



Northern University of Business and Technology Khulna

NUBTK \$ (^w^) \$

Sourav Mondal, Rahul Kumar Ghosh, Ayon Adikary Arko

NWU CSE FEST 2025

November 11, 2025

**1 Contest****2 Mathematics****3 Data Structures****4 Number Theory****5 Combinatorial****6 Numerical****7 Graphs****8 Geometry****9 Strings****10 Various**

## Contest (1)

## template.cpp

13 lines

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

#ifndef LOCAL
#include "debug.h"
#else
#define dbg(...)
#endif

int32_t main() {
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
}
```

## debug.h

16 lines

```
template <typename A, typename B>
ostream &operator<<(ostream &os, const pair<A, B> &p) {
    return os << '(' << p.first << ", " << p.second << ')';
}

template <typename C, typename T = typename enable_if<!is_same<
    →C, string>::value, typename C::value_type>::type>
ostream &operator<<(ostream &os, const C &v) {
    os << '{';
    string sep;
    for (const T &x : v) os << sep << x, sep = ", ";
    return os << '}';
}

void dbg_out() { cerr << "\n"; }

template<typename Head, typename... Tail> void dbg_out(Head H,
    →Tail... T) {
    cerr << " | " << H; dbg_out(T...);
}

#define dbg(...) cerr << "[" << #__VA_ARGS__ << "] =", dbg_out(
    →__VA_ARGS__)
```

**template debug gen .bashrc hash stress troubleshoot**

<pre>1 gen.h 2 3 .bashrc 4 5 hash.sh 6 7 stress.sh 8 9 10</pre>		<p>2 lines</p> <p>5 lines</p> <p>3 lines</p> <p>20 lines</p> <p>75 lines</p>	<ul style="list-style-type: none"> <li>* Any uninitialized variables?</li> <li>* Any undefined behavior (array out of bounds)?</li> <li>* Any overflows or NaNs (or shifting 1l by &gt;=64 bits)?</li> <li>* Confusing N and M, i and j, etc.?</li> <li>* Confusing ++i and i++?</li> <li>* Return vs continue vs break?</li> <li>* Are you sure the STL functions you use work as you think?</li> <li>* Add some assertions, maybe resubmit.</li> <li>* Create some test cases to run your algorithm on.</li> <li>* Go through the algorithm for a simple case.</li> <li>* Go through this list again.</li> <li>* Explain your algorithm to a teammate.</li> <li>* Ask the teammate to look at your code.</li> <li>* Go for a small walk, e.g. to the toilet.</li> <li>* Rewrite your solution from the start or let a teammate do it.</li> </ul> <p>Geometry:</p> <ul style="list-style-type: none"> <li>* Work with ints if possible.</li> <li>* Correctly account for numbers close to (but not) zero.</li> <li>    →Related:</li> <li>for functions like acos make sure absolute val of input is not (slightly) greater than one.</li> <li>* Correctly deal with vertices that are collinear, concyclic, coplanar (in 3D), etc.</li> <li>* Subtracting a point from every other (but not itself)?</li> </ul> <p>Runtime error:</p> <ul style="list-style-type: none"> <li>* Have you tested all corner cases locally?</li> <li>* Any uninitialized variables?</li> <li>* Are you reading or writing outside the range of any vector?</li> <li>* Any assertions that might fail?</li> <li>* Any possible division by 0? (mod 0 for example)</li> <li>* Any possible infinite recursion?</li> <li>* Invalidated pointers or iterators?</li> <li>* Are you using too much memory?</li> <li>* Debug with resubmits (e.g. remapped signals, see Various).</li> </ul> <p>Time limit exceeded:</p> <ul style="list-style-type: none"> <li>* Do you have any possible infinite loops?</li> <li>* What's your complexity? Large TL does not mean that something simple (like NlogN) isn't intended.</li> <li>* Are you copying a lot of unnecessary data? (References)</li> <li>* Avoid vector, map. (use arrays/unordered_map)</li> <li>* How big is the input and output? (consider FastIO)</li> <li>* What do your teammates think about your algorithm?</li> <li>* Calling count() on multiset?</li> </ul> <p>Memory limit exceeded:</p> <ul style="list-style-type: none"> <li>* What is the max amount of memory your algorithm should need?</li> <li>* Are you clearing all data structures between test cases?</li> <li>* If using pointers try BumpAllocator.</li> </ul>
---	--	--	---

## 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  $x = -b/2a$ .

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

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

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

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

## 2.3 Trigonometry

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

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

$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$

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

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

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

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

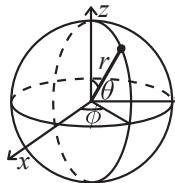
### 2.4.2 Quadrilaterals

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

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

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

### 2.4.3 Spherical coordinates



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

## 2.5 Derivatives/Integrals

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

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

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

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

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

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

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

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

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

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

## 2.7 Series

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

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

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

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

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

## 2.8 Probability theory

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

Expectation is linear:

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

For independent  $X$  and  $Y$ ,

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

### 2.8.1 Discrete distributions

#### Binomial distribution

The number of successes in  $n$  independent yes/no experiments, each which yields success with probability  $p$  is

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

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

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

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

## First success distribution

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

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

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

## Poisson distribution

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

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

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

## 2.8.2 Continuous distributions

### Uniform distribution

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

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

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

### Exponential distribution

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

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

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

### Normal distribution

Most real random values with mean  $\mu$  and variance  $\sigma^2$  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  $X_1, X_2, \dots$  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.

### Stationary distribution

$\pi$  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  $i$ .  $\pi_j/\pi_i$  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,  $\pi_i$  is proportional to node  $i$ 's degree.

### Ergodicity

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).  $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$ .

## Data Structures (3)

### 3.1 STL

#### MapComparator.h

Description: example of function object (functor) for map or set

Usage: `set<int, cmp> s; map<int, int, cmp> m;` 5bfa6c, 1 lines

struct cmp{bool operator()(int l,int r) const{return l>r;}}

#### HashMap.h

Description: Hash map with similar API as `unordered_map`. Initial capacity must be a power of 2 if provided.

Usage: `gp.hash_table<int, int, chash> h({},{},{},{},{1<<16});`

Memory: ~1.5x `unordered_map`

Time: ~3x faster than `unordered_map`

<[ext/pb.ds/assoc\\_container.hpp](#)> b6e07b, 6 lines

```
using namespace __gnu_pbds;
struct chash {
    const uint64_t C = 4e18*acos(0)+71;
    const int RANDOM = chrono::steady_clock::now().
        __time_since_epoch().count();
    int64_t operator()(int64_t x) const { return
        __builtin_bswap64((x^RANDOM)*C); }
};
```

#### OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the  $n$ 'th element, and finding the index of an element. Change `null_type` to get a map.

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

<[ext/pb.ds/assoc\\_container.hpp](#)> 257533, 12 lines

```
using namespace __gnu_pbds;
template<class T> using Tree = tree<T, null_type, less<T>,
```

```
rb_tree_tag, tree_order_statistics_node_update>;
#define ook order_of_key // position of the given element
#define fbo find_by_order // return an iterator

int atMost(Tree<pair<int, int>>& T, int r) {
    return T.order_of_key({r,MOD});
}
int getSum(Tree<pair<int, int>>& T, int l, int r) {
    return atMost(T,r)-atMost(T,l-1);
}
```

## 3.2 1D Range Queries

#### SparseTable.h

Description: Static 1D range (min/max/gcd/lcm/or/and) query. If TL is an issue, use arrays instead of vectors and store values instead of indices.

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

Time:  $\mathcal{O}(1)$

471d4b, 19 lines

```
template<class T> struct SparseTable {
    int N, LOG;
    vector<vector<T>> jmp;
    T cmb(T a, T b) { return min(a, b); }
    SparseTable(const vector<T>& v) {
        N = v.size(); LOG = __lg(N);
        jmp.resize(N, vector<T>(LOG + 1));
        for (int i = 0; i < N; ++i) jmp[i][0] = v[i];
        for (int j = 1; j <= LOG; ++j) {
            for (int i = 0; i + (1<<j) <= N; ++i) {
                jmp[i][j] = cmb(jmp[i][j-1], jmp[i+(1<<(j-1))][j-1]);
            }
        }
    }
    T query(int l, int r) {
        int d = __lg(r-l+1);
        return cmb(jmp[l][d], jmp[r-(1<<d)+1][d]);
    }
};
```

#### FenwickTree.h

Description: Computes partial sums  $a[0] + a[1] + \dots + a[\text{pos} - 1]$ , and updates single elements  $a[i]$ , taking the difference between the old and new value. 0-Indexed.

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

122df3, 23 lines

```
template<class T> struct FenwickTree {
    int N; vector<T> bit;
    FenwickTree(int _N) { N = _N; bit.resize(N); }
    void add(int p, T x) {
        for (++p; p <= N; p += p&-p) bit[p-1] += x;
    }
    T sum(int l, int r) { return sum(r+1)-sum(l); }
    T sum(int r) {
        T res = 0;
        for(; r; r -= r&-r) res += bit[r-1];
        return res;
    }
    int lower_bound(T sum) {
        if (sum <= 0) return -1;
        int pos = 0;
        for (int pw = 1<<25; pw; pw >>= 1) {
            int npow = pos+pw;
            if (npow <= N && bit[npow-1] < sum)
                pos = npow, sum -= bit[npow-1];
        }
        return pos;
    }
};
```

## SegmentTree.h

**Description:** 1D point update and range query where  $\text{cmb}$  is any associative operation.  $\text{seg}[1] == \text{query}(0, N-1)$ .

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

749f3e, 29 lines

```
template<class T> struct SegTree {
    const T ID{};
    T cmb(T a, T b) { return a + b; }
    int n; vector<T> seg;
    SegTree(int _n) {
        for (int i = 1; i < _n; ) n *= 2;
        seg.assign(2 * n, ID);
    }
    void pull(int p) { seg[p] = cmb(seg[2 * p], seg[2 * p + 1]); }
    void update(int p, T val) { // set val at position p
        seg[p += n] = val;
        for (p /= 2; p; p /= 2) pull(p);
    }
    T query(int l, int r) { // zero-indexed, inclusive
        T ra = ID, rb = ID;
        for (l += n, r += n + 1; l < r; l /= 2, r /= 2) {
            if (l & 1) ra = cmb(ra, seg[l++]);
            if (r & 1) rb = cmb(seg[--r], rb);
        }
        return cmb(ra, rb);
    }
    // int first_at_least(int lo, int val, int ind, int l, int r)
    //    ↪ { // if seg stores max across range
    //        if (r < lo || val > seg[ind]) return -1;
    //        if (l == r) return l;
    //        int m = (l + r) / 2;
    //        int res = first_at_least(lo, val, 2 * ind, l, m); if (res != -1) return res;
    //        return first_at_least(lo, val, 2 * ind + 1, m + 1, r);
    //    }
};
```

## LazySegmentTree.h

**Description:** 1D range increment and sum query.

**Usage:** LazySeg<int64\_t, 1<<20> T; T.update(l, r, val);

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

e0742f, 42 lines

```
template<class T, int SZ> struct LazySeg {
    // SZ must be power of 2
    static_assert(__builtin_popcount(SZ) == 1);
    const T ID{};
    T cmb(T a, T b) { return a + b; }
    T seg[2 * SZ], lazy[2 * SZ];
    LazySeg() {
        for (int i = 0; i < 2 * SZ; ++i) seg[i] = lazy[i] = ID;
    }
    // modify values for current node
    void push(int ind, int L, int R) {
        seg[ind] += (R - L + 1) * lazy[ind]; // dependent on operation
        if (L != R) for (int i = 0; i < 2; ++i) lazy[2 * ind + i] += lazy[ind];
        lazy[ind] = 0;
    }
    void pull(int ind) {
        seg[ind] = cmb(seg[2 * ind], seg[2 * ind + 1]);
    }
    void build() {
        for (int i = SZ - 1; i >= 1; --i) pull(i);
    }
    void update(int lo, int hi, T inc, int ind = 1, int L = 0,
               int R = SZ - 1) {
        push(ind, L, R);
        if (hi < L || R < lo) return;
        if (lo <= L & R <= hi) {
            lazy[ind] = inc;
        }
```

```
        push(ind, L, R);
        return;
    }
    int M = (L + R) / 2;
    update(lo, hi, inc, 2 * ind, L, M);
    update(lo, hi, inc, 2 * ind + 1, M + 1, R);
    pull(ind);
}
T query(int lo, int hi, int ind = 1, int L = 0, int R = SZ - 1) {
    push(ind, L, R);
    if (lo > R || L > hi) return ID;
    if (lo <= L & R <= hi) return seg[ind];
    int M = (L + R) / 2;
    return cmb(query(lo, hi, 2 * ind, L, M), query(lo, hi, 2 * ind + 1, M + 1, R));
}
```

## 3.3 2D Range Queries

## PrefixSum2D.h

**Description:** calculates rectangle sums in constant time

65b070, 17 lines

```
template<typename T> struct PrefixSum2D {
    vector<vector<T>> sum;
    PrefixSum2D(const vector<vector<T>> &v) {
        int n = v.size(), m = v[0].size();
        sum.assign(n + 1, vector<T>(m + 1, T{}));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                sum[i + 1][j + 1] = v[i][j]
                    - sum[i][j] + sum[i + 1][j] + sum[i][j + 1];
            }
        }
    }
    T query(int x1, int y1, int x2, int y2) {
        return sum[x2][y2] + sum[x1 - 1][y1 - 1]
            - sum[x1 - 1][y2] - sum[x2][y1 - 1];
    }
};
```

## Number Theory (4)

## 4.1 Modular Arithmetic

Modular arithmetic operations are defined as follows:

Addition:  $(a + b) \bmod m$

Subtraction:  $((a - b) \bmod m + m) \bmod m$  (ensures non-negative result)

Multiplication:  $(a \cdot b) \bmod m$

Division:  $a/b \bmod m$  is defined as  $a \cdot b^{-1} \bmod m$  if  $\gcd(b, m) = 1$ , otherwise it is impossible.

Modular inverse:  $b \cdot b^{-1} \equiv 1 \pmod{m}$

## ModIntShort.h

**Description:** Modular arithmetic. Assumes  $MOD$  is prime.

**Usage:** mint a = MOD+5; inv(a); // 400000003

7c1e94, 30 lines

```
struct mint {
    int v;
    static const int MOD = 1E9 + 7;
    explicit operator int() const { return v; }
    mint() : v(0) {}
    mint(int64_t _v) : v(_v % MOD) { v += (v < 0) * MOD; }
    mint &operator+=(mint o) {
```

```
        if ((v += o.v) >= MOD) v -= MOD;
        return *this;
    }
    mint &operator-=(mint o) {
        if ((v -= o.v) < 0) v += MOD;
        return *this;
    }
    mint &operator*=(mint o) {
        v = int((int64_t)v * o.v % MOD);
        return *this;
    }
    friend mint pow(mint a, int64_t p) {
        assert(p >= 0);
        return p == 0 ? 1 : pow(a * a, p / 2) * (p & 1 ? a : 1);
    }
    friend mint inv(mint a) {
        assert(a.v != 0);
        return pow(a, MOD - 2);
    }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
};
```

## ModMulLL.h

**Description:** Multiply two 64-bit integers mod another if 128-bit is not available.  $\text{modMul}$  is equivalent to  $(\text{ul}) \cdot \text{int128}(a) * \text{b \% mod}$ . Works for  $0 \leq a, b < \text{mod} < 2^{63}$ . NOTE : Only works if system supports 80-bit floating point.

d22eee, 10 lines

```
using ul = uint64_t;
ul modMul(ul a, ul b, const ul mod) {
    return __int128(a) * __int128(b) % mod;
}
ul modPow(ul a, ul b, const ul mod) {
    if (b == 0) return 1;
    ul res = modPow(a, b / 2, mod);
    res = modMul(res, res, mod);
    return b & 1 ? modMul(res, a, mod) : res;
}
```

## FastMod.h

**Description:** Barrett reduction computes  $a \% b$  about 4 times faster than usual where  $b > 1$  is constant but not known at compile time. Division by  $b$  is replaced by multiplication by  $m$  and shifting right 64 bits.

ce43ec, 9 lines

```
using ul = uint64_t;
struct FastMod {
    ul b, m;
    FastMod(ul b) : b(b), m(-1ULL / b) {}
    ul reduce(ul a) {
        ul q = (ul)((__uint128_t(m) * a) >> 64), r = a - q * b;
        return r - (r >= b) * b;
    }
};
```

## 4.2 Primality

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

## 4.2.2 Divisors

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

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  $n < 1e19$ .

**Dirichlet Convolution:** Given a function  $f(x)$ , let

$$(f * g)(x) = \sum_{d|x} g(d)f(x/d).$$

If the partial sums  $s_{f*g}(n), s_g(n)$  can be computed in  $O(1)$  and  $s_f(1 \dots n^{2/3})$  can be computed in  $O(n^{2/3})$  then all  $s_f(\frac{n}{d})$  can as well. Use

$$s_{f*g}(n) = \sum_{d=1}^n g(d)s_f(n/d).$$

If  $f(x) = \mu(x)$  then  $g(x) = 1, (f * g)(x) = (x == 1)$ , and  $s_f(n) = 1 - \sum_{i=2}^n s_f(n/i)$ .

If  $f(x) = \phi(x)$  then  $g(x) = 1, (f * g)(x) = x$ , and  $s_f(n) = \frac{n(n+1)}{2} - \sum_{i=2}^n s_f(n/i)$ .

### Sieve.h

**Description:** Tests primality up to  $SZ$ . Runs faster if only odd indices are stored.

**Time:**  $\mathcal{O}(SZ \log \log SZ)$  or  $\mathcal{O}(SZ)$

935b91, 22 lines

```
template<int SZ> struct Sieve {
    bitset<SZ> is_prime; vector<int> primes;
    Sieve() {
        is_prime.set(); is_prime[0] = is_prime[1] = 0;
        for (int i = 4; i < SZ; i += 2) is_prime[i] = 0;
        for (int i = 3; i*i < SZ; i += 2) if (is_prime[i])
            for (int j = i*i; j < SZ; j += i*2) is_prime[j] = 0;
        for (int i = 0; i < SZ; i++) {
            if (is_prime[i]) primes.push_back(i);
        }
    }
    // array<int, SZ> spf{} // smallest prime that divides
    // Sieve() { // above is faster
    //     for (int i = 2; i < SZ; i++) {
    //         if (spf[i] == 0) spf[i] = i, primes.push_back(i);
    //         for (int p: primes) {
    //             if (p > spf[i] || i*p >= SZ) break;
    //             spf[i*p] = p;
    //         }
    //     }
    // }
};
```

## 4.3 Euclidean Algorithm

### 4.3.1 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

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

## Sieve Euclid

### Euclid.h

**Description:** Generalized Euclidean algorithm. `euclid` and `invGeneral` work for  $A, B < 2^{62}$ .

**Time:**  $\mathcal{O}(\log AB)$

8bab06, 13 lines

```
// For A,B>=0, finds (x,y) s.t.
pair<int64_t, int64_t> euclid(int64_t A, int64_t B) {
    // Ax+By=gcd(A,B), |Ax|, |By|<=AB/gcd(A,B)
    if (!B) return {1, 0};
    pair<int64_t, int64_t> p = euclid(B, A%B);
    return {p.second, p.first - A/B*p.second};
}
// find x in [0,B) such that Ax=1 mod B
int64_t invGeneral(int64_t A, int64_t B) {
    pair<int64_t, int64_t> p = euclid(A, B);
    assert(p.first*A + p.second*B == 1);
    return p.first + (p.first<0)*B;
} // must have gcd(A,B)=1
```

## 4.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

## 4.5 Lifting the Exponent

For  $n > 0, p$  prime, and ints  $x, y$  s.t.  $p \nmid x, y$  and  $p|x-y$ :

- $p \neq 2$  or  $p = 2, 4|x-y \implies v_p(x^n - y^n) = v_p(x-y) + v_p(n)$ .
- $p = 2, 2|n \implies v_2(x^n - y^n) = v_2((x^2)^{n/2} - (y^2)^{n/2})$ .

## Combinatorial (5)

### 5.1 Permutations

#### 5.1.1 Factorial

$n$	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
$n$	11	12	13	14	15	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		

#### 5.1.2 Cycles

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

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

#### 5.1.3 Derangements

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

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

### 5.1.4 Burnside's lemma

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

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

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

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

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

## 5.2 Partitions and subsets

### 5.2.1 Partition function

Number of ways of writing  $n$  as a sum of positive integers, disregarding the order of the summands.

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

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

$n$	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	$\sim 2e5$	$\sim 2e8$

### 5.2.2 Lucas' Theorem

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

### 5.3 General purpose numbers

#### 5.3.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t - 1}$  (FFT-able).

$$B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

#### 5.3.2 Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

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

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

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

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

### 5.3.3 Eulerian numbers

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

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

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

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

### 5.3.4 Stirling numbers of the second kind

Partitions of  $n$  distinct elements into exactly  $k$  groups.

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

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

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

### 5.3.5 Bell numbers

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

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

### 5.3.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$

# on  $k$  existing trees of size  $n_i$ :  $n_1 n_2 \cdots n_k n^{k-2}$

# with degrees  $d_i$ :  $(n - 2)! / ((d_1 - 1)! \cdots (d_n - 1)!)$

### 5.3.7 Catalan numbers

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

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

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

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

## BigInt

### 5.4 Young Tableaux

Let a **Young diagram** have shape  $\lambda = (\lambda_1 \geq \dots \geq \lambda_k)$ , where  $\lambda_i$  equals the number of cells in the  $i$ -th (left-justified) row from the top. A **Young tableau** of shape  $\lambda$  is a filling of the  $n = \sum \lambda_i$  cells with a permutation of  $1 \dots n$  such that each row and column is increasing.

**Hook-Length Formula:** For the cell in position  $(i, j)$ , let  $h_\lambda(i, j) = |\{(I, J) | i \leq I, j \leq J, (I = i \text{ or } J = j)\}|$ . The number of Young tableaux of shape  $\lambda$  is equal to  $f^\lambda = \frac{n!}{\prod h_\lambda(i, j)}$ .

**Schensted's Algorithm:** converts a permutation  $\sigma$  of length  $n$  into a pair of Young Tableaux  $(S(\sigma), T(\sigma))$  of the same shape. When inserting  $x = \sigma_i$ ,

1. Add  $x$  to the first row of  $S$  by inserting  $x$  in place of the largest  $y$  with  $x < y$ . If  $y$  doesn't exist, push  $x$  to the end of the row, set the value of  $T$  at that position to be  $i$ , and stop.
2. Add  $y$  to the second row using the same rule, keep repeating as necessary.

All pairs  $(S(\sigma), T(\sigma))$  of the same shape correspond to a unique  $\sigma$ , so  $n! = \sum (f^\lambda)^2$ . Also,  $S(\sigma^R) = S(\sigma)^T$ .

Let  $d_k(\sigma), a_k(\sigma)$  be the lengths of the longest subseqs which are a union of  $k$  decreasing/ascending subseqs, respectively. Then  $a_k(\sigma) = \sum_{i=1}^k \lambda_i, d_k(\sigma) = \sum_{i=1}^k \lambda_i^*$ , where  $\lambda_i^*$  is size of the  $i$ -th column.

## Numerical (6)

### 6.1 Arithmetic

BigInt.h

Description: Big Integer

1cbd2c, 289 lines

```
// base and base_digits must be consistent
const int base = 1e9, base_digits = 9;
struct bigint { // value == 0 is represented by empty z
    vi z; // digits
    int sign; // sign == 1 <==> value >= 0
    bigint() : sign(1) {} // sign == -1 <==> value < 0
    bigint(ll v) { *this = v; }
    bigint &operator=(ll v) {
        sign = v < 0 ? -1 : 1; v *= sign; // make v positive
        z.clear(); for (; v; v/=base) z.pb(v%base);
        return *this;
    }
    bigint(const str &s) { read(s); } // add char by char
    bigint &operator+=(const bigint &other) {
        //dbg("ADDING", *this, other, sign, other.sign);
        if (sign == other.sign) {
            for (int i = 0, carry = 0; i < sz(other.z) || carry; ++i)
                {
                    if (i == sz(z)) z.pb(0);
                    z[i] += carry+(i<sz(other.z)?other.z[i]:0);
                    carry = z[i] >= base; if (carry) z[i] -= base;
                }
        }
        else {
            for (int i = 0, carry = 0; i < sz(other.z) || carry; ++i)
                {
                    if (i == sz(z)) z.pb(0);
                    z[i] -= carry+(i<sz(other.z)?other.z[i]:0);
                    carry = z[i] < 0; if (carry) z[i] += base;
                }
        }
    }
    bigint &operator*=(int v) {
        if (v < 0) sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < sz(z) || carry; ++i)
            if (i == sz(z)) z.pb(0);
            ll cur = (ll)z[i]*v+carry;
            carry = cur/base; z[i] = cur%base;
        }
        trim(); return *this;
    }
    bigint &operator*=(const bigint &b) {
        if (sign == -b.sign) sign = 1;
        for (int i = 0, carry = 0; i < sz(z) || carry; ++i)
            if (i == sz(z)) z.pb(0);
            ll cur = (ll)z[i]*b.z[i]+carry;
            carry = cur/base; z[i] = cur%base;
        }
        trim(); return *this;
    }
    bigint &operator/=(const bigint &b) {
        if (sign == b.sign) sign = 1;
        for (int i = 0, carry = 0; i < sz(z) || carry; ++i)
            if (i == sz(z)) z.pb(0);
            ll cur = (ll)z[i]/b.z[i];
            carry = cur%base; z[i] = cur/base;
        }
        trim(); return *this;
    }
    bigint &operator%=(const bigint &b) {
        if (sign == b.sign) sign = 1;
        for (int i = 0, carry = 0; i < sz(z) || carry; ++i)
            if (i == sz(z)) z.pb(0);
            ll cur = (ll)z[i]-b.z[i];
            carry = cur%base; z[i] = cur/base;
        }
        trim(); return *this;
    }
    bigint &operator/=(int v) {
        if (v < 0) sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < sz(z) || carry; ++i)
            if (i == sz(z)) z.pb(0);
            ll cur = (ll)z[i]/v;
            carry = cur%base; z[i] = cur/base;
        }
        trim(); return *this;
    }
    bigint &operator%=(int v) {
        if (v < 0) sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < sz(z) || carry; ++i)
            if (i == sz(z)) z.pb(0);
            ll cur = (ll)z[i]-v;
            carry = cur%base; z[i] = cur/base;
        }
        trim(); return *this;
    }
    bigint &operator/=(const str &s) {
        if (s == "0") sign = 0;
        else if (s == "-0") sign = -1;
        else if (s[0] == '+') sign = 1;
        else if (s[0] == '-') sign = -1;
        else if (s[0] == '0') sign = 1;
        else {
            int neg = 1;
            for (int i = 1; i < sz(s); i++)
                if (s[i] < '0' || s[i] > '9') {
                    if (s[i] == '-') neg = -1;
                    break;
                }
            sign = neg;
        }
        for (int i = 0; i < sz(s); i++) {
            if (s[i] < '0' || s[i] > '9') {
                if (s[i] == '-' || s[i] == '+') continue;
                if (s[i] == '.') continue;
                if (i == 0) sign = -1;
                else if (i == 1) sign = 1;
                else if (i == 2) sign = 1;
                else if (i == 3) sign = 1;
                else if (i == 4) sign = 1;
                else if (i == 5) sign = 1;
                else if (i == 6) sign = 1;
                else if (i == 7) sign = 1;
                else if (i == 8) sign = 1;
                else if (i == 9) sign = 1;
                else if (i == 10) sign = 1;
                else if (i == 11) sign = 1;
                else if (i == 12) sign = 1;
                else if (i == 13) sign = 1;
                else if (i == 14) sign = 1;
                else if (i == 15) sign = 1;
                else if (i == 16) sign = 1;
                else if (i == 17) sign = 1;
                else if (i == 18) sign = 1;
                else if (i == 19) sign = 1;
                else if (i == 20) sign = 1;
                else if (i == 21) sign = 1;
                else if (i == 22) sign = 1;
                else if (i == 23) sign = 1;
                else if (i == 24) sign = 1;
                else if (i == 25) sign = 1;
                else if (i == 26) sign = 1;
                else if (i == 27) sign = 1;
                else if (i == 28) sign = 1;
                else if (i == 29) sign = 1;
                else if (i == 30) sign = 1;
                else if (i == 31) sign = 1;
                else if (i == 32) sign = 1;
                else if (i == 33) sign = 1;
                else if (i == 34) sign = 1;
                else if (i == 35) sign = 1;
                else if (i == 36) sign = 1;
                else if (i == 37) sign = 1;
                else if (i == 38) sign = 1;
                else if (i == 39) sign = 1;
                else if (i == 40) sign = 1;
                else if (i == 41) sign = 1;
                else if (i == 42) sign = 1;
                else if (i == 43) sign = 1;
                else if (i == 44) sign = 1;
                else if (i == 45) sign = 1;
                else if (i == 46) sign = 1;
                else if (i == 47) sign = 1;
                else if (i == 48) sign = 1;
                else if (i == 49) sign = 1;
                else if (i == 50) sign = 1;
                else if (i == 51) sign = 1;
                else if (i == 52) sign = 1;
                else if (i == 53) sign = 1;
                else if (i == 54) sign = 1;
                else if (i == 55) sign = 1;
                else if (i == 56) sign = 1;
                else if (i == 57) sign = 1;
                else if (i == 58) sign = 1;
                else if (i == 59) sign = 1;
                else if (i == 60) sign = 1;
                else if (i == 61) sign = 1;
                else if (i == 62) sign = 1;
                else if (i == 63) sign = 1;
                else if (i == 64) sign = 1;
                else if (i == 65) sign = 1;
                else if (i == 66) sign = 1;
                else if (i == 67) sign = 1;
                else if (i == 68) sign = 1;
                else if (i == 69) sign = 1;
                else if (i == 70) sign = 1;
                else if (i == 71) sign = 1;
                else if (i == 72) sign = 1;
                else if (i == 73) sign = 1;
                else if (i == 74) sign = 1;
                else if (i == 75) sign = 1;
                else if (i == 76) sign = 1;
                else if (i == 77) sign = 1;
                else if (i == 78) sign = 1;
                else if (i == 79) sign = 1;
                else if (i == 80) sign = 1;
                else if (i == 81) sign = 1;
                else if (i == 82) sign = 1;
                else if (i == 83) sign = 1;
                else if (i == 84) sign = 1;
                else if (i == 85) sign = 1;
                else if (i == 86) sign = 1;
                else if (i == 87) sign = 1;
                else if (i == 88) sign = 1;
                else if (i == 89) sign = 1;
                else if (i == 90) sign = 1;
                else if (i == 91) sign = 1;
                else if (i == 92) sign = 1;
                else if (i == 93) sign = 1;
                else if (i == 94) sign = 1;
                else if (i == 95) sign = 1;
                else if (i == 96) sign = 1;
                else if (i == 97) sign = 1;
                else if (i == 98) sign = 1;
                else if (i == 99) sign = 1;
                else if (i == 100) sign = 1;
                else if (i == 101) sign = 1;
                else if (i == 102) sign = 1;
                else if (i == 103) sign = 1;
                else if (i == 104) sign = 1;
                else if (i == 105) sign = 1;
                else if (i == 106) sign = 1;
                else if (i == 107) sign = 1;
                else if (i == 108) sign = 1;
                else if (i == 109) sign = 1;
                else if (i == 110) sign = 1;
                else if (i == 111) sign = 1;
                else if (i == 112) sign = 1;
                else if (i == 113) sign = 1;
                else if (i == 114) sign = 1;
                else if (i == 115) sign = 1;
                else if (i == 116) sign = 1;
                else if (i == 117) sign = 1;
                else if (i == 118) sign = 1;
                else if (i == 119) sign = 1;
                else if (i == 120) sign = 1;
                else if (i == 121) sign = 1;
                else if (i == 122) sign = 1;
                else if (i == 123) sign = 1;
                else if (i == 124) sign = 1;
                else if (i == 125) sign = 1;
                else if (i == 126) sign = 1;
                else if (i == 127) sign = 1;
                else if (i == 128) sign = 1;
                else if (i == 129) sign = 1;
                else if (i == 130) sign = 1;
                else if (i == 131) sign = 1;
                else if (i == 132) sign = 1;
                else if (i == 133) sign = 1;
                else if (i == 134) sign = 1;
                else if (i == 135) sign = 1;
                else if (i == 136) sign = 1;
                else if (i == 137) sign = 1;
                else if (i == 138) sign = 1;
                else if (i == 139) sign = 1;
                else if (i == 140) sign = 1;
                else if (i == 141) sign = 1;
                else if (i == 142) sign = 1;
                else if (i == 143) sign = 1;
                else if (i == 144) sign = 1;
                else if (i == 145) sign = 1;
                else if (i == 146) sign = 1;
                else if (i == 147) sign = 1;
                else if (i == 148) sign = 1;
                else if (i == 149) sign = 1;
                else if (i == 150) sign = 1;
                else if (i == 151) sign = 1;
                else if (i == 152) sign = 1;
                else if (i == 153) sign = 1;
                else if (i == 154) sign = 1;
                else if (i == 155) sign = 1;
                else if (i == 156) sign = 1;
                else if (i == 157) sign = 1;
                else if (i == 158) sign = 1;
                else if (i == 159) sign = 1;
                else if (i == 160) sign = 1;
                else if (i == 161) sign = 1;
                else if (i == 162) sign = 1;
                else if (i == 163) sign = 1;
                else if (i == 164) sign = 1;
                else if (i == 165) sign = 1;
                else if (i == 166) sign = 1;
                else if (i == 167) sign = 1;
                else if (i == 168) sign = 1;
                else if (i == 169) sign = 1;
                else if (i == 170) sign = 1;
                else if (i == 171) sign = 1;
                else if (i == 172) sign = 1;
                else if (i == 173) sign = 1;
                else if (i == 174) sign = 1;
                else if (i == 175) sign = 1;
                else if (i == 176) sign = 1;
                else if (i == 177) sign = 1;
                else if (i == 178) sign = 1;
                else if (i == 179) sign = 1;
                else if (i == 180) sign = 1;
                else if (i == 181) sign = 1;
                else if (i == 182) sign = 1;
                else if (i == 183) sign = 1;
                else if (i == 184) sign = 1;
                else if (i == 185) sign = 1;
                else if (i == 186) sign = 1;
                else if (i == 187) sign = 1;
                else if (i == 188) sign = 1;
                else if (i == 189) sign = 1;
                else if (i == 190) sign = 1;
                else if (i == 191) sign = 1;
                else if (i == 192) sign = 1;
                else if (i == 193) sign = 1;
                else if (i == 194) sign = 1;
                else if (i == 195) sign = 1;
                else if (i == 196) sign = 1;
                else if (i == 197) sign = 1;
                else if (i == 198) sign = 1;
                else if (i == 199) sign = 1;
                else if (i == 200) sign = 1;
                else if (i == 201) sign = 1;
                else if (i == 202) sign = 1;
                else if (i == 203) sign = 1;
                else if (i == 204) sign = 1;
                else if (i == 205) sign = 1;
                else if (i == 206) sign = 1;
                else if (i == 207) sign = 1;
                else if (i == 208) sign = 1;
                else if (i == 209) sign = 1;
                else if (i == 210) sign = 1;
                else if (i == 211) sign = 1;
                else if (i == 212) sign = 1;
                else if (i == 213) sign = 1;
                else if (i == 214) sign = 1;
                else if (i == 215) sign = 1;
                else if (i == 216) sign = 1;
                else if (i == 217) sign = 1;
                else if (i == 218) sign = 1;
                else if (i == 219) sign = 1;
                else if (i == 220) sign = 1;
                else if (i == 221) sign = 1;
                else if (i == 222) sign = 1;
                else if (i == 223) sign = 1;
                else if (i == 224) sign = 1;
                else if (i == 225) sign = 1;
                else if (i == 226) sign = 1;
                else if (i == 227) sign = 1;
                else if (i == 228) sign = 1;
                else if (i == 229) sign = 1;
                else if (i == 230) sign = 1;
                else if (i == 231) sign = 1;
                else if (i == 232) sign = 1;
                else if (i == 233) sign = 1;
                else if (i == 234) sign = 1;
                else if (i == 235) sign = 1;
                else if (i == 236) sign = 1;
                else if (i == 237) sign = 1;
                else if (i == 238) sign = 1;
                else if (i == 239) sign = 1;
                else if (i == 240) sign = 1;
                else if (i == 241) sign = 1;
                else if (i == 242) sign = 1;
                else if (i == 243) sign = 1;
                else if (i == 244) sign = 1;
                else if (i == 245) sign = 1;
                else if (i == 246) sign = 1;
                else if (i == 247) sign = 1;
                else if (i == 248) sign = 1;
                else if (i == 249) sign = 1;
                else if (i == 250) sign = 1;
                else if (i == 251) sign = 1;
                else if (i == 252) sign = 1;
                else if (i == 253) sign = 1;
                else if (i == 254) sign = 1;
                else if (i == 255) sign = 1;
                else if (i == 256) sign = 1;
                else if (i == 257) sign = 1;
                else if (i == 258) sign = 1;
                else if (i == 259) sign = 1;
                else if (i == 260) sign = 1;
                else if (i == 261) sign = 1;
                else if (i == 262) sign = 1;
                else if (i == 263) sign = 1;
                else if (i == 264) sign = 1;
                else if (i == 265) sign = 1;
                else if (i == 266) sign = 1;
                else if (i == 267) sign = 1;
                else if (i == 268) sign = 1;
                else if (i == 269) sign = 1;
                else if (i == 270) sign = 1;
                else if (i == 271) sign = 1;
                else if (i == 272) sign = 1;
                else if (i == 273) sign = 1;
                else if (i == 274) sign = 1;
                else if (i == 275) sign = 1;
                else if (i == 276) sign = 1;
                else if (i == 277) sign = 1;
                else if (i == 278) sign = 1;
                else if (i == 279) sign = 1;
                else if (i == 280) sign = 1;
                else if (i == 281) sign = 1;
                else if (i == 282) sign = 1;
                else if (i == 283) sign = 1;
                else if (i == 284) sign = 1;
                else if (i == 285) sign = 1;
                else if (i == 286) sign = 1;
                else if (i == 287) sign = 1;
                else if (i == 288) sign = 1;
                else if (i == 289) sign = 1;
                else if (i == 290) sign = 1;
                else if (i == 291) sign = 1;
                else if (i == 292) sign = 1;
                else if (i == 293) sign = 1;
                else if (i == 294) sign = 1;
                else if (i == 295) sign = 1;
                else if (i == 296) sign = 1;
                else if (i == 297) sign = 1;
                else if (i == 298) sign = 1;
                else if (i == 299) sign = 1;
                else if (i == 300) sign = 1;
                else if (i == 301) sign = 1;
                else if (i == 302) sign = 1;
                else if (i == 303) sign = 1;
                else if (i == 304) sign = 1;
                else if (i == 305) sign = 1;
                else if (i == 306) sign = 1;
                else if (i == 307) sign = 1;
                else if (i == 308) sign = 1;
                else if (i == 309) sign = 1;
                else if (i == 310) sign = 1;
                else if (i == 311) sign = 1;
                else if (i == 312) sign = 1;
                else if (i == 313) sign = 1;
                else if (i == 314) sign = 1;
                else if (i == 315) sign = 1;
                else if (i == 316) sign = 1;
                else if (i == 317) sign = 1;
                else if (i == 318) sign = 1;
                else if (i == 319) sign = 1;
                else if (i == 320) sign = 1;
                else if (i == 321) sign = 1;
                else if (i == 322) sign = 1;
                else if (i == 323) sign = 1;
                else if (i == 324) sign = 1;
                else if (i == 325) sign = 1;
                else if (i == 326) sign = 1;
                else if (i == 327) sign = 1;
                else if (i == 328) sign = 1;
                else if (i == 329) sign = 1;
                else if (i == 330) sign = 1;
                else if (i == 331) sign = 1;
                else if (i == 332) sign = 1;
                else if (i == 333) sign = 1;
                else if (i == 334) sign = 1;
                else if (i == 335) sign = 1;
                else if (i == 336) sign = 1;
                else if (i == 337) sign = 1;
                else if (i == 338) sign = 1;
                else if (i == 339) sign = 1;
                else if (i == 340) sign = 1;
                else if (i == 341) sign = 1;
                else if (i == 342) sign = 1;
                else if (i == 343) sign = 1;
                else if (i == 344) sign = 1;
                else if (i == 345) sign = 1;
                else if (i == 346) sign = 1;
                else if (i == 347) sign = 1;
                else if (i == 348) sign = 1;
                else if (i == 349) sign = 1;
                else if (i == 350) sign = 1;
                else if (i == 351) sign = 1;
                else if (i == 352) sign = 1;
                else if (i == 353) sign = 1;
                else if (i == 354) sign = 1;
                else if (i == 355) sign = 1;
                else if (i == 356) sign = 1;
                else if (i == 357) sign = 1;
                else if (i == 358) sign = 1;
                else if (i == 359) sign = 1;
                else if (i == 360) sign = 1;
                else if (i == 361) sign = 1;
                else if (i == 362) sign = 1;
                else if (i == 363) sign = 1;
                else if (i == 364) sign = 1;
                else if (i == 365) sign = 1;
                else if (i == 366) sign = 1;
                else if (i == 367) sign = 1;
                else if (i == 368) sign = 1;
                else if (i == 369) sign = 1;
                else if (i == 370) sign = 1;
                else if (i == 371) sign = 1;
                else if (i == 372) sign = 1;
                else if (i == 373) sign = 1;
                else if (i == 374) sign = 1;
                else if (i == 375) sign = 1;
                else if (i == 376) sign = 1;
                else if (i == 377) sign = 1;
                else if (i == 378) sign = 1;
                else if (i == 379) sign = 1;
                else if (i == 380) sign = 1;
                else if (i == 381) sign = 1;
                else if (i == 382) sign = 1;
                else if (i == 383) sign = 1;
                else if (i == 384) sign = 1;
                else if (i == 385) sign = 1;
                else if (i == 386) sign = 1;
                else if (i == 387) sign = 1;
                else if (i == 388) sign = 1;
                else if (i == 389) sign = 1;
                else if (i == 390) sign = 1;
                else if (i == 391) sign = 1;
                else if (i == 392) sign = 1;
                else if (i == 393) sign = 1;
                else if (i == 394) sign = 1;
                else if (i == 395) sign = 1;
                else if (i == 396) sign = 1;
                else if (i == 397) sign = 1;
                else if (i == 398) sign = 1;
                else if (i == 399) sign = 1;
                else if (i == 400) sign = 1;
                else if (i == 401) sign = 1;
                else if (i == 402) sign = 1;
                else if (i == 403) sign = 1;
                else if (i == 404) sign = 1;
                else if (i == 405) sign = 1;
                else if (i == 406) sign = 1;
                else if (i == 407) sign = 1;
                else if (i == 408) sign = 1;
                else if (i == 409) sign = 1;
                else if (i == 410) sign = 1;
                else if (i == 411) sign = 1;
                else if (i == 412) sign = 1;
                else if (i == 413) sign = 1;
                else if (i == 414) sign = 1;
                else if (i == 415) sign = 1;
                else if (i == 416) sign = 1;
                else if (i == 417) sign = 1;
                else if (i == 418) sign = 1;
                else if (i == 419) sign = 1;
                else if (i == 420) sign = 1;
                else if (i == 421) sign = 1;
                else if (i == 422) sign = 1;
                else if (i == 423) sign = 1;
                else if (i == 424) sign = 1;
                else if (i == 425) sign = 1;
                else if (i == 426) sign = 1;
                else if (i == 427) sign = 1;
                else if (i == 428) sign = 1;
                else if (i == 429) sign = 1;
                else if (i == 430) sign = 1;
                else if (i == 431) sign = 1;
                else if (i == 432) sign = 1;
                else if (i == 433) sign = 1;
                else if (i == 434) sign = 1;
                else if (i == 435) sign = 1;
                else if (i == 436) sign = 1;
                else if (i == 437) sign = 1;
                else if (i == 438) sign = 1;
                else if (i == 439) sign = 1;
                else if (i == 440) sign = 1;
                else if (i == 441) sign = 1;
                else if (i == 442) sign = 1;
                else if (i == 443) sign = 1;
                else if (i == 444) sign = 1;
                else if (i == 445) sign = 1;
                else if (i == 446) sign = 1;
                else if (i
```

```

res *= base; res += q; // add a bit to sqrt
if (j > 0) {
    int d1 = sz(res.z)+2 < sz(r.z) ? r.z[sz(res.z)+2] : 0;
     $\rightarrow$ // always 0/1?
    int d2 = sz(res.z)+1 < sz(r.z) ? r.z[sz(res.z)+1] : 0;
    int d3 = sz(res.z) < sz(r.z) ? r.z[sz(res.z)] : 0;
    q = ((ll) d1*base*base+(ll)d2*base+d3)/(firstDigit*2);
}
res.trim(); return res/norm;

bigint operator/(const bigint &v) const { return divmod(*this
 $\rightarrow$ , v.f); }
bigint operator%(const bigint &v) const { return divmod(*this
 $\rightarrow$ , v.s); }
bigint &operator/=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = sz(z)-1, rem = 0; i >= 0; --i) {
        ll cur = z[i]+rem*(ll)base;
        z[i] = cur/v; rem = cur%v;
    }
    trim(); return *this;
}
bigint operator/(int v) const { return bigint(*this) /= v; }
int operator%(int v) const {
    if (v < 0) v = -v;
    int m = 0; R0F(i,sz(z)) m = (z[i]+m*(ll)base)%v;
    return m*sign;
}
bigint &operator*=(const bigint &v) { return *this = *this*v;
 $\rightarrow$  }
bigint &operator/=(const bigint &v) { return *this = *this/v;
 $\rightarrow$  }

bool operator<(const bigint &v) const {
    if (sign != v.sign) return sign < v.sign;
    if (sz(z) != sz(v.z)) return sz(z)*sign < sz(v.z) * v.sign;
    R0F(i,sz(z)) if (z[i] != v.z[i]) return z[i]*sign < v.z[i]*sign;
     $\rightarrow$ sign;
    return 0; // equal
}
bool operator>(const bigint &v) const { return v < *this; }
bool operator<=(const bigint &v) const { return !(v < *this);
 $\rightarrow$  }
bool operator>=(const bigint &v) const { return !(*this < v);
 $\rightarrow$  }
bool operator==(const bigint &v) const { return !(*this < v)
 $\rightarrow \&& !(v < *this); }
bool operator!=(const bigint &v) const { return *this < v ||
 $\rightarrow v < *this; }
void trim() {
    while (sz(z) && z.bk == 0) z.pop_back();
    if (!sz(z)) sign = 1; // don't output -0
}
bool isZero() const { return !sz(z); }
friend bigint operator-(bigint v) {
    if (sz(v.z)) v.sign = -v.sign;
    return v;
}
bigint abs() const { return sign == 1 ? *this : -*this; }
ll longValue() const {
    ll res = 0; R0F(i,sz(z)) res = res*base+z[i];
    return res*sign;
}
friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b); } // euclidean algo
friend bigint lcm(const bigint &a, const bigint &b) {
    return a/gcd(a, b) * b;
}

void read(const str &s) {
    sign = 1; z.clear(); int pos = 0;
    while (pos < sz(s) && (s[pos] == '-' || s[pos] == '+')) {$$ 
```

```

        if (s[pos] == '-') sign = -sign;
         $\rightarrow$ pos; } // account for sign
for (int i = sz(s)-1; i >= pos; i -= base_digits) {
    int x = 0;
    for (int j = max(pos, i-base_digits+1); j <= i; j++)
        x = x*10+s[j]-'0';
    z.pb(x);
}
trim();
friend istream &operator>>(istream &is, bigint &v) {
    str s; is >> s; v.read(s); return is;
}
friend ostream &operator<<(ostream &os, const bigint &v) {
    if (v.sign == -1) os << '-';
    os << (!sz(v.z) ? 0 : v.z.bk);
    R0F(i,sz(v.z)-1) os << setfill(base_digits) << setfill('0') <<
        v.z[i];
    return os; // pad with zeroes
}
static vi convert_base(const vi &a, int old_digits, int
 $\rightarrow$ new_digits) {
    vl p(max(old_digits, new_digits) + 1); // blocks of  $10^{old}$ 
 $\rightarrow$ p[0] ->  $10^{new}$ 
    p[0] = 1; FOR(i,1,sz(p)) p[i] = p[i-1]*10;
    vi res; ll cur = 0; int cur_digits = 0;
    for (int v:a) {
        cur += v*p[cur_digits]; cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.pb(cur%p[new_digits]);
            cur /= p[new_digits]; cur_digits -= new_digits;
        }
    }
    res.pb(cur); while (sz(res) && res.bk == 0) res.pop_back();
    return res;
}
static vl karatMul(const vl &a, const vl &b) { // karatsuba
    int n = sz(a); vl res(2*n);
    if (n <= 32) { // naive multiply
        FOR(i,n) FOR(j,n) res[i+j] += a[i]*b[j];
        return res;
    }
    int k = n/2;
    vl a1(begin(a),begin(a)+k), a2(k+all(a));
    vl b1(begin(b),begin(b)+k), b2(k+all(b));
    vl a1b1 = karatMul(a1, b1), a2b2 = karatMul(a2, b2);
    FOR(i,k) a2[i] += a1[i], b2[i] += b1[i];
    vl r = karatMul(a2, b2); // three instead of four products
    FOR(i,sz(a1b1)) r[i] -= a1b1[i];
    FOR(i,sz(a2b2)) r[i] -= a2b2[i];
    FOR(i,sz(r)) res[i+k] += r[i];
    FOR(i,sz(a1b1)) res[i] += a1b1[i];
    FOR(i,sz(a2b2)) res[i+n] += a2b2[i];
    return res;
}
bigint operator*(const bigint &v) const {
    if (min(sz(z),sz(v.z)) < 150) return mul_simple(v);
    bigint res; res.sign = sign*v.sign; // should work as long
 $\rightarrow$ as # of digits isn't too large (> LLONG_MAX/10^12)
    vi a6 = convert_base(*this>z, base_digits, 6); // blocks of
 $\rightarrow$ 10^6 instead of 10^9
    vi b6 = convert_base(v.z, base_digits, 6);
    vl a(all(a6)), b(all(b6));
    while (sz(a) < sz(b)) a.pb(0);
    while (sz(b) < sz(a)) b.pb(0);
    while (sz(a)&(sz(a)-1)) a.pb(0), b.pb(0); // make size
 $\rightarrow$ power of 2
    vl c = karatMul(a, b);
    ll cur = 0; FOR(i,sz(c)) { // process carries
        cur += c[i]; res.z.pb(cur%1000000); cur /= 1000000;
    }
    res.z = convert_base(res.z,6,base_digits);
}
res.trim(); return res;
bigint mul_simple(const bigint &v) const {
    bigint res; res.sign = sign*v.sign;
    res.z.rsz(sz(z)+sz(v.z));
    FOR(i,sz(z)) if (z[i]) {
        ll cur = 0; for (int j = 0; j < sz(v.z) || cur; ++j) {
            cur += res.z[i+j]+(ll)z[i]*(j<sz(v.z)?v.z[j]:0);
            res.z[i+j] = cur%base; cur /= base;
        }
    }
    res.trim(); return res;
}
friend str ts(const bigint& v) {
    stringstream ss; ss << v;
    str s; ss >> s; return s;
}

bigint random_bigint(int n) {
    str s; FOR(i,n) s += rand() % 10 + '0';
    return bigint(s);

// random tests
void bigintTest() {
    bigint x = bigint("120");
    bigint y = bigint("5");
    cout << x / y << endl;
    FOR(i,1000) {
        int n = rand() % 100 + 1;
        bigint a = random_bigint(n), res = sqrt(a);
        bigint xx = res * res, yy = (res + 1) * (res + 1);
        if (xx > a || yy <= a) {
            dbg("SQRT FAILED",i);
            dbg(a,res);
            break;
        }
        int m = rand() % n + 1;
        bigint b = random_bigint(m)+1;
        res = a/b; xx = res*b, yy = b*(res+1);
        if (xx > a || yy <= a) {
            dbg("DIVISION FAILED",i);
            dbg(a,b,res);
            break;
        }
    }
    bigint a = random_bigint(10000);
    bigint b = random_bigint(2000);
    auto t1 = chrono::high_resolution_clock::now();
    bigint c = a / b;
    auto t2 = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> duration = t2 - t1;
    dbg(duration.count(),"ms");
}
bigint a = random_bigint(500000);
bigint b = random_bigint(500000);
bigint c1, c2;
{
    auto t1 = chrono::high_resolution_clock::now();
    c1 = a * b;
    auto t2 = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> duration = t2 - t1;
    dbg(duration.count(),"ms");
}
auto t1 = chrono::high_resolution_clock::now();
c2 = a.mul_simple(b);
auto t2 = chrono::high_resolution_clock::now();
chrono::duration<double, milli> duration = t2 - t1;

```

```

    dbg(duration.count(), "ms");
}dbg(c1 == c2);
{
    auto t1 = chrono::high_resolution_clock::now();
FOR(i,1000000) {
    a = random_bignum(30);
    b = random_bignum(30);
    c2 = a.mul_simple(b);
}
auto t2 = chrono::high_resolution_clock::now();
chrono::duration<double, milli> duration = t2 - t1;
dbg(duration.count(), "ms");
}

```

## 6.2 Matrix

Matrix.h

**Description:** 2D matrix operations.

"ModInt.h" b18e29, 21 lines

```

using T = mi;
using Mat = V<V<T>>; // use array instead if tight TL

Mat makeMat(int r, int c) { return Mat(r, V<T>(c)); }
Mat makeId(int n) {
    Mat m = makeMat(n, n); FOR(i, n) m[i][i] = 1;
    return m;
}

Mat operator*(const Mat& a, const Mat& b) {
    int x = sz(a), y = sz(a[0]), z = sz(b[0]);
    assert(y == sz(b)); Mat c = makeMat(x, z);
    FOR(i, x) FOR(j, y) FOR(k, z) c[i][k] += a[i][j]*b[j][k];
    return c;
}
Mat& operator*=(Mat& a, const Mat& b) { return a = a*b; }
Mat pow(Mat m, ll p) {
    int n = sz(m); assert(n == sz(m[0]) && p >= 0);
    Mat res = makeId(n);
    for (; p; p /= 2, m *= m) if (p&1) res *= m;
    return res;
}

```

## Graphs (7)

**Erdos-Gallai:**  $d_1 \geq \dots \geq d_n$  can be degree sequence of simple graph on  $n$  vertices iff their sum is even and

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

## 7.1 Basics

Dijkstra.h

**Description:** shortest path

Time:  $\mathcal{O}(N \log N + M)$  5afc43, 27 lines

```

template<class T, bool directed> struct Dijkstra {
    int SZ;
    vector<T> dist;
    vector<vector<pair<int, T>>> adj;
    Dijkstra(int _SZ) {
        SZ = _SZ;
        adj.resize(SZ);
    }
    void ae(int u, int v, T w) {
        adj[u].push_back({v, w});
        if (!directed) adj[v].push_back({u, w});
    }
}

```

## Matrix Dijkstra TopoSort TreeDiameter LCAjump

```

void gen(int st) {
    dist.assign(SZ, numeric_limits<T>::max());
    priority_queue<pair<T, int>> PQ;
    auto relax = [&](int v, T d) {
        if (dist[v] <= d) return;
        dist[v] = d;
        PQ.push({-dist[v], v});
    };
    relax(st, 0);
    while (!PQ.empty()) {
        auto [d, u] = PQ.top(); PQ.pop(); d = -d;
        if (dist[u] < d) continue;
        for (auto& [v, w]: adj[u]) relax(v, w+d);
    }
}

```

TopoSort.h

**Description:** sorts vertices such that if there exists an edge  $x \rightarrow y$ , then  $x$  goes before  $y$

6779f6, 23 lines

```

struct TopoSort {
    int SZ;
    vector<int> in, res;
    vector<vector<int>> adj;
    TopoSort(int _SZ) {
        SZ = _SZ;
        in.resize(SZ);
        adj.resize(SZ);
    }
    void ae(int u, int v) {
        adj[u].push_back(v), ++in[v];
    }
    bool sort() {
        queue<int> todo;
        for (int u = 0; u < SZ; ++u) if (!in[u]) todo.push(u);
        while (!todo.empty()) {
            int u = todo.front(); todo.pop();
            res.push_back(u);
            for (auto &v : adj[u]) if (!--in[v]) todo.push(v);
        }
        return (int)res.size() == SZ;
    }
};

```

## 7.2 Trees

TreeDiameter.h

**Description:** Calculates longest path in tree. The vertex furthest from 0 must be an endpoint of the diameter.

3c22a4, 38 lines

```

struct TreeDiameter {
    int N, diaLen;
    vector<int> dist, dia;
    vector<vector<int>> adj;
    TreeDiameter(int _N) {
        N = _N;
        dia = {0, 0};
        adj.resize(N);
        dist.resize(N);
    }
    void ae(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void genDist(int R) { dist[R] = 0; dfs(R); }
    void gen() {
        genDist(0);
        for (int i = 0; i < N; ++i) {
            if (dist[i] > dist[dia[0]]) dia[0] = i;
        }
    }
}

```

```

genDist(dia[0]);
for (int i = 0; i < N; ++i) {
    if (dist[i] > dist[dia[1]]) dia[1] = i;
}
diaLen = dist[dia[1]];
// center vertex of dia[0] -> dia[1]
// int cen = dia[1];
// for (int i = 0; i < diaLen/2; ++i) cen = par[cen];
// center = {cen};
// if (diaLen&1) center.push_back(par[cen]);
}
void dfs(int u, int p = -1) {
    for (auto& v: adj[u]) {
        if (v == p) continue;
        dist[v] = dist[u] + 1; dfs(v, u);
    }
}
};

LCAjump.h
```

**Description:** Calculates least common ancestor in tree with verts  $0 \dots N-1$  and root  $R$  using binary jumping.

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

**Time:**  $\mathcal{O}(N \log N)$  build,  $\mathcal{O}(\log N)$  query

5c0112, 47 lines

```

struct LCA {
    int N, LOG;
    vector<int> depth;
    vector<vector<int>> par, adj;
    LCA(int _N) {
        N = _N; LOG = 1;
        while ((1 << LOG) < N) ++LOG;
        adj.resize(N);
        depth.resize(N);
        par.resize(N, vector<int>(LOG));
    }
    void ae(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void gen(int R = 0) { par[R][0] = R; dfs(R); }
    void dfs(int u = 0) {
        for (int j = 1; j < LOG; ++j) {
            par[u][j] = par[par[u][j-1]][j-1];
        }
        for (auto& v: adj[u]) {
            if (v == par[u][0]) continue;
            depth[v] = depth[u]+1;
            par[v][0] = u;
            dfs(v);
        }
    }
    int jump(int x, int d) {
        for (int j = 0; j < LOG; ++j) {
            if ((d>>j) & 1) x = par[x][j];
        }
        return x;
    }
    int lca(int u, int v) {
        if (depth[u] > depth[v]) swap(u, v);
        v = jump(v, depth[v]-depth[u]);
        if (u == v) return u;
        for (int j = LOG-1; j >= 0; --j) {
            int U = par[u][j], V = par[v][j];
            if (U != V) u = U, v = V;
        }
        return par[u][0];
    }
    int dist(int u, int v) {

```

```

        return depth[u] + depth[v] - 2*depth[lca(u, v)];
    }
);

```

## 7.3 DSU

### DSU.h

**Description:** Disjoint Set Union with path compression and union by size. Add edges and test connectivity. Use for Kruskal's or Boruvka's minimum spanning tree.

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

a876ce, 12 lines

```

struct DSU {
    vector<int> e;
    DSU(int N) { e = vector<int>(N, -1); }
    int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }
    bool sameSet(int a, int b) { return get(a) == get(b); }
    int size(int x) { return -e[get(x)]; }
    bool unite(int x, int y) { // union by size
        x = get(x), y = get(y); if (x == y) return false;
        if (e[x] > e[y]) swap(x, y);
        e[x] += e[y]; e[y] = x; return true;
    }
};

```

### DSUrb.h

**Description:** Disjoint Set Union with Rollback

d21534, 25 lines

```

struct DSUrb {
    vector<int> e;
    vector<array<int,4>> mod;
    DSUrb(int n) { e = vector<int>(n, -1); }
    int get(int x) { return e[x] < 0 ? x : get(e[x]); }
    bool sameSet(int a, int b) { return get(a) == get(b); }
    int size(int x) { return -e[get(x)]; }
    bool unite(int x, int y) { // union-by-rank
        x = get(x), y = get(y);
        if (x == y) {
            mod.push_back({-1,-1,-1,-1});
            return 0;
        }
        if (e[x] > e[y]) swap(x,y);
        mod.push_back({x,y,e[x],e[y]});
        e[x] += e[y]; e[y] = x; return 1;
    }
    void rollback() {
        auto a = mod.back(); mod.pop_back();
        if (a[0] != -1) {
            e[a[0]] = a[2];
            e[a[1]] = a[3];
        }
    }
};

```

### 7.3.1 SqrtDecompton

HLD generally suffices. If not, here are some common strategies:

- Rebuild the tree after every  $\sqrt{N}$  queries.
- Consider vertices with  $>$  or  $<$   $\sqrt{N}$  degree separately.
- For subtree updates, note that there are  $O(\sqrt{N})$  distinct sizes among child subtrees of any node.

**Block Tree:** Use a DFS to split edges into contiguous groups of size  $\sqrt{N}$  to  $2\sqrt{N}$ .

## DSU DSUrb DFSBasic Point ConvexHull KMP

**Mo's Algorithm for Tree Paths:** Maintain an array of vertices where each one appears twice, once when a DFS enters the vertex (st) and one when the DFS exists (en). For a tree path  $u \leftrightarrow v$  such that  $st[u] < st[v]$ ,

- If  $u$  is an ancestor of  $v$ , query  $[st[u], st[v]]$ .
- Otherwise, query  $[en[u], st[v]]$  and consider  $LCA(u, v)$  separately.

Solutions with worse complexities can be faster if you optimize the operations that are performed most frequently. Use arrays instead of vectors whenever possible. Iterating over an array in order is faster than iterating through the same array in some other order (ex. one given by a random permutation) or DFSing on a tree of the same size. Also, the difference between  $\sqrt{N}$  and the optimal block (or buffer) size can be quite large. Try up to 5x smaller or larger (at least).

## 7.4 DFS Algorithms

### DFSBasic.h

**Description:** Basic DFS implementation for undirected graphs

**Time:**  $\mathcal{O}(N + M)$

14b944, 21 lines

```

struct DFS {
    int N;
    vector<bool> visited;
    vector<vector<int>> adj;
    DFS(int _N) {
        N = _N;
        adj.resize(N);
        visited.resize(N);
    }
    void ae(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void dfs(int u) {
        visited[u] = true;
        for (auto& v: adj[u]) {
            if (visited[v]) continue;
            dfs(v);
        }
    }
};

```

## Geometry (8)

### 8.1 Geometric primitives

#### Point.h

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

47ec0a, 28 lines

```

template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template <class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
}

```

```

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

```

## 8.2 Polygons

### ConvexHull.h

**Description:**

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

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



310954, 13 lines

```

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

```

## Strings (9)

### 9.1 Light

#### KMP.h

**Description:**  $f[i]$  is length of the longest proper suffix of the  $i$ -th prefix of  $s$  that is a prefix of  $s$

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

70d0fa, 14 lines

```

vector<int> kmp(str s) {
    int N = sz(s); vi f(N+1); f[0] = -1;
    FOR(i, 1, N+1) {
        for (f[i]=f[i-1]; f[i]!=-1&&s[f[i]]!=s[i-1]; f[i]=f[f[i]]; f[i]++;
    }
    return f;
}
vi getOc(str a, str b) { // find occurrences of a in b
    vi f = kmp(a+"@"+b), ret;
    FOR(i, sz(a), sz(b)+1) if (f[i+sz(a)+1] == sz(a))
        ret.pb(i-sz(a));
    return ret;
}

```

Z.h  
**Description:**  $f[i]$  is the max len such that  $s.substr(0, len) == s.substr(i, len)$   
**Time:**  $\mathcal{O}(N)$

---

```
566170, 15 lines
vi z(str s) {
    int N = sz(s), L = 1, R = 0; s += '#';
    vi ans(N); ans[0] = N;
    FOR(i, 1, N) {
        if (i <= R) ans[i] = min(R-i+1, ans[i-L]);
        while (s[i+ans[i]] == s[ans[i]]) ++ans[i];
        if (i+ans[i]-1 > R) L = i, R = i+ans[i]-1;
    }
    return ans;
}
vi getPrefix(str a, str b) { // find prefixes of a in b
    vi t = z(a+b); t = vi(sz(a)+all(t));
    each(u, t) ckmin(u, sz(a));
    return t;
}
```

## Various (10)

### 10.1 Binary search

fstTrue.h

**Description:** Finds the first value  $x$  in  $[lo, hi]$  such that predicate  $f(x)$  is true.

**Usage:** `int ans = fstTrue()`  
**Time:**  $\mathcal{O}(\log(hi - lo))$

2654fa, 9 lines

```
template<typename T, typename U> T fstTrue(T lo, T hi, U f) {
    ++hi;
    assert(lo <= hi);
    while (lo < hi) {
        T mid = lo + (hi - lo) / 2;
        f(mid) ? hi = mid : lo = mid + 1;
    }
    return lo;
};
```

lstTrue.h

**Description:** Finds the last value  $x$  in  $[lo, hi]$  such that predicate  $f(x)$  is true.

**Usage:** `int ans = lstTrue()`  
**Time:**  $\mathcal{O}(\log(hi - lo))$

8aff85, 9 lines

```
template<typename T, typename U> T lstTrue(T lo, T hi, U f) {
    --lo;
    assert(lo <= hi);
    while (lo < hi) {
        T mid = lo + (hi - lo + 1) / 2;
        f(mid) ? lo = mid : hi = mid - 1;
    }
    return lo;
};
```

realTrue.h

**Description:** Binary search on continuous (floating-point) domain.

**Usage:** `int ans = realTrue()`

**Time:**  $\mathcal{O}(\log(precision))$  — write 100 iterations and have a relax

69fe1d, 7 lines

```
template<typename T, typename U> T realTrue(T lo, T hi, U f) {
    for (int i = 0; i < 100; ++i) {
        T mid = 0.5 * (hi + lo);
        f(mid) ? lo = mid : hi = mid;
    }
    return lo;
};
```

## Z fstTrue lstTrue realTrue FastInput Python3

### 10.2 Dynamic programming

When doing DP on intervals:

$a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$ , where the (minimal) optimal  $k$  increases with both  $i$  and  $j$ ,

- one can solve intervals in increasing order of length, and search  $k = p[i][j]$  for  $a[i][j]$  only between  $p[i][j-1]$  and  $p[i+1][j]$ .
- This is known as Knuth DP. Sufficient criteria for this are if  $f(b, c) \leq f(a, d)$  and  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a \leq b \leq c \leq d$ .
- Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

### 10.3 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).

### 10.4 Optimization tricks

#### 10.4.1 Bit hacks

- $x \& -x$  is the least bit in  $x$ .
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of  $m$  (except  $m$  itself).
- $c = x \& -x$ ,  $r = x + c$ ;  $((r^x) >> 2) / c$  |  $r$  is the next number after  $x$  with the same number of bits set.
- `FOR(b, k) FOR(i, 1<<K) if (i&1<<b) D[i] += D[i^(1<<b)];` computes all sums of subsets.

#### 10.4.2 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. Also consider older `#pragma GCC target ("sse4")`.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

FastInput.h

**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.

**Usage:** `./a.out < input.txt`

**Time:** About 5x as fast as `cin/scanf`.

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

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

### 10.5 Other languages

Python3.py

**Description:** Python review.

40 lines

```
from math import *
import sys, random
def nextInt():
    return int(input())
def nextStrs():
    return input().split()
def nextInts():
    return list(map(int, nextStrs()))

n = nextInt()
v = [n]
def process(x):
    global v
    x = abs(x)
    V = []
    for t in v:
        g = gcd(t, x)
        if g != 1:
            V.append(g)
        if g != t:
            V.append(t//g)
    v = V
for i in range(50):
    x = random.randint(0, n-1)
    if gcd(x, n) != 1:
        process(x)
    else:
        sx = x*x%n # assert(gcd(sx, n) == 1)
        print(f"sqrt({sx})")
        sys.stdout.flush()
        X = nextInt()
        process(x+X)
        process(x-X)
print(f'{len(v)}', end='')
for i in v:
    print(f'{i}', end='')
print()
sys.stdout.flush() # sys.exit(0) -> exit
# sys.setrecursionlimit(int(1e9)) -> stack size
# print(f'{ans:.6f}') -> print ans to 6 decimal places
```

7b3c70, 17 lines