



Northern University of Bussiness and Technology Khulna

NUBTK $\$(\hat{w})\$$

Sourav Mondal, Rahul Kumar Ghosh, Ayon Adikary Arko

BUET IUPC 2026

February 5, 2026

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	f321ac, 13 lines
#include <bits/stdc++.h> using namespace std; #ifdef LOCAL #include "debug.h" #else #define dbg(...) #endif int32_t main() { cin.tie(0)->sync_with_stdio(0); cin.exceptions(cin.failbit); }	
DebugShort.h	16 lines
template <typename A, typename B> ostream &operator<<(ostream &os, const pair<A, B> &p) { return os << '(' << p.first << ", " << p.second << ')'; } // pair 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 << '}'; } // ds void dbg_out() { cerr << " \n"; } template<typename H, typename... T> void dbg_out(H h, T... t) { cerr << " "<< h; dbg_out(t...); } // basic #define dbg(...) cerr << __LINE__<< ":[" << #__VA_ARGS__ << "] ⇨=", dbg_out(__VA_ARGS__)	

keybindings.json	23 lines
[{ "key": "alt+b", // build "command": "workbench.action.terminal.sendSequence", "args": { "text": "./compile.sh \${fileBasenameNoExtension}\n" } }, { "key": "alt+d", // file input/output "command": "workbench.action.terminal.sendSequence", "args": { "text": "./compile.sh \${fileBasenameNoExtension} && ./\${ ⇨fileBasenameNoExtension} < input.txt > output.txt\n" } }, { "key": "alt+c", // manual input "command": "workbench.action.terminal.sendSequence", "args": { "text": "./compile.sh \${fileBasenameNoExtension} && ./\${ ⇨fileBasenameNoExtension}\n" } }]	
gen.h	2 lines
#define uid(l, r) uniform_int_distribution<int>(l, r)(rng) mt19937 rng(chrono::steady_clock::now().time_since_epoch()). ⇨count();	
compile.sh	1 lines
g++ -std=c++23 -O2 -Wall -Wextra -Wshadow -Wconversion - ⇨fsanitize=address -fsanitize=undefined -fsanitize=signed- ⇨integer-overflow -D_GLIBCXX_DEBUG -o \$1 \$1.cpp;	
hash.sh	3 lines
# Hash file ignoring whitespace and comments. Verifies that # code was correctly typed. Usage: 'sh hash.sh < A.cpp' cpp -dD -P -fpreprocessed tr -d '[:space:]' md5sum cut -c-6	
stress.sh	20 lines
#!/usr/bin/env bash for ((testNum=0;testNum<\$4;testNum++)) do "./\$3".out > input "./\$2".out < input > outSlow "./\$1".out < input > outWrong if ! (cmp -s "outWrong" "outSlow") then echo "Error found!" echo "Input:" cat input echo "Wrong Output:" cat outWrong echo "Slow Output:" cat outSlow exit fi done echo Passed \$4 tests	

troubleshoot.txt	75 lines
General: * Write down most of your thoughts, even if you’re not sure whether they’re useful. * Give your variables (and files) meaningful names. * Stay organized and don’t leave papers all over the place! * You should know what your code is doing ... 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? * Remove debug output. * Make sure to submit the right file. Wrong answer: * Print your solution! Print debug output as well. * Read the full problem statement again. * Have you understood the problem correctly? * Are you sure your algorithm works? * Try writing a slow (but correct) solution. * Can your algorithm handle the whole range of input? * Did you consider corner cases (ex. n=1)? * Is your output format correct? (including whitespace) * Are you clearing all data structures between test cases? * Any uninitialized variables? * Any undefined behavior (array out of bounds)? * Any overflows or NaNs (or shifting ll by >=64 bits)? * Confusing N and M, i and j, etc.? * Confusing ++i and i++? * Return vs continue vs break? * Are you sure the STL functions you use work as you think? * Add some assertions, maybe resubmit. * Create some test cases to run your algorithm on. * Go through the algorithm for a simple case. * Go through this list again. * Explain your algorithm to a teammate. * Ask the teammate to look at your code. * Go for a small walk, e.g. to the toilet. * Rewrite your solution from the start or let a teammate do it. Geometry: * Work with ints if possible. * Correctly account for numbers close to (but not) zero. ⇨Related: for functions like acos make sure absolute val of input is not (slightly) greater than one. * Correctly deal with vertices that are collinear, concyclic, coplanar (in 3D), etc. * Subtracting a point from every other (but not itself)? 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’s your complexity? Large TL does not mean that something simple (like NlogN) isn’t intended. * Are you copying a lot of unnecessary data? (References)	

- * Avoid vector, map. (use arrays/unordered_map)
- * How big is the input and output? (consider FastIO)
- * What do your teammates think about your algorithm?
- * Calling count() on multiset?

Memory limit exceeded:
* What is the max amount of memory your algorithm should need?
* Are you clearing all data structures between test cases?
* If using pointers try BumpAllocator.

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 $x = -b/2a$.

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

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

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

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

2.2 Recurrences

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

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

Non-distinct roots r become polynomial factors, e.g.

$$a_n = (d_1n + d_2)r^n.$$

2.3 Trigonometry

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

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

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

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

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

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

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

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

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

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

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

2.4.2 Quadrilaterals Law of tangents:
$$\tan \frac{\alpha + \beta}{2} = \frac{\tan \frac{\alpha}{2} + \tan \frac{\beta}{2}}{1 - \tan \frac{\alpha}{2} \tan \frac{\beta}{2}}$$

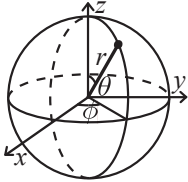
With side lengths a, b, c, d , diagonals e, f , diagonals angle θ , area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

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

2.4.3 Spherical coordinates

For cyclic quadrilaterals the sum of opposite angles is 180° ,

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



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

2.5 Derivatives/Integrals

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

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

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

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

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

Expectation is linear:

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

For independent X and Y ,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

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

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

Poisson distribution

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

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

Stationary 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.

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> bd4c35, 16 lines

```
using namespace __gnu_pbds;
template<class T> using IndexedSet = 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
// Number of element <= k
int atMost (IndexedSet<pair<int, int>>& T, int k) {
    return T.order_of_key({k, INT_MAX});
}
// Number of element grater >= k
int atLeast (IndexedSet<pair<int, int>>& T, int k) {
    return T.size() - T.order_of_key({k, INT_MIN});
}
int getSum (Tree<pair<int, int>>& T, int l, int r) {
    return atMost (T, r) - atMost (T, l - 1);
}
```

CoordCompress.h

Description: coordinate compression, get index of x by counting number of element less than x
Time: $\mathcal{O}(N \log N)$ 14ce69, 7 lines

```
template<class T> void compress (vector<T>& v) {
    ranges::sort (v);
    v.erase(unique (v.begin(), v.end()), v.end());
}
template<class T> int get (vector<T>& v, T x) {
    return ranges::lower_bound (v, x) - begin (v);
}
```

```
}

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

```
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 $\mathcal{O}(\log N)$.

```
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 npos = pos+pw;
            if (npos <= N && bit[npos-1] < sum)
                pos = npos, sum -= bit[pos-1];
        }
        return pos;
    }
};
```

SegmentTree.h

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

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

```
template<class T> struct SegmentTree {
    const T ID{};
    T cmb(T a, T b) { return a + b; }
    int N = 1; vector<T> seg;
    SegmentTree(int _N) {
```

```
        while (N < _N) N *= 2;
        seg.assign(2*N, ID);
    }
    void update(int p, T val) { // set val at position p
        seg[p += N] = val;
        for (p /= 2; p > 0; p /= 2) {
            seg[p] = cmb(seg[2*p], seg[2*p+1]);
        }
    }
    T query(int l, int r) { // zero-indexed, inclusive
        T lf = ID, rf = ID;
        for (l += N, r += N + 1; l < r; l /= 2, r /= 2) {
            if (l & 1) lf = cmb(lf, seg[l++]);
            if (r & 1) rf = cmb(seg[--r], rf);
        }
        return cmb(lf, rf);
    }
    // int first_at_least(int lo, int val, int ind, int l, int r)
    //      $\hookrightarrow$  { // 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 !=
    //              $\hookrightarrow$  -1) return res;
    //         return first_at_least(lo, val, 2*ind+1, m+1, r);
    //     }
    //     int k_th_one(int k, int ind, int l, int r) {
    //         if (l == r) return l;
    //         int m = (l + r) / 2;
    //         int lc = 2*ind, rc = 2*ind+1;
    //         if (seg[lc] >= k) return k_th_one(k, lc, l, m);
    //         else return k_th_one(k-seg[lc], rc, 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: $\mathcal{O}(\log N)$

```
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] +=
             $\hookrightarrow$  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,
         $\hookrightarrow$  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 -
     $\hookrightarrow$  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
         $\hookrightarrow$  +1, M+1, R));
}
};
```

3.3 2D Range Queries

PrefixSum2D.h

Description: calculates rectangle sums in constant time

```
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

ModIntShort.h

Description: Modular arithmetic. Assumes MOD is prime.

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

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

ModFactInt.h	
Description: Combinations modulo a prime MOD . Assumes $2 \leq N \leq MOD$.	
Usage: F.gen(100), F.C(6, 4); // 15	
Time: $\mathcal{O}(N)$	232d42, 22 lines
<pre>struct { vector<int> inv, fac, ifac; void gen(int N) { inv.resize(N); fac.resize(N); ifac.resize(N); inv[1] = fac[0] = ifac[0] = 1; for (int i = 2; i < N; i++){ inv[i] = int(MOD-(int64_t)MOD/i*inv[MOD%i]%MOD); } for (int i = 1; i < N; i++) { fac[i] = int((int64_t)fac[i-1]*i%MOD); ifac[i] = int((int64_t)ifac[i-1]*inv[i]%MOD); } } int P(int n, int r) { // nPr if (n < r r < 0) return 0; return (int64_t)fac[n]*ifac[n-r]%MOD; } int C(int n, int r) { // nCr if (n < r r < 0) return 0; return (int64_t)fac[n]*ifac[r]%MOD*ifac[n-r]%MOD; } } F;</pre>	

ModMulLL.h	
Description: Multiply two 64-bit integers mod another if 128-bit is not available. modMul is equivalent to (ul)(__int128(a)*b%mod). Works for $0 \leq a, b < mod < 2^{63}$. 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;
}
```

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

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

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

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

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 $O\left(n^{2/3}\right)$ then all $s_f\left(\frac{n}{d}\right)$ 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 N . Runs faster if only odd indices are stored.	
Time: $\mathcal{O}(N \log \log N)$ or $\mathcal{O}(N)$	5a9b1c, 20 lines

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

PrimeCnt.h
Description: Counts number of primes up to N . Can also count sum of primes.

Time: $\mathcal{O}\left(N^{3/4} / \log N\right)$, 60ms for $N = 10^{11}$, 2.5s for $N = 10^{13}$	
	c04e96, 20 lines
<pre>ll count_primes(ll N) { // count_primes(1e13) == 346065536839 if (N <= 1) return 0; int sq = (int)sqrt(N);</pre>	

```
vl big_ans((sq+1)/2), small_ans(sq+1);
FOR(i,1,sq+1) small_ans[i] = (i-1)/2;
F0R(i,sz(big_ans)) big_ans[i] = (N/(2*i+1)-1)/2;
vb skip(sq+1); int prime_cnt = 0;
for (int p = 3; p <= sq; p += 2) if (!skip[p]) { // primes
    for (int j = p; j <= sq; j += 2*p) skip[j] = 1;
    F0R(j,min((ll)sz(big_ans), (N/p/p+1)/2)) {
        ll prod = (ll)(2*j+1)*p;
        big_ans[j] -= (prod > sq ? small_ans[(double)N/prod]
            : big_ans[prod/2]) - prime_cnt;
    }
    for (int j = sq, q = sq/p; q >= p; --q) for (;j >= q*p;--j)
        small_ans[j] -= small_ans[q] - prime_cnt;
    ++prime_cnt;
}
return big_ans[0]+1;
}
```

MillerRabin.h	
Description: Deterministic primality test, works up to 2^{64} . For larger numbers, extend A randomly.	
"ModMulLL.h"	f5b0fe, 11 lines

```
bool prime(ul n) { // not ll!
    if (n < 2 || n % 6 % 4 != 1) return n - 2 < 2;
    ul s = __builtin_ctzll(n - 1), d = n >> s;
    ul A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    for (auto a : A) {
        ul p = modPow(a, d, n), i = s;
        while (p != 1 && p != n-1 && a % n && i--) p = modMul(p, p,
            ↳n);
        if (p != n - 1 && i != s) return false;
    }
    return true;
}
```

FactorBasic.h	
Description: Factors integers.	
Time: $\mathcal{O}\left(\sqrt{N}\right)$	8b0341, 53 lines

```
inline namespace factorBasic {
template<class T> vector<pair<T, int>>> factor(T x) {
    vector<pair<T, int>>> primes;
    for (T i = 2; i*i <= x; ++i) if (x % i == 0) {
        int t = 0;
        while (x % i == 0) x /= i, t++;
        primes.push_back({i, t});
    }
    if (x > 1) primes.push_back({x, 1});
    return primes;
}
```

```
/* Note:
 * number of operations needed s.t.
 *     phi(phi(...phi(n)...))=1
 * is  $O(\log n)$ .
 * Euler's theorem:  $a^{\wedge\{\phi(p)\}} \equiv 1 \pmod p$ ,  $\gcd(a,p)=1$ 
 */
template<class T> T phi(T x) {
    for (auto& [p, exp]: factor(x)) x -= x/p;
    return x;
}
```

```
template<class T> void tour(vector<pair<T, int>>> &v,
vector<T> &V, int ind, T cur) {
    if (ind == (int)size(v)) V.push_back(cur);
    else {
        T mul = 1;
        for (int i = 0; i <= v[ind].second; i++) {
```

```
        tour(v, V, ind + 1, cur * mul);
        mul *= v[ind].first;
    }
}

template<class T> vector<T> getDiv(T x) {
    auto v = factor(x);
    vector<T> V;
    tour(v, V, 0, (T)1);
    ranges::sort(V);
    return V;
}

template<class T> vector<T> getDiv(T x) {
    vector<T> V;
    for (T i = 1; i*i <= x; ++i) if (x % i == 0) {
        V.push_back(i);
        if (x / i != i) V.push_back(x / i);
    }
    ranges::sort(V);
    return V;
}
}
```

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

Time: $\mathcal{O}\left(N^{1/4}\right)$, less for numbers with small factors
"MillerRabin.h" 113e1c, 23 lines

```
ul pollard(ul n) { // return some nontrivial factor of n
    auto f = [n](ul x) { return modMul(x, x, n) + 1; };
    ul x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modMul(prd, max(x, y)-min(x, y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return gcd(prd, n);
}
```

```
void factor_rec(ul n, map<ul, int> &cnt) {
    if (n == 1) return;
    if (prime(n)) { ++cnt[n]; return; }
    ul u = pollard(n);
    factor_rec(u, cnt), factor_rec(n/u, cnt);
}
```

```
vector<pair<ul, int>> factor(ul n) {
    map<ul, int> cnt;
    factor_rec(n, cnt);
    return vector<pair<ul, int>>(cnt.begin(), cnt.end());
}
```

4.3 Euclidean Algorithm

Euclid.h
Description: Generalized Euclidean algorithm. euclid and invGeneral work for $A, B < 2^{62}$.
Time: $\mathcal{O}(\log AB)$

```
// 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 $p \mid x - y$:

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

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 ($g.x = x$).

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	$\sim 2\text{e}5$	$\sim 2\text{e}8$

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:

$$\begin{aligned} \sum_{i=m}^{\infty} f(i) &= \int_m^{\infty} f(x) dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m) \\ &\approx \int_m^{\infty} f(x) dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m)) \end{aligned}$$

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 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.

5.4 Young Tableaux

Let a **Young diagram** have shape $\lambda = (\lambda_1 \geq \cdots \geq \lambda_k)$, where λ_i equals the number of cells in the i -th (left-justified) row from the top. A **Young tableau** of shape λ 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 λ is equal to $f^\lambda = \frac{n!}{\prod h_\lambda(i, j)}$.

Schensted’s Algorithm: converts a permutation σ 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 σ , 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 λ_i^* is size of the i -th column.

Numerical (6)

6.1 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); F0R(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);
    F0R(i,x) F0R(j,y) F0R(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: $d_1 \geq \cdots \geq d_n$ 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

DFS.h

Description: Basic DFS implementation for undirected graphs

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

a4b944, 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);
        }
    }
};
```

BFS.h

Description: Performs a standard BFS on an undirected graph.

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

a2f9d4, 26 lines

```
struct BFS {
    int SZ;
    vector<bool> vis;
    vector<vector<int>> adj;
    BFS(int _SZ) {
        SZ = _SZ;
        adj.resize(SZ);
        vis.resize(SZ);
    }
    void ae(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void bfs(int src) {
        vis[src] = 1;
        queue<int> todo; todo.push(src);
        while (!todo.empty()) {
            auto u = todo.front(); todo.pop();
            for (auto& v: adj[u]) {
                if (vis[v]) continue;
                vis[v] = 1;
                todo.push(v);
            }
        }
    }
};
```

Bipartite.h

Description: Checks if an undirected graph is bipartite. Colors nodes with 1/2. If not bipartite, returns false.

Time: $\mathcal{O}(N + M)$	4f7998, 28 lines
<pre>struct Bipartite { int N; vector<int> color; vector<vector<int>>> adj; Bipartite(int _N) { N = _N; adj.resize(N); color.resize(N); } void ae(int u, int v) { adj[u].push_back(v); adj[v].push_back(u); } bool check() { queue<int> todo; for (int src = 0; src < N; src++) { if (color[src]) continue; color[src] = 1; todo.push(src); while (!todo.empty()) { int u = todo.front(); todo.pop(); for (auto &v: adj[u]) { if (color[v] == color[u]) return false; if (!color[v]) color[v] = 3 - color[u], todo.push(v); } } } return true; } };</pre>	

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

<pre>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; } };</pre>	6779f6, 23 lines
--	------------------

DirectedCycle.h
Description: stack

<pre>struct DirectedCycle { int SZ; vector<bool> inStk, vis; vector<int> stk, cyc; vector<vector<int>>> adj; DirectedCycle(int _SZ) { SZ = _SZ; adj.resize(SZ);</pre>	90d8f6, 29 lines
--	------------------

<pre> inStk.resize(SZ), vis.resize(SZ); } void ae(int u, int v) { adj[u].push_back(v); } vector<int> gen() { for (int u = 0; u < SZ; u++) { if (!vis[u] && empty(cyc)) dfs(u); } return cyc; } void dfs(int u) { stk.push_back(u); inStk[u] = vis[u] = 1; for (auto& v: adj[u]) { if (inStk[v]) cyc = {ranges::find(stk, v), end(stk)}; else if (!vis[v]) dfs(v); if (size(cyc)) return; } stk.pop_back(); inStk[u] = 0; } };</pre>	
---	--

Dijkstra.h
Description: shortest path
Time: $\mathcal{O}(N \log N + M)$

<pre>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}); } 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); } } };</pre>	5afe43, 27 lines
---	------------------

FloydWarshall.h
Description: All-Pairs Shortest Path

<pre>void floydWarshall(V<v1> &m) { int n = sz(m); F0R(i, n) ckmin(m[i][i], 0LL); F0R(k, n) F0R(i, n) F0R(j, n) if (m[i][k] != BIG && m[k][j] != BIG) { auto newDist = max(m[i][k] + m[k][j], -BIG); ckmin(m[i][j], newDist); } F0R(k, n)</pre>	a2bfcc, 14 lines
---	------------------

<pre> if (m[k][k] < 0) F0R(i, n) F0R(j, n) if (m[i][k] != BIG && m[↪k][j] != BIG) m[i][j] = -BIG; }</pre>	
--	--

7.2 Trees

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

<pre>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]]; // 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); } } };</pre>	3c22a4, 37 lines
--	------------------

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

<pre>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); }</pre>	5c0112, 47 lines
--	------------------

```
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 DFS Algorithms

SCCK.h
Description: Kosaraju’s Algorithm, DFS twice to generate strongly connected components in topological order. *a, b* in same component if both *a* → *b* and *b* → *a* exist.
Time: $\mathcal{O}(N + M)$

```
struct SCC {
    int N;
    vector<bool> vis;
    vector<vector<int>>> adj, radj;
    vector<int> todo, comp, comps;
    SCC(int _N) {
        N = _N;
        adj.resize(N), radj.resize(N);
        comp.resize(N, -1), vis.resize(N);
    }
    void ae(int u, int v) {
        adj[u].push_back(v);
        radj[v].push_back(u);
    }
    void dfs(int u) {
        vis[u] = true;
        for (auto &v: adj[u]) if (!vis[v]) dfs(v);
        todo.push_back(u);
    }
    void dfs2(int u, int id) {
        comp[u] = id;
        for (auto &v: radj[u]) if (comp[v] == -1) dfs2(v, id);
    }
    void gen() {
        for (int u = 0; u < N; u++) if (!vis[u]) dfs(u);
        ranges::reverse(todo);
        for (auto &u: todo) if (comp[u] == -1) {
            dfs2(u, u);
            comps.push_back(u);
        }
    }
};
```

```
};

TwoSAT.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)&&... becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).
Usage: TwoSat ts;
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setVal(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(N); // Returns true iff it is solvable
ts.ans[0..N-1] holds the assigned values to the vars
"SCCK.h" ff0f3d, 31 lines

struct TwoSAT {
    int N = 0; vpi edges;
    void init(int _N) { N = _N; }
    int addVar() { return N++; }
    void either(int x, int y) {
        x = max(2*x,-1-2*x), y = max(2*y,-1-2*y);
        edges.eb(x,y); }
    void implies(int x, int y) { either(~x,y); }
    void must(int x) { either(x,x); }
    void atMostOne(const vi& li) {
        if (sz(li) <= 1) return;
        int cur = ~li[0];
        FOR(i,2,sz(li)) {
            int next = addVar();
            either(cur,~li[i]); either(cur,next);
            either(~li[i],next); cur = ~next;
        }
        either(cur,~li[1]);
    }
    vb solve() {
        SCC S; S.init(2*N);
        each(e,edges) S.ae(e.f^1,e.s), S.ae(e.s^1,e.f);
        S.gen(); reverse(all(S.comps)); // reverse topo order
        for (int i = 0; i < 2*N; i += 2)
            if (S.comp[i] == S.comp[i^1]) return {};
        vi tmp(2*N); each(i,S.comps) if (!tmp[i])
            tmp[i] = 1, tmp[S.comp[i^1]] = -1;
        vb ans(N); F0R(i,N) ans[i] = tmp[S.comp[2*i]] == 1;
        return ans;
    }
};
```

7.4 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: $\mathcal{O}(\alpha(N))$

```
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;
    }
};
```

Kruskal.h
Description: Computes the weight of a Minimum Spanning Tree (MST) using Kruskal’s algorithm. Requires DSU.
Time: $\mathcal{O}(M \log M)$, where M = number of edges

```
"DSU.h" f6c35c, 8 lines

template<class T> T Kruskal(int N, vector<array<T, 3>> eg) {
    ranges::sort(eg);
    T ans = 0; DSU D(N);
    for (auto &[w, u, v] : eg) {
        if (D.unite(u, v)) ans += w;
    }
    return ans;
};

DSUrb.h
Description: Disjoint Set Union with Rollback
```

```
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];
        }
    }
};
```

Geometry (8)

8.1 Primitives

PointShort.h
Description: Use in place of complex<T>.

```
using T = db; // or ll
const T EPS = 1e-9; // adjust as needed
using P = pair<T,T>; using vP = V<P>; using Line = pair<P,P>;
int sgn(T a) { return (a>EPS)-(a<-EPS); }
T sq(T a) { return a*a; }

T abs2(P p) { return sq(p.f)+sq(p.s); }
T abs(P p) { return sqrt(abs2(p)); }
T arg(P p) { return atan2(p.s,p.f); }
P conj(P p) { return P(p.f,-p.s); }
P perp(P p) { return P(-p.s,p.f); }
P dir(T ang) { return P(cos(ang),sin(ang)); }

P operator+(P l, P r) { return P(l.f+r.f,l.s+r.s); }
P operator-(P l, P r) { return P(l.f-r.f,l.s-r.s); }
P operator*(P l, T r) { return P(l.f*r,l.s*r); }
P operator/(P l, T r) { return P(l.f/r,l.s/r); }
P operator*(P l, P r) { // complex # multiplication
```

```
    return P(l.f*r.f-l.s*r.s,l.s*r.f+l.f*r.s); }
P operator/(P l, P r) { return l*conj(r)/abs2(r); }

P unit(const P& p) { return p/abs(p); }
T dot(const P& a, const P& b) { return a.f*b.f+a.s*b.s; }
T dot(const P& p, const P& a, const P& b) { return dot(a-p,b-p)
    ↪; }
T cross(const P& a, const P& b) { return a.f*b.s-a.s*b.f; }
T cross(const P& p, const P& a, const P& b) {
    return cross(a-p,b-p); }
P reflect(const P& p, const Line& l) {
    P a = l.f, d = l.s-l.f;
    return a+conj((p-a)/d)*d; }
P foot(const P& p, const Line& l) {
    return (p+reflect(p,l))/(T)2; }
bool onSeg(const P& p, const Line& l) {
    return sgn(cross(l.f,l.s,p)) == 0 && sgn(dot(p,l.f,l.s)) <= 0
    ↪; }
ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.f << ", " << p.s << ")"; }
```

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: $\mathcal{O}(N)$

```
4538e4, 13 lines
vi 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 relaxed id, 7 lines

```
887c1d, 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;
};
```

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* & $\neg x$ is the least bit in *x*.
- for (int *x* = *m*; *x*;) { $\neg x$ &= *m*; ... } loops over all subset masks of *m* (except *m* itself).
- *c* = *x* & $\neg x$, *r* = *x* + *c*; ((*r* ^ *x*) >> 2) / *c* | *r* is the next number after *x* with the same number of bits set.
- F0R(*b*,*k*) F0R(*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.

```
7b3c70, 17 lines
```

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

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

10.5 Miscellaneous

GridDirection.h

Description: Predefined direction for grid-based problems. Includes common movement sets for 2D and 3D grids such as 4-directional, 8-directional, diagonal-only, knight moves, 3D etc.

efed5e, 22 lines

```
// Down, Right, Up, Left
const int dx[4] = {1, 0, -1, 0};
const int dy[4] = {0, 1, 0, -1};
const char dc[4] = {'D', 'R', 'U', 'L'};

// Down, Down-Right, Right, Up-Right, Up, Up-Left, Left, Down-
  ↖Left
const int dx8[8] = {1, 1, 0, -1, -1, -1, 0, 1};
const int dy8[8] = {0, 1, 1, 1, 0, -1, -1, -1};

// Diagonals only
const int dxDiag[4] = {1, 1, -1, -1};
const int dyDiag[4] = {1, -1, 1, -1};

// Knight moves
const int dxK[8] = {2, 2, 1, 1, -1, -1, -2, -2};
const int dyK[8] = {1, -1, 2, -2, 2, -2, 1, -1};

// 3D Grid
const int dx[6] = {1, -1, 0, 0, 0, 0};
const int dy[6] = {0, 0, 1, -1, 0, 0};
const int dz[6] = {0, 0, 0, 0, 1, -1};
```