
    if(abs(eqs[i][j]) < EPS) { --i; continue; }

    for(int k = i + 1; k < m; ++k) {
      ld x = -eqs[k][j] / eqs[i][j];
      for(int l = j; l < n; ++l) eqs[k][l] += eqs[i][l] * x;
    }
  }

  vector<ld> x(n - 1, -1);
  for(int i = m - 1; i >= 0; --i) {
    int j;
    for(j = 0; j < n - 1; ++j) if(abs(eqs[i][j]) > EPS) break;
    if(j == n - 1) continue;

    x[j] = eqs[i][n - 1];
    for(int l = j + 1; l < n - 1; ++l)
      if(abs(eqs[i][l]) > EPS && x[l] < 0) { x[j] = -1; break;
          }
      else x[j] -= x[l] * eqs[i][l];
    if(x[j] >= 0) x[j] /= eqs[i][j];
  }

  return x;
}
```

modMatrix.cpp
Description: ceva

106 lines

```
int pw(int base, int exp, int mod) {
  int res;
  for(res = 1; exp; exp >>= 1) {
    if(exp & 1) res = (1LL * res * base) % mod;
    base = (1LL * base * base) % mod;
  }

  return res;
}

int modInv(int base, int mod) {
  return pw(base, mod - 2, mod);
}

template<class T = int>
struct ModMatrix {
  int m, n, p;
  vector<vector<T>> a;

  ModMatrix(int m, int n, int p) : m(m), n(n), p(p) {
    for(int i = 0; i < m; ++i)
      a.push_back(vector<T>(n, 0));
  }

  static ModMatrix identity(int n, int p) {
    ModMatrix res(n, n, p);
    for(int i = 0; i < n; ++i) res[i][i] = 1;
    return res;
  }

  vector<T>& operator [](int index) {
    return a[index];
  }

  const vector<T>& operator [](int index) const {
    return a[index];
  }

  friend ModMatrix operator *(const ModMatrix &a, const
      ModMatrix &b) {
    ModMatrix c(a.m, b.n, a.p);
    mul(a, b, c);

    return c;
  }

  friend ModMatrix mul(const ModMatrix &a, const ModMatrix &b,
      ModMatrix &c) {
    for(int i = 0; i < c.m; ++i)
      for(int j = 0; j < c.n; ++j) {
        c[i][j] = 0;
        for(int k = 0; k < a.n; ++k)
          c[i][j] = (c[i][j] + 1LL * a[i][k] * b[k][j]) % a.p;
      }

    return c;
  }

  friend ModMatrix pw(ModMatrix base, int exp) {
    ModMatrix res = identity(base.m, base.p), aux(base.m, base.
        m, base.p);
    for(; exp; exp >>= 1) {
      if(exp & 1) mul(res, base, aux), res.a.swap(aux.a);
      mul(base, base, aux), base.a.swap(aux.a);
    }

    return res;
  }

  friend ModMatrix modInv(ModMatrix a) { // assumes a is
      invertible
    ModMatrix inv = identity(a.n, a.p);

    for(int i = 0; i < a.m; ++i) {
      int k;
      for(k = i; k < a.m && !a[k][i]; ++k);
      if(i != k) {
        a[i].swap(a[k]);
        inv[i].swap(inv[k]);
      }

      int x = modInv(a[i][i], a.p);
      for(int j = 0; j < a.n; ++j) {
        a[i][j] = (1LL * a[i][j] * x) % a.p;
        inv[i][j] = (1LL * inv[i][j] * x) % a.p;
      }

      for(int k = 0; k < a.m; ++k) {
        if(k == i) continue;

        int x = a[k][i];
        for(int j = 0; j < a.n; ++j) {
          a[k][j] -= (1LL * a[i][j] * x) % a.p; if(a[k][j] < 0)
              a[k][j] += a.p;
          inv[k][j] -= (1LL * inv[i][j] * x) % a.p; if(inv[k][j
              ] < 0) inv[k][j] += a.p;
        }
      }
    }

    return inv;
  }

  friend ostream& operator <<(ostream &out, const ModMatrix &a)
      {
    for(int i = 0; i < a.m; ++i) {
      for(int j = 0; j < a.n; ++j) out << a[i][j] << ' ';
      if(i + 1 < a.m) out << '\n';
    }
    return out;
  }
};
```

5.7 Divisibility

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

8 lines

```
using ll = long long;

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

5.7.1 Bézout's identity

For $a \neq, b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

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

phiFunction.h

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

11 lines

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

void ComputePhi() {
  for (int i = 0; i < kLim; ++i)
    phi[i] = (i % 2) ? i : i / 2;
  for (int i = 3; i < kLim; i += 2)
    if (phi[i] == i)
      for (int j = i; j < kLim; j += i)
        (phi[j] /= i) *= i - 1;
}
```

5.8 Chinese remainder theorem

CRT.h
Description: wtf

33 lines

```
void extendedGCD(int a, int b, int &x, int &y, int mod) {
  if(b == 0) { x = 1; y = 0; return; }

  int xx, yy;
  extendedGCD(b, a % b, xx, yy, mod);

  x = yy;
  y = (xx - 1LL * a / b * yy) % mod;
  if(y < 0) y += mod;
}

int fastCrt(const vector<pii> &eqs) {
  int currm = 1, currr = 0;

  for(const auto &[m, r] : eqs) {
    if(currm == 1) {
      currm = m;
      currr = r;
      continue;
    }

    int x, y;
    extendedGCD(currm, m, x, y, currm * m);

    int a = (1LL * r * x) % m;
    int b = (1LL * currr * y) % currm;

    currr = (1LL * a * currm + 1LL * b * m) % (currm * m);
    currm *= m;
  }

  return currr;
}
```

5.9 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.

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

5.11 Estimates

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Cycles

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

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

6.1.2 Derangements

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

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

derangements.h
Description: Generates the i:th derangement of S_n (in lexicographical order). (perm a.i. v[i] != i)

38 lines

```
template <class T, int N>
struct derangements {
  T dgen[N][N], choose[N][N], fac[N];
  derangements() {
    fac[0] = choose[0][0] = 1;
    memset(dgen, 0, sizeof(dgen));
    rep(m,1,N) {
      fac[m] = fac[m-1] * m;
      choose[m][0] = choose[m][m] = 1;
      rep(k,1,m)
        choose[m][k] = choose[m-1][k-1] + choose[m-1][k];
    }
  }
  T DGen(int n, int k) {
    T ans = 0;
    if (dgen[n][k]) return dgen[n][k];
    rep(i,0,k+1)
      ans += (i&1?-1:1) * choose[k][i] * fac[n-i];
    return dgen[n][k] = ans;
  }
  void generate(int n, T idx, int *res) {
    int vals[N];
    rep(i,0,n) vals[i] = i;
    rep(i,0,n) {
      int j, k = 0, m = n - i;
      rep(j,0,m) if (vals[j] > i) ++k;
      rep(j,0,m) {
        T p = 0;
        if (vals[j] > i) p = DGen(m-1, k-1);
        else if (vals[j] < i) p = DGen(m-1, k);
        if (idx <= p) break;
        idx -= p;
      }
      res[i] = vals[j];
      memmove(vals + j, vals + j + 1, sizeof(int)*(m-j-1));
    }
  }
};
```

6.1.3 Involutions

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$

$$a(0) = a(1) = 1$$

1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152

6.1.4 Stirling numbers of the first kind

$$s(n,k) = (-1)^{n-k} c(n,k)$$

$c(n,k)$ is the unsigned Stirling numbers of the first kind, and they count the number of permutations on n items with k cycles.

$$s(n,k) = s(n-1,k-1) - (n-1)s(n-1,k)$$

$$s(0,0) = 1, s(n,0) = s(0,n) = 0$$

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

$$c(0,0) = 1, c(n,0) = c(0,n) = 0$$

6.1.5 Eulerian numbers

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

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

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

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

6.1.6 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

Partitions of n with exactly k parts, $p(n,k)$, i.e., writing n as a sum of k positive integers, disregarding the order of the summands.

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

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

For partitions with any number of parts, $p(n)$ obeys

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

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 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.2.3 Bell numbers

Total number of partitions of n distinct elements.

$$B(n) = \sum_{k=1}^{n}\binom{n-1}{k-1}B(n-k) = \sum_{k=1}^{n}S(n,k)$$

$$B(0) = B(1) = 1$$

The first are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597. For a prime p

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

6.2.4 Triangles

Given rods of length $1, \ldots, n$,

$$T(n) = \frac{1}{24}\begin{cases} n(n-2)(2n-5) & n \text{ even} \\ (n-1)(n-3)(2n-1) & n \text{ odd} \end{cases}$$

is the number of distinct triangles (positive are) that can be constructed, i.e., the # of 3-subsets of $[n]$ s.t. $x \leq y \leq z$ and $z \neq x + y$.

6.3 General purpose numbers

6.3.1 Catalan numbers

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

$$C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C_0 = 1, C_{n+1} = \sum C_i C_{n-i}$$

First few are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900.

- # of monotonic lattice paths of a $n \times n$-grid which do not pass above the diagonal.
- # of expressions containing n pairs of parenthesis which are correctly matched.
- # of full binary trees with with $n+1$ leaves (0 or 2 children).
- # of non-isomorphic ordered trees with $n+1$ vertices.
- # of ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- # of permutations of $[n]$ with no three-term increasing subsequence.

6.3.2 Super Catalan numbers

The number of monotonic lattice paths of a $n \times n$-grid that do not touch the diagonal.

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$

$$S(1) = S(2) = 1$$

1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859

6.3.3 Motzkin numbers

Number of ways of drawing any number of nonintersecting chords among n points on a circle. Number of lattice paths from $(0,0)$ to $(n,0)$ never going below the x-axis, using only steps NE, E, SE.

$$M(n) = \frac{3(n-1)M(n-2) + (2n+1)M(n-1)}{n+2}$$

$$M(0) = M(1) = 1$$

1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634

6.3.4 Narayana numbers

Number of lattice paths from $(0,0)$ to $(2n,0)$ never going below the x-axis, using only steps NE and SE, and with k peaks.

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

$$N(n,1) = N(n,n) = 1$$

$$\sum_{k=1}^{n} N(n,k) = C_n$$

1, 1, 1, 1, 3, 1, 1, 6, 6, 1, 1, 10, 20, 10, 1, 1, 15, 50

6.3.5 Schröder numbers

Number of lattice paths from $(0,0)$ to (n,n) using only steps N,NE,E, never going above the diagonal. Number of lattice paths from $(0,0)$ to $(2n,0)$ using only steps NE, SE and double east EE, never going below the x-axis. Twice the Super Catalan number, except for the first term. 1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098

Graph (7)

7.1 Fundamentals

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices (array idx), such that there are edges only from left to right. The function returns false if there is a cycle in the graph.

Time: $\mathcal{O}(|V| + |E|)$ 18 lines

```
template <class E, class I>
bool topo_sort(const E &edges, I &idx, int n) {
  vi indeg(n);
  rep(i,0,n)
    trav(e, edges[i])
      indeg[e]++;
  queue<int> q; // use priority queue for lexic. smallest ans.
  rep(i,0,n) if (indeg[i] == 0) q.push(-i);
  int nr = 0;
  while (q.size() > 0) {
    int i = -q.front(); // top() for priority queue
    idx[i] = ++nr;
    q.pop();
    trav(e, edges[i])
      if (--indeg[e] == 0) q.push(-e);
  }
  return nr == n;
}
```

7.2 Euler walk

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. For a directed / undirected graph. For each path/cycle found, calls a callback. You can check cycle by checking path endpoints. To transform into undirected, toggle comment on lines (*)

Time: $\mathcal{O}(E)$ where E is the number of edges. 46 lines

```
struct EulerWalk {
  int n;
  vector<multiset<int>> G;
  vector<int> deg;

  EulerWalk(int n) : n(n), G(n + 1), deg(n + 1, 0) {}

  void AddEdge(int a, int b) {
    G[b].insert(a);
    deg[a] += 1; deg[b] -= 1;
    // G[a].insert(b); (*)
  }

  vector<int> walk;
  void dfs(int node) {
    while (G[node].size()) {
      auto vec = *G[node].begin();
      G[node].erase(G[node].begin());
      // G[vec].erase(G[vec].find(node)); (*)
```

```
      dfs(vec);
    }
    walk.push_back(node);
  }

  template<typename Callback>
  void Solve(Callback cb) {
    for (int i = 1; i <= n; ++i) {
      while (deg[i] < 0) AddEdge(i, n); // (*)
      while (deg[i] > 0) AddEdge(n, i); // (*)
      // if (deg[i] % 2) AddEdge(i, n); (*)
    }
    // Paths
    vector<int> buff; dfs(n);
    for (auto node : walk) {
      if (node < n) buff.push_back(node);
      else if (buff.size()) {
        cb(buff); buff.clear();
      }
    }
    // Cycles
    for (int i = 0; i < n; ++i) {
      walk.clear(); dfs(i);
      if (walk.size() > 1) cb(walk);
    }
  }
};
```

7.3 Network flow

Dinic.h

Description: wtf

81 lines

```
struct NetworkFlow {
  const int INT_INF = 0x3f3f3f3f;
  const ll LL_INF = 1e18;

  struct Edge {
    int to, flow;
  };

  int n, source, sink;
  vector<int> dst, ptr;
  vector<Edge> edges;
  vector<vector<int>> adj;

  NetworkFlow(int n) : n(n) {
    source = 0;
    sink = n - 1;

    dst.resize(n);
    adj.resize(n);
  }

  void addEdge(int a, int b, int cap) {
    adj[a].push_back(edges.size());
    edges.push_back({b, cap});

    adj[b].push_back(edges.size());
    edges.push_back({a, 0});
  }

  bool bfs(int v) {
    dst.assign(n, INT_INF);

    queue<int> q;
    for(dst[v] = 0, q.push(v); !q.empty(); q.pop()) {
      v = q.front();
      for(auto id : adj[v])
        if(dst[edges[id].to] > 1 + dst[v] && edges[id].flow) {
          dst[edges[id].to] = 1 + dst[v];
          q.push(edges[id].to);
        }
    }

    return dst[sink] != INT_INF;
  }

  ll dfs(int v, ll flow) {
    if(v == sink || !flow) return flow;

    for(; ptr[v] < adj[v].size(); ++ptr[v]) {
      int id = adj[v][ptr[v]];
      int u = edges[id].to;
      if(edges[id].flow && dst[u] == 1 + dst[v]) {
        int pushed = dfs(u, min(flow, (ll) edges[id].flow));
        if(pushed) {
          edges[id].flow -= pushed;
          edges[id ^ 1].flow += pushed;
          return pushed;
        }
      }
    }

    return 0;
  }

  ll dinic() {
    ll flow, total;

    for(total = 0; bfs(source); ) {
      ptr.assign(n, 0);
      while(flow = dfs(source, LL_INF)) total += flow;
    }

    return total;
  }

  void clear() {
    edges.clear();
    for(int i = 0; i < n; ++i) adj[i].clear();
  }
};
```

flowWithLowerBound.cpp

Description: wtf

8 lines

```
Min floww with lower bounds on edges:
Add two new vertices s', t'
Add edge from s' to v with capacity sum{u}(lower_bound(u->v))
Add edge from v to t' with capacity sum{w}(lower_bound(v->w))
Add edge from v to u with capacity cap(v->u) - lower_bound(v->u
    )
Add edge from s to t with capacity INF
After finding a feasible flow, run Dinic again to find a mximum
      feasible flow, ensuring the flow is feasible at every
     step
i.e. do not substract flow from an edge such that the new value
      is less than the lower bound
```

edmonsblossomValeriu.h

Description: None

Usage: ask Djok

<bits/stdc++.h> 94 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 105;

int i, match[N], p[N], base[N], q[N];
bool used[N], viz[N], blossom[N];
vector<int> lda[N];

int lca(int a, int b) {
  memset(viz, 0, sizeof(viz));
  while(1) {
    a = base[a];
    viz[a] = 1;
    if(match[a] == -1) break;
    a = p[match[a]];
  }
  while(1) {
    b = base[b];
    if(viz[b]) break;
    b = p[match[b]];
  }
  return b;
}

void markPath(int x, int y, int children) {
  while(base[x] != y) {
    blossom[base[x]] = blossom[base[match[x]]] = 1;
    p[x] = children;
    children = match[x];
    x = p[match[x]];
  }
}

int findPath(int x) {
  memset(used, 0, sizeof(used));
  memset(p, -1, sizeof(p));
  for(int i = 0; i < N; ++i) base[i] = i;

  int qh = 0, qt = 0;
  q[qt++] = x; used[x] = 1;
  while(qh < qt) {
    int v = q[qh++];
    for(int to : lda[v]) {
      if(base[v] == base[to] || match[v] == to) continue;
      if(to == x || match[to] != -1 && p[match[to]] != -1) {
        int curbase = lca(v, to);
        memset(blossom, 0, sizeof(blossom));
        markPath(v, curbase, to);
        markPath(to, curbase, v);
        for(int i = 0; i < N; ++i)
          if(blossom[base[i]]) {
            base[i] = curbase;
            if(!used[i]) {
              used[i] = 1;
              q[qt++] = i;
            }
          }
      }
      else if(p[to] == -1) {
        p[to] = v;
        if(match[to] == -1) return to;
        to = match[to];
        used[to] = 1;
        q[qt++] = to;
      }
    }
  }
  return -1;
```

```
}

int main() {
  // add edge x, y and y, x to lda
  memset(match, -1, sizeof(match));
  for(i = 0; i < N; ++i)
    if(match[i] == -1)
      for(int to : lda[i])
        if(match[to] == -1) {
          match[to] = i;
          match[i] = to;
          break;
        }

  for(i = 0; i < N; ++i)
    if(match[i] == -1) {
      int v = findPath(i);
      while(v != -1) {
        int pv = p[v], ppv = match[pv];
        match[v] = pv; match[pv] = v; v = ppv;
      }
    }

  return 0;
}
```

MinCostMaxFlow.h

Description: wtf

97 lines

```
const int INF = 0x3f3f3f3f;

struct MCMF {
  struct Edge {
    int to, flow, cst;
  };

  int n, source, sink;
  vector<int> d, reald, newd, prv;
  vector<bool> vis;
  vector<Edge> edges;
  vector<vector<int>> adj;

  MCMF(int n) : n(n), source(0), sink(n - 1), d(n), reald(n),
       newd(n), prv(n), vis(n), adj(n) {}

  void addEdge(int a, int b, int cap, int cst) {
    adj[a].push_back(edges.size());
    edges.push_back({b, cap, cst});

    adj[b].push_back(edges.size());
    edges.push_back({a, 0, -cst});
  }

  void bellman() {
    priority_queue<pii> q;
    fill(all(d), INF);

    for(d[source] = 0, q.push({0, source}); !q.empty(); ) {
      int dst = -q.top().fi;
      int v = q.top().se;
      q.pop();

      if(dst != d[v]) continue;

      for(auto id : adj[v]) {
        int u = edges[id].to;
        if(edges[id].flow && d[u] > d[v] + edges[id].cst) {
          d[u] = d[v] + edges[id].cst;
          q.push({-d[u], u});
        }
      }
    }
  }

  bool dijkstra() {
    priority_queue<pii> q;
    fill(all(newd), INF);
    fill(all(vis), false);

    for(reald[source] = newd[source] = 0, q.push({0, source});
        !q.empty(); ) {
      int dst = -q.top().fi;
      int v = q.top().se;
      q.pop();

      if(vis[v]) continue;
      vis[v] = true;

      for(auto id : adj[v]) {
        int u = edges[id].to;
        int w = d[v] + edges[id].cst - d[u];
        if(edges[id].flow && newd[u] > newd[v] + w) {
          newd[u] = newd[v] + w;
          reald[u] = reald[v] + edges[id].cst;
          prv[u] = id;
          q.push({-newd[u], u});
        }
      }
    }

    return newd[sink] < INF;
  }

  pii get() {
    int flow, cst;

    bellman();

    for(flow = cst = 0; dijkstra(); ) {
      int pushed = INF;
      for(int v = sink; v != source; v = edges[prv[v] ^ 1].to)
          {
        pushed = min(pushed, edges[prv[v]].flow);
      }

      flow += pushed;

      for(int v = sink; v != source; v = edges[prv[v] ^ 1].to)
          {
        cst += pushed * edges[prv[v]].cst;
        edges[prv[v]].flow -= pushed;
        edges[prv[v] ^ 1].flow += pushed;
      }

      d = reald;
    }

    return { flow, cst };
  }
};
```

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.

1 lines

GlobalMinCut.h

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

Time: $\mathcal{O}(V^3)$

<bits/stdc++.h> 40 lines

```
using T = long long;
pair<T, vector<int>> GetMinCut(vector<vector<T>> weights) {
  int n = weights.size();
  vector<int> used(n), best_cut, cut;
  T best_weight = numeric_limits<T>::max();

  for (int phase = n - 1; phase > 0; phase--) {
    auto w = weights[0];
    auto added = used;
    int prev, k = 0;

    for (int i = 0; i < phase; ++i) {
      prev = k; k = -1;

      for (int j = 1; j < n; ++j)
        if (!added[j] && (k == -1 || w[j] > w[k]))
          k = j;

      if (i != phase - 1) {
        for (int j = 0; j < n; ++j)
          w[j] += weights[k][j];
        added[k] = true;
        continue;
      }

      for (int j = 0; j < n; ++j)
          weights[prev][j] += weights[k][j];
      for (int j = 0; j < n; ++j)
          weights[j][prev] = weights[prev][j];

      used[k] = true; cut.push_back(k);

      if (w[k] < best_weight) {
        best_cut = cut;
        best_weight = w[k];
      }
    }
  }
  return {best_weight, best_cut};
}
```

GomoryHu.h

Description: Computes Gomory-Hu tree of a graph (ans[i][j] = min cut intre i si j)

Time: $\mathcal{O}(V)$ calls of flow algorithm

14 lines

```
void GomoryHu() {
    int parent[n]; //initialized to 0
    int answer[n][n]; //initialize this one to infinity
    for(int i=1;i<n;++i){
        //Compute the minimum cut between i and parent[i].
        //Let the i-side of the min cut be S, and the value of
             the min-cut be F
        for (int j=i+1;j<n;++j)
            if ((j is in S) && parent[j]==parent[i])
                parent[j]=i;
        answer[i][parent[i]]=answer[parent[i]][i]=F;
        for (int j=0;j<i;++j)
            answer[i][j]=answer[j][i]=min(F,answer[parent[i]][j
                 ]);
    }
}
```

7.4 Matching

matching.cpp

Description: wtf

73 lines

```cpp
struct Matching {
  int m, n;
  vector<int> l, r;
  vector<bool> vis, ok, coverL, coverR;
  vector<vector<int>> adj, adjt;

  Matching(int m, int n) : m(m), n(n), l(n), r(m), vis(m), ok(m
      ), adj(m), adjt(n), coverL(m), coverR(n) {}

  bool pairUp(int v) {
    if(vis[v]) return false;
    vis[v] = true;

    for(auto u : adj[v])
      if(l[u] == -1) return l[u] = v, r[v] = u, true;

    for(auto u : adj[v])
      if(pairUp(l[u])) return l[u] = v, r[v] = u, true;

    return false;
  }

  void bfs(vector<vector<int>> adj, vector<int> l, vector<int>
      r) {
    queue<int> q;
    vis.assign(r.size(), false);
    for(int i = 0; i < r.size(); ++i) if(r[i] == -1) q.push(i),
         vis[i] = true;
    for(; !q.empty(); q.pop()) {
      int v = q.front();
      ok[v] = true;
      for(auto u : adj[v])
        if(!vis[l[u]]) q.push(l[u]), vis[l[u]] = true;
    }
  }

  void cover(int v) {
    for(auto u : adj[v])
      if(!coverR[u]) {
        coverR[u] = true;
        coverL[l[u]] = false;
        cover(l[u]);
      }
  }

  void addEdge(int a, int b) {
    adj[a].push_back(b);
    adjt[b].push_back(a);
  }

  int matching() {
    int sz;
    bool changed;

    l.assign(n, -1);
    r.assign(m, -1);
    for(sz = 0, changed = true; changed; ) {
      vis.assign(m, false);
      changed = false;
      for(int i = 0; i < m; ++i)
        if(r[i] == -1 && pairUp(i)) ++sz, changed = true;
    }
    return sz;
  }

  // if ok[i] == false => i belongs to all maximum matchings
  void computeVerticesBelongingToAllmaximumMatchings() {
    bfs(adj, l, r);
    bfs(adjt, r, l);
  }

  void computeMinimumVertexCover() {
    for(int i = 0; i < m; ++i) if(r[i] != -1) coverL[i] = true;
    for(int i = 0; i < m; ++i) if(r[i] == -1) cover(i);
  }
};
```

WeightedMatching.h

Description: Min cost perfect bipartite matching. Negate costs for max cost.

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

57 lines

```cpp
template<typename T>
int MinAssignment(const vector<vector<T>> &c) {
  int n = c.size(), m = c[0].size();        // assert(n <= m);
  vector<T> v(m), dist(m);                  // v: potential
  vector<int> L(n, -1), R(m, -1);           // matching pairs
  vector<int> index(m), prev(m);
  iota(index.begin(), index.end(), 0);

  auto residue = [&](int i, int j) { return c[i][j] - v[j]; };
  for (int f = 0; f < n; ++f) {
    for (int j = 0; j < m; ++j) {
      dist[j] = residue(f, j); prev[j] = f;
    }
    T w; int j, l;
    for (int s = 0, t = 0;;) {
      if (s == t) {
        l = s; w = dist[index[t++]];
        for (int k = t; k < m; ++k) {
          j = index[k]; T h = dist[j];
          if (h <= w) {
            if (h < w) { t = s; w = h; }
            index[k] = index[t]; index[t++] = j;
          }
        }
        for (int k = s; k < t; ++k) {
          j = index[k];
          if (R[j] < 0) goto aug;
        }
      }
      int q = index[s++], i = R[q];
      for (int k = t; k < m; ++k) {
        j = index[k];
        T h = residue(i,j) - residue(i,q) + w;
        if (h < dist[j]) {
          dist[j] = h; prev[j] = i;
          if (h == w) {
            if (R[j] < 0) goto aug;
            index[k] = index[t]; index[t++] = j;
          }
        }
      }
    }
  aug:
    for(int k = 0; k < l; ++k)
      v[index[k]] += dist[index[k]] - w;
    int i;
    do {
      R[j] = i = prev[j];
      swap(j, L[i]);
    } while (i != f);
  }
  T ret = 0;
  for (int i = 0; i < n; ++i) {
    ret += c[i][L[i]]; // (i, L[i]) is a solution
  }
  return ret;
}
```

Hungarian.cpp

Description: Computes the min-cost perfect matching of a graph, —L—=—R—=n

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

<bits/stdc++.h> 46 lines

```cpp
const int N = 505;
const long long INF = 1e18;

int n;
long long a[N][N]; // a[1..n][1..n], a[i][j] = cost(i, j)
long long u[N], v[N];
int p[N], way[N];

int main(){
  long long res = 0;
  for(int i = 1; i <= n; ++i){
    p[0] = i;
    int j0 = 0;
    vector<long long> minv (n + 1, INF);
    vector<char> used (n + 1, false);
    do{
      used[j0] = true;
      int i0 = p[j0],   j1;
      long long delta = INF;
      for (int j = 1; j <= n; ++j)
        if (!used[j]){
          long long cur = a[i0][j] - u[i0] - v[j];
          if (cur < minv[j])
            minv[j] = cur,  way[j] = j0;
          if (minv[j] < delta)
            delta = minv[j],  j1 = j;
        }
      for (int j = 0; j <= n; ++j)
        if (used[j])
          u[p[j]] += delta,  v[j] -= delta;
        else
          minv[j] -= delta;
      j0 = j1;
    }while (p[j0] != 0);
    do {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while (j0);

    res = max(res, v[0]);
  }

  cout << res << endl;
  return 0;
}
```

7.5 DFS algorithms

SCC.h

Description: wtf

27 lines

```cpp
int comp[N];
bool vis[N];
vector<int> adj[N], adjt[N], ord;

void dfs(int v) {
  vis[v] = true;
```

```
  for(auto u : adj[v])
    if(!vis[u]) dfs(u);
  ord.push_back(v);
}

void dfst(int v, int nrc) {
  vis[v] = true;
  comp[v] = nrc;
  for(auto u : adjt[v])
    if(!vis[u]) dfst(u, nrc);
}

for(int i = 1; i <= n; ++i)
  if(!vis[i]) dfs(i);

reverse(all(ord));
memset(vis, 0, sizeof vis);

int nrc = 0;
for(auto i : ord) if(!vis[i]) dfst(i, ++nrc);
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected multigraph, 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. HOWEVER, note that we are outputting bridges as BCC's here, because we might be interested in vertex bcc's, not edge bcc's.
To get the articulation points, look for vertices that are in more than 1 BCC.
To get the bridges, look for biconnected components with one edge
Time: $\mathcal{O}(E+V)$ 54 lines

```
struct BCC {
  vector<pair<int, int>> edges;
  vector<vector<int>> G;
  vector<int> enter, low, stk;

  BCC(int n) : G(n), enter(n, -1) {}

  int AddEdge(int a, int b) {
    int ret = edges.size();
    edges.emplace_back(a, b);
    G[a].push_back(ret);
    G[b].push_back(ret);
    return ret;
  }

  template<typename Iter>
  void Callback(Iter bg, Iter en) {
    for (Iter it = bg; it != en; ++it) {
      auto edge = edges[*it];
      // Do something useful
    }
  }

  void Solve() {
    for (int i = 0; i < (int)G.size(); ++i)
      if (enter[i] == -1) {
        dfs(i, -1);
      }
  }

  int timer = 0;
  int dfs(int node, int pei) {
    enter[node] = timer++;
    int ret = enter[node];

    for (auto ei : G[node]) if (ei != pei) {
      int vec = (edges[ei].first ^ edges[ei].second ^ node);
      if (enter[vec] != -1) {
        ret = min(ret, enter[vec]);
        if (enter[vec] < enter[node])
          stk.push_back(ei);
      } else {
        int sz = stk.size(), low = dfs(vec, ei);
        ret = min(ret, low);
        stk.push_back(ei);
        if (low >= enter[node]) {
          Callback(stk.begin() + sz, stk.end());
          stk.resize(sz);
        }
      }
    }
    return ret;
  }
};
```

2sat.h

Description: wtf 47 lines

```
struct Sat {
  int n;
  vector<int> ord, val, compId;
  vector<bool> vis;
  vector<vector<int>> adj, adjt;

  Sat(int n) : n(2 * n), vis(2 * n), compId(2 * n), adj(2 * n),
       adjt(2 * n) {}

  void addEdge(int x, int y) {
    x = (x < 0 ? -2 * x - 2 : 2 * x - 1);
    y = (y < 0 ? -2 * y - 2 : 2 * y - 1);
    adj[x].push_back(y);
    adjt[y].push_back(x);
  }

  void addClause(int x, int y) {
    addEdge(-x, y);
    addEdge(-y, x);
  }

  void dfs(int v) {
    vis[v] = true;
    for(auto u : adj[v]) if(!vis[u]) dfs(u);
    ord.push_back(v);
  }

  void dfst(int v, int id) {
    vis[v] = false;
    compId[v] = id;
    if(val[v] == -1) val[v] = 0, val[v ^ 1] = 1;
    for(auto u : adjt[v]) if(vis[u]) dfst(u, id);
  }

  bool solve() {
    val.assign(n, -1);

    for(int i = 0; i < n; ++i) if(!vis[i]) dfs(i);
    for(int nr = 0, i = n - 1; i >= 0; --i) if(vis[ord[i]])
         dfst(ord[i], nr++);

    for(int i = 0; i < n; i += 2) if(compId[i] == compId[i +
         1]) return false;
    return true;
  }

  int get(int i) {
    return val[2 * i - 1];
  }
};
```

7.6 Trees

TreePower.h

Description: Calculate power of two jumps in a tree. Assumes the root node points to itself.
Time: $\mathcal{O}(|V|\log|V|)$ 14 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;
}
```

centroid.cpp

Description: wtf 50 lines

```
int fth[N], sz[N];
bool used[N];
vector<pii> adj[N];

void computeSz(int v, int p = -1) {
  sz[v] = 1;
  for(auto [u, w] : adj[v])
    if(u != p && !used[u]) {
      computeSz(u, v);
      fth[u] = v;
      sz[v] += sz[u];
    }
}

int findCentroid(int v, int n, int p = -1) {
  while(true) {
    int heavyCh = -1;
    for(auto [u, w] : adj[v])
      if(u != p && !used[u] && (heavyCh == -1 || sz[u] > sz[
           heavyCh])) heavyCh = u;

    if(heavyCh == -1 || sz[heavyCh] <= n / 2) return v;
    p = v;
    v = heavyCh;
  }

  return -1;
}

void dfs(int v, int p = -1) {
  // do something with node v

  for(auto [u, w] : adj[v])
    if(u != p && !used[u])
      dfs(u, v);
}

void solve(int v, int n) {
  fth[v] = 0;
  computeSz(v);
```

```
  int centroid = findCentroid(v, n);

  dfs(centroid);

  used[centroid] = true;

  for(auto [u, w] : adj[centroid])
    if(!used[u]) solve(u, sz[u]);

  if(fth[centroid] && !used[fth[centroid]]) solve(fth[centroid
      ], n - sz[centroid]);
}
```

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 the nodes of the reduced tree, while at the same time populating a link array that stores the new parents. The root points to -1.

Time: $\mathcal{O}(|S| * (\log |S| + LCA\_Q))$

"LCA.h" 18 lines

```
vector<int> CompressTree(vector<int> v, LCA& lca,
                         vector<int>& link) {
  auto cmp = [&](int a, int b) {
    return lca.enter[a] < lca.enter[b];
  };
  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());

  for (int i = (int)v.size() - 1; i > 0; --i)
    v.push_back(lca.Query(v[i - 1], v[i]));

  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());

  for (int i = 0; i < (int)v.size(); ++i)
    link[v[i]] = (i == 0 ? -1 : lca.Query(v[i - 1], v[i]));
  return v;
}
```

HLD.h

Description: wtf

73 lines

```
struct HLD {
  int n, t;
  vector<int> in, out, head, fth, h, sz;
  vector<vector<int>> adj;
  SegTree segTree;
  // in[i] = time entering node i
  // out[i] = time leaving node i
  // head[i] = head of path containing node i
  // fth[i] = parent of node i in original tree
  // h[i] = height of node i in original tree starting from 0
  // sz[i] = size of subtree of i in original tree

  HLD(int n) : n(n), in(n), out(n), head(n), fth(n), h(n), sz(n
      ), adj(n), segTree(n) {}

  void addEdge(int a, int b) {
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  void dfsSize(int v, int p = -1) {
    sz[v] = 1;
    for(auto &u : adj[v])
      if(u != p) {
        fth[u] = v;
        h[u] = 1 + h[v];
        dfsSize(u, v);
        sz[v] += sz[u];
        if(sz[u] > sz[adj[v][0]]) swap(u, adj[v][0]);
      }
  }

  void dfsHld(int v, int p = -1) {
    in[v] = t++;
    for(auto u : adj[v])
      if(u != p) {
        head[u] = (u == adj[v][0] ? head[v] : u);
        dfsHld(u, v);
      }
    out[v] = t;
  }

  void build(const vector<int> &v) {
    t = 0;
    sz.assign(n, 0);

    dfsSize(0);
    dfsHld(0);

    for(int i = 0; i < n; ++i) segTree.update(in[i], v[i]);
  }

  void update(int v, int val) {
    segTree.update(in[v], val);
  }

  int query(int v, int u) {
    int res = 0;
    while(head[v] != head[u]) {
      if(h[head[v]] > h[head[u]]) swap(v, u);

      res = max(res, segTree.query(in[head[u]], in[u] + 1));
      u = fth[head[u]];
    }

    if(h[v] > h[u]) swap(v, u);
    res = max(res, segTree.query(in[v], in[u] + 1));

    return res;
  }

  // subtree of v: [in_v, out_v)
  // path from v to the heavy path head: [in_head_v, in_v]
};
```

LinkCutTree.h

Description: 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.

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

96 lines

```
struct Node { // Splay tree. Root's pp contains tree's parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc. if wanted)
  }
  void push_flip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this : -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      z->c[h ^ 1] = b ? x : this;
    }
    y->c[i ^ 1] = b ? this : x;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
  }
  void splay() {
    for (push_flip(); p; ) {
      if (p->p) p->p->push_flip();
      p->push_flip(); push_flip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() {
    push_flip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};

struct LinkCut {
  vector<Node> node;
  LinkCut(int N) : node(N) {}

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    make_root(&node[u]);
    node[u].pp = &node[v];
  }

  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    make_root(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }

  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }

  void make_root(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
```

```
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

7.7 Matrix tree theorem

MatrixTree.h

Description: To count the number of spanning trees in an undirected graph G: create an $N \times N$ matrix `mat`, and for each edge $(a, b) \in G$, do `mat[a][a]++, mat[b][b]++, mat[a][b]--, mat[b][a]--`. Remove the last row and column, and take the determinant.

1 lines

Geometry (8)

8.1 Geometric primitives

Point.h

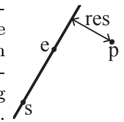
Description: Point declaration, and basic operations.

32 lines

```
using Point = complex<double>;

const double kPi = 4.0 * atan(1.0);
const double kEps = 1e-9; // Good eps for long double is ~1e-11

#define X() real()
#define Y() imag()

double dot(Point a, Point b) { return (conj(a) * b).X(); }
double cross(Point a, Point b) { return (conj(a) * b).Y(); }
double dist(Point a, Point b) { return abs(b - a); }
Point perp(Point a) { return Point{-a.Y(), a.X()}; } // +90deg

double rotateCCW(Point a, double theta) {
  return a * polar(1.0, theta); }
double det(Point a, Point b, Point c) {
  return cross(b - a, c - a); }

// abs() is norm (length) of vector
// norm() is square of abs()
// arg() is angle of vector
// det() is twice the signed area of the triangle abc
// and is > 0 iff c is to the left as viewed from a towards b.
// polar(r, theta) gets a vector from abs() and arg()

void ExampleUsage() {
  Point a{1.0, 1.0}, b{2.0, 3.0};
  cerr << a << " " << b << endl;
  cerr << "Length of ab is: " << dist(a, b) << endl;
  cerr << "Angle of a is: " << arg(a) << endl;
  cerr << "axb is: " << cross(a, b) << endl;
}
```

lineDistance.h

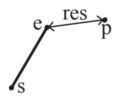
Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan, although don't rely on that. Also works in 3D. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance.



"Point.h" 8 lines

```
double LineDistance(Point a, Point b, Point p) {
  return det(a, b, p) / abs(b - a);
}

// Projects point p on line (a, b)
Point ProjectPointOnLine(Point p, Point a, Point b) {
  return a + (b - a) * dot(p - a, b - a) / norm(b - a);
}
```

SegmentDistance.h



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

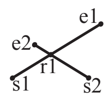
Usage: `Point a{0, 0}, b{2, 2}, p{1, 1};`
`bool onSegment = SegmentDistance(p, a, b) < kEps;`

"Point.h" 15 lines

```
double SegmentDistance(Point p, Point a, Point b) {
  if (a == b) return abs(p - a); // Beware of precision!!!
  double d = norm(b - a);
  double t = min(d, max(.0, dot(p - a, b - a)));
  return abs((p - a) * d - (b - a) * t) / d;
}

// Projects point p on segment [a, b]
Point ProjectPointOnSegment(Point p, Point a, Point b) {
  double d = norm(b - a);
  if (d == 0) return a; // Beware of precision!!!

  double r = dot(p - a, b - a) / d;
  return (r < 0) ? a : (r > 1) ? b : (a + (b - a) * r);
}
```

SegmentIntersection.h

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



Usage: `Point<double> intersection, dummy;`
`if (segmentIntersection(s1,e1,s2,e2,intersection,dummy)==1)`
`cout << "segments intersect at " << intersection << endl;`

"Point.h" 27 lines

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

SegmentIntersectionQ.h

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

"Point.h" 16 lines

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

LineIntersectionCheck.h

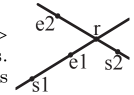
Description: Checks if two lines intersect, and returns 1 if one intersection, 0 if lines are parallel (no intersection), and -1 if they coincide (infinite intersections).

"Point.h" 6 lines

```
int LineIntersection(Point a, Point b, Point p, Point q) {
  double c1 = det(a, b, p), c2 = det(a, b, q);
  if (sgn(c1 - c2)) return 1;
  if (sgn(c1) == 0) return -1;
  return 0;
}
```

lineIntersection.h

Description:
Returns the intersection between non-parallel lines. If unsure if lines are concurrent, check with LineIntersectionCheck. The wrong position will be returned if P is complex<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



"Point.h" 5 lines

```
Point LineIntersection(Point a, Point b, Point p, Point q) {
  double c1 = det(a, b, p), c2 = det(a, b, q);
  assert(sgn(c1 - c2)); // undefined if parallel
  return (q * c1 - p * c2) / (c1 - c2);
}
```

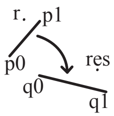
onSegment.h

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

"Point.h" 4 lines

```
bool OnSegment(Point s, Point e, Point p) {
  Point ds = p - s, de = p - e;
  return cross(ds, de) == 0 && dot(ds, de) <= 0;
}
```

linearTransformation.h



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

"Point.h" 6 lines

```
Point LinearTransformation(Point p0, Point p1,
                           Point q0, Point q1, Point r) {
  Point dp = p1 - p0, dq = q1 - q0,
        num = dp * conj(dq);
  return q0 + (r - p0) * conj(num) / norm(dp);
}
```

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.

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
```

34 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 a) const { return {x-a.x, y-a.y, t}; }
  int quad() const {
    assert(x || y);
    if (y < 0) return (x >= 0) + 2;
    if (y > 0) return (x <= 0);
    return (x <= 0) * 2;
  }
  Angle t90() const { return {-y, x, t + (quad() == 3)}; }
  Angle t180() const { return {-x, -y, t + (quad() >= 2)}; }
  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.quad(), 1LL * a.y * b.x) <
         make_tuple(b.t, b.quad(), 1LL * a.x * 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) { // where b is a vector
  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;
}
```

Caliper.h

Description: A class for simulating the rotating calipers technique. Calipers will rotate covering the smallest arc. To change that, edit the code at (*) to add 2*kPi if return value is < 0

"Point.h" 18 lines

```
struct Caliper {
  Point pivot; double angle;

  double AngleTo(Point oth) {
    double new_ang = arg(oth - pivot);
    return remainder(new_ang - angle, 2.0 * kPi); // (*)
  }

  void RotateCCW(double ang) { angle += ang; }
  void ChangePivot(Point oth) { pivot = oth; }

  // Need to have same angle
  double DistanceTo(Caliper oth) {
    Point a = RotateCCW(pivot, -angle);
    Point b = RotateCCW(oth.pivot, -angle);
    return abs(a.imag() - b.imag());
  }
};
```

8.2 Circles

Circle.h

Description: Circle

"Point.h" 1 lines

```
struct Circle { Point c; double r; };
```

CircleIntersection.h

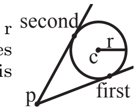
Description: Computes the intersection between two circles and other circle-related geometry

"Circle.h" 30 lines

```
// Computes the intersection of two circles.
// Can be 0(non-intersecting), 1(tangent), or 2 points
void CircleCircleIntersect(Circle c, Circle d,
                           vector<Point>& inter) {
  Point a = c.c, b = d.c, delta = b - a;
  double r1 = c.r, r2 = d.r;
  if (sgn(norm(delta)) == 0) return;
  double r = r1 + r2, d2 = norm(delta);
  double p = (d2 + r1 * r1 - r2 * r2) / (2.0 * d2);
  double h2 = r1 * r1 - p * p * d2;
  if (sgn(d2 - r * r) > 0 || sgn(h2) < 0) return;
  Point mid = a + delta * p,
        per = perp(delta) * sqrt(abs(h2) / d2);
  inter.push_back(mid - per);
  if (sgn(per) != 0)
    inter.push_back(mid + per);
}

// Computes the intersection between a line pq and a circle
// Can be 0(non-intersecting), 1(tangent), or 2 points
void LineCircleIntersect(Circle c, Point p, Point q,
                         vector<Point>& inter) {
  Point mid = ProjectPointOnLine(c.c, p, q);
  double d2 = norm(mid - c.c), dist = c.r * c.r - d2;
  if (sgn(dist) < 0) return;
  Point dir = (q - p) * sqrt(dist) / abs(q - p);
  inter.push_back(mid - dir);
  if (sgn(dist) != 0)
    inter.push_back(mid + dir);
}
```

circleTangents.h

Description:



Returns a pair of the two points on the circle with radius r centered around c whose tangent lines intersect p. If p lies within the circle NaN-points are returned. The first point is the one to the right as seen from the p towards c.

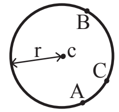
Usage: `auto p = Tangents(Point(100, 2), Point(0, 0), 2);`

"Circle.h" 6 lines

```
pair<Point, Point> Tangents(Point p, Circle c) {
  p -= c.c;
  double x = c.r * c.r / norm(p), y = sqrt(x - x * x);
  return make_pair(c.c + p * x + perp(p) * y,
                   c.c + p * x - perp(p) * y);
}
```

circumcircle.h

Description:



The circumcirle of a triangle is the circle intersecting all three vertices. CircumRadius returns the radius of the circle going through points a, b and c and CircumCenter returns the center of the same circle.

"Circle.h" 12 lines

```
double CircumRadius(Point a, Point b, Point c) {
  return dist(a, b) * dist(b, c) * dist(c, a) /
    abs(det(a, b, c)) / 2.;
}
Point CircumCenter(Point a, Point b, Point c) {
  c -= a; b -= a;
  return a + perp(c*norm(b) - b*norm(c)) / cross(c, b) / 2.;
}
Circle CircumCircle(Point a, Point b, Point c) {
  Point p = CircumCenter(a, b, c);
  return {p, abs(p - a)};
}
```

MinimumEnclosingCircle.h

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

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

"Circumcircle.h" 21 lines

```
// IMPORTANT: random_shuffle(pts.begin(), pts.end())
Circle MEC(vector<Point>& pts, vector<Point> ch = {}) {
  if (pts.empty() || ch.size() == 3) {
    switch (ch.size()) {
      case 0: return {0, -1};
      case 1: return {ch[0], 0};
      case 2: return {(ch[0] + ch[1])/2, abs(ch[0] - ch[1])/2};
      case 3: return CircumCircle(ch[0], ch[1], ch[2]);
      default: assert(false);
    }
  }

  auto p = pts.back(); pts.pop_back();
  auto c = MEC(pts, ch);
  if (sgn(abs(p - c.c) - c.r) > 0) {
    ch.push_back(p);
    c = MEC(pts, ch);
  }
  pts.push_back(p);
  return c;
}
```

8.3 Polygons

insidePolygon.h

Description: Returns true if p lies within the polygon described by the points between iterators begin and end. Returns 0 if on polygon, 1 if inside polygon and -1 if outside. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within epsilon from an edge should be considered as on the edge replace the line ”if (onSegment...” with the comment bellow it (this will cause overflow for int and long long).
Usage: `typedef Point<int> pi;`
```
vector<pi> v; v.push_back(pi(4,4));
v.push_back(pi(1,2)); v.push_back(pi(2,1));
bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);
```
Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h" 10 lines
```
int InsidePolygon(vector<Point> P, const Point& p) {
  int ic = 0, n = P.size();
  for (int i = 0, j = n - 1; i < n; j = i++) {
    if (OnSegment(P[i], P[j], p)) return 0;
    ic += (max(P[i].Y(), P[j].Y()) > p.Y() &&
           min(P[i].Y(), P[j].Y()) <= p.Y() &&
          (det(P[i], P[j], p) > 0) == (P[i].Y() <= p.Y()));
  }
  return ic % 2 ? 1 : -1; //inside if odd number of
       intersections
}
```

InsidePolygonMulti.h

Description: Given a (possibly non-convex) polygon P and Q query points, computes if the points are inside P or not. Returns -1 for strictly outside, 0 for edge, 1 for strictly inside. If no points are on the polygon, you can remove the events of type 2 completely.
Time: $\mathcal{O}((N+Q)\log N)$

<bits/stdc++.h>, <bits/extc++.h> 57 lines
```
using namespace __gnu_pbds;

vector<int> PointsInPolygon(vector<Point> P, vector<Point> Q) {
  int n = P.size(), q = Q.size();

  // Step 1: add events to sweepline
  vector<tuple<Point, int, int, int>> events;
  auto process = [&](int i, int j) {
    events.emplace_back(P[i], 2, i, i);
    if (P[j] < P[i]) swap(i, j);
    if (P[i].real() == P[j].real()) {
      events.emplace_back(P[i], 2, i, j);
    } else {
      events.emplace_back(P[i], 1, i, j);
      events.emplace_back(P[j], 0, i, j);
    }
  };
  for (int i = 0; i < n; ++i) process(i, (i + 1) % n);
  for (int i = 0; i < q; ++i)
    events.emplace_back(Q[i], 3, i, -1);

  // Step 2: Prepare sweepline status
  sort(events.begin(), events.end());
  auto cmp = [](pair<Point, Point> p1, pair<Point, Point> p2) {
    Point a, b, p, q; tie(a, b) = p1; tie(p, q) = p2;

    int v = sgn(det(a, b, p)) + sgn(det(a, b, q));
    if (v != 0) return v > 0;
    return sgn(det(p, q, a)) + sgn(det(p, q, b)) < 0;
  };

  tree<pair<Point, Point>, null_type, decltype(cmp),
    rb_tree_tag, tree_order_statistics_node_update> s(cmp);
  vector<int> ans(q);
  Point vert{-1, -1};
  vert *= (int)(2e9);

  // Step 3: Solve
  for (auto itr : events) {
    int tp, i, j; tie(ignore, tp, i, j) = itr;

    if (tp == 0) s.erase({P[i], P[j]});
    if (tp == 1) s.insert({P[i], P[j]});
    if (tp == 2) vert = max(vert, P[j]);
    if (tp == 3) {
      auto q = Q[i];
      auto it = s.lower_bound({q, q});
      int dist = s.order_of_key({q, q});
      ans[i] = (dist % 2 ? 1 : -1);

      if ((it != s.end() && det(it->first, it->second, q) == 0)
          || (vert.real() == q.real() && vert.imag() >= q.imag
            ()))
        ans[i] = 0;
    }
  }
  return ans;
}
```

PolygonArea.h

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

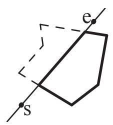
"Point.h" 6 lines
```
double SignedArea(const vector<Point> &P) {
  double area = cross(P.back(), P.front());
  for (int i = 1; i < (int)P.size(); ++i)
    area += cross(P[i - 1], P[i]);
  return area; // Divide by 2 for proper area
}
```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

"Point.h" 8 lines
```
Point PolygonCenter(vector<Point>& P) {
  int n = P.size(); Point res{0, 0}; double area = 0;
  for (int i = 0, j = n - 1; i < n; j = i++) {
    res += (P[i] + P[j]) * cross(P[j], P[i]);
    area += cross(P[j], P[i]);
  }
  return res / area / 3.0;
}
```

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<Point> p = ...;`
```
p = PolygonCut(p, Point(0, 0), Point(1, 0));
```

"Point.h", "LineIntersection.h" 13 lines
```
vector<Point> PolygonCut(vector<Point>& P, Point s, Point e) {
    vector<Point> res;
    for (int i = 0; i < (int)P.size(); ++i) {
        Point cur = P[i], prev = i ? P[i - 1] : P.back();
        int side1 = sgn(det(s, e, cur));
        int side2 = sgn(det(s, e, prev));
        if (side1 * side2 == -1) {
            res.push_back(LineIntersection(s, e, cur, prev));
        }
        if (side1 <= 0) res.push_back(cur);
    }
    return res;
}
```

Voronoi.h

Description: Determines the voronoi cell of a point with a list of other points. If the cell is unbounded, check for points with very high coordinates.
Time: $\mathcal{O}(N^2)$

"Point.h", "PolygonCut.h", <bits/stdc++.h> 18 lines
```
const double kInf = 1e9;

// To the right of mediator is region closer to b
pair<Point, Point> Mediator(Point a, Point b) {
  Point m = (a + b) * .5;
  return make_pair(m, m + perp(b - a));
}

vector<Point> VoronoiCell(Point p, vector<Point> P) {
  vector<Point> ret = {{-kInf, -kInf}, {kInf, -kInf},
    {kInf, kInf}, {-kInf, kInf}};

  for (auto oth : P) {
    Point a, b; tie(a, b) = Mediator(p, oth);
    ret = PolygonCut(ret, b, a);
  }
  return ret;
}
```

PolygonDiameter.h

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

"ConvexHull.h" 19 lines
```
vector<pii> antipodal(const vector<P>& S, vi& U, vi& L) {
  vector<pii> ret;
  int i = 0, j = sz(L) - 1;
  while (i < sz(U) - 1 || j > 0) {
    ret.emplace_back(U[i], L[j]);
    if (j == 0 || (i != sz(U)-1 && (S[L[j]] - S[L[j-1]])
          .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
    else --j;
  }
  return ret;
}

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

PointInsideHull.h

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

"Point.h", "sideOf.h", "onSegment.h" 22 lines
```
typedef Point<ll> P;
int insideHull2(const vector<P>& H, int L, int R, const P& p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return -1;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == sz(H)))
      return 0;
    return 1;
  }
  int mid = L + len / 2;
```

```
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}

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

LineHullIntersection.h

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

"Point.h" 63 lines

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

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

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

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

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

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

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

HalfplaneSet.h

Description: Data structure that dynamically keeps track of the intersection of halfplanes. Use is straigntforward. Area should be able to be kept dynamically with some modifications.
Usage: HalfplaneSet hs;
hs.Cut({0, 0}, {1, 1});
double best = hs.Maximize({1, 2});
Time: $\mathcal{O}(\log n)$

"Point.h", "LineIntersection.h", "Angle.h" 62 lines

```
struct HalfplaneSet : multimap<Angle, Point> {
  using Iter = multimap<Angle, Point>::iterator;

  HalfplaneSet() {
    insert({{+1, 0}, {-kInf, -kInf}});
    insert({{0, +1}, {+kInf, -kInf}});
    insert({{-1, 0}, {+kInf, +kInf}});
    insert({{0, -1}, {-kInf, +kInf}});
  }

  Iter get_next(Iter it) {
    return (next(it) == end() ? begin() : next(it)); }
  Iter get_prev(Iter it) {
    return (it == begin() ? prev(end()) : prev(it)); }
  Iter fix(Iter it) { return it == end() ? begin() : it; }

  // Cuts everything to the RIGHT of a, b
  // For LEFT, just swap a with b
  void Cut(Angle a, Angle b) {
    if (empty()) return;
    int old_size = size();

    auto eval = [&](Iter it) {
      return sgn(det(a.p(), b.p(), it->second)); };
    auto intersect = [&](Iter it)  {
      return LineIntersection(a.p(), b.p(),
          it->second, it->first.p() + it->second);
    };

    auto it = fix(lower_bound(b - a));
    if (eval(it) >= 0) return;

    while (size() && eval(get_prev(it)) < 0)
      fix(erase(get_prev(it)));
    while (size() && eval(get_next(it)) < 0)
      it = fix(erase(it));

    if (empty()) return;

    if (eval(get_next(it)) > 0) it->second = intersect(it);
    else it = fix(erase(it));
    if (old_size <= 2) return;
    it = get_prev(it);
    insert(it, {b - a, intersect(it)});
    if (eval(it) == 0) erase(it);
  }

  // Maximizes dot product
  double Maximize(Angle c) {
    assert(!empty());
    auto it = fix(lower_bound(c.t90()));
    return dot(it->second, c.p());
  }

  double Area() {
    if (size() <= 2) return 0;
    double ret = 0;
    for (auto it = begin(); it != end(); ++it)
      ret += cross(it->second, get_next(it)->second);
    return ret;
  }
};
```

hullCorect.cpp

Description: wtf

53 lines

```
struct pt {
    long long x, y;

    bool operator<(const pt& b) const
    {
        if (x != b.x)
            return x < b.x;
        return y < b.y;
    }
};

int orientation(pt a, pt b, pt c) {
    long long v = a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x *
        (a.y - b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c)
    == 0; }

void convex_hull(vector<pt>& a, bool include_collinear = false)
    {
    if (a.size() <= 2)
        return;

    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
        });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x - a.x) * (p0.x - a.x) + (p0.y - a.y) *
                (p0.y - a.y)
            < (p0.x - b.x) * (p0.x - b.x) + (p0.y - b.y) * (p0.
                y - b.y);
        return o < 0;
        });
    if (include_collinear) {
        int i = (int)a.size() - 1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
```

```
        reverse(a.begin() + 1 + i, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size() - 2], st.back
             (), a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    a = st;
}
```

8.4 Misc. Point Set Problems

closestPair.h

Description: Returns the indices to the closest pair of points in the point vector *pts* after the call. The distance can be easily computed. Might fail when using floating point (distance should be arbitrarily close though).

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

"Point.h" 41 lines

```
using T = long long;
using Point = complex<T>;

pair<int, int> ClosestPair(vector<Point> pts) {
  int n = pts.size();

  vector<int> order(n);
  iota(order.begin(), order.end(), 0);
  sort(order.begin(), order.end(), [&](int a, int b) {
    return pts[a].real() < pts[b].real();
  });
  set<pair<T, int>> s;

  T best_dist = numeric_limits<T>::max();
  pair<int, int> sol;

  int ii = 0, jj = 0;
  while (ii < n) {
    T d = ceil(sqrt(best_dist));
    int i = order[ii], j = order[jj];

    if (i != j && pts[i].real() - pts[j].real() >= best_dist) {
      s.erase({pts[j].imag(), j});
      jj += 1;
    } else {
      auto it1 = s.lower_bound({pts[i].imag() - d, -1});
      auto it2 = s.upper_bound({pts[i].imag() + d, n});

      for (auto it = it1; it != it2; ++it) {
        T now_dist = norm(pts[i] - pts[it->second]);
        if (best_dist > now_dist) {
          best_dist = now_dist;
          sol = {i, it->second};
        }
      }
      s.insert({pts[i].imag(), i});
      ii += 1;
    }
  }
  return sol;
}
```

kdTree.h

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

"Point.h" 63 lines

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

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

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

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

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

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

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

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

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

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

DelaunayTriangulation.h

Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are colinear or any four are on the same circle, behavior is undefined.

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

"Point.h", "3dHull.h" 10 lines

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  trav(p, ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) trav(t, hull3d(p3)) if ((p3[t.b]-p3[t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

8.5 3D

PolyhedronVolume.h

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

6 lines

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

Point3D.h

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

28 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 norm() const { return x*x + y*y + z*z; }
  double abs() const { return sqrt((double)norm()); }
  P unit() const { return *this / (T)abs(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u * dot(u) * (1-c) + (*this) * c - cross(u) * s;
  }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.

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

"Point3D.h" 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);
    }
  }
  trav(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: Conversions to/from spherical coordinates and great circle distance formula

18 lines

```
Point3D FromSpherical(double r, double lat, double lon) {
  return Point3D{
    r * cos(lat) * cos(lon),
    r * cos(lat) * sin(lon),
    r * sin(lat)};
}

void ToSpherical(Point3D p, double& r,
                 double& lat, double& lon) {
  r = p.abs(); lat = asin(p.z / r); lon = atan2(p.y, p.x);
}

double SphericalDistance(double r, double lat1, double lon1,
                         double lat2, double lon2) {
  double d = (FromSpherical(1.0, lat1, lon1)
           - FromSpherical(1.0, lat2, lon2)).abs();
  return 2 * r * asin(d / 2);
}
```

Strings (9)

KMP.h

Description: pi[x] is the length of the longest prefix of s that ends at x (exclusively), other than s[0..x) itself. This is used by Match() to find all occurences of a string.

Usage: `ComputePi("alabala") => {-1, 0, 0, 1, 0, 1, 2, 3}`
`Match("atoat", "atoatoat") => {4, 7}`

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

24 lines

```
vector<int> ComputePi(string s) {
  int n = s.size();
  vector<int> pi(n + 1, -1);

  for (int i = 0; i < n; ++i) {
    int j = pi[i];
    while (j != -1 && s[j] != s[i]) j = pi[j];
    pi[i + 1] = j + 1;
  }

  return pi;
}

vector<int> Match(string text, string pat) {
  vector<int> pi = ComputePi(pat), ret;
  int j = 0;

  for (int i = 0; i < (int)text.size(); ++i) {
    while (j != -1 && pat[j] != text[i]) j = pi[j];
    if (++j == pat.size())
      ret.push_back(i), j = pi[j];
  }
  return ret;
}
```

ZFunction.h

Description: z[i] is the length of the longest string that is, at the same time, a prefix of s and a prefix of the suffix of s starting at i

14 lines

```
int z[N];

void computeZFunction(const string &s) {
  int n = s.length();
  for(int i = 0; i < n; ++i) {
    int l = -1, r = -1;
    for(int j = i + 1; j < n; ++j) {
      int k = (j > r ? 0 : min(z[j - l + i], r - j + 1));
      while(j + k < n && s[i + k] == s[j + k]) ++k;
      z[j] = k;
      if(j + k - 1 > r) l = j, r = j + k - 1;
    }
  }
}
```

Manacher.h

Description: wtf

25 lines

```
int odd[N], even[N];
/*
 * [i - odd[i], i + odd[i]] - longest palindrome with center in
     i
 * [i - even[i], i + even[i] - 1] - longest palindrome with
     center in (i-1, i)
 */

void manacher(const string &s) {
  int l = 0, r = -1, n = s.length();
  for(int i = 0; i < n; ++i) {
    odd[i] = (i > r ? 1 : min(odd[l + r - i], r - i));
    while(i - odd[i] >= 0 && i + odd[i] < n && s[i - odd[i]] ==
        s[i + odd[i]]) ++odd[i];
    --odd[i];

    if(i + odd[i] > r) l = i - odd[i], r = i + odd[i];
  }

  l = 0; r = -1;
  for(int i = 0; i < n; ++i) {
    even[i] = (i > r ? 1 : min(even[l + r - i + 1], r - i));
    while(i - even[i] >= 0 && i + even[i] - 1 < n && s[i - even
        [i]] == s[i + even[i] - 1]) ++even[i];
    --even[i];

    if(i + even[i] - 1 > r) l = i - even[i], r = i + even[i] -
        1;
  }
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin()+MinRotation(v), v.end());`

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

11 lines

```
int MinRotation(string s) {
  int a = 0, n = s.size(); s += s;
  for (int b = 0; b < n; ++b)
    for (int i = 0; i < n; ++i) {
      if (a + i == b || s[a + i] < s[b + i]) {
        b += max(0, i - 1); break;
      }
      if (s[a + i] > s[b + i]) { a = b; break; }
    }
  return a;
}
```

SuffixAutomaton.h

Description: wtf

26 lines

```
const int N = 2000010;
const int A = 26;

int k, last;
int len[N], link[N];
int go[N][A];

void addLetter(int ch) {
  int p = last;
  last = k++;
  len[last] = len[p] + 1;

  while(!go[p][ch]) go[p][ch] = last, p = link[p];
  if(go[p][ch] == last) return void(link[last] = 0);

  int q = go[p][ch];
  if(len[q] == len[p] + 1) return void(link[last] = q);

  int cl = k++;
  memcpy(go[cl], go[q], sizeof go[q]);
  link[cl] = link[q];
  len[cl] = len[p] + 1;
  link[last] = link[q] = cl;

  while(go[p][ch] == q) go[p][ch] = cl, p = link[p];
}
```

SuffixArray.h

Description: wtf

29 lines

```
const int L = 200010;
const int LOGL = 18;

int sa[L];
int p[LOGL][L];

void buildSA(const string &s) {
  int n = s.length();
  for(int j = 0; j < n; ++j) p[0][j] = s[j];

  for(int i = 0; i + 1 < LOGL; ++i) {
    vector<pair<pii, int>> v;
    for(int j = 0; j < n; ++j)
      v.push_back({{p[i][j], j + (1 << i) < n ? p[i][j + (1 <<
          i)] : -1}, j});
    sort(all(v));

    for(int j = 0; j < n; ++j)
      p[i + 1][v[j].se] = (j && v[j - 1].fi == v[j].fi ? p[i +
          1][v[j - 1].se] : j);
  }

  for(int j = 0; j < n; ++j) sa[p[LOGL - 1][j]] = j;

  for(int i = 0, k = 0; i < n; ++i)
    if(p[LOGL - 1][i] != n - 1) {
      for(int j = sa[p[LOGL - 1][i] + 1]; i + k < n && j + k <
          n && s[i + k] == s[j + k]; ) ++k;
      //lcp[p[LOGL - 1][i]] = k;
      if(k) --k;
    }
}
```

AhoCorasickBicsi.h

Description: Aho-Corasick algorithm builds an automaton for multiple pattern string matching

Time: $\mathcal{O}(N * log(sigma))$ where N is the total length

<bits/stdc++.h> 48 lines

```
struct AhoCorasick {
  struct Node {
    int link;
    map<char, int> leg;
  };
  vector<Node> T;
  int root = 0, nodes = 1;

  AhoCorasick(int sz) : T(sz) {}

  // Adds a word to trie and returns the end node
  int AddWord(const string &word) {
    int node = root;
    for (auto c : word) {
      auto &nxt = T[node].leg[c];
      if (nxt == 0) nxt = nodes++;
      node = nxt;
    }
    return node;
  }

  // Advances from a node with a character (like an automaton)
  int Advance(int node, char chr) {
    while (node != -1 && T[node].leg.count(chr) == 0)
      node = T[node].link;
    if (node == -1) return root;
    return T[node].leg[chr];
  }

  // Builds links
  void BuildLinks() {
    queue<int> Q;
    Q.push(root);
    T[root].link = -1;

    while (!Q.empty()) {
      int node = Q.front();
      Q.pop();

      for (auto &p : T[node].leg) {
        int vec = p.second;
        char chr = p.first;
        T[vec].link = Advance(T[node].link, chr);
        Q.push(vec);
      }
    }
  }
};
```

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: $\mathcal{O}(26N)$

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;
  }
};
```

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

35 lines

```
struct IntervalContainer {
  map<int, int> s;
  using Iter = map<int, int>::iterator;

  Iter AddInterval(int l, int r) {
    if (l == r) return s.end();
    Iter it = s.lower_bound(l);
    while (it != s.end() && it->first <= r) {
      r = max(r, it->second);
      it = s.erase(it);
    }
    while (it != s.begin() && (--it)->second >= l) {
      l = min(l, it->first);
      r = max(r, it->second);
      it = s.erase(it);
    }
    return s.insert({l, r}).first;
  }

  Iter FindInterval(int x) {
    auto it = s.upper_bound(x);
    if (it == s.begin() or (--it)->second <= x)
      return s.end();
    return it;
  }

  void RemoveInterval(int l, int r) {
    if (l == r) return;
    auto it = AddInterval(l, r);
    int l2 = it->first, r2 = it->second;
    s.erase(it);
    if (l != l2) s.insert({l2, l});
    if (r != r2) s.insert({r, r2});
  }
};
```

IntervalCover.h

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

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

19 lines

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
  vi S(sz(I)), R;
  iota(all(S), 0);
  sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
```

```
  T cur = G.first;
  int at = 0;
  while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <= cur) {
      mx = max(mx, make_pair(I[S[at]].second, S[at]));
      at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. 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 = TernarySearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$ 12 lines

```
template<class Func>
int TernarySearch(int a, int b, Func 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;
  }
  for (int i = a + 1; i <= b; ++i)
    if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

Karatsuba.h

Description: Faster-than-naive convolution of two sequences: $c[x] = \sum a[i]b[x-i]$. Uses the identity $(aX + b)(cX + d) = acX^2 + bd + ((a + c)(b+d) - ac - bd)X$. Doesn't handle sequences of very different length well. See also FFT, under the Numerical chapter.
Time: $\mathcal{O}(N^{1.6})$ 1 lines

LIS.h

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

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

10.3 Dynamic programming

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)}(f(i, k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$ 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); }
};
```

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 $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
$dp[i] = minj < idp[j] + b[j] * a[i]whenb[j] >= b[j+1], a[i] <= a[i+1] : O(n^2) - > O(n)$
$dp[i][j] = mink < jdp[i-1][k] + b[k] * a[j]whenb[k] >= b[k+1], a[j] <= a[j+1] : O(kn^2) - > O(kn)$
$dp[i][j] = mink < jdp[i-1][k] + C[k][j], optim[i][j] <= optim[i][j+1] : O(kn^2) - > O(knlogn)$

Time: $\mathcal{O}(N^2)$ 1 lines

10.4 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` violations generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

10.5 Optimization tricks

10.5.1 Bit hacks

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

10.5.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
- `#pragma GCC target ("avx,avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

rand.cpp

Description: wtf 6 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count()); // for int - returns in [0, 2^32)
//mt19937_64 rng(chrono::steady_clock::now().time_since_epoch()
    .count()); // for long long - returns in [0, 2^64)
//uniform_int_distribution<> uniform(A, B);
//uniform_real_distribution<> uniform(A, B);
// usage: rng()
// usage: uniform(rng)
```

Techniques (A)

techniques.txt 159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contigous subvector sum
  Invariants
  Huffman encoding
Graph teory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algoritm
  MST: Prim's algoritm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Eulercykler
  Flow networks
  * Augumenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cutvertices, cutedges och biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programmering
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euklidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's small theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynom hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```