

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

Contest (1)

template.cpp	14 lines
<pre>#include &lt;bits/stdc++.h&gt; using namespace std;  #define rep(i, a, b) for(int i = a; i &lt; (b); ++i) #define all(x) begin(x), end(x) #define sz(x) (int)(x).size() typedef long long ll; typedef pair&lt;int, int&gt; pii; typedef vector&lt;int&gt; vi;  int main() {     cin.tie(0)-&gt;sync_with_stdio(0);     cin.exceptions(cin.failbit); }</pre>	
.bashrc	3 lines
<pre>alias g++='g++ -Wall -Wextra -Wshadow -D_GLIBCXX_DEBUG - ggdb3 -std=gnu++23 -fmax-errors=2 -fsanitize=address, undefined' xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =&lt; # g++ -x c++-header template.cpp</pre>	
.vimrc	4 lines
<pre>set ic ts=4 sw=4 nu im jk &lt;esc&gt; no ; : ca hash w !cpp -dD -P -fpreprocessed \\\ tr -d '[:space:]' \\ md5sum \\\ cut -c-6</pre>	
hash.sh	3 lines
<pre># Hashes a file, ignoring all whitespace and comments. Use for # verifying that code was correctly typed.</pre>	

template .bashrc .vimrc hash troubleshoot	
1	cpp -dD -P -fpreprocessed   tr -d '[:space:]'   md5sum   cut -c-6
1	troubleshoot.txt
3	Pre-submit: Write a few simple test cases if sample is not enough. Are time limits close? If so, generate max cases. Is the memory usage fine? Could anything overflow? Make sure to submit the right file.
8	
10	Wrong answer: Print your solution! Print debug output, as well. Are you clearing all data structures between test cases? Can your algorithm handle the whole range of input?
11	Read the full problem statement again. Do you handle all corner cases correctly? Have you understood the problem correctly?
17	Any uninitialized variables? Any overflows? Confusing N and M, i and j, etc.?
21	Are you sure your algorithm works? What special cases have you not thought of?
22	Are you sure the STL functions you use work as you think? Add some assertions, maybe resubmit. Create some testcases to run your algorithm on.
24	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. Is your output format correct? (including whitespace) Rewrite your solution from the start or let a teammate do it.
	Runtime error: Have you tested all corner cases locally? Any uninitialized variables? Are you reading or writing outside the range of any vector? Any assertions that might fail? Any possible division by 0? (mod 0 for example) Any possible infinite recursion? Invalidated pointers or iterators? Are you using too much memory? Debug with resubmits (e.g. remapped signals, see Various).
	Time limit exceeded: Do you have any possible infinite loops? What is the complexity of your algorithm? Are you copying a lot of unnecessary data? (References) How big is the input and output? (consider scanf) Avoid vector, map. (use arrays/unordered_map) What do your teammates think about your algorithm?
	Memory limit exceeded: What is the max amount of memory your algorithm should need? Are you clearing all data structures between test cases?
Mathematics (2)	
2.1 Equations	
$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	
The extremum is given by $x = -b/2a$ .	

	$\begin{aligned} ax + by &= e & \Rightarrow & x = \frac{ed - bf}{ad - bc} \\ cx + dy &= f & & y = \frac{af - ec}{ad - bc} \end{aligned}$
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	
$\sin(v + w) = \sin v \cos w + \cos v \sin w$	
$\cos(v + w) = \cos v \cos w - \sin v \sin w$	
$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$	
$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$	
$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$	
$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$	
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:  $a^2 = b^2 + c^2 - 2bc \cos \alpha$

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

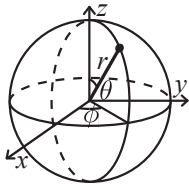
2.4.2 Quadrilaterals

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



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

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

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

2.7 Series

$$\begin{aligned} e^x &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, (-\infty < x < \infty) \\ \ln(1+x) &= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots, (-1 < x \leq 1) \\ \sqrt{1+x} &= 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \cdots, (-1 \leq x \leq 1) \\ \sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, (-\infty < x < \infty) \\ \cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, (-\infty < x < \infty) \end{aligned}$$

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  $Y$ ,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

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.

$\pi$  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.

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

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

## Data structures (3)

### OrderStatisticTree.h

**Description:** A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null\_type.  
**Time:**  $\mathcal{O}(\log N)$

```
#include <bits/extc++.h> // 893
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>; // 988

void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9)); // 6bd
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
} // cbb
```

### HashMap.h

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

#include <bits/extc++.h> // 1e4
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
    const uint64_t C = 1l(4e18 * acos(0)) | 7l;
    ll operator()(ll x) const { return __builtin_bswap64(x*C)
        ; }
}; // 198
__gnu_pbds::gp_hash_table<ll,int,chash> h({}, {}, {}, {}, {
    1<<16});
```

### SegmentTree.h

**Description:** Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.  
**Time:**  $\mathcal{O}(\log N)$

```
struct Tree { // 026
    typedef int T;
    static constexpr T unit = INT_MIN;
    T f(T a, T b) { return max(a, b); } // (any associative
        fn)
    vector<T> s; int n;
    Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {} // c86
    void update(int pos, T val) {
        for (s[pos += n] = val; pos /= 2;)
            s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
    }
    T query(int b, int e) { // query [b, e) // e90
        T ra = unit, rb = unit;
        for (b += n, e += n; b < e; b /= 2, e /= 2) {
            if (b % 2) ra = f(ra, s[b++]);
            if (e % 2) rb = f(s[--e], rb);
        } // 490
        return f(ra, rb);
    }
};
```

### LazySegmentTree.h

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

```
"../various/BumpAllocator.h"
34cef5, 50 lines

const int inf = 1e9; // 317
struct Node {
    Node *l = 0, *r = 0;
    int lo, hi, mset = inf, madd = 0, val = -inf;
    Node(int lo, int hi) : lo(lo), hi(hi) {} // Large interval of
        -inf
    Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {} // f58
    if (lo + 1 < hi) {
        int mid = lo + (hi - lo) / 2;
        l = new Node(v, lo, mid); r = new Node(v, mid, hi);
        val = max(l->val, r->val);
    } // 22c
    else val = v[lo];
}
int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val; // 2ff
    push();
    return max(l->query(L, R), r->query(L, R));
}
void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return; // 1bd
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
```

```
else {
    push(), l->set(L, R, x), r->set(L, R, x);
    val = max(l->val, r->val);
} // f3e
}
void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
        if (mset != inf) mset += x; // 415
        else madd += x;
        val += x;
    }
    else {
        push(), l->add(L, R, x), r->add(L, R, x); // cac
        val = max(l->val, r->val);
    }
}
}
void push() {
    if (!l) {} // 53d
    int mid = lo + (hi - lo) / 2;
    l = new Node(lo, mid); r = new Node(mid, hi);
}
if (mset != inf)
    l->set(lo, hi, mset), r->set(lo, hi, mset), mset = inf; // 333
else if (madd)
    l->add(lo, hi, madd), r->add(lo, hi, madd), madd = 0;
}
};
```

### UnionFindRollback.h

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

```
de4ad0, 21 lines

struct RollbackUF { // f73
    vi e; vector<pii> st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return sz(st); } // cbd
    void rollback(int t) {
        for (int i = time(); i --> t;)
            e[st[i].first] = st[i].second;
        st.resize(t);
    } // e73
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        st.push_back({a, e[a]}); // 274
        st.push_back({b, e[b]});
        e[a] += e[b]; e[b] = a;
        return true;
    }
}; // 214
```

### SubMatrix.h

**Description:** Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).  
**Usage:** SubMatrix<int> m(matrix);  
m.sum(0, 0, 2, 2); // top left 4 elements  
**Time:**  $\mathcal{O}(N^2 + Q)$

```
c59ada, 13 lines

template<class T> // 03e
struct SubMatrix {
    vector<vector<T>> p;
    SubMatrix(vector<vector<T>>& v) {
        int R = sz(v), C = sz(v[0]);
        p.assign(R+1, vector<T>(C+1)); // 4c9
```

```
    rep(r,0,R) rep(c,0,C)
        p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
    }
    T sum(int u, int l, int d, int r) {
        return p[d][r] - p[d][l] - p[u][r] + p[u][l]; //286
    }
};
```

Matrix.h  
**Description:** Basic operations on square matrices.  
**Usage:** Matrix<int, 3> A;

```
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
vector<int> vec = {1,2,3};
vec = (A^N) * vec;
```

```
template<class T, int N> struct Matrix { //1aa
    typedef Matrix M;
    array<array<T, N>, N> d{};
    M operator*(const M& m) const {
        M a;
        rep(i,0,N) rep(j,0,N) //683
            rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
        return a;
    }
    vector<T> operator*(const vector<T>& vec) const {
        vector<T> ret(N); //9bd
        rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
        return ret;
    }
    M operator^(ll p) const {
        assert(p >= 0); //358
        M a, b(*this);
        rep(i,0,N) a.d[i][i] = 1;
        while (p) {
            if (p&1) a = a*b;
            b = b*b; //1d8
            p >>= 1;
        }
        return a;
    }
}; //214
```

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

```
struct Line { //7e3
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
}; //d77
struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); //66e
    }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k, x->k - y->k);
        return x->p >= y->p; //bec
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y)); //890
    }
};
```

```
    while ((y = x) != begin() && (--x)->p >= y->p)
        isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty()); //b07
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

Treap.h  
**Description:** A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.  
**Time:**  $\mathcal{O}(\log N)$

```
struct Node { //829
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int val) : val(val), y(rand()) {}
    void recalc();
}; //3ef

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) { //5d5
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {}; //ca5
    if (cnt(n->l) >= k) { // "n->val >= k" for lower-bound(k)
        auto pa = split(n->l, k);
        n->l = pa.second;
        n->recalc();
        return {pa.first, n}; //b54
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = pa.first;
        n->recalc();
        return {n, pa.second}; //86d
    }
}
```

```
Node* merge(Node* l, Node* r) {
    if (!l) return r; //fbf
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        l->recalc();
        return l; //780
    } else {
        r->l = merge(l, r->l);
        r->recalc();
        return r;
    } //96d
}
```

```
Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second); //99b
}
```

```
// Example application: move the range [l, r) to index k
void move(Node& t, int l, int r, int k) {
    Node *a, *b, *c; //99c
    tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}
```

```
}
```

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

```
struct FT { //711
    vector<ll> s;
    FT(int n) : s(n) {}
    void update(int pos, ll dif) { // a[pos] += dif
        for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
    } //cc4
    ll query(int pos) { // sum of values in [0, pos)
        ll res = 0;
        for (; pos > 0; pos &= pos - 1) res += s[pos-1];
        return res;
    } //477
    int lower_bound(ll sum) { // min pos st sum of [0, pos] >= sum
        // Returns n if no sum is >= sum, or -1 if empty sum is .
        if (sum <= 0) return -1;
        int pos = 0;
        for (int pw = 1 << 25; pw; pw >>= 1) { //fc5
            if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
                pos += pw, sum -= s[pos-1];
        }
        return pos;
    } //e03
};
```

FenwickTree2d.h  
**Description:** Computes sums  $a[i,j]$  for all  $i < I, j < J$ , and increases single elements  $a[i,j]$ . Requires that the elements to be updated are known in advance (call `fakeUpdate()` before `init()`).  
**Time:**  $\mathcal{O}(\log^2 N)$ . (Use persistent segment trees for  $\mathcal{O}(\log N)$ .)

```
"FenwickTree.h" 157f07, 22 lines

struct FT2 { //e22
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
    } //57f
    void init() {
        for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
    }
    int ind(int x, int y) {
        return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); //358
    }
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) { //688
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum;
    } //e03
};
```

RMQ.h  
**Description:** Range Minimum Queries on an array. Returns  $\min(V[a], V[a + 1], \dots, V[b - 1])$  in constant time.  
**Usage:** RMQ rmq(values);  
`rmq.query(inclusive, exclusive);`  
**Time:**  $\mathcal{O}(|V| \log |V| + Q)$

```
template<class T> //722
```

```
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k)
            jmp.emplace_back(sz(V) - pw * 2 + 1); //f6c
        rep(j,0,sz(jmp[k]))
            jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
    }
    T query(int a, int b) { //a3d
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
}; //214
```

### MoQueries.h

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

**Time:**  $O(N\sqrt{Q})$

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1) //342
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer
```

```
vi mo(vector<pii> Q) {
    int L = 0, R = 0, blk = 350; // ~N/sqrt(Q) //cb0
    vi s(sz(Q)), res = s;
    #define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) { //623
        pii q = Q[qi];
        while (L > q.first) add(--L, 0);
        while (R < q.second) add(R++, 1);
        while (L < q.first) del(L++, 0);
        while (R > q.second) del(--R, 1); //d22
        res[qi] = calc();
    }
    return res;
} //842
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root = 0) {
    int n = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
    vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
    add(0, 0), in[0] = 1;
    auto dfs = [&](int x, int p, int dep, auto& f) -> void {
        //263
        par[x] = p;
        L[x] = N;
        if (dep) I[x] = N++;
        for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
        if (!dep) I[x] = N++; //23e
        R[x] = N;
    };
    dfs(root, -1, 0, dfs);
    #define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
    iota(all(s), 0); //064
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) rep(end,0,2) {
        int &a = pos[end], b = Q[qi][end], i = 0;
```

```
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                  else { add(c, end); in[c] = 1; } a = c; } \
                //440
    while (!(L[b] <= L[a] && R[a] <= R[b]))
        I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc(); //695
}
return res;
}
```

## Numerical (4)

### 4.1 Polynomials and recurrences

Polynomial.h

```
struct Poly { //1b7
    vector<double> a;
    double operator()(double x) const {
        double val = 0;
        for (int i = sz(a); i--;) (val += x) += a[i];
        return val; //06d
    }
    void diff() {
        rep(i,1,sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    } //b82
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b,
            b=c;
        a.pop_back();
    } //e03
};
```

### PolyRoots.h

**Description:** Finds the real roots to a polynomial.

**Usage:** polyRoots({{2,-3,1}},-1e9,1e9) // solve x<sup>2</sup>-3x+2 = 0

**Time:**  $O(n^2 \log(1/\epsilon))$

```
"Polynomial.h"
vector<double> polyRoots(Poly p, double xmin, double xmax)
{ //840
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax); //9c1
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    rep(i,0,sz(dr)-1) {
        double l = dr[i], h = dr[i+1]; //189
        bool sign = p(l) > 0;
        if (sign ^ (p(h) > 0)) {
            rep(it,0,60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m; //a7f
                else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    } //808
    return ret;
}
```

### PolyInterpolate.h

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

**Time:**  $O(n^2)$

```
typedef vector<double> vd; //159
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1; //746
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    } //0e1
    return res;
}
```

### BerlekampMassey.h

**Description:** Recovers any  $n$ -order linear recurrence relation from the first  $2n$  terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size  $\leq n$ .

**Usage:** berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}

**Time:**  $O(N^2)$

"./number-theory/ModPow.h"

```
vector<ll> berlekampMassey(vector<ll> s) { //b21
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1; //4c7
    rep(i,0,n) { ++m;
        ll d = s[i] % mod;
        rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod; //1b2
        rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }
    //255
    C.resize(L + 1); C.erase(C.begin());
    for (ll& x : C) x = (mod - x) % mod;
    return C;
}
```

### LinearRecurrence.h

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

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

**Time:**  $O(n^2 \log k)$

"f4e444, 26 lines"

```
typedef vector<ll> Poly; //bb1
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1); //251
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1); //12f
        return res;
    };
}
```

```
Poly pol(n + 1), e(pol);
pol[0] = e[1] = 1; //df7

for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
} //c0e

ll res = 0;
rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
return res;
} //cbb
```

4.2 Optimization

GoldenSectionSearch.h

**Description:** Finds the argument minimizing the function  $f$  in the interval  $[a,b]$  assuming  $f$  is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is  $\epsilon$  $\epsilon$ s. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

**Usage:** double func(double x) { return 4+x+.3\*x\*x; }  
double xmin = gss(-1000,1000,func);  
**Time:**  $\mathcal{O}(\log((b-a)/\epsilon))$

```
double gss(double a, double b, double (*f)(double)) { //40b
    double r = (sqrt(5)-1)/2, eps = 1e-7;
    double x1 = b - r*(b-a), x2 = a + r*(b-a);
    double f1 = f(x1), f2 = f(x2);
    while (b-a > eps)
        if (f1 < f2) { //change to > to find maximum//707
            b = x2; x2 = x1; f2 = f1;
            x1 = b - r*(b-a); f1 = f(x1);
        } else {
            a = x1; x1 = x2; f1 = f2;
            x2 = a + r*(b-a); f2 = f(x2); //ec9
        }
    return a;
}
```

HillClimbing.h

**Description:** Poor man's optimization for unimodal functions.

**typedef** array<double, 2> P; *//68a*

```
template<class F> pair<double, P> hillClimb(P start, F f) {
    pair<double, P> cur(f(start), start);
    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
        rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) { //2dc
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        } //a63
    }
    return cur;
}
```

Integrate.h

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

```
template<class F> //e93
double quad(double a, double b, F f, const int n = 1000) {
    double h = (b - a) / 2 / n, v = f(a) + f(b);
    rep(i,1,n*2)
        v += f(a + i*h) * (i%1 ? 4 : 2);
    return v * h / 3; //2d2
```

```
}

IntegrateAdaptive.h
Description: Fast integration using an adaptive Simpson's rule.
Usage: double sphereVolume = quad(-1, 1, [](double x) {
    return quad(-1, 1, [&](double y) {
        return quad(-1, 1, [&](double z) {
            return x*x + y*y + z*z < 1; });});});
} 92dd79, 15 lines
```

```
typedef double d; //e70
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
    d c = (a + b) / 2; //b17
    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
} //836
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
    return rec(f, a, b, eps, S(a, b));
}
```

Simplex.h

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

**Usage:** vvd A = {{1,-1}, {-1,1}, {-1,-2}};  
vd b = {1,1,-4}, c = {-1,-1}, x;  
T val = LPSolver(A, b, c).solve(x);  
**Time:**  $\mathcal{O}(NM * \#\text{pivots})$ , where a pivot may be e.g. an edge relaxation.  $\mathcal{O}(2^n)$  in the general case.

```
typedef double T; // long double, Rational, double + modKP
>... //629
typedef vector<T> vd;
typedef vector<vd> vvd;
```

```
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair //94e
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s]))
    s=j
```

```
struct LPSolver {
    int m, n;
    vi N, B; //282
    vvd D;

    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
        rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j]; //108
        rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
        rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m+1][n] = 1;
    }
} //9c3
void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
        T *b = D[i].data(), inv2 = b[s] * inv;
        rep(j,0,n+2) b[j] -= a[j] * inv2; //d0d
        b[s] = a[s] * inv2;
    }
```

```
rep(j,0,n+2) if (j != s) D[r][j] *= inv;
rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
D[r][s] = inv; //aa5
swap(B[r], N[s]);
}
```

```
bool simplex(int phase) {
    int x = m + phase - 1; //c51
    for (;;) {
        int s = -1;
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        if (D[x][s] >= -eps) return true;
        int r = -1; //bc0
        rep(i,0,m) {
            if (D[i][s] <= eps) continue;
            if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                < MP(D[r][n+1] / D[r][s], B[r])) r = i;
        } //00c
        if (r == -1) return false;
        pivot(r, s);
    }
} //d2f
T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
        pivot(r, n); //f81
        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
        rep(i,0,m) if (B[i] == -1) {
            int s = 0;
            rep(j,1,n+1) ltj(D[i]);
            pivot(i, s); //866
        }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf; //401
}
};
```

4.3 Matrices

Determinant.h

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

```
double det(vector<vector<double>>& a) //309
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1; //454
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k]; //07b
        }
    }
    return res;
}
```

IntDeterminant.h

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

```
const ll mod = 12345; //cab
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
```

```
rep(i,0,n) {
    rep(j,i+1,n) {
        while (a[j][i] != 0) { // gcd step//c65
            ll t = a[i][i] / a[j][i];
            if (t) rep(k,i,n)
                a[i][k] = (a[i][k] - a[j][k] * t) % mod;
            swap(a[i], a[j]);
            ans *= -1;//bc6
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }//b19
    return (ans + mod) % mod;
}
```

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

```
typedef vector<double> vd;//2cf
const double eps = 1e-12;
```

```
int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);//940
    vi col(m); iota(all(col), 0);
```

```
rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)//ddb
        if ((v = fabs(A[r][c])) > bv)
            br = r, bc = c, bv = v;
    if (bv <= eps) {
        rep(j,i,n) if (fabs(b[j]) > eps) return -1;
        break;//de0
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);//328
    bv = 1/A[i][i];
    rep(j,i+1,n) {
        double fac = A[j][i] * bv;
        b[j] -= fac * b[i];
        rep(k,i+1,m) A[j][k] -= fac*A[i][k];//af1
    }
    rank++;
}
```

```
x.assign(m, 0);//3c5
for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
}//807
return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h  
**Description:** To get all uniquely determined values of  $x$  back from SolveLinear, make the following changes:

```
"SolveLinear.h"
08e495, 7 lines

rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)//22b
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];//4e3
```

```
fail;; }
```

SolveLinearBinary.h  
**Description:** Solves  $Ax = b$  over  $\mathbb{F}_2$ . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys  $A$  and  $b$ .  
**Time:**  $\mathcal{O}(n^2m)$

```
typedef bitset<1000> bs;//d90

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);//2c9
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;//13e
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);//b88
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];//76c
            A[j] ^= A[i];
        }
        rank++;
    }
    //7a7
    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
        rep(j,0,i) b[j] ^= A[j][i];//df7
    }
    return rank; // (multiple solutions if rank < m)
}
```

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

```
int matInv(vector<vector<double>>& A) //{9a9
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) { //214
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
            if (fabs(A[j][k]) > fabs(A[r][c]))
                r = j, c = k;
        if (fabs(A[r][c]) < 1e-12) return i;//e5b
        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
        rep(j,0,n)
            swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
        swap(col[i], col[c]);
        double v = A[i][i];//afc
        rep(j,i+1,n) {
            double f = A[j][i] / v;
            A[j][i] = 0;
            rep(k,i+1,n) A[j][k] -= f*A[i][k];
            rep(k,0,n) tmp[j][k] -= f*tmp[i][k];//c80
```

```
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
} //bfb

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
} //e74

rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
return n;
}
```

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

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

This is useful for solving problems on the type

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

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

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

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

```
8f9fa8, 26 lines

typedef double T;//399
vector<T> tridiagonal(vector<T> diag, const vector<T>&
    super,
    const vector<T>& sub, vector<T> b) {
    int n = sz(b); vi tr(n);
    rep(i,0,n-1) {
        if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i]
            = 0//464
            b[i+1] -= b[i] * diag[i+1] / super[i];
            if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
            diag[i+1] = sub[i]; tr[++i] = 1;
        } else {
            diag[i+1] -= super[i]*sub[i]/diag[i];//d50
            b[i+1] -= b[i]*sub[i]/diag[i];
        }
    }
    for (int i = n; i--;) {
        if (tr[i]) { //054
            swap(b[i], b[i-1]);
            diag[i-1] = diag[i];
            b[i] /= super[i-1];
        } else {
            b[i] /= diag[i]; //20b
            if (i) b[i-1] -= b[i]*super[i-1];
        }
    }
    return b;
} //cbb
```

## 4.4 Fourier transforms

### FastFourierTransform.h

**Description:**  $\text{fft}(a)$  computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution:  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by  $n$ , reverse(start+1, end), FFT back. Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFT-Mod.  
**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1\text{s}$  for  $N = 2^{22}$ )

```
typedef complex<double> C; //1ec
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% faster if double) //
    c50
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i, k, 2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
    } //292
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) { //577
            C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
            rolled)
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
    } //15f
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n); //d93
    copy(all(a), begin(in));
    rep(i, 0, sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]); //36e
    fft(out);
    rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}
```

### FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as  $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (in practice  $10^{16}$  or higher). Inputs must be in  $[0, \text{mod})$ .  
**Time:**  $\mathcal{O}(N \log N)$ , where  $N = |A| + |B|$  (twice as slow as NTT or FFT)

```
"FastFourierTransform.h"
b82773, 22 lines

typedef vector<ll> vl; //2c4
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M))
    ;
    vector<C> L(n), R(n), outs(n), outl(n); //c4f
    rep(i, 0, sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut)
    ;
    rep(i, 0, sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut)
    ;
    fft(L), fft(R);
    rep(i, 0, n) {
        int j = -i & (n - 1); //3eb
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / li;
    }
}
```

```
fft(outl), fft(outs);
rep(i, 0, sz(res)) { //58f
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5)
    ;
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
}
return res; //510
}
```

### NumberTheoreticTransform.h

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

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

const ll mod = (119 << 23) + 1, root = 62; // =
998244353 //0ca
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 <<
21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n); //cc5
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i, k, 2*k) rt[i] = rt[i / 2] * z[i & 1] % mod; //4a0
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2) //ed7
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j
            ];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        } //dfc
    }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
    n = 1 << B; //d58
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i, 0, n) //f18
        out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

### FastSubsetTransform.h

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

```
void FST(vi& a, bool inv) { //ae8
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j, i, i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
            inv ? pii(v - u, u) : pii(v, u + v); // AND
        }
    }
}
```

```
        inv ? pii(v, u - v) : pii(u + v, u); // OR/0af
        pii(u + v, u - v); // XOR
    }
}
if (inv) for (int& x : a) x /= sz(a); // XOR only
} //dc4
vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
    rep(i, 0, sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
} //cbb
```

## Number theory (5)

### 5.1 Modular arithmetic

#### ModularArithmetic.h

**Description:** Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

```
"euclid.h"
35bfea, 18 lines

const ll mod = 17; // change to something else //9af
struct Mod {
    ll x;
    Mod(ll xx) : x(xx) {}
    Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
    Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
    } //dd1
    Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
    Mod operator/(Mod b) { return *this * invert(b); }
    Mod invert(Mod a) {
        ll x, y, g = euclid(a.x, mod, x, y);
        assert(g == 1); return Mod((x + mod) % mod); //13e
    }
    Mod operator^(ll e) {
        if (!e) return Mod(1);
        Mod r = *this ^ (e / 2); r = r * r;
        return e&1 ? *this * r : r; //935
    }
};
```

#### ModInverse.h

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

```
6f684f, 3 lines

const ll mod = 1000000007, LIM = 200000; //6f6
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i, 2, LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

#### ModPow.h

```
b83c45, 8 lines

const ll mod = 1000000007; // faster if const //8bc

ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod; //7e5
    return ans;
}
```

#### ModLog.h

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

```
Time: O(sqrt(m))
c040b8, 11 lines

ll modLog(ll a, ll b, ll m) { //260
    ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
    unordered_map<ll, ll> A;
    while (j <= n && (e = f * e % m) != b % m)
        A[e * b % m] = j++;
}
```



```
    if (e == b % m) return j;//d16
    if (__gcd(m, e) == __gcd(m, b))
        rep(i,2,n+2) if (A.count(e == e * f % m))
            return n * i - A[e];
    return -1;
} //cbb
```

### ModSum.h

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

<pre>typedef unsigned long long ull; //df3 ull sumsq(ull to) { return to / 2 * ((to-1)   1); }</pre>	5c5bc5, 16 lines
--	------------------

ull divsum(ull to, ull c, ull k, ull m) {  
 ull res = k / m \* sumsq(to) + c / m \* to;  
 k %= m; c %= m; //e1a  
 if (!k) return res;  
 ull to2 = (to \* k + c) / m;  
 return res + (to - 1) \* to2 - divsum(to2, m-1 - c, m, k);  
}  
//1ae  
ll modsum(ull to, ll c, ll k, ll m) {  
 c = ((c % m) + m) % m;  
 k = ((k % m) + m) % m;  
 return to \* c + k \* sumsq(to) - m \* divsum(to, c, k, m);  
} //cbb

### ModMulLL.h

**Description:** Calculate  $a \cdot b \bmod c$  (or  $a^b \bmod c$ ) for  $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$ .  
**Time:**  $\mathcal{O}(1)$  for modmul,  $\mathcal{O}(\log b)$  for modpow

<pre>typedef unsigned long long ull; //a9c ull modmul(ull a, ull b, ull M) {     ll ret = a * b - M * ull(1.L / M * a * b);     return ret + M * (ret &lt; 0) - M * (ret &gt;= (ll)M); } ull modpow(ull b, ull e, ull mod) { //51d     ull ans = 1;     for (; e; b = modmul(b, b, mod), e /= 2)         if (e &amp; 1) ans = modmul(ans, b, mod);     return ans; } //cbb</pre>	bbbd8f, 11 lines
--	------------------

### ModSqrt.h

**Description:** Tonelli-Shanks algorithm for modular square roots. Finds  $x$  s.t.  $x^2 = a \pmod p$  ( $-x$  gives the other solution).  
**Time:**  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$

<pre>"ModPow.h" ll sqrt(ll a, ll p) { //473     a %= p; if (a &lt; 0) a += p;     if (a == 0) return 0;     assert(modpow(a, (p-1)/2, p) == 1); // else no solution     if (p % 4 == 3) return modpow(a, (p+1)/4, p);     // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5//a48     ll s = p - 1, n = 2;     int r = 0, m;     while (s % 2 == 0)         ++r, s /= 2;     while (modpow(n, (p - 1) / 2, p) != p - 1) ++n; //c4b     ll x = modpow(a, (s + 1) / 2, p);     ll b = modpow(a, s, p), g = modpow(n, s, p);     for (; r = m) {         ll t = b;         for (m = 0; m &lt; r &amp;&amp; t != 1; ++m) //faf             t = t * t % p;         if (m == 0) return x;</pre>	19a793, 24 lines
--	------------------

```
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p; //a28
    b = b * g % p;
}
}
```

## 5.2 Primality

**FastEratosthenes.h**  
**Description:** Prime sieve for generating all primes smaller than LIM.  
**Time:** LIM=1e9  $\approx$  1.5s

<pre>const int LIM = 1e6; //058 bitset&lt;LIM&gt; isPrime; vi eratosthenes() {     const int S = (int)round(sqrt(LIM)), R = LIM / 2;     vi pr = {2}, sieve(S+1); pr.reserve((int)(LIM/log(LIM)*1.1)     );     vector&lt;pii&gt; cp; //083     for (int i = 3; i &lt;= S; i += 2) if (!sieve[i]) {         cp.push_back({i, i * i / 2});         for (int j = i * i; j &lt;= S; j += 2 * i) sieve[j] = 1;     }     for (int L = 1; L &lt;= R; L += S) { //62d         array&lt;bool, S&gt; block{};         for (auto &amp;[p, idx] : cp)             for (int i=idx; i &lt; S+L; idx = (i+=p)) block[i-L] = 1;         rep(i,0,min(S, R - L))             if (!block[i]) pr.push_back((L + i) * 2 + 1); //c68     }     for (int i : pr) isPrime[i] = 1;     return pr; }</pre>	6b2912, 20 lines
--	------------------

### MillerRabin.h

**Description:** Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to  $7 \cdot 10^{18}$ ; for larger numbers, use Python and extend A randomly.  
**Time:** 7 times the complexity of  $a^b \bmod c$ .

<pre>"ModMulLL.h" bool isPrime(ull n) { //60a     if (n &lt; 2    n % 6 % 4 != 1) return (n   1) == 3;     ull A[] = {2, 325, 9375, 28178, 450775, 9780504,         1795265022},         s = __builtin_ctzll(n-1), d = n &gt;&gt; s;     for (ull a : A) { // ^ count trailing zeroes         ull p = modpow(a%n, d, n), i = s; //81c         while (p != 1 &amp;&amp; p != n - 1 &amp;&amp; a % n &amp;&amp; i--)             p = modmul(p, p, n);         if (p != n-1 &amp;&amp; i != s) return 0;     }     return 1; //84a }</pre>	60dcd1, 12 lines
---	------------------

### Factor.h

**Description:** Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).  
**Time:**  $\mathcal{O}\left(n^{1/4}\right)$ , less for numbers with small factors.

<pre>"ModMulLL.h", "MillerRabin.h" ull pollard(ull n) { //47d     ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;     auto f = [&amp;](ull x) { return modmul(x, x, n) + i; };     while (t++ % 40    __gcd(prd, n) == 1) {         if (x == y) x = ++i, y = f(x);         if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;         //049         x = f(x), y = f(f(y));     } }</pre>	d8d98d, 18 lines
--	------------------

```
    return __gcd(prd, n);
}
vector<ull> factor(ull n) { //c19
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r)); //363
    return l;
}
}
```

## 5.3 Divisibility

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

<pre>ll euclid(ll a, ll b, ll &amp;x, ll &amp;y) { //33b     if (!b) return x = 1, y = 0, a;     ll d = euclid(b, a % b, y, x);     return y -= a/b * x, d; }</pre>	33ba8f, 5 lines
---	-----------------

### CRT.h

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

<pre>"euclid.h" ll crt(ll a, ll m, ll b, ll n) { //eae     if (n &gt; m) swap(a, b), swap(m, n);     ll x, y, g = euclid(m, n, x, y);     assert((a - b) % g == 0); // else no solution     x = (b - a) % n * x % n / g * m + a;     return x &lt; 0 ? x + m*n/g : x; //6ac }</pre>	04d93a, 7 lines
---	-----------------

#### 5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

### phiFunction.h

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

<pre>const int LIM = 5000000; //70b int phi[LIM];  void calculatePhi() {     rep(i,0,LIM) phi[i] = i&amp;1 ? i : i/2;     for (int i = 3; i &lt; LIM; i += 2) if (phi[i] == i) //103         for (int j = i; j &lt; LIM; j += i) phi[j] -= phi[j] / i; }</pre>	cf7d6d, 8 lines
--	-----------------

## 5.4 Fractions

### ContinuedFractions.h

**Description:** Given  $N$  and a real number  $x \geq 0$ , finds the closest rational approximation  $p/q$  with  $p, q \leq N$ . It will obey  $|p/q - x| \leq 1/qN$ . For consecutive convergents,  $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$ . ( $p_k/q_k$  alternates between  $> x$  and  $< x$ .) If  $x$  is rational,  $y$  eventually becomes  $\infty$ ; if  $x$  is the root of a degree 2 polynomial the  $a$ 's eventually become cyclic. **Time:**  $\mathcal{O}(\log N)$

```

                                     dd6c5e, 21 lines
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
//32b
pair<ll, ll> approximate(d x, ll N) {
    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x
    ;
    for (;;) {
        ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf
        ),
        a = (ll)floor(y), b = min(a, lim), //5ad
        NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If b > a/2, we have a semi-convergent that gives
            us a
            // better approximation; if b = a/2, we *may* have
            one.
            // Return {P, Q} here for a more canonical
            approximation. //8fe
            return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)
            ) ?
                make_pair(NP, NQ) : make_pair(P, Q);
        }
        if (abs(y = 1/(y - (d)a)) > 3*N) {
            return {NP, NQ}; //5c7
        }
        LP = P; P = NP;
        LQ = Q; Q = NQ;
    }
} //cbb
```

### FracBinarySearch.h

**Description:** Given  $f$  and  $N$ , finds the smallest fraction  $p/q \in [0, 1]$  such that  $f(p/q)$  is true, and  $p, q \leq N$ . You may want to throw an exception from  $f$  if it finds an exact solution, in which case  $N$  can be removed.

**Usage:** fracBS([f](Frac f) { return f.p>=3\*f.q; }, 10); // {1,3}  
**Time:**  $\mathcal{O}(\log(N))$

```

                                     27ab3e, 25 lines
struct Frac { ll p, q; }; //386
```

```

template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N
    //262
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >= si) { //7e2
            adv += step;
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                adv -= step; si = 2;
            } //bf0
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi); //f58
        A = B; B = !adv;
    }
}
```

## 5.5 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.

## 5.6 Primes

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

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

## 5.7 Estimates

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

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

## 5.8 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

# Combinatorial (6)

## 6.1 Permutations

### 6.1.1 Factorial

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

### IntPerm.h

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

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

```

                                     044568, 6 lines
int permToInt(vi& v) { //cf9
    int use = 0, i = 0, r = 0;
    for (int x:v) r = r * ++i + __builtin_popcount(use & -(1<<
        x)),
        use |= 1 << x; // (note: minus, not
        ~!)
    return r;
} //cbb
```

### 6.1.2 Cycles

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

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

### 6.1.3 Derangements

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

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

### 6.1.4 Burnside’s lemma

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

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

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

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

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

## 6.2 Partitions and subsets

### 6.2.1 Partition function

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

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

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

$n$	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	~2e5	~2e8

### 6.2.2 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  $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$ .

### 6.2.3 Binomials

multinomial.h

**Description:** Computes  $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n}=\frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$ .a0a312, 5 lines

```
ll multinomial(vi& v) { //a0a
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i]) c = c * ++m / (j+1);
  return c;
}
```

## 6.3 General purpose numbers

### 6.3.1 Bernoulli numbers

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

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

### 6.3.2 Stirling numbers of the first kind

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

$$\begin{aligned} c(n,k) &= c(n-1,k-1) + (n-1)c(n-1,k), \quad c(0,0) = 1 \\ \sum_{k=0}^n c(n,k)x^k &= x(x+1)\ldots(x+n-1) \end{aligned}$$

$$\begin{aligned} c(8,k) &= 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1 \\ c(n,2) &= 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots \end{aligned}$$

### 6.3.3 Eulerian numbers

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

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

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

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

### 6.3.4 Stirling numbers of the second kind

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

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

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

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

### 6.3.5 Bell numbers

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

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

### 6.3.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$   
# on  $k$  existing trees of size  $n_i$ :  $n_1n_2\cdots n_kn^{k-2}$   
# with degrees  $d_i$ :  $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

### 6.3.7 Catalan numbers

$$\begin{aligned} C_n &= \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!} \\ C_0 &= 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum C_i C_{n-i} \end{aligned}$$

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

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

## Graph (7)

### 7.1 Fundamentals

BellmanFord.h

**Description:** Calculates shortest paths from  $s$  in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes  $V^2 \max |w_i| < \sim 2^{63}$ .  
**Time:**  $\mathcal{O}(VE)$ 830a8f, 23 lines

```
const ll inf = LLONG_MAX; //019
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
  nodes[s].dist = 0; //3a0
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
  rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b]; //e21
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf); //69b
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf; //943
  }
}
```

}

FloydWarshall.h

**Description:** Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix  $m$ , where  $m[i][j] = \text{inf}$  if  $i$  and  $j$  are not adjacent. As output,  $m[i][j]$  is set to the shortest distance between  $i$  and  $j$ , inf if no path, or -inf if the path goes through a negative-weight cycle.  
**Time:**  $\mathcal{O}(N^3)$ 531245, 12 lines

```
const ll inf = 1LL << 62; //279
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) { //ef8
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf; //eec
}
```

TopoSort.h

**Description:** Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than  $n$  – nodes reachable from cycles will not be returned.  
**Time:**  $\mathcal{O}(|V|+|E|)$ d678d8, 8 lines

```
vi topoSort(const vector<vi>& gr) { //c7a
  vi indeg(sz(gr)), q;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push_back(i);
  rep(j,0,sz(q)) for (int x : gr[q[j]])
    if (--indeg[x] == 0) q.push_back(x); //28e
  return q;
}
```

## 7.2 Network flow

PushRelabel.h

**Description:** Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.  
**Time:**  $\mathcal{O}(V^2\sqrt{E})$ 0ae1d4, 48 lines

```
struct PushRelabel { //d82
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g; //bef
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
  //07d
  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  } //a02

  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f; //124
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
}
```

```
11 calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;//a96
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
        while (hs[hi].empty()) if (!hi--) return -ec[s];//e2e
        int u = hs[hi].back(); hs[hi].pop_back();
        while (ec[u] > 0) // discharge u
            if (cur[u] == g[u].data() + sz(g[u])) {
                H[u] = 1e9;
                for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]
                    ]+1)//9ff
                    H[u] = H[e.dest]+1, cur[u] = &e;
                if (++co[H[u]],!--co[hi] && hi < v)
                    rep(i,0,v) if (hi < H[i] && H[i] < v)
                        --co[H[i]], H[i] = v + 1;
                hi = H[u];//7ed
            } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
                addFlow(*cur[u], min(ec[u], cur[u]->c));
            else ++cur[u];
    }
} //a5b
bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h  
**Description:** Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.  
**Time:**  $\mathcal{O}(FE \log(V))$  where  $F$  is max flow.  $\mathcal{O}(VE)$  for setpi.

```
#include <bits/extc++.h>//2fb

const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
    struct edge { //219
        int from, to, rev;
        ll cap, cost, flow;
    };
    int N;
    vector<vector<edge>> ed; //252
    vi seen;
    vector<ll> dist, pi;
    vector<edge*> par;

    MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {} //98d

    void addEdge(int from, int to, ll cap, ll cost) {
        if (from == to) return;
        ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
        ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 }); //6ab
    }

    void path(int s) {
        fill(all(seen), 0);
        fill(all(dist), INF); //da3
        dist[s] = 0; ll di;

        __gnu_pbds::priority_queue<pair<ll, int>> q;
        vector<decltype(q)::point_iterator> its(N);
        q.push({ 0, s }); //aa9

        while (!q.empty()) {
            s = q.top().second; q.pop();
            seen[s] = 1; di = dist[s] + pi[s];
```

```
for (edge& e : ed[s]) if (!seen[e.to]) { //344
    ll val = di - pi[e.to] + e.cost;
    if (e.cap - e.flow > 0 && val < dist[e.to]) {
        dist[e.to] = val;
        par[e.to] = &e;
        if (its[e.to] == q.end()) //b01
            its[e.to] = q.push({ -dist[e.to], e.to });
        else
            q.modify(its[e.to], { -dist[e.to], e.to });
    }
} //f01
rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
}

pair<ll, ll> maxflow(int s, int t) { //10b
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (edge* x = par[t]; x; x = par[x->from])
            fl = min(fl, x->cap - x->flow); //64a

        totflow += fl;
        for (edge* x = par[t]; x; x = par[x->from]) {
            x->flow += fl;
            ed[x->to][x->rev].flow -= fl; //897
        }
    }
    rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
    return {totflow, totcost/2};
} //ca9

// If some costs can be negative, call this before maxflow:
void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v; //486
    while (ch-- && it--)
        rep(i,0,N) if (pi[i] != INF)
            for (edge& e : ed[i]) if (e.cap)
                if ((v = pi[i] + e.cost) < pi[e.to])
                    pi[e.to] = v, ch = 1; //222
    assert(it >= 0); // negative cost cycle
}
};
```

EdmondsKarp.h  
**Description:** Flow algorithm with guaranteed complexity  $\mathcal{O}(VE^2)$ . To get edge flow values, compare capacities before and after, and take the positive values only.

```
template<class T> T edmondsKarp(vector<unordered_map<int, T>>& //324
    graph, int source, int sink) {
    assert(source != sink);
    T flow = 0;
    vi par(sz(graph)), q = par;
    //cf9
    for (;) {
        fill(all(par), -1);
        par[source] = 0;
        int ptr = 1;
        q[0] = source; //623

        rep(i,0,ptr) {
            int x = q[i];
            for (auto e : graph[x]) {
                if (par[e.first] == -1 && e.second > 0) { //3a4
                    par[e.first] = x;
                    q[ptr++] = e.first;
```

```
if (e.first == sink) goto out;
        }
    } //3cd
}
return flow;
out:
T inc = numeric_limits<T>::max();
for (int y = sink; y != source; y = par[y]) //d19
    inc = min(inc, graph[par[y]][y]);

flow += inc;
for (int y = sink; y != source; y = par[y]) {
    int p = par[y]; //b79
    if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
    graph[y][p] += inc;
}
} //cbb
```

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

GlobalMinCut.h  
**Description:** Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.  
**Time:**  $\mathcal{O}(V^3)$

```
pair<int, vi> globalMinCut(vector<vi> mat) { //f64
    pair<int, vi> best = {INT_MAX, {}};
    int n = sz(mat);
    vector<vi> co(n);
    rep(i,0,n) co[i] = {i};
    rep(ph,1,n) { //c8f
        vi w = mat[0];
        size_t s = 0, t = 0;
        rep(it,0,n-ph) { //  $\mathcal{O}(V^2) \rightarrow \mathcal{O}(E \log V)$  with prio.
            queue
            w[t] = INT_MIN;
            s = t, t = max_element(all(w)) - w.begin(); //0bb
            rep(i,0,n) w[i] += mat[t][i];
        }
        best = min(best, {w[t] - mat[t][t], co[t]});
        co[s].insert(co[s].end(), all(co[t]));
        rep(i,0,n) mat[s][i] += mat[t][i]; //a2c
        rep(i,0,n) mat[i][s] = mat[s][i];
        mat[0][t] = INT_MIN;
    }
    return best;
} //cbb
```

GomoryHu.h  
**Description:** Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.  
**Time:**  $\mathcal{O}(V)$  Flow Computations

```
"PushRelabel.h"
typedef array<ll, 3> Edge; //34e
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works //3fd
        for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
        tree.push_back({i, par[i], D.calc(i, par[i])});
        rep(j,i+1,N)
```

```
        if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i
            ;
    } //eec
    return tree;
}
```

### 7.3 Matching

#### hopcroftKarp.h

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

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

**Time:**  $\mathcal{O}(\sqrt{VE})$

f612e4, 42 lines

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) { //d9e
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            //613
            return btoa[b] = a, 1;
    }
    return 0;
} //ad4
int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0); //db3
        fill(all(B), 0);
        cur.clear();
        for (int a : btoa) if (a != -1) A[a] = -1;
        rep(a, 0, sz(g)) if (A[a] == 0) cur.push_back(a);
        for (int lay = 1;; lay++) { //559
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay; //1ca
                    islast = 1;
                }
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                    next.push_back(btoa[b]); //1eb
                }
            }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay; //4f3
            cur.swap(next);
        }
        rep(a, 0, sz(g))
            res += dfs(a, 0, g, btoa, A, B);
    } //67c
}
```

#### DFSMatching.h

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

**Usage:** vi btoa(m, -1); dfsMatching(g, btoa);

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

522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) { //400
    if (btoa[j] == -1) return 1;
    vis[j] = 1; int di = btoa[j];
    for (int e : g[di])
        if (!vis[e] && find(e, g, btoa, vis)) {
            btoa[e] = di; //a0e
            return 1;
        }
    return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) { //52f
    vi vis;
    rep(i, 0, sz(g)) {
        vis.assign(sz(btoa), 0);
        for (int j : g[i])
            if (find(j, g, btoa, vis)) { //e5b
                btoa[j] = i;
                break;
            }
    }
    return sz(btoa) - (int)count(all(btoa), -1); //ff5
}
```

#### MinimumVertexCover.h

**Description:** Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h" da4196, 20 lines

```
vi cover(vector<vi>& g, int n, int m) { //60f
    vi match(m, -1);
    int res = dfsMatching(g, match);
    vector<bool> lfound(n, true), seen(m);
    for (int it : match) if (it != -1) lfound[it] = false;
    vi q, cover; //0db
    rep(i, 0, n) if (lfound[i]) q.push_back(i);
    while (!q.empty()) {
        int i = q.back(); q.pop_back();
        lfound[i] = 1;
        for (int e : g[i]) if (!seen[e] && match[e] != -1) { //dc5
            seen[e] = true;
            q.push_back(match[e]);
        }
    }
    rep(i, 0, n) if (!lfound[i]) cover.push_back(i); //849
    rep(i, 0, m) if (seen[i]) cover.push_back(n+i);
    assert(sz(cover) == res);
    return cover;
}
```

#### WeightedMatching.h

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

**Time:**  $\mathcal{O}(N^2M)$

1e0fe9, 31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) { //64f
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i, 1, n) {
        p[0] = i; //0b5
        int j0 = 0; // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true; //14f
            int i0 = p[j0], j1, delta = INT_MAX;
```

```
        rep(j, 1, m) if (!done[j]) {
            auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
            if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
            if (dist[j] < delta) delta = dist[j], j1 = j; //865
        }
        rep(j, 0, m) {
            if (done[j]) u[p[j]] += delta, v[j] -= delta;
            else dist[j] -= delta;
        } //aa1
        j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
        int j1 = pre[j0];
        p[j0] = p[j1], j0 = j1; //88f
    }
}
rep(j, 1, m) if (p[j]) ans[p[j] - 1] = j - 1;
return {-v[0], ans}; // min cost
} //cbb
```

#### GeneralMatching.h

**Description:** Matching for general graphs. Fails with probability  $N/\text{mod}$ .

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

"./numerical/MatrixInverse-mod.h" cb1912, 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) { //19e
    vector<vector<ll>> mat(N, vector<ll>(N)), A;
    for (pii pa : ed) {
        int a = pa.first, b = pa.second, r = rand() % mod;
        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
    } //063

    int r = matInv(A = mat), M = 2*N - r, fi, fj;
    assert(r % 2 == 0);

    if (M != N) do { //f88
        mat.resize(M, vector<ll>(M));
        rep(i, 0, N) {
            mat[i].resize(M);
            rep(j, N, M) {
                int r = rand() % mod; //338
                mat[i][j] = r, mat[j][i] = (mod - r) % mod;
            }
        }
    } while (matInv(A = mat) != M);
    //92b
    vi has(M, 1); vector<pii> ret;
    rep(it, 0, M/2) {
        rep(i, 0, M) if (has[i])
            rep(j, i+1, M) if (A[i][j] && mat[i][j]) {
                fi = i; fj = j; goto done; //e0a
            }
        assert(0); done;
        if (fj < N) ret.emplace_back(fi, fj);
        has[fi] = has[fj] = 0;
        rep(sw, 0, 2) {
            ll a = modpow(A[fi][fj], mod-2); //b7f
            rep(i, 0, M) if (has[i] && A[i][fj]) {
                ll b = A[i][fj] * a % mod;
                rep(j, 0, M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
            }
            swap(fi, fj); //3c7
        }
    }
    return ret;
}
```

## 7.4 DFS algorithms

### SCC.h

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

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

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

```
vi val, comp, z, cont; //ed2
int Time, ncmps;
template<class G, class F> int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for (auto e : g[j]) if (comp[e] < 0)
        low = min(low, val[e] ?: dfs(e, g, f)); //b9e

    if (low == val[j]) {
        do {
            x = z.back(); z.pop_back();
            comp[x] = ncmps; //f1f
            cont.push_back(x);
        } while (x != j);
        f(cont); cont.clear();
        ncmps++;
    } //658
    return val[j] = low;
}

template<class G, class F> void scc(G& g, F f) {
    int n = sz(g);
    val.assign(n, 0); comp.assign(n, -1); //5bc
    Time = ncmps = 0;
    rep(i, 0, n) if (comp[i] < 0) dfs(i, g, f);
}
```

76b5c9, 24 lines

### BiconnectedComponents.h

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

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

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

```
vi num, st; //3e8
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, top = me; //112
    for (auto [y, e] : ed[at]) if (e != par) {
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e); //c2b
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);
            if (up == me) //c92
                st.push_back(e);
            f(vi(st.begin() + si, st.end()));
            st.resize(si);
        }
    }
}
```

c6b7c7, 32 lines

```
    }
    else if (up < me) st.push_back(e); //1a1
    else { /* e is a bridge */ }
}
}
return top;
} //85e

template<class F>
void bicomps(F f) {
    num.assign(sz(ed), 0);
    rep(i, 0, sz(ed)) if (!num[i]) dfs(i, -1, f); //888
}
```

### 2sat.h

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

**Usage:** `TwoSat ts(number of boolean variables);`  
`ts.either(0, ~3);` // Var 0 is true or var 3 is false  
`ts.setValue(2);` // Var 2 is true  
`ts.atMostOne({0, ~1, 2});` //  $\leq 1$  of vars 0, ~1 and 2 are true  
`ts.solve();` // Returns true iff it is solvable  
`ts.values[0..N-1]` holds the assigned values to the vars  
**Time:**  $\mathcal{O}(N + E)$ , where  $N$  is the number of boolean variables, and  $E$  is the number of clauses.

5f9706, 56 lines

```
struct TwoSat { //7c0
    int N;
    vector<vi> gr;
    vi values; // 0 = false, 1 = true

    TwoSat(int n = 0) : N(n), gr(2*n) {} //54e

    int addVar() { // (optional)
        gr.emplace_back();
        gr.emplace_back();
        return N++; //662
    }

    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j); //3b0
        gr[f].push_back(j^1);
        gr[j].push_back(f^1);
    }

    void setValue(int x) { either(x, x); }
} //41c

void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i, 2, sz(li)) {
        int next = addVar(); //f5e
        either(cur, ~li[i]);
        either(cur, next);
        either(~li[i], next);
        cur = ~next;
    } //276
    either(cur, ~li[1]);
}
```

```
vi val, comp, z; int time = 0;
int dfs(int i) { //7e3
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
        low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back(); z.pop_back(); //0c0
        comp[x] = low;
    }
}
```

```
    if (values[x>>1] == -1)
        values[x>>1] = x&1;
} while (x != i);
return val[i] = low; //749
}

bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val; //4fa
    rep(i, 0, 2*N) if (!comp[i]) dfs(i);
    rep(i, 0, N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
}
} //214
```

### EulerWalk.h

**Description:** Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with `src` at both start and end, or empty list if no cycle/path exists. To get edge indices back, add `.second` to `s` and `ret`.

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

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src
=0) { //fda
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        //e35
        if (it == end) { ret.push_back(x); s.pop_back();
            continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.push_back(y); //8f2
        }
    }
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return
    {};
    return {ret.rbegin(), ret.rend()};
}
```

## 7.5 Coloring

### EdgeColoring.h

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

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

e210e2, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) { //d26
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) { //945
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1) //665
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = c; at != -1; cd ^= c ^ d, at = adj[at][cd
        ])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) { //e70
            int left = fan[i], right = fan[++i], e = cc[i];
        }
    }
}
```

```
adj[u][e] = left;
adj[left][e] = u;
adj[right][e] = -1;
free[right] = e;//75c
}
adj[u][d] = fan[i];
adj[fan[i]][d] = u;
for (int y : {fan[0], u, end})
    for (int& z = free[y] = 0; adj[y][z] != -1; z++);//
        b06
}
rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i]
        ;
return ret;
};//cbb
```

## 7.6 Heuristics

### MaximalCliques.h

**Description:** Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.

**Time:**  $\mathcal{O}\left(3^{n/3}\right)$ , much faster for sparse graphs

```
typedef bitset<128> B;//abb
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={
    }) {
    if (!P.any()) { if (!X.any()) f(R); return; }
    auto q = (P | X)._Find_first();
    auto cands = P & ~eds[q];//7d8
    rep(i,0,sz(eds)) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    };//67c
}
```

### MaximumClique.h

**Description:** Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

**Time:** Runs in about 1s for n=155 and worst case random graphs (p=90). Runs faster for sparse graphs.

```
typedef vector<bitset<200>> vb;//b92
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;//5b2
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;//dab
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
        rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
    };//aba
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (sz(R)) {
            if (sz(q) + R.back().d <= sz(qmax)) return;//6b0
            q.push_back(R.back().i);
            vv T;
            for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
        }
```

```
if (sz(T)) {
    if (S[lev]++ / ++pk < limit) init(T);//feb
    int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
    C[1].clear(), C[2].clear();
    for (auto v : T) {
        int k = 1;
        auto f = [&](int i) { return e[v.i][i]; };//547
        while (any_of(all(C[k]), f)) k++;
        if (k > mxk) mxk = k, C[mxk + 1].clear();
        if (k < mnk) T[j++].i = v.i;
        C[k].push_back(v.i);
    };//08b
    if (j > 0) T[j - 1].d = 0;
    rep(k,mnk,mxk + 1) for (int i : C[k])
        T[j].i = i, T[j++].d = k;
    expand(T, lev + 1);
    } else if (sz(q) > sz(qmax)) qmax = q;//15f
    q.pop_back(), R.pop_back();
    }
}
vi maxClique() { init(V), expand(V); return qmax; }
Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S)
    ) {//02b
    rep(i,0,sz(e)) V.push_back({i});
}
};
```

### MaximumIndependentSet.h

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

//d41

## 7.7 Trees

### BinaryLifting.h

**Description:** Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.

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

```
vector<vi> treeJump(vi& P){//bcb
    int on = 1, d = 1;
    while(on < sz(P)) on *= 2, d++;
    vector<vi> jmp(d, P);
    rep(i,1,d) rep(j,0,sz(P))
        jmp[i][j] = jmp[i-1][jmp[i-1][j]];//47a
    return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
    rep(i,0,sz(tbl))//66f
        if(steps&(1<<i)) nod = tbl[i][nod];
    return nod;
}
```

```
int lca(vector<vi>& tbl, vi& depth, int a, int b) {//57b
    if (depth[a] < depth[b]) swap(a, b);
    a = jmp(tbl, a, depth[a] - depth[b]);
    if (a == b) return a;
    for (int i = sz(tbl); i--;) {
        int c = tbl[i][a], d = tbl[i][b];//30e
        if (c != d) a = c, b = d;
    }
    return tbl[0][a];
}
```

### LCA.h

**Description:** Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.

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

```
"../data-structures/RMQ.h"
0f62fb, 21 lines

struct LCA {//169
    int T = 0;
    vi time, path, ret;
    RMQ<int> rmq;

    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret))
        {}//e10
    void dfs(vector<vi>& C, int v, int par) {
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.push_back(v), ret.push_back(time[v]);
            dfs(C, y, v);//3f8
        }
    }

    int lca(int a, int b) {
        if (a == b) return a;//3f5
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)];
    }

    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)
        ];}
};//214
```

### CompressTree.h

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

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

```
"LCA.h"
9775a0, 21 lines

typedef vector<pair<int, int>> vpi;//386
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);//a92
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }//c76
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {//ff8
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
};//cbb
```

### HLD.h

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

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

```
"../data-structures/LazySegmentTree.h"
03139d, 46 lines

template <bool VALS_EDGES> struct HLD {//431
    int N, tim = 0;
    vector<vi> adj;
    vi par, siz, rt, pos;
    Node *tree;
```

```

HLD(vector<vi> adj_)//567
: N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
rt(N), pos(N), tree(new Node(0, N)) { dfsSz(0); dfsHld
(0); }
void dfsSz(int v) {
if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]
));
for (int& u : adj[v]) { //8da
par[u] = v;
dfsSz(u);
siz[v] += siz[u];
if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
} //09d
}
void dfsHld(int v) {
pos[v] = tim++;
for (int u : adj[v]) {
rt[u] = (u == adj[v][0] ? rt[v] : u); //0b4
dfsHld(u);
}
}
template <class B> void process(int u, int v, B op) {
for (; rt[u] != rt[v]; v = par[rt[v]]) { //0db
if (pos[rt[u]] > pos[rt[v]]) swap(u, v);
op(pos[rt[v]], pos[v] + 1);
}
if (pos[u] > pos[v]) swap(u, v);
op(pos[u] + VALS_EDGES, pos[v] + 1); //31c
}
void modifyPath(int u, int v, int val) {
process(u, v, [&](int l, int r) { tree->add(l, r, val);
});
}
int queryPath(int u, int v) { // Modify depending on
problem//ad4
int res = -1e9;
process(u, v, [&](int l, int r) {
res = max(res, tree->query(l, r));
});
return res; //4b8
}
int querySubtree(int v) { // modifySubtree is similar
return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]
);
}
} //214

```

## LinkCutTree.h

**Description:** Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

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

0fb462, 90 lines

```

struct Node { // Splay tree. Root's pp contains tree's
parent. //4e
Node *p = 0, *pp = 0, *c[2];
bool flip = 0;
Node() { c[0] = c[1] = 0; fix(); }
void fix() {
if (c[0]) c[0]->p = this; //b8f
if (c[1]) c[1]->p = this;
// (+ update sum of subtree elements etc. if wanted)
}
void pushFlip() {
if (!flip) return; //dfd
flip = 0; swap(c[0], c[1]);
if (c[0]) c[0]->flip ^= 1;
if (c[1]) c[1]->flip ^= 1;
}
int up() { return p ? p->c[1] == this : -1; } //3a9
void rot(int i, int b) {

```

```

int h = i ^ b;
Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y :
x;
if ((y->p = p)) p->c[up()] = y;
c[i] = z->c[i ^ 1]; //eb7
if (b < 2) {
x->c[h] = y->c[h ^ 1];
y->c[h ^ 1] = x;
}
z->c[i ^ 1] = this; //430
fix(); x->fix(); y->fix();
if (p) p->fix();
swap(pp, y->pp);
}
void splay() { //4c8
for (pushFlip(); p; ) {
if (p->p) p->p->pushFlip();
p->pushFlip(); pushFlip();
int c1 = up(), c2 = p->up();
if (c2 == -1) p->rot(c1, 2); //9e8
else p->p->rot(c2, c1 != c2);
}
}
Node* first() {
pushFlip(); //828
return c[0] ? c[0]->first() : (splay(), this);
}
}
struct LinkCut { //d99
vector<Node> node;
LinkCut(int N) : node(N) {}

void link(int u, int v) { // add an edge (u, v)
assert(!connected(u, v)); //166
makeRoot(&node[u]);
node[u].pp = &node[v];
}
void cut(int u, int v) { // remove an edge (u, v)
Node *x = &node[u], *top = &node[v]; //0b9
makeRoot(top); x->splay();
assert(top == (x->pp ? x->c[0]));
if (x->pp) x->pp = 0;
else {
x->c[0] = top->p = 0; //158
x->fix();
}
}
bool connected(int u, int v) { // are u, v in the same
tree?
Node* nu = access(&node[u])->first(); //781
return nu == access(&node[v])->first();
}
void makeRoot(Node* u) {
access(u);
u->splay(); //09d
if (u->c[0]) {
u->c[0]->p = 0;
u->c[0]->flip ^= 1;
u->c[0]->pp = u;
u->c[0] = 0; //41e
u->fix();
}
}
Node* access(Node* u) {
u->splay(); //4e7
while (Node* pp = u->pp) {
pp->splay(); u->pp = 0;
if (pp->c[1]) {
pp->c[1]->p = 0; pp->c[1]->pp = pp; }
pp->c[1] = u; pp->fix(); u = pp; //f4d
}
}

```

```

}
return u;
}
};

DirectedMST.h
Description: Finds a minimum spanning tree/arborescence of a di-
rected graph, given a root node. If no MST exists, returns -1.
Time:  $\mathcal{O}(E \log V)$ 
../data-structures/UnionFindRollback.h 39e620, 60 lines
struct Edge { int a, b; ll w; }; //59f
struct Node {
Edge key;
Node *l, *r;
ll delta;
void prop() { //936
key.w += delta;
if (l) l->delta += delta;
if (r) r->delta += delta;
delta = 0;
} //5dc
Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
if (!a || !b) return a ? : b;
a->prop(), b->prop(); //72a
if (a->key.w > b->key.w) swap(a, b);
swap(a->l, (a->r = merge(b, a->r)));
return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); } //8
e9

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
RollbackUF uf(n);
vector<Node*> heap(n);
for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e}
); //0f3
ll res = 0;
vi seen(n, -1), path(n, par(n);
seen[r] = r;
vector<Edge> Q(n), in(n, {-1, -1}), comp;
deque<tuple<int, int, vector<Edge>>> cyps; //4c6
rep(s, 0, n) {
int u = s, qi = 0, w;
while (seen[u] < 0) {
if (!heap[u]) return {-1, {}};
Edge e = heap[u]->top(); //2b0
heap[u]->delta -= e.w, pop(heap[u]);
Q[qi] = e, path[qi++] = u, seen[u] = s;
res += e.w, u = uf.find(e.a);
if (seen[u] == s) {
Node* cyc = 0; //fff
int end = qi, time = uf.time();
do cyc = merge(cyc, heap[w = path[--qi]]);
while (uf.join(u, w));
u = uf.find(u), heap[u] = cyc, seen[u] = -1;
cyps.push_front({u, time, {&Q[qi], &Q[end]}}); //984
}
}
rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
}
} //b55
for (auto& [u, t, comp] : cyps) { // restore sol (optional)
uf.rollback(t);
Edge inEdge = in[u];
for (auto& e : comp) in[uf.find(e.b)] = e;
in[uf.find(inEdge.b)] = inEdge; //ffd
}
rep(i, 0, n) par[i] = in[i].a;
return {res, par};
}

```



```
}
```

## 7.8 Math

### 7.8.1 Number of Spanning Trees

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

### 7.8.2 Erdős–Gallai theorem

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

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

## Geometry (8)

### 8.1 Geometric primitives

**Point.h**

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

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
//fa7
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}//4f8
    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); }//e11
    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; }//0c3
    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//9f3
    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) {}//25e
    return os << "(" << p.x << ", " << p.y << ")"; }
};
```

**lineDistance.h**

**Description:**

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

```
"Point.h"
f6bf6b, 4 lines
template<class P> //f6b
double lineDist(const P& a, const P& b, const P& p) {
    return (double) (b-a).cross(p-a) / (b-a).dist();
}
```

**SegmentDistance.h**

**Description:**

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

```
"Point.h"
5c88f4, 6 lines
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
typedef Point<double> P; //b95
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0, (p-s).dot(e-s)));
    return ((p-s)*d-(e-s)*t).dist()/d;
} //cbb
```

**SegmentIntersection.h**

**Description:**

If a unique intersection point between the line segments going from  $s_1$  to  $e_1$  and from  $s_2$  to  $e_2$  exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if  $P$  is `Point<ll>` and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

```
"Point.h", "OnSegment.h"
9d57f2, 13 lines
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
    cout << "segments intersect at " << inter[0] << endl;
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    //dec
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b * oa) / (ob - oa)}; //8a0
    set<P> s;
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d); //814
    return {all(s)};
```

```
}
```

**lineIntersection.h**

**Description:**

If a unique intersection point of the lines going through  $s_1, e_1$  and  $s_2, e_2$  exists  $\{1, \text{point}\}$  is returned. If no intersection point exists  $\{0, (0,0)\}$  is returned and if infinitely many exists  $\{-1, (0,0)\}$  is returned. The wrong position will be returned if  $P$  is `Point<ll>` and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.

```
"Point.h"
a01f81, 8 lines
Usage: auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
    cout << "intersection point at " << res.second << endl;
template<class P> //47e
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // if parallel
        return {-(s1.cross(e1, s2) == 0), P(0, 0)};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1); //16d
    return {1, (s1 * p + e1 * q) / d};
}
```

**sideOf.h**

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

```
"Point.h"
3af81c, 9 lines
Usage: bool left = sideOf(p1,p2,q)==1;
template<class P> //059
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps)
{
    auto a = (e-s).cross(p-s); //7c7
    double l = (e-s).dist()*eps;
    return (a > l) - (a < -l);
}
```

**OnSegment.h**

**Description:** Returns true iff  $p$  lies on the line segment from  $s$  to  $e$ . Use `(segDist(s,e,p)<=epsilon)` instead when using `Point<double>`.

```
"Point.h"
c597e8, 3 lines
template<class P> bool onSegment(P s, P e, P p) { //c59
    return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

**linearTransformation.h**

**Description:**

Apply the linear transformation (translation, rotation and scaling) which takes line  $p_0$ - $p_1$  to line  $q_0$ - $q_1$  to point  $r$ .

```
"Point.h"
03a306, 6 lines
typedef Point<double> P; //d52
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
    P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
```

```
    return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.
        dist2();
} //cbb
```

**Angle.h**  
**Description:** A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.  
**Usage:** vector<Angle> v = {w[0], w[0].t360() ...}; // sorted

```
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of
positively oriented triangles with vertices at 0 and i
0f602, 35 lines
```

```
struct Angle { //6c9
    int x, y;
    int t;
    Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
    Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}
        ; }
    int half() const { //a5b
        assert(x || y);
        return y < 0 || (y == 0 && x < 0);
    }
    Angle t90() const { return {-y, x, t + (half() && x >= 0)
        }; }
    Angle t180() const { return {-x, -y, t + half()}; } //de0
    Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (1l)b.x) < //41b
        make_tuple(b.t, b.half(), a.x * (1l)b.y);
}
```

```
// Given two points, this calculates the smallest angle
    between
// them, i.e., the angle that covers the defined line
    segment. //f86
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
} //b11
Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
} //073
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a
        )};
}
```

## 8.2 Circles

### CircleIntersection.h

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

```
"Point.h" 84d6d3, 11 lines
typedef Point<double> P; //deb
bool circleInter(P a,P b,double r1,double r2,pair<P, P>*
    out) {
    if (a == b) { assert(r1 != r2); return false; }
    P vec = b - a;
    double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*
            d2; //367
    if (sum*sum < d2 || dif*dif > d2) return false;
```

```
P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) /
    d2);
*out = {mid + per, mid - per};
return true;
} //cbb
```

### CircleTangents.h

**Description:** Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

```
"Point.h" b0153d, 13 lines
template<class P> //c18
vector<pair<P, P>> tangents(P c1, double r1, P c2, double
    r2) {
    P d = c2 - c1;
    double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) return {};
    vector<pair<P, P>> out; //446
    for (double sign : {-1, 1}) {
        P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    if (h2 == 0) out.pop_back(); //918
    return out;
}
```

### CirclePolygonIntersection.h

**Description:** Returns the area of the intersection of a circle with a ccw polygon.  
**Time:**  $\mathcal{O}(n)$

```
"../../content/geometry/Point.h" alee63, 19 lines
typedef Point<double> P; //a6c
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p; //eda
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.
            dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det
            ));
        if (t < 0 || 1 <= s) return arg(p, q) * r2; //174
        P u = p + d * s, v = p + d * t;
        return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
    };
    auto sum = 0.0;
    rep(i,0,sz(ps)) //a61
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

### circumcircle.h

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

```

"Point.h" 1caa3a, 9 lines
typedef Point<double> P; //032
```

```
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist() /
        abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) { //793
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

### MinimumEnclosingCircle.h

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

```
"circumcircle.h" 09dd0a, 17 lines
pair<P, double> mec(vector<P> ps) { //b50
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0; //d54
        rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;
            r = (o - ps[i]).dist();
            rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
                o = ccCenter(ps[i], ps[j], ps[k]); //4ec
                r = (o - ps[i]).dist();
            }
        }
    }
    return {o, r}; //2ac
}
```

## 8.3 Polygons

### InsidePolygon.h

**Description:** Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

**Usage:** vector<P> v = {P{4,4}, P{1,2}, P{2,1}};  
bool in = inPolygon(v, P{3, 3}, false);  
**Time:**  $\mathcal{O}(n)$

```
"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines
template<class P> //1c1
bool inPolygon(vector<P> &p, P a, bool strict = true) {
    int cnt = 0, n = sz(p);
    rep(i,0,n) {
        P q = p[(i + 1) % n];
        if (onSegment(p[i], q, a) return !strict; //fa7
        //or: if (segDist(p[i], q, a) <= eps) return !strict;
        cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) >
            0;
    }
    return cnt;
} //cbb
```

### PolygonArea.h

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

```
"Point.h" f12300, 6 lines
template<class T> //b19
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
} //cbb
```

### PolygonCenter.h

**Description:** Returns the center of mass for a polygon.


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

"Point.h"	9706dc, 9 lines
<pre>typedef Point&lt;double&gt; P; //082 P polygonCenter(const vector&lt;P&gt;&amp; v) {     P res(0, 0); double A = 0;     for (int i = 0, j = sz(v) - 1; i &lt; sz(v); j = i++) {         res = res + (v[i] + v[j]) * v[j].cross(v[i]);         A += v[j].cross(v[i]); //168     }     return res / A / 3; }</pre>	

### PolygonCut.h

**Description:**

Returns a vector with the vertices of a polygon with every-thing to the left of the line going from s to e cut away.




"Point.h", "lineIntersection.h"	f2b7d4, 13 lines
<pre>Usage: vector&lt;P&gt; p = ...; p = polygonCut(P, P(0,0), P(1,0));  typedef Point&lt;double&gt; P; //366 vector&lt;P&gt; polygonCut(const vector&lt;P&gt;&amp; poly, P s, P e) {     vector&lt;P&gt; res;     rep(i,0,sz(poly)) {         P cur = poly[i], prev = i ? poly[i-1] : poly.back();         bool side = s.cross(e, cur) &lt; 0; //44d         if (side != (s.cross(e, prev) &lt; 0))             res.push_back(lineInter(s, e, cur, prev).second);         if (side)             res.push_back(cur);     } //0e1     return res; }</pre>	

### 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.



"Point.h"	310954, 13 lines
<pre>typedef Point&lt;ll&gt; P; //3e3 vector&lt;P&gt; convexHull(vector&lt;P&gt; pts) {     if (sz(pts) &lt;= 1) return pts;     sort(all(pts));     vector&lt;P&gt; h(sz(pts)+1);     int s = 0, t = 0; //f18     for (int it = 2; it--; s = --t, reverse(all(pts)))         for (P p : pts) {             while (t &gt;= s + 2 &amp;&amp; h[t-2].cross(h[t-1], p) &lt;= 0) t--;             h[t++] = p;         } //aa0     return {h.begin(), h.begin() + t - (t == 2 &amp;&amp; h[0] == h[1])}; }</pre>	

### HullDiameter.h

**Description:** Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

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

"Point.h"	c571b8, 12 lines
<pre>typedef Point&lt;ll&gt; P; //5c7 array&lt;P, 2&gt; hullDiameter(vector&lt;P&gt; S) {</pre>	

<pre>int n = sz(S), j = n &lt; 2 ? 0 : 1; pair&lt;ll, array&lt;P, 2&gt;&gt; res({0, {S[0], S[0]}}); rep(i,0,j)     for (; j = (j + 1) % n) { //56c         res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}})         ;         if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) &gt;= 0)             break;     }     return res.second; //52a }</pre>	
--	--

### PointInsideHull.h

**Description:** Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

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

"Point.h", "sideOf.h", "OnSegment.h"	71446b, 14 lines
<pre>typedef Point&lt;ll&gt; P; //7a3  bool inHull(const vector&lt;P&gt;&amp; l, P p, bool strict = true) {     int a = 1, b = sz(l) - 1, r = !strict;     if (sz(l) &lt; 3) return r &amp;&amp; onSegment(l[0], l.back(), p);     if (sideOf(l[0], l[a], l[b]) &gt; 0) swap(a, b); //4a6     if (sideOf(l[0], l[a], p) &gt;= r    sideOf(l[0], l[b], p) &lt;= -r)         return false;     while (abs(a - b) &gt; 1) {         int c = (a + b) / 2;         (sideOf(l[0], l[c], p) &gt; 0 ? b : a) = c; //0da     }     return sgn(l[a].cross(l[b], p)) &lt; r; }</pre>	

### LineHullIntersection.h

**Description:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon:  $\bullet (-1, -1)$  if no collision,  $\bullet (i, -1)$  if touching the corner  $i$ ,  $\bullet (i, i)$  if along side  $(i, i + 1)$ ,  $\bullet (i, j)$  if crossing sides  $(i, i + 1)$  and  $(j, j + 1)$ . In the last case, if a corner  $i$  is crossed, this is treated as happening on side  $(i, i + 1)$ . The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.

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

"Point.h"	7cf45b, 39 lines
<pre>#define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n])) //b9d #define extr(i) cmp(i + 1, i) &gt;= 0 &amp;&amp; cmp(i, i - 1 + n) &lt; 0 template &lt;class P&gt; int extrVertex(vector&lt;P&gt;&amp; poly, P dir) {     int n = sz(poly), lo = 0, hi = n;     if (extr(0)) return 0;     while (lo + 1 &lt; hi) { //51a         int m = (lo + hi) / 2;         if (extr(m)) return m;         int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);         (ls &lt; ms    (ls == ms &amp;&amp; ls == cmp(lo, m)) ? hi : lo) = m;     } //e8c     return lo; }  #define cmpL(i) sgn(a.cross(poly[i], b)) template &lt;class P&gt; //7fd array&lt;int, 2&gt; lineHull(P a, P b, vector&lt;P&gt;&amp; poly) {     int endA = extrVertex(poly, (a - b).perp());     int endB = extrVertex(poly, (b - a).perp());     if (cmpL(endA) &lt; 0    cmpL(endB) &gt; 0)         return {-1, -1}; //04b     array&lt;int, 2&gt; res;</pre>	

<pre>rep(i,0,2) {     int lo = endB, hi = endA, n = sz(poly);     while ((lo + 1) % n != hi) {         int m = (lo + hi + (lo &lt; hi ? 0 : n)) / 2; //ec0         (cmpL(m) == cmpL(endB) ? lo : hi) = m;     }     res[i] = (lo + !cmpL(hi)) % n;     swap(endA, endB); } //6ab if (res[0] == res[1]) return {res[0], -1}; if (!cmpL(res[0]) &amp;&amp; !cmpL(res[1]))     switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {         case 0: return {res[0], res[0]};         case 2: return {res[1], res[1]}; //08a     }     return res; }</pre>	
--	--

## 8.4 Misc. Point Set Problems

### ClosestPair.h

**Description:** Finds the closest pair of points.

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

"Point.h"	ac41a6, 17 lines
<pre>typedef Point&lt;ll&gt; P; //9e7 pair&lt;P, P&gt; closest(vector&lt;P&gt; v) {     assert(sz(v) &gt; 1);     set&lt;P&gt; S;     sort(all(v), [](P a, P b) { return a.y &lt; b.y; });     pair&lt;ll, pair&lt;P, P&gt;&gt; ret{LLONG_MAX, {P(), P()}}; //e83     int j = 0;     for (P p : v) {         P d(1 + (ll)sqrt(ret.first), 0);         while (v[j].y &lt;= p.y - d.x) S.erase(v[j++]);         auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d); //cb2         for (; lo != hi; ++lo)             ret = min(ret, {(lo - p).dist2(), {lo, p}});         S.insert(p);     }     return ret.second; //982 }</pre>	

### kdTree.h

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

"Point.h"	bac5b0, 63 lines
<pre>typedef long long T; //632 typedef Point&lt;T&gt; P; const T INF = numeric_limits&lt;T&gt;::max();  bool on_x(const P&amp; a, const P&amp; b) { return a.x &lt; b.x; } bool on_y(const P&amp; a, const P&amp; b) { return a.y &lt; b.y; } //c56  struct Node {     P pt; // if this is a leaf, the single point in it     T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds     Node *first = 0, *second = 0; //5b4      T distance(const P&amp; p) { // min squared distance to a point         T x = (p.x &lt; x0 ? x0 : p.x &gt; x1 ? x1 : p.x);         T y = (p.y &lt; y0 ? y0 : p.y &gt; y1 ? y1 : p.y);         return (P(x,y) - p).dist2(); //a82     }      Node(vector&lt;P&gt;&amp;&amp; vp) : pt(vp[0]) {         for (P p : vp) {             x0 = min(x0, p.x); x1 = max(x1, p.x); //151             y0 = min(y0, p.y); y1 = max(y1, p.y);         }     } }</pre>	

```
    if (vp.size() > 1) {
        // split on x if width >= height (not ideal...)
        sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y); //1d2
        // divide by taking half the array for each child (
        not
        // best performance with many duplicates in the
        middle)
        int half = sz(vp)/2;
        first = new Node({vp.begin(), vp.begin() + half});
        second = new Node({vp.begin() + half, vp.end()}); //ace
    }
};

struct KDTree { //72b
    Node* root;
    KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {
    }

    pair<T, P> search(Node *node, const P& p) {
        if (!node->first) { //119
            // uncomment if we should not find the point itself:
            // if (p == node->pt) return (INF, P());
            return make_pair(p - node->pt).dist2(), node->pt);
        }
        //a89
        Node *f = node->first, *s = node->second;
        T bfirst = f->distance(p), bsec = s->distance(p);
        if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

        // search closest side first, other side if needed //bfa
        auto best = search(f, p);
        if (bsec < best.first)
            best = min(best, search(s, p));
        return best;
    } //13a

    // find nearest point to a point, and its squared
    distance
    // (requires an arbitrary operator< for Point)
    pair<T, P> nearest(const P& p) {
        return search(root, p); //213
    }
};
```

### FastDelaunay.h

**Description:** Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ... }, all counter-clockwise. **Time:**  $O(n \log n)$

"Point.h"	cefdff5, 88 lines
-----------	-------------------

```
typedef Point<ll> P; //503
typedef struct Quad* Q;
typedef __int128_t ll1; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point

struct Quad { //8bb
    Q rot, o; P p = arb; bool mark;
    P& F() { return r()->p; }
    Q& r() { return rot->rot; }
    Q prev() { return rot->o->rot; }
    Q next() { return r()->prev(); } //0bd
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle
    ?
    ll1 p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2; //520
```

### FastDelaunay PolyhedronVolume Point3D 3dHull

```
    return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B >
        0;
}

Q makeEdge(P orig, P dest) {
    Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
    H = r->o; r->r()->r() = r; //60f
    rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->
        r();
    r->p = orig; r->F() = dest;
    return r;
}

void splice(Q a, Q b) { //5b1
    swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}

Q connect(Q a, Q b) {
    Q q = makeEdge(a->F(), b->p);
    splice(q, a->next()); //3cc
    splice(q->r(), b);
    return q;
}

pair<Q,Q> rec(const vector<P>& s) { //a03
    if (sz(s) <= 3) {
        Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back()
            );
        if (sz(s) == 2) return { a, a->r() };
        splice(a->r(), b);
        auto side = s[0].cross(s[1], s[2]); //d54
        Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
    }

    #define H(e) e->F(), e->p //f35
    #define valid(e) (e->F().cross(H(base)) > 0)
    Q A, B, ra, rb;
    int half = sz(s) / 2;
    tie(ra, A) = rec({all(s) - half});
    tie(B, rb) = rec({sz(s) - half + all(s)}); //c17
    while ((B->p.cross(H(A)) < 0 && (A = A->next()) ||
        (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
    Q base = connect(B->r(), A);
    if (A->p == ra->p) ra = base->r();
    if (B->p == rb->p) rb = base; //a99

    #define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
        while (circ(e->dir->F(), H(base), e->F())) { \
            Q t = e->dir; \
            splice(e, e->prev()); //475
            splice(e->r(), e->r()->prev()); \
            e->o = H; H = e; e = t; \
        }
    for (;) {
        DEL(LC, base->r(), o); DEL(RC, base, prev()); //031
        if (!valid(LC) && !valid(RC)) break;
        if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
            base = connect(RC, base->r());
        else
            base = connect(base->r(), LC->r()); //907
    }
    return { ra, rb };
}
```

```
vector<P> triangulate(vector<P> pts) { //e5d
    sort(all(pts)); assert(unique(all(pts)) == pts.end());
    if (sz(pts) < 2) return {};
    Q e = rec(pts).first;
    vector<Q> q = {e};
    int qi = 0; //02b
    while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
    #define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p
        ); \
```

```
q.push_back(c->r()); c = c->next(); } while (c != e); }
ADD; pts.clear();
while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD; //24a
return pts;
}
```

### 8.5 3D

#### PolyhedronVolume.h

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

template<class V, class L> //27c	3058c3, 6 lines
----------------------------------	-----------------

```
double signedPolyVolume(const V& p, const L& trilst) {
    double v = 0;
    for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.
        c]);
    return v / 6;
} //cbb
```

#### Point3D.h

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

template<class T> <b>struct</b> Point3D { //c7b	8058ae, 32 lines
---	------------------

```
typedef Point3D P;
typedef const P& R;
T x, y, z;
explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z)
{
bool operator<(R p) const { //5e8
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
} //9b1
P operator*(T d) const { return P(x*d, y*d, z*d); }
P operator/(T d) const { return P(x/d, y/d, z/d); }
T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
    //58a
}
T dist2() const { return x*x + y*y + z*z; }
double dist() const { return sqrt((double)dist2()); }
//Azimuthal angle (longitude) to x-axis in interval [-pi,
    pi]
double phi() const { return atan2(y, x); } //a2c
//Zenith angle (latitude) to the z-axis in interval [0,
    pi]
double theta() const { return atan2(sqrt(x*x+y*y),z); }
P unit() const { return *this/(T)dist(); } //makes dist()
    =1
//returns unit vector normal to *this and p
P normal(P p) const { return cross(p).unit(); } //e88
//returns point rotated 'angle' radians ccw around axis
P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit
        ();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
} //e03
};
```

#### 3dHull.h

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

**Time:**  $O(n^2)$

"Point3D.h"	5b45fc, 49 lines
-------------	------------------

```
typedef Point3D<double> P3; //e28
```

```
struct PR {
    void ins(int x) { (a == -1 ? a : b) = x; }
    void rem(int x) { (a == x ? a : b) = -1; }
    int cnt() { return (a != -1) + (b != -1); }//c34
    int a, b;
};

struct F { P3 q; int a, b, c; };
//36b
vector<F> hull3d(const vector<P3>& A) {
    assert(sz(A) >= 4);
    vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
    #define E(x,y) E[f.x][f.y]
    vector<F> FS;//de0
    auto mf = [&](int i, int j, int k, int l) {
        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i]))
            q = q * -1;
        F f{q, i, j, k};//923
        E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
        FS.push_back(f);
    };
    rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
        mf(i, j, k, 6 - i - j - k);//e21

    rep(i,4,sz(A)) {
        rep(j,0,sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) //b63
                E(a,b).rem(f.c);
                E(a,c).rem(f.b);
                E(b,c).rem(f.a);
                swap(FS[j--], FS.back());
                FS.pop_back();//0df
        }
        int nw = sz(FS);
        rep(j,0,nw) {
            F f = FS[j];//945
            #define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f
                .c);
                C(a, b, c); C(a, c, b); C(b, c, a);
        }
        for (F& it : FS) if ((A[it.b] - A[it.a]).cross(//ab3
            A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
        return FS;
    };
};
```

### sphericalDistance.h

**Description:** Returns the shortest distance on the sphere with radius  $r$  between the points with azimuthal angles (longitude)  $\phi_1$  ( $\phi_1$ ) and  $\phi_2$  ( $\phi_2$ ) from  $x$  axis and zenith angles (latitude)  $\theta_1$  ( $\theta_1$ ) and  $\theta_2$  ( $\theta_2$ ) from  $z$  axis ( $0 = \text{north pole}$ ). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows.  $dx \cdot \text{radius}$  is then the difference between the two points in the  $x$  direction and  $d \cdot \text{radius}$  is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,//6da
    double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
    double dz = cos(t2) - cos(t1);
    double d = sqrt(dx*dx + dy*dy + dz*dz);//65e
    return radius*2*asin(d/2);
}
```

## Strings (9)

### KMP.h

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

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

d4375c, 16 lines

```
vi pi(const string& s) //f6d
    vi p(sz(s));
    rep(i,1,sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]);//off
    }
    return p;
}
```

```
vi match(const string& s, const string& pat) //752
    vi p = pi(pat + '\0' + s), res;
    rep(i,sz(p)-sz(s),sz(p))
        if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
    return res;
}//cbb
```

### Zfunc.h

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

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

ee09e2, 12 lines

```
vi Z(const string& S) //fc3
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - l]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])//8ec
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;//939
}
```

### Manacher.h

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

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

e7ad79, 13 lines

```
array<vi, 2> manacher(const string& s) //510
    int n = sz(s);
    array<vi,2> p = {vi(n+1), vi(n)};
    rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
        int t = r-i!z;
        if (i<r) p[z][i] = min(t, p[z][l+t]);//f50
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }//291
    return p;
}
```

### MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.

**Usage:** rotate(v.begin(), v.begin()+minRotation(v), v.end());

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

d07a42, 8 lines

```
int minRotation(string s) //20f
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
            break;}
    }
```

```
        if (s[a+k] > s[b+k]) { a = b; break; }
    }//3a8
    return a;
}
```

### SuffixArray.h

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

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

bc716b, 22 lines

```
struct SuffixArray //7a7
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)), y(n), ws(max(n, lim));
        x.push_back(0), sa = lcp = y, iota(all(sa), 0);//7c9
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;//f08
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[i]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] = y[a] == y[b] && y[a + j] == y[b + j] ? p - 1 : p
                ++;//726
        }
        for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
            for (k && k--, j = sa[x[i] - 1];
                s[i + k] == s[j + k]; k++);
    }//e03
};
```

### SuffixTree.h

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

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

aae0b8, 50 lines

```
struct SuffixTree //b1f
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;
//b11
    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m++]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];//99f
        }
        if (q==-1 || c==toi(a[q])) q++; else {
            l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
            p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
            l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;//604
            v=s[p[m]]; q=l[m];
            while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }
            if (q==r[m]) s[m]=v; else s[m]=m+2;
            q=r[v]-(q-r[m]); m+=2; goto suff;
        }//478
    }
```

```
SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);//f11
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] =
        0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
} //d1a

// example: find longest common substring (uses ALPHA =
// 28)
pii best;
int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;//636
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) :
        0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
        mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3) //a3a
        best = max(best, {len, r[node] - len});
    return mask;
}
static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2)
    ); //78c
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
};
```

Hashing.h

Description: Self-explanatory methods for string hashing

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// d41
// code, but works on evil test data (e.g. Thue-Morse,
// where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^
// 64).
// "typedef ull H;" instead if you think test data is
// random,
// or work mod 10^9+7 if the Birthday paradox is not a
// problem.
typedef uint64_t ull; //98c
struct H {
    ull x; H(ull x=0) : x(x) {}
    H operator+(H o) { return x + o.x + (x + o.x < x); }
    H operator-(H o) { return *this + ~o.x; }
    H operator*(H o) { auto m = (__uint128_t)x * o.x; //884
        return H((ull)m) + (ull)(m >> 64); }
    ull get() const { return x + !~x; }
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
}; //7dd
static const H C = (1l)1e11+3; // (order ~ 3e9; random also
// ok)

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), pw(ha) { //c1e
        pw[0] = 1;
        rep(i,0,sz(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    } //b8f
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};
```

```
};
//4b7
vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
    rep(i,0,length)
        h = h * C + str[i], pw = pw * C; //7ab
    vector<H> ret = {h};
    rep(i,length,sz(str)) {
        ret.push_back(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret; //413
}

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

AhoCorasick.h

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

**Time:** construction takes  $\mathcal{O}(26N)$ , where  $N$  = sum of length of patterns. find(x) is  $\mathcal{O}(N)$ , where  $N$  = length of x. findAll is  $\mathcal{O}(NM)$ .

```
struct AhoCorasick { //724
    enum {alpha = 26, first = 'A'}; // change this!
    struct Node {
        // (nmatches is optional)
        int back, next[alpha], start = -1, end = -1, nmatches =
            0;
        Node(int v) { memset(next, v, sizeof(next)); } //cc2
    };
    vector<Node> N;
    vi backp;
    void insert(string& s, int j) {
        assert(!s.empty()); //757
        int n = 0;
        for (char c : s) {
            int& m = N[n].next[c - first];
            if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
            else n = m; //20b
        }
        if (N[n].end == -1) N[n].start = j;
        backp.push_back(N[n].end);
        N[n].end = j;
        N[n].nmatches++; //77c
    }
    AhoCorasick(vector<string>& pat) : N(1, -1) {
        rep(i,0,sz(pat)) insert(pat[i], i);
        N[0].back = sz(N);
        N.emplace_back(0); //12a

        queue<int> q;
        for (q.push(0); !q.empty(); q.pop()) {
            int n = q.front(), prev = N[n].back;
            rep(i,0,alpha) { //57b
                int &ed = N[n].next[i], y = N[prev].next[i];
                if (ed == -1) ed = y;
                else {
                    N[ed].back = y;
                    (N[ed].end == -1 ? N[ed].end : backp[N[ed].start
                    ]) //338
                        = N[y].end;
                    N[ed].nmatches += N[y].nmatches;
                    q.push(ed);
                }
            }
        }
    }
};
```

```
    }
    } //c05
}

vi find(string word) {
    int n = 0;
    vi res; // 11 count = 0; //a68
    for (char c : word) {
        n = N[n].next[c - first];
        res.push_back(N[n].end);
        // count += N[n].nmatches;
    } //bb1
    return res;
}

vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word)); //008
    rep(i,0,sz(word)) {
        int ind = r[i];
        while (ind != -1) {
            res[i - sz(pat[ind]) + 1].push_back(ind);
            ind = backp[ind]; //8f0
        }
    }
    return res;
}
}; //214
```

Various (10)

10.1 Intervals

IntervalContainer.h

**Description:** Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

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

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R)
{ //ba1
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}), before = it;
    while (it != is.end() && it->first <= R) {
        R = max(R, it->second);
        before = it = is.erase(it); //ea6
    }
    if (it != is.begin() && (--it)->second >= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it); //05d
    }
    return is.insert(before, {L,R});
}
```

```
void removeInterval(set<pii>& is, int L, int R) { //858
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L; //61f
    if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h

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

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



```
template<class T>//0e2
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
    vi S(sz(I)), R;
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
    T cur = G.first;//ed8
    int at = 0;
    while (cur < G.second) { // (A)
        pair<T, int> mx = make_pair(cur, -1);
        while (at < sz(I) && I[S[at]].first <= cur) {
            mx = max(mx, make_pair(I[S[at]].second, S[at]));//607
            at++;
        }
        if (mx.second == -1) return {};
        cur = mx.first;
        R.push_back(mx.second);//26b
    }
    return R;
}
```

### ConstantIntervals.h

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

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

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

753a4c, 19 lines

```
template<class F, class G, class T>//570
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
    if (p == q) return;
    if (from == to) {
        g(i, to, p);
        i = to; p = q;//05f
    } else {
        int mid = (from + to) >> 1;
        rec(from, mid, f, g, i, p, f(mid));
        rec(mid+1, to, f, g, i, p, q);
    }//729
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
    if (to <= from) return;
    int i = from; auto p = f(i), q = f(to-1);//a6c
    rec(from, to-1, f, g, i, p, q);
    g(i, to, q);
}
```

## 10.2 Misc. algorithms

### TernarySearch.h

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

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

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

9155b4, 11 lines

```
template<class F>//7d4
int ternSearch(int a, int b, F f) {
    assert(a <= b);
    while (b - a >= 5) {
        int mid = (a + b) / 2;
        if (f(mid) < f(mid+1)) a = mid; // (A)//ec4
        else b = mid+1;
    }
    rep(i, a+1, b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
} //cbb
```

### LIS.h

**Description:** Compute indices for the longest increasing subsequence.

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

2932a0, 17 lines

```
template<class I> vi lis(const vector<I>& S) { //47f
    if (S.empty()) return {};
    vi prev(sz(S));
    typedef pair<I, int> p;
    vector<p> res;
    rep(i, 0, sz(S)) { //a50
        // change 0 -> i for longest non-decreasing subsequence
        auto it = lower_bound(all(res), p{S[i], 0});
        if (it == res.end()) res.emplace_back(), it = res.end()
            -1;
        *it = {S[i], i};
        prev[i] = it == res.begin() ? 0 : (it-1)->second; //476
    }
    int L = sz(res), cur = res.back().second;
    vi ans(L);
    while (L--) ans[L] = cur, cur = prev[cur];
    return ans; //342
}
```

### FastKnapsack.h

**Description:** Given  $N$  non-negative integer weights  $w$  and a non-negative target  $t$ , computes the maximum  $S \leq t$  such that  $S$  is the sum of some subset of the weights.

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

b20ccc, 16 lines

```
int knapsack(vi w, int t) { //e2b
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1); //14a
    v[a+m-t] = b;
    rep(i, b, sz(w)) {
        u = v;
        rep(x, 0, m) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for (x = 2*m; --x > m;) rep(j, max(0, u[x]), v[x]) //45b
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
    return a;
} //cbb
```

## 10.3 Dynamic programming

### KnuthDP.h

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

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

d41d8c, 1 lines

//d41

### DivideAndConquerDP.h

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

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

d38d2b, 18 lines

```
struct DP { // Modify at will: //ff9
    int lo(int ind) { return 0; }
    int hi(int ind) { return ind; }
    ll f(int ind, int k) { return dp[ind][k]; }
    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }
}
```

//ec8

```
void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO, lo(mid)), min(HI, hi(mid))) //680
        best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
} //a30
void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

## 10.4 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

## 10.5 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

### 10.5.1 Bit hacks

- $x \ \& \ -x$  is the least bit in  $x$ .
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of  $m$  (except  $m$  itself).
- $c = x \ \& \ -x$ ,  $r = x + c$ ; `((r^x) >> 2) / c` |  $r$  is the next number after  $x$  with the same number of bits set.
- `rep(b, 0, K) rep(i, 0, (1 << K))`  
if `(i & 1 << b) D[i] += D[i^(1 << b)];`  
computes all sums of subsets.

### 10.5.2 Pragmas

- `#pragma GCC optimize ("Ofast,unroll-loops")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2,tune=native")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

### FastMod.h

**Description:** Compute  $a \% b$  about 5 times faster than usual, where  $b$  is constant but not known at compile time. Returns a value congruent to  $a \pmod b$  in the range  $[0, 2b)$ .

751a02, 8 lines

```
typedef unsigned long long ull; //010
```

```
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(-1ULL / b) {}
    ull reduce(ull a) { // a % b + (0 or b)
        return a - (ull)((__uint128_t(m) * a) >> 64) * b;//430
    }
};
```

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();//c51
    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);//818
    }
    return buf[bc++]; // returns 0 on EOF
}

int readInt() { //f26
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 48;
    return a - 48;//d34
}
```

BumpAllocator.h  
**Description:** When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

745db2, 8 lines

```
// Either globally or in a single class://c17
static char buf[450 << 20];
void* operator new(size_t s) {
    static size_t i = sizeof buf;
    assert(s < i);
    return (void*)&buf[i -= s];//ef5
}

void operator delete(void*) {}
```

SmallPtr.h  
**Description:** A 32-bit pointer that points into BumpAllocator memory.

2dd6c9, 10 lines

```
"BumpAllocator.h"

template<class T> struct ptr { //bda
    unsigned ind;
    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
        assert(ind < sizeof buf);
    }
    T& operator*() const { return *(T*)(buf + ind); } //95f
    T* operator->() const { return &*this; }
    T& operator[](int a) const { return (&*this)[a]; }
    explicit operator bool() const { return ind; }
};
```

BumpAllocatorSTL.h  
**Description:** BumpAllocator for STL containers.  
**Usage:** vector<vector<int, small<int>>> ed(N);

bb66d4, 14 lines

```
char buf[450 << 20] alignas(16); //2c8
size_t buf_ind = sizeof buf;

template<class T> struct small {
    typedef T value_type;
    small() {} //8ec
```

```
template<class U> small(const U&) {}
T* allocate(size_t n) {
    buf_ind -= n * sizeof(T);
    buf_ind &= 0 - alignof(T);
    return (T*)(buf + buf_ind);//ad1
}

void deallocate(T*, size_t) {}
};
```

SIMD.h  
**Description:** Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "\_mm(256)?name\_(si(128|256)|epi(8|16|32|64)|pd|ps)". Not all are described here; grep for \_mm\_ in /usr/lib/gcc/\*/4.9/include/ for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h" and #define \_SSE\_ and \_MMX\_ before including it. For aligned memory use \_mm\_malloc(size, 32) or int buf[N] alignas(32), but prefer loadu/storeu.

551b82, 43 lines

```
#pragma GCC target ("avx2") // or sse4.1//c6d
#include "emmintrin.h"

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))
//d41
// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256,
// _mm_malloc
// blendv_(epi8|ps|pd) (z?y:x), movemask_epi8 (hibits of
// bytes)
// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts
// of x
// sad_epu8: sum of absolute differences of u8, outputs 4
// xi64//d41
// maddubs_epi16: dot product of unsigned i7's, outputs 16
// xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtssi128_si32 (128->
// lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane//
// d41
// shuffle_epi8(x, y) takes a vector instead of an imm

// Methods that work with most data types (append e.g.
// _epi32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/
// or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|
// hi)//512

int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
    int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); } //28e
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }

ll example_filteredDotProduct(int n, short* a, short* b) {
    int i = 0; ll r = 0; //730
    mi zero = _mm256_setzero_si256(), acc = zero;
    while (i + 16 <= n) {
        mi va = L(a[i]), vb = L(b[i]); i += 16;
        va = _mm256_and_si256(_mm256_cmpgt_epil6(vb, va), va);
        mi vp = _mm256_madd_epi16(va, vb); //b47
        acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
            _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)
            ));
    }
```

```
union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[
    i];
    for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <-
        equiv//c30
    return r;
}
```

## JHU (11)

### 11.1 Extra Equations

Legendres: the largest  $x$  s.t.  $k^x|n!$  is

$$\left\lfloor \frac{n}{k} \right\rfloor + \left\lfloor \frac{n}{k^2} \right\rfloor + \dots + \left\lfloor \frac{n}{k^i} \right\rfloor + \dots$$

Chicken McNugget: for any two pos relatively prime integers  $m, n$ , the greatest integer that cannot be written as  $am + bn$  for non-neg  $a, b$  is  $mn - m - n$ . Additionally there are exactly  $\frac{(m-1)(n-1)}{2}$  pos integers which cannot be expressed in the form  $am + bn$ .

Pick's: area of a polygon whose vertices are all lattice points in a coord plane is

$$A = I + \frac{1}{2}B - 1$$

$I$  is number of interior lattice points,  $B$  is number of border lattice points

Shoelace: Suppose polygon  $P$  has vertices  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , listed clockwise. Then area of  $P$  is

$$A = \frac{1}{2}|(x_1y_2 + x_2y_3 + \dots + x_ny_1) - (y_1x_2 + y_2x_3 + \dots + y_nx_1)|$$

$$\binom{2n}{n} = \sum_{k=0}^n \binom{n}{k}^2$$

Dense.h  
**Description:** Dense Dijkstra's  
**Time:**  $\mathcal{O}(V^2)$

5eeb07, 24 lines

```
vector<vector<pair<int, int>>> vec; //270
vector<ll> dists;
vector<int> paths;
vector<bool> visited;
void denseDjik(int start) {
    dists[start] = 0; //358
    for (int i = 0; i < n; i++) {
        int v = -1;
        for (int j = 0; j < n; j++) {
            if (!visited[j] && (v == -1 || dists[j] < dists
                [v]))
                v = j; //ab6
        }
        if (dists[v] == INF) break;
        visited[v] = true;
        for (auto e : vec[v]) {
            int to = e.first; //235
            int len = e.second;
            if (dists[v] + len < dists[to]) {
                dists[to] = dists[v] + len;
                paths[to] = v;
            }
        }
    }
```



```
        } //088
    }
}
```

## 11.2 Pi

numbers::pi

## 11.3 Dynamic bitset

Dynamic-bitset.h

4edc0d, 10 lines

```

#include <tr2/dynamic_bitset> //4ed
tr2::dynamic_bitset<__uint128_t> bs;
// void    append (block_type __block)
// void    append (initializer_list< block_type > __il)
// size_type find_first () const
// size_type find_next (size_t __prev) const //d41
// bool    is_proper_subset_of (const dynamic_bitset &__b)
//         const
// bool    is_subset_of (const dynamic_bitset &__b) const
// void    push_back (bool __bit)
// void    resize (size_type __nbits, bool __value=false)
```

Techniques (A)

techniques.txt	159 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Scheduling	
Max contiguous subvector sum	
Invariants	
Huffman encoding	
Graph theory	
Dynamic graphs (extra book-keeping)	
Breadth first search	
Depth first search	
* Normal trees / DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Konig's theorem and vertex cover	
Min-cost max flow	
Lovasz toggle	
Matrix tree theorem	
Maximal matching, general graphs	
Hopcroft-Karp	
Hall's marriage theorem	
Graphical sequences	
Floyd-Warshall	
Euler cycles	
Flow networks	
* Augmenting paths	
* Edmonds-Karp	
Bipartite matching	
Min. path cover	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Edge coloring	
* Trees	
Vertex coloring	
* Bipartite graphs (=> trees)	
* 3^n (special case of set cover)	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Coin change	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
Dynprog over intervals	
Dynprog over subsets	
Dynprog over probabilities	
Dynprog over trees	
3^n set cover	
Divide and conquer	
Knuth optimization	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Bitonic cycle	
Log partitioning (loop over most restricted)	
Combinatorics	
Computation of binomial coefficients	
Pigeon-hole principle	

Inclusion/exclusion
Catalan number
Pick's theorem
Number theory
Integer parts
Divisibility
Euclidean algorithm
Modular arithmetic
* Modular multiplication
* Modular inverses
* Modular exponentiation by squaring
Chinese remainder theorem
Fermat's little theorem
Euler's theorem
Phi function
Frobenius number
Quadratic reciprocity
Pollard-Rho
Miller-Rabin
Hensel lifting
Vieta root jumping
Game theory
Combinatorial games
Game trees
Mini-max
Nim
Games on graphs
Games on graphs with loops
Grundy numbers
Bipartite games without repetition
General games without repetition
Alpha-beta pruning
Probability theory
Optimization
Binary search
Ternary search
Unimodality and convex functions
Binary search on derivative
Numerical methods
Numeric integration
Newton's method
Root-finding with binary/ternary search
Golden section search
Matrices
Gaussian elimination
Exponentiation by squaring
Sorting
Radix sort
Geometry
Coordinates and vectors
* Cross product
* Scalar product
Convex hull
Polygon cut
Closest pair
Coordinate-compression
Quadtrees
KD-trees
All segment-segment intersection
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Discrete second derivatives
Strings
Longest common substring
Palindrome subsequences
Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Suffix array

Suffix tree
Aho-Corasick
Manacher's algorithm
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Best-first (A*)
Bidirectional search
Iterative deepening DFS / A*
Data structures
LCA (2^k-jumps in trees in general)
Pull/push-technique on trees
Heavy-light decomposition
Centroid decomposition
Lazy propagation
Self-balancing trees
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks
Persistent segment tree