Universitat Politècnica de Catalunya

UPC 2

Michael Sammler, Eric Valls, Dean Zhu

SWERC 2017

November 26, 2017

Contest (1)

content.txt

```
Contest (1)
Mathematics (2)
Data structures (3)
Numerical (4)
Number theory (5)
Combinatorial (6)
Graph (7)
Geometry (8)
Strings (9)
Various (10)
```

template.cpp

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

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

const ll oo = 0x3f3f3f3f3f3f3f3fLL;

#define FOR(i, a, b) for(ll i = (a); i < int(b); i++)
#define FORD(i, a, b) for(ll i = (b)-1; i >= int(a); i--)
#define has(c, e) ((c).find(e) != (c).end())
#define sz(c) ll((c).size())
#define all(c) c.begin(),c.end()

int main() {
  cin.sync_with_stdio(0); cin.tie(0);
  cin.exceptions(cin.failbit);
  return 0;
}
```

Makefile

```
CXXFLAGS += -g -Wall -Wextra -Wshadow -std=c++14
```

troubleshoot.txt

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

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

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

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

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

Mathematics (2)

2.1 Equations

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

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

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

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

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

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

2.2 Recurrences

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

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

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

2.3 Trigonometry

$$\begin{aligned} \sin(v + w) &= \sin v \cos w + \cos v \sin w \\ cos(v + w) &= \cos v \cos w - \sin v \sin w \end{aligned}$$

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

$$\begin{aligned} \sin v + \sin w &= 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2} \\ \cos v + \cos w &= 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2} \end{aligned}$$

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

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

$$\begin{aligned} a \cos x + b \sin x &= r \cos(x - \phi) \\ a \sin x + b \cos x &= r \sin(x + \phi) \end{aligned}$$

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

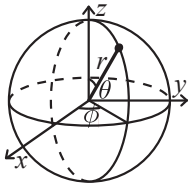
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.4.4 Pick's theorem

$A = i + b/2 - 1$

2.4.5 Volumes

	Sphere	Cube	Tetrahedron
Area	$4\pi r^2$	$6a^2$	$a^2\sqrt{3}$
Volume	$\frac{4}{3}\pi r^3$	a^3	$\frac{1}{12}a^3\sqrt{2}$

	Octahedron	Dodecahedron	Icosahedron
Area	$2\sqrt{3}a^2$	$3a^2\sqrt{25+10\sqrt{5}}$	$5\sqrt{3}a^2$
Volume	$\frac{1}{3}\sqrt{2}a^3$	$\frac{1}{4}(15+7\sqrt{5})a^3$	$\frac{5}{12}(3+\sqrt{5})a^3$

2.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \quad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

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

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \quad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

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

Integration by parts:

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

2.6 Sums

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

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

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

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

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

2.7 Series

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

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

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

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

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

2.7.1 Faulhaber's formulae

$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$
$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} = \frac{2n^3+3n^2+n}{6}$
$1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n^2+n}{2}\right)^2 = \frac{n^4+2n^3+n^2}{4}$
$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{6n^5+15n^4+10n^3-n}{30}$
$1^5 + 2^5 + 3^5 + \cdots + n^5 = \frac{2n^6+6n^5+5n^4-n^2}{12}$
$1^6 + 2^6 + 3^6 + \cdots + n^6 = \frac{6n^7+21n^6+21n^5-7n^3+n}{42}$

2.8 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

Poisson distribution

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

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

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

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

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

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

Data structures (3)

LineContainer.h

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

32 lines

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

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

SegmentTree.h

Description: Some useful functions for segment trees
Time: $\mathcal{O}(\log N)$

65 lines

```
const ll tot = (1 << 19); // bigger than N
ll N = 1; // ...
ll minv[tot * 2];
ll lazy[tot * 2];

void init(ll x, ll l, ll r) {
  if(l >= N) return;
  if(r - l <= 1) {
    minv[x] = 1; // ...
    lazy[x] = 0;
    return;
  }
  init(2*x, l, (l+r)/2);
  init(2*x+1, (l+r)/2, r);
  minv[x] = min(minv[2*x], minv[2*x+1]);
  lazy[x] = 0;
}

// propagates lazy, msut be called before recursing
void prop(ll x, ll l, ll r) {
  if(r - l <= 1) return;
  minv[2*x] += lazy[x];
  lazy[2*x] += lazy[x];
  minv[2*x+1] += lazy[x];
  lazy[2*x+1] += lazy[x];
  lazy[x] = 0;
}

// add v to [a, b)
void sadd(ll x, ll l, ll r, ll a, ll b, ll v) {
  if(b <= l || r <= a) return;
  if(a <= l && r <= b) {
    minv[x] += v;
    lazy[x] += v;
    return;
  }
  prop(x, l, r);
  sadd(2*x, l, (l+r)/2, a, b, v);
  sadd(2*x+1, (l+r)/2, r, a, b, v);
  minv[x] = min(minv[2*x], minv[2*x+1]);
}

// finds the lowest value res, such that all values
// in [res, end) are >= 2 (condition can be adjusted)
ll squery(ll x, ll l, ll r, ll end) {
  if(end <= l) return end;
  if(minv[x] >= 2) return l; // ...
  if(r - l <= 1) {
    return r;
  }
  prop(x, l, r);
  if(minv[2*x+1] >= 2) { // ...
    return squery(2*x, l, (l+r)/2, end);
  }
  ll rend = squery(2*x+1, (l+r)/2, r, end);
  if(rend > (l+r)/2) return rend;
  return squery(2*x, l, (l+r)/2, end);
}

void example() {
  ll a = 0, b = 1;
  init(1, 0, tot);
```

```
  sadd(1, 0, tot, a, b, -2);
  ll res = squery(1, 0, tot, a);
}
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$.

22 lines

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

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j].
Time: $\mathcal{O}(\log^2 N)$.

22 lines

```
struct FT2 {
    ll R, C;
    vector<vector<ll>> tree;

  // note r+1 & c+1
    NumMatrix(ll r, ll c) : R(r), C(c), tree(r+1, vector<ll>(c
        +1)) { }

    void update(int row, int col, int diff) {
        for(int i = row+1;i<=R;i += (i & -i))
            for(int j = col+1;j<=C;j += (j & -j))
                tree[i][j] += diff;
    }

    int sum(int row, int col) {
        int r = 0;
        // Not i >= 0
        for(int i = row;i > 0;i -= (i & -i))
            for(int j = col;j > 0;j -= (j & -j))
                r += tree[i][j];
        return r;
    }
};
```

OrderedSet.h

Description: Order Statistics tree
Usage: Set.find_by_order(k) returns an iterator to the k-th object, Set.order_of_key finds its rank in the set. (its index in the sorted set)

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

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

4 lines

```
using namespace __gnu_pbds;

template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;
```

RMQ.h

Description: Range minimum query
Usage: Call function build and then use query to return the smallest value, ST stores the index to the value.
Time: $< \mathcal{O}(N \log N), \mathcal{O}(1) >$

35 lines

```
struct SparseTable {
    vector<vector<int> > ST;
    vector<int> P;
    vector<int> v;
    int N;
    int MAXLOG = 0;

    void build(int n, const vector<int>& V){
        N = n;
        v = V;
        while ((1 << MAXLOG) <= N) ++MAXLOG;
        ST = vector<vector<int> > (N, vector<int> (MAXLOG));
        P = vector<int> (N+1);
        int LOG = 0;
        for(int i = 1; i < N + 1; ++i){
            P[i] = ((1<<LOG) > i ?  LOG-1 : ++LOG-1);
        }

        for(int i = 0; i < N; ++i) ST[i][0] = i;
        for (int j = 1; j < MAXLOG; ++j) {
            for (int i = 0; i + (1 << j) - 1 < N; ++i) {
                if (V[ST[i][j - 1]] < V[ST[i + (1 << (j - 1))][
                    j - 1]])
                    ST[i][j] = ST[i][j - 1];
                else
                    ST[i][j] = ST[i + (1 << (j - 1))][j - 1];
            }
        }
    }

  // minimum in range [l, r] (both inclusive)
    int query(int l, int r){
        int LOG = P[r-l+1];
        return min(v[ST[l][LOG]], v[ST[r - (1 << LOG) + 1][LOG
            ]]);
    }
};
```

Treap.h

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

103 lines

```
struct Tree {
  Tree *left, *right, *parent; // if parent needed.
  ll x, y, count;
  Tree (ll x) : x(x), y(rand()), count(1) {
    left = right = parent = nullptr;
  }
};

inline int card(Tree* t) {return (t ? t->count : 0);}
inline void setp(Tree *t, Tree *p) {
  if(t) t->parent = p; }

void update(Tree* t) { //Update when pointers change
  if (!t) return;
  t->count = 1 + card(t->left) + card(t->right);
  setp(t->left, t);
  setp(t->right, t);
}

Tree* merge(Tree* t1, Tree* t2) {
  if (t1 == nullptr) return t2;
  if (t2 == nullptr) return t1;
  if (t1->y >= t2->y) {
    t1->right = merge(t1->right, t2);
    update(t1);
    return t1;
  } else {
    t2->left = merge(t1, t2->left);
    update(t2);
    return t2;
  }
}

// leaves on the left all nodes less than x.
pair<Tree*, Tree*> split(Tree* t, ll x) {
  if (t == nullptr) return {nullptr, nullptr};
  if (t->x < x) {// if (card(t->left) + 1 <= x) {
  // auto p = split(t->right, x - card(t->left) - 1);
    auto p = split(t->right, x);
    t->right = p.first;
    update(t);
    setp(t, nullptr);
    return {t, p.second};
  } else {
    auto p = split(t->left, x);
    t->left = p.second;
    update(t);
    setp(t, nullptr);
    return {p.first, t};
  }
}

Tree* insert(Tree* t, Tree* n) {
  auto p = split(t, n->x);
  t = merge(p.first, n);
  t = merge(t, p.second);
  return t;
}

// Devuelve cuantos hay <= x
int count(Tree* t, int x) {
  if (!t) return 0;
  if (t->x <= x)
    return 1 + card(t->left) + count(t->right, x);
  else return count(t->left, x);
}

Tree* update_node(Tree* &root, Tree* node, ll nx) {
  setp(node->left, nullptr);
  setp(node->right, nullptr);
  auto m = merge(node->left, node->right);
  auto p = node->parent;
  node->left = node->right = node->parent = nullptr;
  node->x = nx;
  update(node);
  if (p) {
    p->left == node ? p->left = m : p->right = m;
    if (m) m->parent = p;
    while (p) { update(p); p = p->parent; }
  } else {
```

```
    root = m;
  }
  return insert(root, node);
}

int main() {
  int n; cin >> n;
  vector<Tree*> nodes(n);
  Tree* treap = nullptr;
  for (int i = 0; i < n; ++i) {
    ll x; cin >> x;
    Tree* n = new Tree(x);
    update(n);
    nodes[i] = n;
    treap = insert(treap, n);
  }
  int m; cin >> m; // Update k-th node with value x
  for (int i = 0; i < m; ++i) {
    int k; ll x; cin >> k >> x;
    Tree* t = nodes[k-1];
    treap = update_node(treap, t, x);
  }
}
```

UnionFind.h

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

13 lines

```
struct UF {
  vi e;
  UF(int n) : e(n, -1) {}
  bool same_set(int a, int b) { return find(a) == find(b); }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
  void join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return;
    if (e[a] > e[b]) swap(a, b);
    e[a] += e[b]; e[b] = a;
  }
};
```

Numerical (4)

Polynomial.h

18 lines

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

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `vector<double> roots; Polynomial p(2);`
`p.a[0] = 2; p.a[1] = -3; p.a[2] = 1;`
`poly_roots(p,-1e10,1e10,roots); // x^2-3x+2=0`

"Polynomial.h" 25 lines

```
void poly_roots(const Polynomial& p, double xmin, double xmax,
     vector<double>& roots) {
  if (p.n == 1) { roots.push_back(-p.a.front()/p.a.back()); }
  else {
    Polynomial d = p;
    d.diff();
    vector<double> dr;
    poly_roots(d, xmin, xmax, dr);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    for (auto i = dr.begin(), j = i++; i != dr.end(); j = i++){
      double l = *j, h = *i, m, f;
      bool sign = p(l) > 0;
      if (sign ^ (p(h) > 0)) {
        //for(int i = 0; i < 60; ++i){
        while(h - l > 1e-8) {
          m = (l + h) / 2, f = p(m);
          if ((f <= 0) ^ sign) l = m;
          else h = m;
        }
        roots.push_back((l + h) / 2);
      }
    }
  }
}
```

PolyInterpolate.h

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

13 lines

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

Integrate.h

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

```
double h(double x) { return x*x + y*y + z*z <= 1; }
double g(double y) { ::y = y; return quad(h, -1, 1); }
double f(double z) { ::z = z; return quad(g, -1, 1); }
double sphereVol = quad(f, -1, 1), pi = sphereVol*3/4;
```

19 lines

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

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1})$ (mod p^k) where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$

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;
}
```

Simplex.h

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

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

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

68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
```

```
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }

  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }

  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

SolveLinear.h

Description: Solves $A*x=b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}\left(n^2m\right)$ 39 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = A.size(), m = x.size(), rank = 0, br, bc;
  if (n) assert(A[0].size() == m);
  // FOR(i, 0, n) FOR(j, 0, m) A[i][j] %= MOD; also b[i]...
  vi col(m); iota(col.begin(), col.end(), 0);

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

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    FOR(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

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

"SolveLinear.h" 7 lines

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

SolveLinearBinary.h

Description: Solves $Ax=b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}\left(n^2m\right)$ 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);
  FOR(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      FOR(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]);
    FOR(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    FOR(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;
    FOR(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

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. For convolutions of integers, 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: $\mathcal{O}\left(N \log N\right)$

<valarray> 29 lines

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

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

FFTIntegers.h

Description: NTT
Time: $\mathcal{O}(N \log N)$
63 lines

```
LL fpw(LL a, LL b, LL p){
    LL r = 1; while(b){if(b&1) r=r*a%p; a=a*a%p; b/=2;} return
        r;
}

const LL MOD = 2013265921; const LL ROOT = 440564289; // MOD ==
     15*(1<<27)+1 (prime)
vector<LL> e, er;                                    // ROOT
    has order 2^27
void FFT(vector<int> &x, LL d = 1){
  int n = x.size();
  if(n != e.size()){
    e.resize(n); er.resize(n);
    e[0] = 1; e[1] = fpw(ROOT,(1<<27)/n,MOD);
    er[0] = 1; er[1] = fpw(e[1],MOD-2,MOD);
    rep(i,2,n) e[i] = e[i-1] * e[1] % MOD;
    rep(i,2,n) er[i] = er[i-1] * er[1] % MOD;
  }
  if(d == -1) swap(e, er);
  rep(i,0,n){
    int j=0; for(int k=1; k<n; k<<=1, j<<=1) if(k&i) j++;   //
        haxy i cheetosy
    j>>=1; if(i<j) swap(x[i], x[j]);                        //
        haxy i cheetosy
  }
  int k=0;
  while((1<<k)<n) k++;
  for(int s=1; s<n; s<<=1){
    --k;
    for(int i=0; i<n; i+=2*s) rep(j,0,s){
      LL u = x[i+j], v = x[i+j+s]*e[j<<k]%MOD;
      x[i+j] = u+v-(u+v>=MOD?MOD:0);
      x[i+j+s] = u-v+(u-v<0?MOD:0);
    }

  }
  if(d == -1) swap(e, er);
}

vector<int> convolution(vector<int> a, vector<int> b){
  int n = 1; while(n < (int)max(a.size(), b.size())) n *= 2;
  n *= 2; a.resize(n); b.resize(n);
  FFT(a); FFT(b); rep(i,0,n) a[i] = (LL)a[i]*b[i]%MOD*fpw(n,MOD
      -2,MOD)%MOD; FFT(a, -1);
  return a;
}

//2**13 : 8192 16384 32768
vector<int> Mult(vector<int> a, vector<int> b) {
  int exponent = 11; //change so that M/2^exp is as small as
       possible
  int sqrt2 = 1 << (exponent);
  int mod = 1e9 + 7; //has to be smaller than mod from ntt
  vector<int> a1(a.size()), a2(a.size()), b1(b.size()), b2(b.
      size());
  for(int i = 0; (int) i < a.size(); ++i) {
    a2[i] = a[i]/sqrt2; a1[i] = a[i]%sqrt2;
  }
  for(int i = 0; (int) i < b.size(); ++i) {
    b2[i] = b[i]/sqrt2; b1[i] = b[i]%sqrt2;
  }
  vector<int> a1b1 = convolution(a1,b1), a1b2 = convolution(a1,
      b2), a2b1 = convolution(a2,b1), a2b2 = convolution(a2,b2
      );
  vector<int> ans(a1b1.size());
  for(int i = 0; i < (int) a1b1.size(); ++i) {
    long long z = a1b1[i]%mod;
    long long x = ((long long)(a1b2[i] + a2b1[i]) * (1LL <<
        exponent)%mod)%mod;
    long long y = ((long long)(a2b2[i]) * (1LL << (2*exponent))
        %mod)%mod;;
    z += x; z %= mod; z += y; z %= mod; ans[i] = z;
  }
  return ans;
}
```

Determinant.h

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

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

const ll mod = 1000000009;
ll det(vector<vector<ll>>& a) {
  int n = sz(a); ll ans = 1;
  rep(i,0,n) {
    rep(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step
        ll t = a[i][i] / a[j][i];
        if (t) rep(k,i,n)
          a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + mod) % mod;
}
```

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

Description: Operators for modular arithmetic. You need to set `mod` to some number first and then you can use the structure.
"euclid.h"
18 lines

```
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

ModInverse.h

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

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

ModPow.h

6 lines

```
const ll mod = 1000000007; // faster if const
ll modpow(ll a, ll e) {
  if (e == 0) return 1;
  ll x = modpow(a * a % mod, e >> 1);
  return e & 1 ? x * a % mod : x;
}
```

ModSum.h

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

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

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

ll modsum(ull to, ll c, ll k, ll m) {
  c %= m;
  k %= m;
  if (c < 0) c += m;
  if (k < 0) k += m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

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

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

```
    x += (a * (b & (po - 1))) % c;
  }
  return x % c;
}
ull mod_pow(ull a, ull b, ull mod) {
  if (b == 0) return 1;
  ull res = mod_pow(a, b / 2, mod);
  res = mod_mul(res, res, mod);
  if (b & 1) return mod_mul(res, a, mod);
  return res;
}
```

ModSqrt.h

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

"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;
  }
}
```

5.2 Primality

eratosthenes.h

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

11 lines

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

MillerRabin.h

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

"ModMulLL.h" 16 lines

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

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `init(bits)`, where bits is the length of the numbers you use. to get factor multiple times, uncomment comments with (*)
Time: Expected running time should be good enough for 50-bit numbers.

"MillerRabin.h", "eratosthenes.h", "euclid.h" 37 lines

```
vector<ull> pr;
ull f(ull a, ull n, ull &has) {
  return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
  vector<ull> res;
  for (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 /* || true (*)*/) res.push_back(d);
        break;
      }
    }
  }
  return res;
}
void init(int bits) {//how many bits do we use?
  vi p = eratosthenes_sieve(1 << ((bits + 2) / 3));
  pr.resize(p.size());
  for (size_t i=0; i<pr.size(); i++)
    pr[i] = p[i];
}
```

5.3 Divisibility

euclid.h

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

7 lines

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

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

5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

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

10 lines

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

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for(int i = 3; i < LIM; i += 2)
    if(phi[i] == i)
      for(int j = i; j < LIM; j += i)
        (phi[j] /= i) *= i-1;
}
```

5.4 Chinese remainder theorem

chinese.h

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

"euclid.h" 13 lines

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

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

5.5 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.

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

5.7 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 The Twelvefold Way

Counts the # of functions $f : N \to K$, $|N| = n$, $|K| = k$. The elements in N and K can be distinguishable or indistinguishable, while f can be injective (one-to-one) of surjective (onto).

N	K	none	injective	surjective
dist	dist	k^n	$\frac{k!}{(k-n)!}$	$k!S(n,k)$
indist	dist	$\binom{n+k-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
dist	indist	$\sum_{t=0}^{k} S(n,t)$	$[n \le k]$	$S(n,k)$
indist	indist	$\sum_{t=1}^{k} p(n,t)$	$[n \le k]$	$p(n,k)$

Here, $S(n,k)$ is the Stirling number of the second kind, and $p(n,k)$ is the partition number.

6.2 Permutations

6.2.1 Factorial

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

intperm.h

Description: Permutations to/from integers. The bijection is order preserving.

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

20 lines

```
int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040}; // etc.
template <class Z, class It>
void perm_to_int(Z& val, It begin, It end) {
  int x = 0, n = 0;
  for (It i = begin; i != end; ++i, ++n)
    if (*i < *begin) ++x;
  if (n > 2) perm_to_int<Z>(val, ++begin, end);
  else val = 0;
  val += factorial[n-1]*x;
}
/* range [begin, end) does not have to be sorted. */
template <class Z, class It>
void int_to_perm(Z val, It begin, It end) {
  Z fac = factorial[end - begin - 1];
  // Note that the division result will fit in an integer!
  int x = val / fac;
  nth_element(begin, begin + x, end);
  swap(*begin, *(begin + x));
  if (end - begin > 2) int_to_perm(val % fac, ++begin, end);
}
```

6.2.2 Cycles

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

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

6.2.3 Derangements

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

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

derangements.h

Description: Generates the i:th derangement of S_n (in lexicographical order).

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));
    }
  }
};
```

6.2.4 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

6.2.5 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.

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

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

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

6.2.6 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. $\pi(j) > j$.

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

6.2.7 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

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

6.3 Partitions and subsets

6.3.1 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

6.3.2 Binomials

binomial.h

Description: The number of k-element subsets of an n-element set, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time: $\mathcal{O}(\min(k, n-k))$

6 lines

```
ll choose(int n, int k) {
  ll c = 1, to = min(k, n-k);
  if (to < 0) return 0;
  rep(i,0,to) c = c * (n - i) / (i + 1);
  return c;
}
```

binomialModPrime.h

Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.

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

10 lines

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

RollingBinomial.h

Description: $\binom{n}{k} \pmod m$ in time proportional to the difference between (n, k) and the previous (n, k).

14 lines

```
const ll mod = 1000000007;
vector<ll> invs; // precomputed up to max n, inclusively
struct Bin {
  int N = 0, K = 0;  ll r = 1;
  void m(ll a, ll b) { r = r * a % mod * invs[b] % mod; }
  ll choose(int n, int k) {
    if (k > n || k < 0) return 0;
    while (N < n) ++N, m(N, N-K);
    while (K < k) ++K, m(N-K+1, K);
    while (K > k) m(K, N-K+1), --K;
    while (N > n) m(N-K, N), --N;
    return r;
  }
};
```

multinomial.h

Description: $\binom{\sum k_i}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$

Time: $\mathcal{O}((\sum k_i) - k_1)$

6 lines

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

6.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

6.3.4 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

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

6.3.5 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 $z \neq x + y$.

6.4 General purpose numbers

6.4.1 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.

6.4.2 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

6.4.3 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

6.4.4 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

6.4.5 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)

EulerianCycle.h

Description: returns de eulerian cycle/tour starting at u, cycle is in reverse order. If its a tour it must start at a vertex with odd degree. It is common to add edges between odd vertex to find a pseudo euler tour.

Usage: Call find cycle with a vertex where a eulerian tour/cycle is possible, when adding edges make sure that two vertex have the same edge iff it is undirected.

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

25 lines

```
typedef vector<int> vi;
struct edge{
  int u, v;
  bool used;
};

void Eulerdfs(int u, vi &nxt, vi &Euler, vector<edge> &E, const
      vector<vi> &adj) {
  while(nxt[u] < adj[u].size()){
    int go = adj[u][nxt[u]++];
    if(!E[go].used){
      E[go].used = 1;
      int to = (E[go].u ^ E[go].v ^ u);
      Eulerdfs(to, nxt, Euler, E, adj);
    }
  }
  Euler.push_back(u);
}

vi Eulerian(int u, vector<edge> &E, const vector<vi> &adj) {
  vi nxt (adj.size(),0);
  vi Euler;
  Eulerdfs(u, nxt, Euler, E, adj);
  reverse(Euler.begin(), Euler.end());
  return Euler;
}
```

SCC.h

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

Usage: Use addedge to addedges in a directed graph(will also add reverse edges), after calling Kosaraju comp will save the component number of each vertex ordered by topological order.

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

27 lines

```
const int MAXN = 100010;
stack<int> st;
int m[MAXN], comp[MAXN];
vector<int> adj[2][MAXN];
int c = 0;

void addedge(vector<vector<int>> &adj, int u, int v) {
    adj[0][u].push_back(v);
    adj[1][v].push_back(u);
}

void dfs(int u, int t, vector<int>& m) {
    m[u] = 1;
    for(int v : adj[t][u]) if(!m[v]) dfs(v,t);
    if(t) comp[u] = c;
    else st.push(u);
}

void kosaraju(int n) {
    vector<int> m(n,0);
    for(int i = 0; i < n; ++i) if(!m[i]) dfs(i,0,m);
    m = vector<int>(n,0);
    for(;st.size();st.pop()) {
        int u = st.top();
        if(!m[u]) dfs(u,1), c++;
    }
}
```

2sat.h

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

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

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

57 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2*n) {}

  int add_var() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
```

```
    return N++;
  }

  void either(int f, int j) {
    f = (f >= 0 ? 2*f : -1-2*f);
    j = (j >= 0 ? 2*j : -1-2*j);
    gr[f^1].push_back(j);
    gr[j^1].push_back(f);
  }
  void set_value(int x) { either(x, x); }

  void at_most_one(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = add_var();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    trav(e, gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    ++time;
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = time;
      if (values[x>>1] == -1)
        values[x>>1] = !(x&1);
    } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph. In a biconnected component there are at least two distinct paths between any two nodes. A node can be in serveral or none components. if(true) at (*) creates pseudo bridge components. With these each node has at least one component and a node is an articulation point iff it has > 1 components. A bridge is not part of any cycle. To find biconnected components, which contain articulation points (see windmill graph), run without (*) and join all components of nodes, which are in more than one component, using union find and create new component for nodes without components. Then the components form a tree.

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

34 lines

```
vi num, st;
// TODO: resize and fill adj
vector<vi> adj, comps; // comps[i]: Components of node i
ll Time, C; // C: number of components
ll dfs(ll at, ll par) {
  ll me = num[at] = ++Time, top = me;
  st.push_back(at);
  bool first = true; // for detecting multi edges
  for(auto y: adj[at]) {
    if(y == par && first) { first = false; continue; }
    if (num[y]) {
      top = min(top, num[y]);
    } else {
      ll si = sz(st);
      ll up = dfs(y, at);
      top = min(top, up);
      if (up >= me) {
        if(up == me) { // (*) if(true) for bridge components
          FOR(p, si, sz(st)) comps[st[p]].push_back(C);
          comps[at].push_back(C++);
        }
        st.resize(si);
      }
      if(up > me) { /*e is a bridge*/ }
    }
  }
  return top;
}

void bicomps() {
  num.assign(sz(adj), 0); C = 0;
  comps.clear(); comps.resize(sz(adj));
  FOR(i,0,sz(adj)) if (!num[i]) dfs(i, -1);
}
```

MinimalArborescence.h

Description: Computes a minimal spanning tree in a directed graph from a root under the condition, that it exists. Set the values marked with TODO. Algo changes N and from!

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

65 lines

```
int N, R; // TODO number of nodes and number of root
vector<pii> from[2*MAXN]; // TODO for each node list of (
     predecessor, cost)
int pred[2*MAXN], label[2*MAXN];
bool del[2*MAXN];

bool cycle(int n, int p) {
  if (label[n] == 2) return false;
  if (label[n] == 1) {
    label[n] = -1;
    return true;
  }
  label[n] = 1;
  if (cycle(pred[n], n)) {
    if (label[n] == 1) {
      label[n] = -1;
    } else if (p != -1) {
      label[p] = 2;
    }
    return true;
  } else {
    label[n] = 2;
    return false;
  }
}
int rek() {
  int res = 0;
  pred[R] = -1;
  FOR(i, 0, N) {
    if (i == R || del[i]) continue;
    int m = oo;
    FOR(j, 0, sz(from[i])) m = min(m, from[i][j].second);
    res += m;
    FOR(j, 0, sz(from[i])) {
      from[i][j].second -= m;
      if (from[i][j].second == 0) pred[i] = from[i][j].first;
    }
  }
  FOR(i, 0, N) label[i] = 0;
  label[R] = 2;
  FOR(i, 0, N) {
    if (del[i] || label[i] == 2) continue;
    if (cycle(i, -1)) {
      FOR(j, 0, N) {
        if (del[j]) continue;
        if (label[j] == -1) {
          FOR(k, 0, sz(from[j])) if (label[from[j][k].first] !=
                -1)
            from[N].pb(from[j][k]);
          from[j].clear();
          del[j] = true;
        } else {
          FOR(k, 0, sz(from[j])) if (label[from[j][k].first] ==
                -1)
            from[j][k].first = N;
        }
      }
      del[N++] = false;
      return res + rek();
    }
  }
  return res;
}

int min_arbor() {
  FOR(i, 0, N) del[i] = false;
  return rek();
}
```

BinaryJumps.h

Description: Computes lca with binaryjumps. Insert nodes in dfs pre-order. NODES MUST BE 1 INDEXED! (or add 1 to the parameters in each function here)

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

29 lines

```
const ll maxN = 201005; // TODO
const ll MAXLOG = 20; // TODO: (1 << MAXLOG) > maxN
struct BinaryLifting {
  ll BL[maxN][MAXLOG];
  ll D[maxN];

  void init(ll root) { D[root] = 1; }
  void insert(ll n, ll par) {
    D[n] = D[par] + 1;
    BL[n][0] = par;
    FOR(j, 1, MAXLOG) BL[n][j] = BL[BL[n][j-1]][j-1];
  }
  ll lift(ll node, ll h) {
    ll cnt = 0;
    while(h) {
      if(h&1) node = BL[node][cnt];
      cnt++; h >>= 1;
    }
    return node;
  }
  int lca(int a, int b) {
    if(D[a] > D[b]) swap(a, b);
    FORD(i, 0, MAXLOG) if(D[BL[b][i]] >= D[a]) b = BL[b][i];
    if(a == b) return a;
    FORD(i, 0, MAXLOG) if(BL[a][i] != BL[b][i])
      a = BL[a][i], b = BL[b][i];
    return BL[a][0];
  }
};
```

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;
  }
};
```

LengauerTarjan.h

Description: Computes dominator tree of a directed graph from a root. Vertex A dominates B, if all possible paths from root to B go through A. Dominator tree contrains exactly one path root -> A -> B. initialize adjancy lists, call dominator_tree(r) with root r. Afterwards dom[i] is the parent of i in the dominator tree.
Time: $\mathcal{O}((E+V)\cdot \log V)$

48 lines

```
vi adj[N], rev[N];
int dom[N], semi[N], par[N], node[N], anc[N], label[N];
stack<int> bucket[N];

void dfs(int n, int &cnt) {
  node[cnt] = n;
  semi[n] = cnt++;
  for (const auto &to : adj[n]) {
    if (semi[to] == -1) { par[to] = n; dfs(to, cnt); }
    rev[to].pb(n);
  }
}

void compress(int n) {
  if (anc[anc[n]] != anc[n]) {
    compress(anc[n]);
    if (semi[label[anc[n]]] < semi[label[n]]) label[n] = label[
        anc[n]];
    anc[n] = anc[anc[n]];
  }
}
int eval(int n) {
  if (anc[n] == n) return n;
  compress(n);
  return label[n];
}
void dominator_tree(int r) {
  FOR(i, 0, N) dom[i] = semi[i] = -1;
  int cnt = 0;
  dfs(r, cnt);
  FOR(i, 0, N) anc[i] = label[i] = i;
  FORD(i, 1, cnt) {
    int n = node[i];
    FOR(i, 0, sz(rev[n])) semi[n] = min(semi[n], semi[eval(rev[
        n][i])]);
    bucket[node[semi[n]]].push(n);
    anc[n] = par[n];
    while (!bucket[par[n]].empty()) {
      int m = bucket[par[n]].top();
      bucket[par[n]].pop();
      int u = eval(m);
      dom[m] = (semi[u] < semi[m]) ? u : par[n];
    }
  }
  FOR(i, 1, cnt) {
    int n = node[i];
    if (dom[n] != node[semi[n]]) dom[n] = dom[dom[n]];
  }
  FOR(i, 0, N) rev[i].clear();
}
```

MaxFlow.h

Description: Returns maximum flow.
Usage: To obtain a cut in the mincut problem one must bfs from the source. All the vertices reached from it using only edges with capacity > 0 are in the same cut
Time: $\mathcal{O}\left(V^2 * E\right)$ for general graphs. For unit capacities $\mathcal{O}\left(min(V^(2/3), E^(1/2)) * E\right)$. For maximum matching $\mathcal{O}(E * sqrt(V))$)(bipartite unit weighted graf). It is generally very fast.

67 lines

```
typedef long long ll;
typedef vector<int> VI;
typedef vector<VI> VVI;
const ll INF = 1000000000000000000LL;

#define VEI(w,e) ((E[e].u == w) ? E[e].v : E[e].u)
#define CAP(w,e) ((E[e].u == w) ? E[e].cap[0] - E[e].flow : E[e
    ].cap[1] + E[e].flow)
#define ADD(w,e,f) E[e].flow += ((E[e].u == w) ? (f) : (-(f)))

struct Edge { int u, v; ll cap[2], flow; };

VI d, act;

bool bfs(int s, int t, VVI& adj, vector<Edge>& E) {
  queue<int> Q;
  d = VI(adj.size(), -1);
  d[t] = 0;
  Q.push(t);
  while (not Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int i = 0; i < int(adj[u].size()); ++i) {
      int e = adj[u][i], v = VEI(u, e);
      if (CAP(v, e) > 0 and d[v] == -1) {
        d[v] = d[u] + 1;
        Q.push(v);
      }
    }
  }
  return d[s] >= 0;
}

ll dfs(int u,int t,ll bot,VVI& adj,vector<Edge>& E) {
  if (u == t) return bot;
  for (; act[u] < int(adj[u].size()); ++act[u]) {
    int e = adj[u][act[u]];
```

```
    if (CAP(u, e) > 0 and d[u] == d[VEI(u, e)] + 1) {
      ll inc=dfs(VEI(u,e),t,min(bot,CAP(u,e)),adj,E);
      if (inc) {
        ADD(u, e, inc);
        return inc;
      }
    }
  }
  return 0;
}

ll maxflow(int s, int t, VVI& adj, vector<Edge>& E) {
  for (int i=0; i<int(E.size()); ++i) E[i].flow = 0;
  ll flow = 0, bot;
  while (bfs(s, t, adj, E)) {
    act = VI(adj.size(), 0);
    while ((bot = dfs(s,t,INF, adj, E))) flow += bot;
  }
  return flow;
}

void addEdge(int u, int v, VVI& adj, vector<Edge>& E, ll cap){
  Edge e;
  e.u = u;
  e.v = v;
  e.cap[0] = cap;
  e.cap[1] = 0;
  e.flow = 0;
  adj[u].push_back(E.size());
  adj[v].push_back(E.size());
  E.push_back(e);
}
```

MaxflowMinCap.h

Description: Normal maxflow but with minimum capacity in each edge
Time: $\mathcal{O}\left(V^2 * E\right)$ for general graphs. For unit capacities $\mathcal{O}\left(min(V^(2/3), E^(1/2)) * E\right)$

108 lines

```
typedef long long ll;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<ll> vll;
const ll INF = 1000000000000000000LL;

#define VEI(w,e) ((E[e].u == w) ? E[e].v : E[e].u)
#define CAP(w,e) ((E[e].u == w) ? E[e].cap[0] - E[e].flow : E[e
    ].cap[1] + E[e].flow)
#define ADD(w,e,f) E[e].flow += ((E[e].u == w) ? (f) : (-(f)))

struct Edge { int u, v; ll cap[2], mincap, flow; };

vi d, act;

bool bfs(int s, int t, vvi& adj, vector<Edge>& E) {
  queue<int> Q;
  d = vi(adj.size(), -1);
  d[t] = 0;
  Q.push(t);
  while (not Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int i = 0; i < int(adj[u].size()); ++i) {
      int e = adj[u][i], v = VEI(u, e);
      if (CAP(v, e) > 0 and d[v] == -1) {
        d[v] = d[u] + 1;
        Q.push(v);
      }
    }
  }
  return d[s] >= 0;
}

ll dfs(int u,int t,ll bot,vvi& adj,vector<Edge>& E) {
  if (u == t) return bot;
  for (; act[u] < int(adj[u].size()); ++act[u]) {
    int e = adj[u][act[u]];
    if (CAP(u, e) > 0 and d[u] == d[VEI(u, e)] + 1) {
      ll inc=dfs(VEI(u,e),t,min(bot,CAP(u,e)),adj,E);
      if (inc) {
        ADD(u, e, inc);
        return inc;
      }
    }
  }
  return 0;
}

ll maxflow(int s, int t, vvi& adj, vector<Edge>& E) {
  ll flow = 0, bot;
  while (bfs(s, t, adj, E)) {
    act = vi(adj.size(), 0);
    while ((bot = dfs(s,t,INF, adj, E))) flow += bot;
  }
  return flow;
}

void addEdge(int x,int y,ll c,ll m,vvi& adj,vector<Edge>& E) {
    Edge e;
    e.u = x;
    e.v = y;
    e.cap[0] = c - m;
    e.cap[1] = 0;
    e.mincap = m;
    adj[x].push_back(E.size());
    adj[y].push_back(E.size());
    E.push_back(e);
}

ll mincap(int s, int t, vvi& adj, vector<Edge>& E) {
    int n = adj.size();
    int m = E.size();
    vll C(n, 0);
    for (int i = 0; i < m; ++i) {
        C[E[i].u] -= E[i].mincap;
        C[E[i].v] += E[i].mincap;
    }
    adj.push_back(vi(0));
    adj.push_back(vi(0));
    ll flowsat = 0;
    for (int i = 0; i < n; ++i) {
        if (C[i] > 0) {
            addEdge(n, i, C[i], 0, adj, E);
            flowsat += C[i];
        }
        else if (C[i] < 0) addEdge(i, n + 1, -C[i], 0, adj, E);
    }
    addEdge(t, s, INF, 0, adj, E);
    for (int i = 0; i < (int)E.size(); ++i) E[i].flow = 0;
    if (flowsat != maxflow(n, n + 1, adj, E)) return -1;
    maxflow(s, t, adj, E);
    while ((int)E.size() > m) E.pop_back();
    adj.pop_back();
    adj.pop_back();
    for (int i = 0; i < n; ++i) {
        int j = (int)adj[i].size() - 1;
        while (j >= 0 and adj[i][j] >= m) {
            --j;
            adj[i].pop_back();
        }
    }
    ll flow = 0;
    for (int i = 0; i < m; ++i) {
        E[i].flow += E[i].mincap;
        if (E[i].u == s) flow += E[i].flow;
        else if (E[i].v == s) flow -= E[i].flow;
    }
    return flow;
}
```

MaxflowMinCost.h

Description: adj is and adjacency list, deg is the degree of the vertex. You shouldn't touch padre. w,cap are matrices, NOT LISTS!

79 lines

```
typedef pair < ll, ll > PLI;

const int INF = 2e9;

#define NN 505
#define pot(u,v) (pi[u]-pi[v])
int adj[NN][NN], deg[NN], padre[NN];
ll w[NN][NN], cap[NN][NN], pi[NN], d[NN], f[NN][NN], dist[NN];
int N;
ll flow, cost;
bool dijkstra(int s, int t) {
  memset(padre, -1, sizeof(padre));
  FOR(i,0,N) d[i] = INF;
  d[s] = 0;
  priority_queue<PLI> Q;
  Q.push(PLI(0,s));
  while (not Q.empty()) {
    int u = Q.top().second;
    ll dist = -Q.top().first;
    Q.pop();
    if (dist != d[u]) continue;
    FOR(i,0,deg[u]) {
      int v = adj[u][i];
      if (f[u][v] >= 0 and cap[u][v] - f[u][v] > 0 and
          d[v] > d[u] + pot(u,v) + w[u][v]) {
        d[v] = d[u] + pot(u,v) + w[u][v];
        Q.push(PLI(-d[v], v));
        padre[v] = u;
      }
      else if (f[u][v] < 0 and d[v] > d[u] + pot(u,v) - w[v][u
          ]) {
        d[v] = d[u] + pot(u,v) - w[v][u];
        Q.push(PLI(-d[v], v));
        padre[v] = u;
      }
    }
  }
  FOR(i,0,N) if (pi[i] < INF) pi[i] += d[i];
  return padre[t] >= 0;
}
void maxmin(int s, int t) {
  memset(f, 0, sizeof(f));
  memset(pi, 0, sizeof(pi));
  flow = cost = 0;
  while (dijkstra(s, t)) {
    ll bot = INF;
    for (int v = t, u = padre[v]; u != -1; v = u, u = padre[u])
        {
      if (f[u][v] >= 0) bot = min(cap[u][v] - f[u][v], bot);
      else bot = min(f[v][u], bot);
    }
    for (int v = t, u = padre[v]; u != -1; v = u, u = padre[u])
        {
      if (f[u][v] >= 0) cost += w[u][v]*bot;
```

```
      else cost -= w[v][u]*bot;
      f[u][v] += bot;
      f[v][u] -= bot;
    }
    flow += bot;
  }
}

void negative_edges(int s, int t) {
  for (int i = 0; i < N; ++i) dist[i] = INF;
  dist[s] = 0;
  for (int k = 0; k < N; ++k) {
    for (int x = 0; x < N; ++x) {
      for (int j = 0; j < deg[x]; ++j) {
        int y = adj[x][j];
        if (!cap[x][y]) continue;
        dist[y] = min(dist[x] + w[x][y], dist[y]);
      }}}
  for (int x = 0; x < N; ++x) {
    for (int j = 0; j < deg[x]; ++j) {
      int y = adj[x][j];
      if (!cap[x][y]) continue;
      w[x][y] += dist[x] - dist[y];
    }
  }
  maxmin(s, t);
  cost += flow*dist[t];
}
```

StableMarriage.h

Description: findMatch finds a stable matching where the first groups is prefered given the preference lists of two groups of persons

Usage: pr1[i] = preference list of person i (from 1. list) (format A), pr2[i][j] = position in the preference list of person j (from 1. list), in preference list of i (from 2. list) (format B)

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

28 lines

```
const int MAXN = 4 * 1024;

int pr1[MAXN][MAXN],pr2[MAXN][MAXN];
int match[MAXN]; // match[i]=partner of person i out of group 2
int res[MAXN];   // res[i]=partner of person i out of group 1
int id[MAXN];
int N;
void findMatch() {
  fill_n(match, N, -1);
  fill_n(id, N, 0);
  FOR(m,0,N) {
    int cur = m;
    while (true) {
      int ot = pr1[cur][id[cur]];
      if (match[ot] == -1) {
        match[ot] = cur;
        break;
      }
      if (pr2[ot][cur] < pr2[ot][match[ot]]) swap(match[ot],cur
          );
      id[cur]++;
    }
  }
  FOR(i,0,N) res[match[i]] = i;
}
// convert preference list of format A to format B and in
    reverse
int ranks[MAXN][MAXN]; // values to convert
int reranks[MAXN][MAXN]; // converted values
void convertRanklist() { FOR(i,0,N) FOR(j,0,N) reranks[i][ranks
    [i][j]] = j; }
```

MaxWeightBipartiteMatching.h

Description: finds a max weight bipartite matching. hungarian() returns the value of the max weight matching. xy contains for each vertex on the left the corresponding node right, yx the other way around.

Usage: initialize cost matrix (profit of matching on of the N left vertexes with one of the N right vertexes)

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

68 lines

```
int cost[N][N];
int xy[N], yx[N];
int lx[N], ly[N];
bool tree[N];
int slack[N], pred[N];

void init_matching() {
  fill_n(xy, N, -1);
  fill_n(yx, N, -1);
  fill_n(lx, N, 0);
  fill_n(ly, N, 0);
  FOR(x, 0, N) FOR(y, 0, N) lx[x] = max(lx[x], cost[x][y]);
}

void init_tree() {
  fill_n(tree, N, false);
  fill_n(pred, N, -1);
  FOR(x, 0, N) {
    if (xy[x] != -1) continue;
    tree[x] = true;
    FOR(y, 0, N) slack[y] = lx[x] + ly[y] - cost[x][y];
    break;
  }
}

bool augment() {
  bool found = true;
  while (found) {
    found = false;
    FOR(y, 0, N) {
      if (pred[y] != -1 || slack[y]) continue;
      found = true;
      FOR(x, 0, N) if (tree[x] && cost[x][y] == lx[x] + ly[y])
          pred[y] = x;
      if (yx[y] == -1) {
        while (y != -1) {
          yx[y] = pred[y];
          swap(y, xy[pred[y]]);
        }
        return true;
      } else if (!tree[yx[y]]) {
        tree[yx[y]] = true;
        FOR(yy, 0, N) {
          int v = lx[yx[y]] + ly[yy] - cost[yx[y]][yy];
          slack[yy] = min(slack[yy], v);
        }
      }
    }
  }
  return false;
}

void update_labels() {
  int delta = INT_MAX;
  FOR(y, 0, N) if (pred[y] == -1) delta = min(delta, slack[y]);
  FOR(x, 0, N) if (tree[x]) lx[x] -= delta;
  FOR(y, 0, N) if (pred[y] != -1) ly[y] += delta; else slack[y]
      -= delta;
}

int hungarian() {
  init_matching();
  FOR(i, 0, N) {
    init_tree();
    while (!augment()) update_labels();
  }
  int res = 0;
  FOR(x, 0, N) res += cost[x][xy[x]];
  return res;
}
```

globalmincut.h

Description: Given an adjacency matrix returns the global mincut and the vertices of one of the cuts.

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

84 lines

```
/*
 * If you dont need the cut you can eliminate every thing with
     this coment "// **********"
 *Explanation of algorithm:
 * -getting the mincut value: it does n-1 iterations. In each
     iteration it starts by a vertex (random) as set A.
 * then it iterates until only two vertices are left by adding
     to set A the most tightly connected vertex to A (vertex
     not in A).
 * it insert this vertex to A. When only two vertices are left
     , the mincut between those two is the weight W of the
     edges between
 * the last vertex and A. mincut = min(mincut, W)
 * We then merge the two last vertices and start again.
 *
 * -getting the cut: basically when we merge two nodes we
     merge them with mfset. When we obtain a new best mincut
     value, a cut
 * is represented buy the nodes in the same component as the
     last node;
 */

// Maximum number of vertices in the graph
#define NN 256
// Maximum edge weight (MAXW * NN * NN must fit into an int)
#define MAXW 1000
// Adjacency matrix and some internal arrays
int v[NN], w[NN];
bool a[NN];

int pare[NN]; // **********
int par (int b){ // **********
  if (pare[b] == b) return b;
  pare[b] = par(pare[b]);
  return pare[b];
}

inline void merge (int b, int c){ // **********
  pare[par(b)] = par(c);
}

pair < int, vi > minCut(vvi& g, int n) {
  int n1 = n;
  // init the remaining vertex set
  for (int i = 0; i < n; i++){
    v[i] = i;
    pare[i] = i; // **********
  }
  // run Stoer-Wagner
  int best = MAXW * n * n;
  vi cut; // **********
  while (n > 1) {
    // initialize the set A and vertex weights
    a[v[0]] = true;
```

```
    for (int i = 1; i < n; i++) {
      a[v[i]] = false;
      w[i] = g[v[0]][v[i]];
    }
    // add the other vertices
    int prev = v[0];
    for (int i = 1; i < n; i++) {
      // find the most tightly connected non-A vertex
      int zj = -1;
      for (int j = 1; j < n; j++)
        if (!a[v[j]] && (zj < 0 || w[j] > w[zj])) zj = j;
      // add it to A
      a[v[zj]] = true;
      // last vertex?
      if (i == n - 1) {
        // remember the cut weight
        if (best > w[zj]){
          best = w[zj];
          cut.clear(); // ********
          for (int ko = 0; ko < n1; ko++) if (par(ko) == par(v[
               zj])) cut.push_back(ko); // ********
        }

        // merge prev and v[zj]
        merge(prev, v[zj]); // *************
        for (int j = 0; j < n; j++)
          g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];
        v[zj] = v[--n];
        break;
      }
      prev = v[zj];
      // update the weights of its neighbours
      for (int j = 1; j < n; j++)
        if (!a[v[j]]) w[j] += g[v[zj]][v[j]];
    }
  }
  return {best, cut};
}
```

7.1 Karp's Minimum Mean Cycle

1. Initialize: $d_0(r) = 0$ and $\forall v \in V - \{r\}, d_0(v) = \infty$.
2. For $i = 1$ to n:
— For every $v \in V$ calculate:
—— $d_i(v) = \min_{u:(u,v)\in E} = \{d_{i-1}(u) + c((u, v))\}$

Optim $= \min_{v\in V} \max_{0\le k\le n-1} \left\{ \frac{d_n(v)-d_k(v)}{n-k} \right\}$

7.2 Planar Graphs

Formula Euler: v-e+f = 2
Theorem 1: If $v >= 3$ then $e <= 3v - 6$.
Theorem 2: If $v > 3$ and there are no cycles of length 3, then $e <= 2v - 4$.

7.3 Bipartite Graphs

let G = (V,E) be a Bipartite graph, we will call each part L and R respectively.

Maximum Matching is equal to **Minimum Vertex Cover**, to get the nodes in the MVC we choose the unmatched vertices of L and run a BFS/DFS in the graph with edges in the matching directed from R to L, and edges not in the matching directed from L to R. Let's all this Z, now MVC = (L\Z) ∪ (R ∩ Z).
The **Minimum Edge Cover** is obtained by doing a Maximum Matching, Then run a BFS/DFS from unmatched vertices of L, the MEC is the unreachable vertices of A and reachable vertices of B.
The **Maximum Independent Set** is the complementary of the **Minimum Vertex Cover**

7.3.1 DAG

The **Minimum Path Cover** is given by the edges in the MM of the bipartite graph after doubling the vertices.

7.4 ASSP : Johnson Algorithm

We have a sparse graph with, possibly, negative edges. We want to compute the all-pairs shortest path. Floyd Warshall may be too slow.
Algorithm:
1) A new node q is added to the graph, connected by zero-weight edges to each of the other nodes.
2) Bellman–Ford algorithm is used, starting from the new vertex q, to find for each vertex v the minimum weight h(v) of a path from q to v.
If this step detects a negative cycle, the algorithm is terminated as the shortest-path is undefined.
3) edges of the original graph are reweighted using the values computed by the Bellman–Ford algorithm:
an edge from u to v, having length w(u,v), is given by the new length w(u,v) + h(u) - h(v).
4) q is removed, and Dijkstra's algorithm is used to find the shortest paths from each node s to every other vertex in the reweighted graph.
5) Dijkstra from every node. The shortest path in this new graph is the same as in the initial graph although it's weight changes.
The idea is to do dijkstra by storing "weight in new graph" and "weight in old graph" sorting by the first one.

Geometry (8)

8.1 Geometric primitives

Point.h

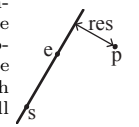
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

25 lines

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

lineDistance.h

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

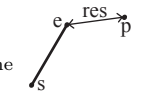


"Point.h" 4 lines

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

SegmentDistance.h

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



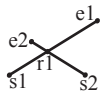
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 6 lines

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

SegmentIntersection.h

Description:
If a unique 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

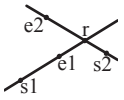
Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

"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);
}
```

lineIntersection.h

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



Usage:
```
point<double> intersection;
if (1 == LineIntersection(s1,e1,s2,e2,intersection))
cout << "intersection point at " << intersection << endl;
```
"Point.h" 9 lines

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

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage:
```
bool left = sideOf(p1,p2,q)==1;
```
"Point.h" 11 lines

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

onSegment.h

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

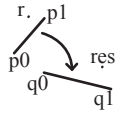
"Point.h" 5 lines

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

linearTransformation.h

Description:

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



"Point.h" 6 lines

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

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
```
37 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(), a.y * (ll)b.x) <
         make_tuple(b.t, b.quad(), a.x * (ll)b.y);
}
bool operator>=(Angle a, Angle b) { return !(a < b); }
bool operator>(Angle a, Angle b) { return b < a; }
bool operator<=(Angle a, Angle b) { return !(b < a); }

// 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 (r > a.t180()) r.t--;
  return r.t180() < a ? r.t360() : r;
}
```

8.2 Circles

CircleIntersection.h

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

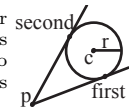
"Point.h" 14 lines

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

circleTangents.h

Description:

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



Usage: `typedef Point<double> P;`
`pair<P,P> p = circleTangents(P(100,2),P(0,0),2);`

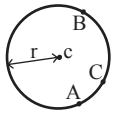
"Point.h" 6 lines

```
template <class P>
pair<P,P> circleTangents(const P &p, const P &c, double r) {
  P a = p-c;
  double x = r*r/a.dist2(), y = sqrt(x-x*x);
  return make_pair(c+a*x+a.perp()*y, c+a*x-a.perp()*y);
}
```

circumcircle.h

Description:

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



"Point.h" 9 lines

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

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h" 28 lines

```
pair<double, P> mec2(vector<P>& S, P a, P b, int n) {
  double hi = INFINITY, lo = -hi;
  rep(i,0,n) {
    auto si = (b-a).cross(S[i]-a);
    if (si == 0) continue;
    P m = ccCenter(a, b, S[i]);
    auto cr = (b-a).cross(m-a);
    if (si < 0) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  double v = (0 < lo ? lo : hi < 0 ? hi : 0);
  P c = (a + b) / 2 + (b - a).perp() * v / (b - a).dist2();
  return {(a - c).dist2(), c};
}
pair<double, P> mec(vector<P>& S, P a, int n) {
  random_shuffle(S.begin(), S.begin() + n);
  P b = S[0], c = (a + b) / 2;
  double r = (a - c).dist2();
  rep(i,1,n) if ((S[i] - c).dist2() > r * (1 + 1e-8)) {
    tie(r,c) = (n == sz(S) ?
      mec(S, S[i], i) : mec2(S, a, S[i], i));
  }
  return {r, c};
}
pair<double, P> enclosingCircle(vector<P> S) {
  assert(!S.empty()); auto r = mec(S, S[0], sz(S));
  return {sqrt(r.first), r.second};
}
```

8.3 Polygons

insidePolygon.h

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

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

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

PolygonArea.h

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

"Point.h" 6 lines

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

PolygonCenter.h

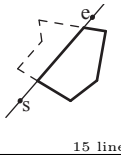
Description: Returns the center of mass for a polygon.

"Point.h" 10 lines

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

PolygonCut.h

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



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

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

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

ConvexHull.h

Description:
Returns a vector of indices of the convex hull in counterclockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

Usage: `vector<P> ps, hull;`
`trav(i, convexHull(ps)) hull.push_back(ps[i]);`
Time: $\mathcal{O}(n \log n)$

"Point.h" 20 lines

```
typedef Point<ll> P;
pair<vi, vi> ulHull(const vector<P>& S) {
  vi Q(sz(S)), U, L;
  iota(all(Q), 0);
  sort(all(Q), [&S](int a, int b){ return S[a] < S[b]; });
  trav(it, Q) {
#define ADDP(C, cmp) while (sz(C) > 1 && S[C[sz(C)-2]].cross(\
  S[it], S[C.back()]) cmp 0) C.pop_back(); C.push_back(it);
    ADDP(U, <=); ADDP(L, >=);
  }
  return {U, L};
}

vi convexHull(const vector<P>& S) {
  vi u, l; tie(u, l) = ulHull(S);
  if (sz(S) <= 1) return u;
  if (S[u[0]] == S[u[1]]) return {0};
  l.insert(l.end(), u.rbegin()+1, u.rend()-1);
  return l;
}
```

PolygonDiameter.h

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

"ConvexHull.h" 19 lines

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

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

PointInsideHull.h

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

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

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

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

LineHullIntersection.h

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

"Point.h" — 63 lines

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

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

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

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

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

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

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

8.4 Misc. Point Set Problems

closestPair.h

Description: $i1$, $i2$ are the indices to the closest pair of points in the point vector p after the call. The distance is returned.
Time: $\mathcal{O}(n\log n)$

"Point.h" — 58 lines

```
template <class It>
bool it_less(const It& i, const It& j) { return *i < *j; }
template <class It>
bool y_it_less(const It& i,const It& j) {return i->y < j->y;}

template<class It, class IIt> /* IIt = vector<It>::iterator */
double cp_sub(IIt ya, IIt yaend, IIt xa, It &i1, It &i2) {
  typedef typename iterator_traits<It>::value_type P;
  int n = yaend-ya, split = n/2;
  if(n <= 3) { // base case
    double a = (*xa[1]-*xa[0]).dist(), b = 1e50, c = 1e50;
    if(n==3) b=(*xa[2]-*xa[0]).dist(), c=(*xa[2]-*xa[1]).dist()
        ;
    if(a <= b) { i1 = xa[1];
      if(a <= c) return i2 = xa[0], a;
      else return i2 = xa[2], c;
    } else { i1 = xa[2];
      if(b <= c) return i2 = xa[0], b;
      else return i2 = xa[1], c;
  } }
  vector<It> ly, ry, stripy;
  P splitp = *xa[split];
  double splitx = splitp.x;
  for(IIt i = ya; i != yaend; ++i) { // Divide
    if(*i != xa[split] && (**i-splitp).dist2() < 1e-12)
      return i1 = *i, i2 = xa[split], 0;// nasty special case!
    if (**i < splitp) ly.push_back(*i);
    else ry.push_back(*i);
  } // assert((signed)lefty.size() == split)
  It j1, j2; // Conquer
  double a = cp_sub(ly.begin(), ly.end(), xa, i1, i2);
  double b = cp_sub(ry.begin(), ry.end(), xa+split, j1, j2);
  if(b < a) a = b, i1 = j1, i2 = j2;
  double a2 = a*a;
  for(IIt i = ya; i != yaend; ++i) { // Create strip (y-sorted)
    double x = (*i)->x;
    if(x >= splitx-a && x <= splitx+a) stripy.push_back(*i);
  }
  for(IIt i = stripy.begin(); i != stripy.end(); ++i) {
    const P &p1 = **i;
    for(IIt j = i+1; j != stripy.end(); ++j) {
      const P &p2 = **j;
      if(p2.y-p1.y > a) break;
      double d2 = (p2-p1).dist2();
      if(d2 < a2) i1 = *i, i2 = *j, a2 = d2;
  } }
  return sqrt(a2);
}

template<class It> // It is random access iterators of point<T>
double closestpair(It begin, It end, It &i1, It &i2 ) {
  vector<It> xa, ya;
  assert(end-begin >= 2);
  for (It i = begin; i != end; ++i)
    xa.push_back(i), ya.push_back(i);
  sort(xa.begin(), xa.end(), it_less<It>);
  sort(ya.begin(), ya.end(), y_it_less<It>);
  return cp_sub(ya.begin(), ya.end(), xa.begin(), i1, i2);
}
```

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.

32 lines

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

3dHull.h

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

"Point3D.h" — 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: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis. All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

8 lines

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

Strings (9)

AhoCorasick.h

Description: Builds an Ahocorasick Trie, with suffix links
Usage: This is an offline Algorithm, Pass the vector of patterns to Trie.init(), find function returns the number of times each string appears
Time: $\mathcal{O}(n + m + z)$

75 lines

```
const int MaxM = 200005;

struct Trie{
    static const int Alpha = 26;
    static const int first = 'a';
    int lst = 1;
    struct node{
        int nxt[Alpha] = {}, p = -1;
        char c;
        vector<int> end; //if 2 patterns must be different,
             change to int end = -1;
        int SuffixLink;
        int cnt = 0;
  };
  vector<node> V;
  int num;
  stack<int> reversebfs;
```

```
  inline int getval(char c) {
    return c - first;
  }

  void CreateSuffixLink() {
    queue<int> q;
    for(q.push(0); q.size(); q.pop()) {
      int pos = q.front();
      reversebfs.push(pos);
      for(int i = 0; i < Alpha; ++i) {
        if(V[pos].nxt[i]) q.push(V[pos].nxt[i]);
        else if(!pos || !V[pos].p) {
          V[pos].SuffixLink = 0;
          V[pos].nxt[i] = V[0].nxt[i];
        }
        else {
          int val = getval(V[pos].c);
          int j = V[V[pos].p].SuffixLink;
          V[pos].SuffixLink = V[j].nxt[val];
          V[pos].nxt[i] = V[V[pos].SuffixLink].nxt[i];
        }
      }
    }
  }

  void init(vector<string> &v) {
    V.resize(MaxM);
    num = v.size();
    int id = 0;
    for(auto &s : v) {
      int pos = 0;
      for(char &c : s) {
        int val = getval(c);
        if(!V[pos].nxt[val]) {
          V[lst].p = pos; V[lst].c = c; V[pos].nxt[val] = lst
              ++;
        }
        pos = V[pos].nxt[val];
      }
      V[pos].end.emplace_back(id++);
    }
    CreateSuffixLink();
  }

  vector<int> find(string& word) {
    int pos = 0;
    vector<int> ans(num, 0);
    for(auto &c : word) {
      int val = getval(c);
      pos = V[pos].nxt[val];
      V[pos].cnt++; //We count the times we reach each node,
          and then do a reverse propagation
    }
    for(;reversebfs.size();reversebfs.pop()) {
      int x = reversebfs.top(); //When we process x, we know we
           have been there V[x].cnt times;
      for(int i : V[x].end) ans[i] += V[x].cnt;
      if(V[x].SuffixLink != -1) V[V[x].SuffixLink].cnt += V[x].
          cnt;
    }
    return ans;
  }
};
```

PalindromeTree.h

Description: Palindrome Tree for string s
Time: $\mathcal{O}(sz(s))$ for building

64 lines

```
const int maxN = 1000010; // at least sz(s) + 3
struct Node {
  int suffix;
  int len;
  map<char, int> children;

  // not needed for construction, add if needed
  char c;
  int parent;
  vector<int> suffixof;
};

int nodeid;
Node tree[maxN]; // 0: -1 root, 1: empty string
int pos2node[maxN]; // not needed for construction

int add(int parent, char c) {
  if(has(tree[parent].children, c)) {
    return tree[parent].children[c];
  }
  int newid = nodeid++;
  tree[newid].suffix = -1;
  tree[newid].len = tree[parent].len + 2;
  tree[newid].parent = parent;
  tree[newid].c = c;
  tree[parent].children[c] = newid;
  return newid;
}

void build(string& s) {
  nodeid = 2;
  tree[0].parent = -1;
  tree[0].len = -1;
  tree[1].parent = -1;
  tree[0].suffixof.push_back(1);
  int cur = 0;
  FOR(i, 0, s.size()) {
    int newn = -1;
    while(1) {
      int curlen = tree[cur].len;
      if(i-1-curlen >= 0 && s[i-1-curlen] == s[i]) {
        newn = add(cur, s[i]);
        break;
      }
      cur = tree[cur].suffix;
    }
    pos2node[i] = newn;
    if(tree[newn].suffix != -1) {
      cur = newn;
      continue;
    }
    if(cur == 0) {
      tree[newn].suffix = 1;
    } else {
      do {
        cur = tree[cur].suffix;
      } while(i-1-tree[cur].len < 0
          || s[i-1-tree[cur].len] != s[i]);
      tree[newn].suffix = tree[cur].children[s[i]];
    }
    tree[tree[newn].suffix].suffixof.push_back(newn);
    cur = newn;
  }
}
```

Manacher.h

Description: call with String str of length n, returns: r[0..2*n-2],r[i] radius of longest palindrome with center i/2 in str
Time: $\mathcal{O}(n)$

11 lines

```
void manacher(int n, char *str, int *r) {
  r[0] = 0;
  int p = 0;
  FOR(i, 1, 2*n-1) {
    r[i] = (p/2 + r[p] >= (i+1)/2) ? min(r[2*p - i], p/2 + r[p]
        - i/2) : 0;
    while (i/2 + r[i] + 1 < n && (i+1)/2 - r[i] - 1 >= 0
    && str[i/2 + r[i] + 1] == str[(i+1)/2 - r[i] - 1]) r[i]++;
    if (i/2 + r[i] > p/2 + r[p]) p = i;
  }
  // FOR(i,0,2*n-1) r[i] = r[i]*2 + !(i&1); // change radius to
      diameter
}
```

SuffixArray.h

Description: lcp (x,y) = min(lcp(x,x+1),lcp(x+1,x+2)...lcp(y-1,y)) to answer queries with RMQ O(1)
Memory: Build $\mathcal{O}(N \log N)$ (Can be optimised as in B you only use the previous row to compute one row. But then you cannot do lcp
Time: Build: $\mathcal{O}(N \log N)$ where N is the length of the string for creation of the SA. LCP $\mathcal{O}(\log N)$ It is not necessary to use raddixsort if the $\mathcal{O}(nlog^2 n)$ fits the time limit, one can just use stl sort.

60 lines

```
struct SF {
  pair<ll, ll> ord;
  ll id;
  bool operator<(const SF& s) const { return ord < s.ord; }
};

ll lcp(ll x, ll y, vector < vector < ll > > &B, ll N, ll step)
    {
  if (x == y) return N - x;
  ll res = 0;
  for (ll i = step - 1; i >= 0 and x < N and y < N; --i)
    if (B[i][x] == B[i][y]) { x += 1<<i; y += 1<<i; res += 1<<i
        ; }
  return res;
}

void raddixSort(vector < SF > & A, vector < vector <ll > > & B,
     vector < ll > &times, vector < ll > & pos, vector < SF >
    & L2, ll N) {
  ll k = max(N, 256LL);
  for (ll i = 0; i < k + 2; ++i) times[i] = 0;
  for (ll i = 0; i < N; ++i)
    times[A[i].ord.second + 1]++;
  pos[0] = 0;
  for (ll i = 1; i < k + 2; ++i)
    pos[i] = pos[i - 1] + times[i - 1];
  for (ll i = 0; i < N; ++i)
    L2[pos[A[i].ord.second + 1]++] = A[i];

  for (ll i = 0; i < k + 2; ++i)
    times[i] = 0;
  for (ll i = 0; i < N; ++i)
    times[L2[i].ord.first + 1]++;
  pos[0] = 0;
  for (ll i = 1; i < k + 2; ++i)
    pos[i] = pos[i - 1] + times[i - 1];
  for (ll i = 0; i < N; ++i)
    A[pos[L2[i].ord.first + 1]++] = L2[i];
}

void compute_suffix_array(vector < SF > & A, vector<vector<ll>
    > & B, ll N, string & S, ll &step) {
  ll MAXN = 3000005; //millor posar numero gran que algo en
      funcio de N pq peta
  vector < SF > L2(MAXN);
  vector < ll > pos(MAXN + 2,0), times(MAXN + 2,0);
```

```
  A.resize(N); B.resize(1); B[0].resize(N);
  for (ll i = 0; i < N; ++i) B[0][i] = S[i];
  step = 1;
  for (ll b = 0, pw = 1; b < N; ++step, pw<<=1) {
    for (ll i = 0; i < N; ++i) {
      A[i].ord.first = B[step - 1][i];
      A[i].ord.second = i + pw < N ? B[step - 1][i + pw] : -1;
      A[i].id = i;
    }
    raddixSort(A, B, times, pos, L2, N); //sort(A.begin(), A.
        end());
    B.resize(step + 1); B[step].resize(N);
    b = B[step][A[0].id] = 1;
    for (ll i = 1; i < N; ++i) {
      if (A[i - 1] < A[i]) ++b;
      B[step][A[i].id] = b;
    }
  }
}
```

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;
  }
};
```

SuffixAutomaton.h

Time: $\mathcal{O}(26N)$

98 lines

```
const int S = 27;
struct node { int len, link, next[S]; };
struct tree_node { int t0[S], t1[S], next[S], k; };

const int N = 200000;
int sz, last;
node v[N];
tree_node tree[N];
int T[N], Tsz;

inline void init() {
  sz = 1; last = 0; v[0].len = 0; v[0].link = -1;
  memset(v[0].next, -1, sizeof(v[0].next));
}

inline void add(int c) {
  int nlast = sz++;
  v[nlast].len = v[last].len + 1;
  memset(v[nlast].next, -1, sizeof(v[nlast].next));
  int p = last;
  while (p != -1 && v[p].next[c] == -1) {
    v[p].next[c] = nlast; p = v[p].link;
  }
  if (p == -1) {
    v[nlast].link = 0;
  } else {
    int q = v[p].next[c];
    if (v[p].len + 1 == v[q].len) {
      v[nlast].link = q;
    } else {
      int clone = sz++;
      v[clone].len = v[p].len + 1;
      v[clone].link = v[q].link;
      memcpy(v[clone].next, v[q].next,
          sizeof(v[q].next));
      while (p != -1 && v[p].next[c] == q) {
        v[p].next[c] = clone; p = v[p].link;
      }
      v[nlast].link = v[q].link = clone;
    }
  }
  last = nlast;
}

int dfs_tree(int x) {
    int p[S];
    for (int i=0;i<tree[x].k;++i)p[i]=tree[x].next[i];
    memset(tree[x].next, -1, sizeof(tree[x].next));
    int minval = Tsz;
    for (int i = 0; i < tree[x].k; ++i) {
        int t1 = dfs_tree(p[i]) - 1;
        int t0 = t1 - v[p[i]].len + v[x].len + 1;
        minval = min(minval, t0);
        tree[x].next[T[t0]] = p[i];
        tree[x].t0[T[t0]] = t0;
        tree[x].t1[T[t0]] = t1;}
    return minval;}

void suffix_tree(string& a) {
    Tsz = (int)a.size();
    for (int i = 0; i < Tsz; ++i) T[i] = a[i] - 'a';
    T[Tsz++] = S-1;
    init();
    for (int i = Tsz-1; i >= 0; --i) add(T[i]);
    for (int i = 0; i < sz; ++i) tree[i].k = 0;
    for (int i = 1; i < sz; ++i)
      tree[v[i].link].next[tree[v[i].link].k++] = i;
    dfs_tree(0);}

//codi per http://codeforces.com/problemset/problem/123/D
typedef long long ll;
ll res;

int dfs(int x) {
    int conter = 0;
    for (int i = 0; i < S; ++i) {
        if (tree[x].next[i] == -1) continue;
        int next = tree[x].next[i];
        ll quants = dfs(next);
        ll dist = tree[x].t1[i] - tree[x].t0[i] + 1;
        res += quants*dist*(quants + 1)/2;
        conter += quants;
    }
    if (!conter) return 1;
    return conter;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    string a;
    cin >> a;
    suffix_tree(a);
    res = 0;
    dfs(0);
    res -= (ll)a.size() + 1;
    cout << res << '\n';
}
```

Hashing.h

Description: Various self-explanatory methods for string hashing.

45 lines

```
typedef unsigned long long H;
static const H C = 123891739; // arbitrary

// Arithmetic mod 2^64-1. 5x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse).
// "typedef H K;" instead if you think test data is random.
struct K {
  typedef __uint128_t H2;
  H x; K(H x=0) : x(x) {}
  K operator+(K o){ return x + o.x + H(((H2)x + o.x)>>64); }
  K operator*(K o){ return K(x*o.x)+ H(((H2)x * o.x)>>64); }
  H operator-(K o) { K a = *this + ~o.x; return a.x + !~a.x; }
};

struct HashInterval {
  vector<K> ha, pw;
  HashInterval(string& str) : ha(str.size()+1), pw(ha) {
    pw[0] = 1;
    FOR(i,0,str.size())
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
```

```
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};

vector<H> getHashes(string& str, int length) {
  if (str.size() < length) return {};
  K h = 0, pw = 1;
  FOR(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h - 0};
  FOR(i,length,str.size()) {
    ret.push_back(h * C + str[i] - pw * str[i-length]);
    h = ret.back();
  }
  return ret;
}

H hashString(string& s) {
  K h = 0;
  for(auto c:s) h = h * C + c;
  return h - 0;
}
```

Z.h

Description: Computes the prefix function

15 lines

```
vector<int> Z(string &s) {
    int n = s.size();
    int L, R;
    L = R = 0;
    vector<int> Z(n, 0);
    for (int i = 1; i < n; ++i){
        if (i < R) Z[i] = min(Z[i-L], R-i);
        else Z[i] = 0;
        while (Z[i] + i < n and s[Z[i]] == s[i+Z[i]]) ++Z[i];
        if (i+Z[i] > R){
            L = i;
            R = i + Z[i];
        }
    }
}
```

KMP.h

Description: string matching

Time: $\mathcal{O}(P+T)$ where P is the length of the pattern, T is length of the text

14 lines

```
string s, p; cin >> s >> p;
vector<int> pi(p.size() + 1, 0);
int k = 0;
for (int i = 2; i <= p.size(); ++i) {
  while (k > 0 and p[i - 1] != p[k]) k = pi[k];
  if (p[i - 1] == p[k]) ++k;
  pi[i] = k;
}
k = 0;
for (int i = 0; i < s.size(); ++i) {
  while (k > 0 and s[i] != p[k]) k = pi[k];
  if (p[k] == s[i]) ++k;
  if (k == p.size()) k = pi[k]; //Matching
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin()+min_rotation(v), v.end());`

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

8 lines

```
int min_rotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(i,0,N) {
    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;
}
```

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

25 lines

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

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

IntervalCover.h

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

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

19 lines

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

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.

Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`

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

19 lines

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

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to $<=$, and reverse the loop at (B). To minimize f, change it to $>$, also at (B).

Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`

Time: $\mathcal{O}(\log(b-a))$

13 lines

```
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) // (A)
      a = mid;
    else
      b = mid+1;
  }
  rep(i,a+1,b+1) 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: $\mathcal{O}\left(N^{1.6}\right)$

1 lines

Josephus.h

Description: if you have n people in a circle and the first person to die is 0, and each time afterwards you kill the person k positions later, outputs the number of the last person to die

3 lines

```
cin >> n >> k; v[1]=0;
for (int i=2;i<=n;++i) v[i]=(v[i-1]+k)%i;
cout<< v[n]+1 << endl;
```

10.3 Dynamic programming

DivideAndConquerDP.h

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

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

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 $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

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

1 lines

10.4 Java

java.java

10 lines

```
BufferedReader reader = new BufferedReader(new
    InputStreamReader(System.in));
String[] tokens = reader.readLine().split("\\s+");
int zahl = Integer.parseInt(tokens[0]);
BigInteger big = new BigInteger(tokens[1]);
import java.awt.geom.*; // lines,points,segments,area,contains,
    intersect,ccw...
import java.util.*; // TreeSet, HashMap, HashSet, Date, Stack,
    ...
import java.util.Arrays / Collections; // sort,binarySearch,
    fill,min,max,reverse
import java.math.*; // BigInteger (mit gcd, modInverse,...),
    BigDecimal
import static java.lang.Math.*;//min,max,abs,cos,cosh,atan2,
    round,random,sqrt,..
import java.io.*; //BufferedReader,..
```

10.5 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`).

10.6 Optimization tricks

10.6.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `x && !(x & (x - 1))` true, if `x` is power of 2.
- `gray_code[x] = x ^ (x >> 1)`
- `checkerboard[y][x] = (x & 1) ^ (y & 1)`
- `ffs(int x), ffs(ll x)` number of the least significant bit, `ffs(1 << i) = i+1`
- `__builtin_clz(uint x), ...__clzll(ull)` number of leading zeros, for x ¿ 0
- `__builtin_ctz(uint x), ...__ctzll(ull)` number of trailing zeros, for x ¿ 0
- `__builtin_popcount(uint x), ...__popcountll(ull)` number of 1 bits
- `#define ld_ll(X) (63-__builtin_clzll(ll(X)))` floor(log2(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.

10.6.2 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).

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
```