

# Contest (1)

template.cpp	19 lines
<pre>#include &lt;bits/stdc++.h&gt; using namespace std;  #define LF '\n' #define SP ' ' #define rep(i, a, b) for(int i = a; i &lt; (b); ++i) #define all(x) begin(x), end(x) using ll = long long; using ull = unsigned long long; using ld = long double; // typedefs used in kactl code: // #define sz(x) x.size() // typedef pair&lt;int, int&gt; pii; // typedef vector&lt;int&gt; vi;  int main() {     ios_base::sync_with_stdio(false);     cin.tie(0); }</pre>	

indexed-set.cpp	6 lines
<pre>#include &lt;ext/pb_ds/assoc_container.hpp&gt; using namespace __gnu_pbds; template&lt;class T, class Comp = std::less&lt;T&gt;&gt; using indexed_set = tree&lt;T, null_type, Comp, rb_tree_tag, tree_order_statistics_node_update&gt;; template&lt;class K, class V, class Comp = std::less&lt;K&gt;&gt; using indexed_map = tree&lt;K, V, Comp, rb_tree_tag, tree_order_statistics_node_update&gt;;</pre>	

run.sh	21 lines
<pre>#!/usr/bin/bash # Usage: ~/run.sh -n A -i input -o output  inn="input" out="output" name="a" while getopts n:i:o: opt do     case "\${opt}" in         n) name="\${OPTARG}";;         i) inn="\${OPTARG}";;         o) out="\${OPTARG}";;     esac done echo \$inn echo \$out echo \$name ulimit -s 600000 g++ -std=c++20 -fno-stack-limit -O2 "\${name}.cpp" -o \${name} time \${name} &lt; \${inn} &gt; \${out} echo \$tim</pre>	

run-int.sh	14 lines
<pre>#!/usr/bin/bash # Usage: ~/run-int.sh -n A  name="a" while getopts n: opt do     case "\${opt}" in         n) name="\${OPTARG}";;</pre>	

stress.sh	23 lines
<pre>esac done ulimit -s 600000 g++ -std=c++20 -fno-stack-limit -O2 "\${name}.cpp" -o \${name} time \${name} echo \$tim</pre>	

stress.sh	23 lines
<pre>#!/usr/bin/sh # Usage: ~/stress.sh -sm smart -st stupid -g gen smrt = "smart" stpid = "stupid" gn = "gen"  while getopts sm:st:g: opt do     case "\${opt}" in         sm) smrt="\${OPTARG}";;         st) stpid="\${OPTARG}";;         g) gn="\${OPTARG}";;     esac done g++ -std=c++20 -O2 "\${smrt}.cpp" -o \${smrt} g++ -std=c++20 -O2 "\${stpid}.cpp" -o \${stpid} g++ -std=c++20 -O2 "\${gn}.cpp" -o \${gn} for ((i=1;i&lt;1000;++i)); do     echo \$i;     ./\${gn} \$i &gt; genIn;     diff &lt;./\${smrt} &lt; genIn &lt;./\${stpid} &lt; genIn    break; done #~/stress.sh -sm smart -st stupid -g gen</pre>	

.vimrc	5 lines
<pre>set nocompatible incsearch ignorecase hlsearch " Select region and then type :Hash to hash your selection. " Useful for verifying that there aren't mistypes. 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. cpp -dD -P -fpreprocessed   tr -d '[:space:]'   md5sum  cut -c-6</pre>	

stress.bat	13 lines
<pre>g++ -std=c++17 -O2 smart.cpp -o smart.exe g++ -std=c++17 -O2 stupid.cpp -o stupid.exe g++ -std=c++17 -O2 gen.cpp -o gen.exe</pre>	

stress.py	17 lines
<pre>:beg gen.exe &gt; input    exit smart.exe &lt; input &gt; smart_output stupid.exe &lt; input &gt; stupid_output fc smart_output stupid_output if errorlevel 1 goto bug goto beg :bug echo found!</pre>	

stress.py	17 lines
<pre>import os, sys  for i in range(1, 100000):     print('Running test', i)</pre>	

stress.py	17 lines
<pre>os.popen('./gen &gt; input') smart_ans = os.popen('./smart &lt; input').readlines() stupid_ans = os.popen('./stupid &lt; input').readlines() if smart_ans != stupid_ans:     print('Outputs are not equal')     print('Input:')     print(*(open('input').readlines()))     print('stupid answer:')     print(*stupid_ans)     print('smart answer:')     print(*smart_ans)     sys.exit() print('All tests passed')</pre>	

troubleshoot.txt	52 lines
<p>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.</p>	

troubleshoot.txt	52 lines
<p>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? Read the full problem statement again. Do you handle all corner cases correctly? Have you understood the problem correctly? Any uninitialized variables? Any overflows? Confusing N and M, i and j, etc.? Are you sure your algorithm works? What special cases have you not thought of? Are you sure the STL functions you use work as you think? Add some assertions, maybe resubmit. Create some testcases to run your algorithm on. Go through the algorithm for a simple case. Go through this list again. Explain your algorithm to a 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.</p>	

troubleshoot.txt	52 lines
<p>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).</p>	

troubleshoot.txt	52 lines
<p>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?</p>	

troubleshoot.txt	52 lines
<p>Memory limit exceeded: What is the max amount of memory your algorithm should need? Are you clearing all data structures between test cases?</p>	

## Mathematics (2)

### 2.1 Recurrences

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

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

Non-distinct roots  $r$  become polynomial factors, e.g.  $a_n = (d_1n + d_2)r^n$ .

### 2.2 Trigonometry

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

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

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

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

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

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

### 2.3 Geometry

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

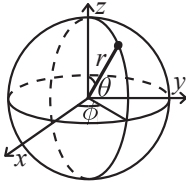
#### 2.3.2 Quadrilaterals

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

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

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

#### 2.3.3 Spherical coordinates



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

### 2.4 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:

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

### 2.5 Sums

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

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

### 2.6 Series

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

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

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

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

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

### 2.7 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  $Y$ ,

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

#### 2.7.1 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 wich yields success with probability  $p$  is  $\operatorname{Fs}(p)$ ,  $0 \leq p \leq 1$ .

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

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

Poisson distribution

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

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

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

2.7.2 Continuous distributions  
Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

Most real random values with mean  $\mu$  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

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

2.8 Additional formulas

2.8.1 Lagrange polynomial

Given  $n + 1$  pairs of numbers  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , such that all  $x_j$  are distinct. Construct polynomial  $L(x)$  of degree at most  $n$ , such that  $L(x_j) = y_j$   
 $l_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \dots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \dots \frac{x - x_n}{x_i - x_n}$   
 $L(x) = \sum_{j=0}^n y_j \ell_j(x)$ .

2.8.2 Wilson’s theorem

$$(n-1)! \equiv -1 \pmod n$$

Data structures (3)

SparseTable.h

Description: Sparse Table for arbitrary associative function

Time:  $\mathcal{O}(N \log N) / \mathcal{O}(1)$  for construction/query

d32011, 27 lines

```
template<class T, class Better = std::less<T>>
struct SparseTable {
    explicit SparseTable(vector<T> vals) {
        log2.push_back(0);
        for (int i = 1; i <= sz(vals); ++i) {
            log2.push_back(log2.back() + (2 << log2.back() < i));
        }

        table.push_back(std::move(vals));
        for (int p = 1; log2.back() >= sz(table); ++p) {
            auto& row = table.emplace_back();
            for (int i = 0; i + (1<<p) <= sz(table[0]); ++i) {
                row.push_back(get(i, i + (1<<p)));
            }
        }

        T get(int begin, int end) const {
            int p = log2[end - begin];
            return min(table[p][begin], table[p][end - (1<<p)], better)
                ;
        }

    private:
        vector<vector<T>> table;
        vector<int> log2;
        Better better;
    };
};
```

Treap.h

Description: Cartesian tree

Time:  $\mathcal{O}(\log N)$  merge, and split

10d1dd, 82 lines

```
typedef int OpT; // some user type
const OpT NEUTRAL = 0;
OpT combine(const OpT &l, const OpT &r) {
    return l + r;
}
inline OpT to_op_val(int val) {
    return val;
}

struct Node;

int get_sz(Node*);
OpT get_op_val(Node*);

struct Node {
    int val;
    int p;
    int sz{1};
    OpT op_val{NEUTRAL};
    Node* L{nullptr};
    Node* R{nullptr};

    Node() { upd(); }
    Node(int val_, int p_): Node() {
        val = val_;
        p = p_;
    }
};
```

```
    }
    void upd() {
        sz = 1 + get_sz(L) + get_sz(R);
        op_val = combine(get_op_val(L), to_op_val(val));
        op_val = combine(op_val, get_op_val(R));
    }
};

int get_sz(Node* v) {
    if (v == nullptr)
        return 0;
    return v->sz;
}

OpT get_op_val(Node* v) {
    if (v == nullptr)
        return NEUTRAL;
    return v->op_val;
}

typedef pair<Node*, Node*> split_t;

// split it so that ans.first.sz == pos
split_t split(Node* t, int pos) {
    if (t == nullptr) {
        return {nullptr, nullptr};
    }

    int sz = get_sz(t->L) + 1;
    if (sz <= pos) {
        split_t v = split(t->R, pos - sz);
        t->R = v.first;
        t->upd();
        return {t, v.second};
    } else {
        split_t v = split(t->L, pos);
        t->L = v.second;
        t->upd();
        return {v.first, t};
    }
}

Node* merge(Node* L, Node* R) {
    if (L == nullptr) return R;
    if (R == nullptr) return L;

    if (L->p < R->p) {
        L->R = merge(L->R, R);
        L->upd();
        return L;
    } else {
        R->L = merge(L, R->L);
        R->upd();
        return R;
    }
}

"Polynomial.h"
b00bfe, 23 lines
vector<double> polyRoots(Poly p, double xmin, double xmax) {
    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);
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];
    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;
            else h = m;
        }
        ret.push_back((l + h) / 2);
    }
}
return ret;
}
```

PolyInterpolate.h

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

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

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  $S[0 \dots \geq n-1]$  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:**  $\mathcal{O}(n^2 \log k)$

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

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        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);
        return res;
    };

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

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

```
}

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

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. The maximum error in the result is  $eps$ . 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 gss(double a, double b, double (\*f)(double)) { 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)) {
    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
            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);
        }
    return a;
}
```

HillClimbing.h

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

```
typedef array<double, 2> P;

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) {
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        }
    }
    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>
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;
}
```

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;
#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;
    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);
}

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 + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
```

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

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

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

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

    bool simplex(int phase) {
```

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

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

4.3 Matrices

**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;
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);
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            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;
        }
        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]);
        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];
        }
        rank++;
    }
}
```

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

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

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        }
        rank++;
    }

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
        rep(j,0,i) b[j] ^= A[j][i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

**4.4 Fourier transforms**  
**FFTRevPrecalc.h**  
**Description:** Linear reverse precalc for FFT.  
**Time:**  $\mathcal{O}(N)$

```
constexpr int MAX_BITS = /* specify this number */;
```

```
int rev[1 << MAX_BITS];

// n = 2^i
void rev_precalc(int n) {
    int bits = __lg(n);
    assert(bits <= MAX_BITS);
    for (int l = bits - 1, step = 1; step < n; l--, step *= 2)
        {
            int v = 1 << l;
            for (int i = step; i < n; i += step) {
                rev[i] ^= v;
            }
        }
    for (int i = 1; i < n; i++) {
        rev[i] ^= rev[i - 1];
    }
}
```

**FastFourierTransform.h**  
**Description:** Fast Fourier Transform with linear reverse precalc.  
**Time:**  $\mathcal{O}(N \log N)$

```
"FFTRevPrecalc.h" ff5dbe, 32 lines

using base_float = double;
using base = complex<base_float>;
const base_float PI = acos(-1.0);

// n = 2^i
void fft(vector<base> &a, int n, bool invert) {
    rev_precalc(__lg(n));
    for (int i = 0; i < n; i++) {
        if (i < rev[i]) {
            swap(a[i], a[rev[i]]);
        }
    }
    for (int len = 1; len < n; len <= 1) {
        int len2 = len << 1;
        base_float angle = 2.01*PI/(base_float)len2;
        if (invert)
            angle = -angle;
        for (int i = 0; i < n; i += len2) {
            for (int j = 0; j < len; j++) {
                base wj = exp(base{0.01, angle*j});
                base x = a[i + j], y = a[i + j + len];
                y *= wj;
                a[i + j] = x + y;
                a[i + j + len] = x - y;
            }
        }
    }
    if (invert) {
        for (int i = 0; i < n; i++)
            a[i] /= n;
    }
}
```

**FastFourierTransformMod.h**  
**Description:** FFT in GF(Prime). Uses inverse(w) to calculate  $w^{-1}$ , and binpow(a, p) to calculate  $a^p$ .  
**Time:**  $\mathcal{O}(N \log N)$

```
"FFTRevPrecalc.h" 373aed, 40 lines

constexpr ll MOD = 998244353;
// 2^ROOT_DEG-th root of 1 modulo MOD
constexpr ll MOD_ROOT = 31;
constexpr ll ROOT_DEG = 23;

ll get_root(ll lg_deg) {
    return binpow(MOD_ROOT, 1ll << (ROOT_DEG - lg_deg));
}
```

```

}

// n = 2^i
void fft(vector<ll> &a, int n, bool invert) {
    rev_precalc(__lg(n));
    for (int i = 0; i < n; i++) {
        if (i < rev[i]) {
            swap(a[i], a[rev[i]]);
        }
    }

    for (ll len = 1; len < n; len <= 1) {
        ll len2 = len << 1;
        ll w = get_root(__lg(len2));
        if (invert)
            w = inverse(w); // w^{n-1}
        for (int i = 0; i < n; i += len2) {
            ll wj = 1;
            for (int j = 0; j < len; j++) {
                ll x = a[i + j], y = a[i + j + len];
                y = (y * wj) % MOD;
                a[i + j] = (x + y) % MOD;
                a[i + j + len] = (x - y + MOD) % MOD;
                wj = (wj * w) % MOD;
            }
        }
    }
    if (invert) {
        ll n_inv = inverse(n); // n^{-1}
        for (int i = 0; i < n; i++)
            a[i] = (a[i] * n_inv) % MOD;
    }
}
```

## Number theory (5)

### 5.1 Modular arithmetic

```

Binpow.h
Description: Binary exponentiation modulo MOD
Time: O(log(power))
3e5630, 13 lines

constexpr ll MOD = 1000000007;

ll binpow(ll a, ll p) {
    ll res = 1;
    for (; e; a = a * a % MOD, p /= 2)
        if (e & 1) res = res * a % MOD;
    return res;
}

// inverse modulo prime
ll inverse(ll a) {
    return binpow(a, MOD - 2);
}
```

```

ModInverse.h
Description: Pre-computation of modular inverses. Assumes LIM ≤ mod
and that mod is a prime.
6f684f, 3 lines

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

ModInverseFact.h
Description: Fact calculation of modular inverses (and inverse factorials).
Time: O(N)
b600f3, 15 lines
```

```

vector<int> get_all_modular_inverses(int p) {
    vector<int> inverse_factorials(p);
    inverse_factorials[p - 1] = p - 1; // -1 mod p = p - 1
    for (int k = p - 2; k > 0; k--) {
        inverse_factorials[k] = 1LL * inverse_factorials[k + 1]
            * (k + 1) % p;
    }

    vector<int> inverses(p);
    int factorial = 1;
    for (int k = 1; k < p; k++) {
        inverses[k] = 1LL * factorial * inverse_factorials[k] %
            p;
        factorial = 1LL * factorial * k % p;
    }
    return inverses;
}
```

```

ModSqrt.h
Description: Tonelli-Shanks algorithm for modular square roots. Finds x
s.t. x^2 = a (mod p) (−x gives the other solution).
Time: O(log^2 p) worst case, O(log p) for most p
19a793, 24 lines

"ModPow.h"
ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // 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
    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;
    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 < r && t != 1; ++m)
            t = t * t % p;
        if (m == 0) return x;
        ll gs = modpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
    }
}
```

### 5.2 Primality

```

MillerRabin.h
Description: Deterministic Miller-Rabin primality test. Guaranteed to
work for numbers up to 7 · 1018; for larger numbers, use Python and
extend A randomly.
Time: 7 times the complexity of ab mod c.
60dcd1, 12 lines

bool isPrime ull n) {
    if (n < 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 >> s;
    for (ull a : A) { // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

```

LinearSieve.h
Description: Linear sieve with linear memory.
Time: O(N)
64ef7a, 14 lines

const int N = 10000000;
int lp[N+1];

void sieve() {
    vector<int> pr;
    for (int i=2; i<=N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back (i);
        }
        for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N; ++j)
            lp[i * pr[j]] = pr[j];
    }
}
```

### 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 (mod b).
33ba8f, 5 lines

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

```

CRT.h
Description: Chinese Remainder Theorem.
crt(a, m, b, n) computes x such that x ≡ a (mod m), x ≡ b (mod n). If
|a| < m and |b| < n, x will obey 0 ≤ x < lcm(m, n). Assumes mn < 262.
Time: log(n)
04d93a, 7 lines

ll crt(ll a, ll m, ll b, ll n) {
    if (n > 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 < 0 ? x + m*n/g : x;
}
```

```

phiFunction.h
Description: Euler’s φ function is defined as φ(n) := # of positive integers
≤ n that are coprime with n. φ(1) = 1, p prime ⇒ φ(pk) = (p − 1)pk−1,
m, n coprime ⇒ φ(mn) = φ(m)φ(n). If n = pk1 pk2 ... pkr then φ(n) =
(p1 − 1)pk1−1 ... (pr − 1)pkr−1. φ(n) = n · ∏p|n (1 − 1/p).
∑d|n φ(d) = n, ∑1 ≤ k ≤ n, gcd(k, n) = 1 k = nφ(n)/2, n > 1
Euler’s thm: a, n coprime ⇒ aφ(n) ≡ 1 (mod n).
Fermat’s little thm: p prime ⇒ ap−1 ≡ 1 (mod p) ∀a.
cf7d6d, 8 lines

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

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

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

### 5.5 Estimates

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

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

### 5.6 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:

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

Other useful formulas/forms:

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

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

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

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

```
int permToInt(vi& v) {
    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;
}
```

### 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	$\sim 2e5$	$\sim 2e8$

#### 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  $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$ . a0a312, 6 lines

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

#### 6.2.4 Stirling numbers of the first kind

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

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

$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$   
 $c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$

## Graph (7)

### 7.1 Network flow

MinCostMaxFlow.h  
**Description:** Min-cost max-flow.  $\text{cap}[i][j] \neq \text{cap}[j][i]$  is allowed; double edges are not. 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:** Approximately  $\mathcal{O}(E^2)$

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

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

struct MCMF {
    int N;
    vector<vi> ed, red;
    vector<VL> cap, flow, cost;
    vi seen;
    VL dist, pi;
    vector<pii> par;
```

```
MCMF(int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}

void addEdge(int from, int to, ll cap, ll cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
}

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

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

    auto relax = [&](int i, ll cap, ll cost, int dir) {
        ll val = di - pi[i] + cost;
        if (cap && val < dist[i]) {
            dist[i] = val;
            par[i] = {s, dir};
            if (its[i] == q.end()) its[i] = q.push({-dist[i], i});
            else q.modify(its[i], {-dist[i], i});
        }
    };

    while (!q.empty()) {
        s = q.top().second; q.pop();
        seen[s] = 1; di = dist[s] + pi[s];
        for (int i : ed[s]) if (!seen[i])
            relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
        for (int i : red[s]) if (!seen[i])
            relax(i, flow[i][s], -cost[i][s], 0);
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
}

pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
            fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x][p]);
        totflow += fl;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
            if (r) flow[p][x] += fl;
            else flow[x][p] -= fl;
    }
    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
}

// 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;
    while (ch-- && it--)
        rep(i,0,N) if (pi[i] != INF)
            for (int to : ed[i]) if (cap[i][to])
                if ((v = pi[i] + cost[i][to]) < pi[to])
                    pi[to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
}
```

DinicFast.h

**Description:** Dinic as taught by Andrew Stankevich.  
**Time:**  $\mathcal{O}(VE \log C_{max})$

<bits/stdc++.h> 2c12a8, 120 lines

```
using namespace std;
using ll = long long;
using ull = unsigned long long;
using ld = long double;

constexpr char LF = '\n';
constexpr char SP = ' ';

constexpr int INF = 1e9 + 100;
constexpr int MAX_N = 500;

struct Edge {
    int u, v, c;
    int f{0};

    inline int can() {
        return c - f;
    }
};

vector<Edge> edges;
vector<int> g[MAX_N];

int s, t;
int BOUND = 1;

array<int, MAX_N> dist;
bool bfs() {
    queue<int> q;
    fill(dist.begin(), dist.end(), MAX_N);
    dist[s] = 0;
    q.push(s);

    while (q.size()) {
        int v = q.front();
        q.pop();

        for (auto eid: g[v]) {
            auto &e = edges[eid];
            if (dist[e.v] == MAX_N && e.can() >= BOUND) {
                dist[e.v] = dist[v] + 1;
                q.push(e.v);
            }
        }
    }

    return dist[t] != MAX_N;
}

bitset<MAX_N> vis;
array<int, MAX_N> edge_cnt;
int dfs(int v, int minc) {
    vis[v] = true;

    if (v == t)
        return minc;

    for (; edge_cnt[v] < g[v].size(); edge_cnt[v]++) {
        int eid = g[v][edge_cnt[v]];
        auto &e = edges[eid];
        if (!vis[e.v] && e.can() >= BOUND && dist[e.v] == dist[v] + 1) {
            int r = dfs(e.v, min(minc, e.can()));
            if (r > 0) {
                auto &er = edges[eid ^ 1];
```

```
                e.f += r;
                er.f -= r;
                return r;
            }
        }
    }

    return 0;
}

ll dinic() {
    ll ans = 0;
    for (BOUND = 1 << 30; BOUND > 0; BOUND /= 2) {
        while (bfs()) {
            fill(edge_cnt.begin(), edge_cnt.end(), 0);
            while (true) {
                vis.reset();
                int r = dfs(s, INF);
                ans += r;
                if (r == 0)
                    break;
            }
        }
        return ans;
    }
}

void add_edge(int u, int v, int c1, int c2 = 0) {
    g[u].push_back(edges.size());
    edges.push_back(Edge{u, v, c1});
    g[v].push_back(edges.size());
    edges.push_back(Edge{v, u, c2});
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int n, m;
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        u--; v--;
        add_edge(u, v, c);
    }

    s = 0;
    t = n - 1;

    ll ans = dinic();
    cout << ans << LF;
    for (int i = 0; i < m; i++) {
        cout << edges[2*i].f << LF;
    }
}
```

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.

7.2 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) {
    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))
            return btoa[b] = a, 1;
    }
    return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0);
        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++) {
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                    next.push_back(btoa[b]);
                }
            }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay;
            cur.swap(next);
        }
        res += dfs(a, 0, g, btoa, A, B);
    }
}
```

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) {
    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;
            return 1;
        }
    return 0;
}

int dfsMatching(vector<vi>& g, vi& btoa) {
```



```
vi vis;
rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
        if (find(j, g, btoa, vis)) {
            btoa[j] = i;
            break;
        }
}
return sz(btoa) - (int)count(all(btoa), -1);
}
```

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

```
1e0fe9, 31 lines
pair<int, vi> hungarian(const vector<vi> &a) {
    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;
        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;
            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;
            }
            rep(j,0,m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            }
            j0 = j1;
        } while (p[j0]);
        while (j0) { // update alternating path
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1;
        }
    }
    rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
}
```

GeneralMatching.h

**Description:** Matching for general graphs. Fails with probability  $N/mod$ .  
**Time:**  $O(N^3)$

```
cb1912, 40 lines
vector<pii> generalMatching(int N, vector<pii>& ed) {
    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;
    }

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

    if (M != N) do {
        mat.resize(M, vector<ll>(M));
        rep(i,0,N) {
```

```
mat[i].resize(M);
        rep(j,N,M) {
            int r = rand() % mod;
            mat[i][j] = r, mat[j][i] = (mod - r) % mod;
        }
    } while (matInv(A = mat) != M);

    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;
            }
        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);
            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);
        }
    }
    return ret;
}
```

7.3 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). ncomps will contain the number of components.

```
76b5c9, 24 lines
vi val, comp, z, cont;
int Time, ncomps;
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));

    if (low == val[j]) {
        do {
            x = z.back(); z.pop_back();
            comp[x] = ncomps;
            cont.push_back(x);
        } while (x != j);
        f(cont); cont.clear();
        ncomps++;
    }
    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);
    Time = ncomps = 0;
    rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

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:**  $O(E + V)$

```
2965e5, 33 lines
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, e, y, top = me;
    for (auto pa : ed[at]) if (pa.second != par) {
        tie(y, e) = pa;
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);
            if (up == me) {
                st.push_back(e);
                f(vi(st.begin() + si, st.end()));
                st.resize(si);
            }
            else if (up < me) st.push_back(e);
            else { /* e is a bridge */ }
        }
    }
    return top;
}
```

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

2sat.h

**Description:** 2-SAT.  
**Time:** ???

```
<bits/stdc++.h>
5d2df2, 85 lines
#define eps 10e-7
#define ll long long
#define pb push_back
#define all(x) (x).begin(), (x).end()

using namespace std;

vector <vector <int>> g, gt;
vector <bool> used;
vector <int> topsort, comp;

void dfs1(int v) {
    used[v] = true;
    for (auto to : g[v])
        if (!used[to])
            dfs1(to);
    topsort.pb(v);
}
```

```
void dfs2(int v, int cl) {
    comp[v] = cl;
    for (auto to : gt[v])
        if (comp[to] == -1)
            dfs2(to, cl);
}

int main()
{
    int n, m; // m— number of vertices
    cin >> n >> m;

    g.resize(2*m); // direct graph
    gt.resize(2*m); // inverse graph

    // graph construction
    // (a || b) && (b || !c)
    // edges
    // !a => b
    // !b => a
    // !b => !c
    // c => b
    vector<int> res(m);
    for (int i = 0; i < 2*m; i++) {
        for (auto item : g[i]) {
            gt[item].pb(i);
        }
    }

    used.assign(2*m, false);
    for (int i = 0; i < 2*m; i++)
        if (!used[i])
            dfs1(i);

    comp.assign(2*m, -1);

    reverse(all(topsort));

    for (int i = 0, j = 0; i < 2*m; i++) {
        int v = topsort[i];
        if (comp[v] == -1) {
            dfs2(v, j);
            j++;
        }
    }

    for (int i = 0; i < m; i++) {
        if (comp[2*i] == comp[2*i + 1] && comp[2*i + 1] != -1)
            {
                cout << "IMPOSSIBLE";
                return 0;
            }
    }

    for (int i = 0; i < m; i++) {
        if (comp[2*i] == -1) {
            res[i] = true;
            continue;
        }
        if (comp[2*i] > comp[2*i + 1])
            res[i] = true;
        else
            res[i] = false;
    }
    return 0;
}
```

7.4 Trees

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.h0f62fb, 21 lines

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

    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
    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);
        }
    }

    int lca(int a, int b) {
        if (a == b) return a;
        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)];}
};
```

7.5 Math

7.5.1 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).

7.5.2 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.)

47ec0a, 28 lines

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

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

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.hf6bf6b, 4 lines

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

SegmentDistance.h

Description:

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

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

Point.h5c88f4, 6 lines

SegmentIntersection.h

Description:

If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 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.

Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
 cout << "segments intersect at " << inter[0] << endl;
"Point.h", "OnSegment.h"9d57f2, 13 lines

template<class P> vector<P> segInter(P a, P b, P c, P d) {
 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)};
    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);
    return {all(s)};
}

```

lineIntersection.h

**Description:** If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, 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. **Usage:** auto res = lineInter(s1,e1,s2,e2); if (res.first == 1) cout << "intersection point at " << res.second << endl;

"Point.h"

a01f81, 8 lines

```

template<class P>
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);
    return {1, (s1 * p + e1 * q) / d};
}

```

sideOf.h

**Description:** Returns where *p* is as seen from *s* towards *e*. 1/0/-1 ⇔ left/on line/right. If the optional argument *eps* is given 0 is returned if *p* is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. **Usage:** bool left = sideOf(p1,p2,q)==1;

"Point.h"

3af81c, 9 lines

```

template<class P>
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);
    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) {
    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 p0-p1 to line q0-q1 to point r.

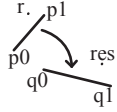
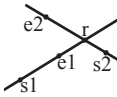
"Point.h"

03a306, 6 lines

```

typedef Point<double> P;

```



```

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

```

Angle.h

**Description:** A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. 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

0f0602, 35 lines

```

struct Angle {
    int x, y;
    int t;
    Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
    Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
    int half() const {
        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()}; }
    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) <
        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.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // 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;
}
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;
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;
    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;
}

```

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>
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;
    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();
    return out;
}

```

CirclePolygonIntersection.h

**Description:** Returns the area of the intersection of a circle with a ccw polygon.

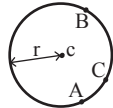
```

Time: O(n)
"../../content/geometry/Point.h"
alee63, 19 lines
typedef Point<double> P;
#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;
        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;
        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))
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}

```

circumcircle.h

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

```

MinimumEnclosingCircle.h

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

"circumcircle.h"09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
    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;
        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]);
                r = (o - ps[i]).dist();
            }
        }
    }
    return {o, r};
}
```

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>
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;
        //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;
}
```

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

PolygonCenter.h

**Description:** Returns the center of mass for a polygon.  
**Time:**  $\mathcal{O}(n)$

"Point.h"9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    }
    return res / A / 3;
}
```

PolygonCut.h

**Description:**  
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.  
**Usage:** vector<P> p = ...;  
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h"f2b7d4, 13 lines

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

ConvexHull.h

**Description:**  
Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.  
**Time:**  $\mathcal{O}(n \log n)$

"Point.h"310954, 13 lines

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

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

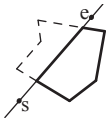
```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
    int n = sz(S), j = n < 2 ? 0 : 1;
    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
    rep(i,0,j)
        for (; j = (j + 1) % n) {
            res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
            if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
                break;
        }
    return res.second;
}
```

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

```
typedef Point<ll> P;
```



```
bool inHull(const vector<P>& l, P p, bool strict = true) {
    int a = 1, b = sz(l) - 1, r = !strict;
    if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
    if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
    if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <= -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
    }
    return sgn(l[a].cross(l[b], p)) < r;
}
```

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

```
#define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
    }
    return lo;
}
```

```
#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i,0,2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !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]};
        }
    return res;
}
```

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; 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()}};     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);         for (; lo != hi; ++lo)             ret = min(ret, {(*lo - p).dist2(), { *lo, p }});         S.insert(p);     }     return ret.second; }</pre>	

kdTree.h

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

"Point.h"	bac5b0, 63 lines
<pre>typedef long long T; 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; }  struct Node {     P pt; // if this is a leaf, the single point in it     T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds     Node *first = 0, *second = 0;      T distance(const P&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();     }      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);             y0 = min(y0, p.y); y1 = max(y1, p.y);         }         if (vp.size() &gt; 1) {             // split on x if width &gt;= height (not ideal...)             sort(all(vp), x1 - x0 &gt;= y1 - y0 ? on_x : on_y);             // divide by taking half the array for each child (not             // best performance with many duplicates in the middle)             int half = sz(vp)/2;             first = new Node({vp.begin(), vp.begin() + half});             second = new Node({vp.begin() + half, vp.end()});         }     } };  struct KDTree {     Node* root;     KDTree(const vector&lt;P&gt;&amp; vp) : root(new Node({all(vp)})) {}      pair&lt;T, P&gt; search(Node *node, const P&amp; p) {         if (!node-&gt;first) {             // uncomment if we should not find the point itself:</pre>	

<pre>// if (p == node-&gt;pt) return {INF, P()}; return make_pair((p - node-&gt;pt).dist2(), node-&gt;pt); }</pre>	
<pre>Node *f = node-&gt;first, *s = node-&gt;second; T bfirst = f-&gt;distance(p), bsec = s-&gt;distance(p); if (bfirst &gt; bsec) swap(bsec, bfirst), swap(f, s);  // search closest side first, other side if needed auto best = search(f, p); if (bsec &lt; best.first)     best = min(best, search(s, p)); return best; }</pre>	
<pre>// find nearest point to a point, and its squared distance // (requires an arbitrary operator&lt; for Point) pair&lt;T, P&gt; nearest(const P&amp; p) {     return search(root, p); }</pre>	
<pre>};</pre>	

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

"Point.h"	eefd5, 88 lines
<pre>typedef Point&lt;ll&gt; P; typedef struct Quad* Q; typedef __int128_t ll1; // (can be ll if coords are &lt; 2e4) P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point  struct Quad {     Q rot, o; P p = arb; bool mark;     P&amp; F() { return r()-&gt;p; }     Q&amp; r() { return rot-&gt;rot; }     Q prev() { return rot-&gt;o-&gt;rot; }     Q next() { return r()-&gt;prev(); } } *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;     return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B &gt; 0; }  Q makeEdge(P orig, P dest) {     Q r = H ? H : new Quad(new Quad{new Quad{new Quad{0}}});     H = r-&gt;o; r-&gt;r()-&gt;r() = r;     rep(i,0,4) r = r-&gt;rot, r-&gt;p = arb, r-&gt;o = i &amp; 1 ? r : r-&gt;r();     r-&gt;p = orig; r-&gt;F() = dest;     return r; }  void splice(Q a, Q b) {     swap(a-&gt;o-&gt;rot-&gt;o, b-&gt;o-&gt;rot-&gt;o); swap(a-&gt;o, b-&gt;o); }  Q connect(Q a, Q b) {     Q q = makeEdge(a-&gt;F(), b-&gt;p);     splice(q, a-&gt;next());     splice(q-&gt;r(), b);     return q; }  pair&lt;Q,Q&gt; rec(const vector&lt;P&gt;&amp; s) {     if (sz(s) &lt;= 3) {         Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());         if (sz(s) == 2) return { a, a-&gt;r() };     }</pre>	

<pre>splice(a-&gt;r(), b); auto side = s[0].cross(s[1], s[2]); Q c = side ? connect(b, a) : 0; return {side &lt; 0 ? c-&gt;r() : a, side &lt; 0 ? c : b-&gt;r() }; }</pre>	
<pre>#define H(e) e-&gt;F(), e-&gt;p #define valid(e) (e-&gt;F().cross(H(base)) &gt; 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)}); while ((B-&gt;p.cross(H(A)) &lt; 0 &amp;&amp; (A = A-&gt;next()))            (A-&gt;p.cross(H(B)) &gt; 0 &amp;&amp; (B = B-&gt;r()-&gt;o))); Q base = connect(B-&gt;r(), A); if (A-&gt;p == ra-&gt;p) ra = base-&gt;r(); if (B-&gt;p == rb-&gt;p) rb = base;  #define DEL(e, init, dir) Q e = init-&gt;dir; if (valid(e)) \     while (circ(e-&gt;dir-&gt;F(), H(base), e-&gt;F())) { \         Q t = e-&gt;dir; \         splice(e, e-&gt;prev()); \         splice(e-&gt;r(), e-&gt;r()-&gt;prev()); \         e-&gt;o = H; H = e; e = t; \     } for (;) {     DEL(LC, base-&gt;r(), o); DEL(RC, base, prev());     if (!valid(LC) &amp;&amp; !valid(RC)) break;     if (!valid(LC)    (valid(RC) &amp;&amp; circ(H(RC), H(LC))))         base = connect(RC, base-&gt;r());     else         base = connect(base-&gt;r(), LC-&gt;r()); } return { ra, rb }; }</pre>	

<pre>vector&lt;P&gt; triangulate(vector&lt;P&gt; pts) {     sort(all(pts)); assert(unique(all(pts)) == pts.end());     if (sz(pts) &lt; 2) return {};     Q e = rec(pts).first;     vector&lt;Q&gt; q = {e};     int qi = 0;     while (e-&gt;o-&gt;F().cross(e-&gt;F(), e-&gt;p) &lt; 0) e = e-&gt;o;     #define ADD { Q c = e; do { c-&gt;mark = 1; pts.push_back(c-&gt;p); \         q.push_back(c-&gt;r()); c = c-&gt;next(); } while (c != e); }     ADD; pts.clear();     while (qi &lt; sz(q)) if (!(e = q[qi++])-&gt;mark) ADD;     return pts; }</pre>	
--	--

8.5 3D

Polyhedron Volume.h

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

3058c3, 6 lines
<pre>template&lt;class V, class L&gt; double signedPolyVolume(const V&amp; p, const L&amp; trilst) {     double v = 0;     for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]);     return v / 6; }</pre>

Point3D.h

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

8058ae, 32 lines
<pre>template&lt;class T&gt; struct Point3D {     typedef Point3D P;     typedef const P&amp; R;</pre>

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

3dHull.h

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

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

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

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

```
struct F { P3 q; int a, b, c; };
```

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

    rep(i,4,sz(A)) {
        rep(j,0,sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                E(a,b).rem(f.c);
                E(a,c).rem(f.b);
```

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

sphericalDistance.h

**Description:** Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 ( $\phi_1$ ) and f2 ( $\phi_2$ ) from x axis and zenith angles (latitude) t1 ( $\theta_1$ ) and t2 ( $\theta_2$ ) from z axis (0 = 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\*radius is then the difference between the two points in the x direction and d\*radius is the total distance between the points.

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

Strings (9)

PrefFunc.h

**Description:** Prefix function  
**Time:**  $O(n)$

	365d92, 12 lines
--	------------------

```
vector<int> prefix_function(string s) {
    int n = (int) s.size();
    vector<int> p(n, 0);
    for (int i = 1; i < n; i++) {
        int cur = p[i - 1];
        while (s[i] != s[cur] && cur > 0)
            cur = p[cur - 1];
        if (s[i] == s[cur])
            p[i] = cur + 1;
    }
    return p;
}
```

Zfunc.h

**Description:** Z-function  
**Time:**  $O(n)$

	ddb08, 16 lines
--	-----------------

```
vector<int> z_function(string s) {
    int n = (int) s.size();
    vector<int> z(n, 0);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            z[i]++;
```

```
        if (i + z[i] - 1 > r) {
            r = i + z[i] - 1;
            l = i;
        }
    }
    return z;
}
```

Manacher.h

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

	e7ad79, 13 lines
--	------------------

```
array<vi, 2> manacher(const string& s) {
    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+l;z;
        if (i<r) p[z][i] = min(t, p[z][l+t]);
        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;
    }
    return p;
}
```

MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.  
**Usage:** rotate(v.begin(), v.begin()+minRotation(v), v.end());  
**Time:**  $O(N)$

	d07a42, 8 lines
--	-----------------

```
int minRotation(string s) {
    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; }
    }
    return a;
}
```

SuffixArray.h

**Description:** Builds suffix array for a string. 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 sa[0] = n. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. The input string must not contain any zero bytes.  
**Time:**  $O(n \log n)$

	38db9f, 23 lines
--	------------------

```
struct SuffixArray {
    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)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        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]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[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++;
        }
        rep(i,1,n) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
```

```
    for (k && k--, j = sa[rank[i] - 1];
         s[i + k] == s[j + k]; k++);
}
};
```

**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 substrng matching, though).  
**Time:**  $\mathcal{O}(26N)$

<pre>struct SuffixTree {     enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10     int toi(char c) { return c - 'a'; }     string a; // v = cur node, q = cur position     int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;      void ukkadd(int i, int c) { suff:         if (r[v]&lt;=q) {             if (t[v][c]==-1) { t[v][c]=m; l[m]=i;                 p[m++]=v; v=s[v]; q=r[v]; goto suff; }             v=t[v][c]; q=l[v];         }         if (q==-1    c==toi(a[q])) q++; else {             l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;             p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;             l[v]=q; p[v]=m; t[p[m]][toi(a[l[m])]]=m;             v=s[p[m]]; q=l[m];             while (q&lt;r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }             if (q==r[m]) s[m]=v; else s[m]=m+2;             q=r[v]-(q-r[m]); m+=2; goto suff;         }     }      SuffixTree(string a) : a(a) {         fill(r, r+N, sz(a));         memset(s, 0, sizeof s);         memset(t, -1, sizeof t);         fill(t[1], t[1]+ALPHA, 0);         s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;         rep(i, 0, sz(a)) ukkadd(i, toi(a[i]));     }      // example: find longest common substring (uses ALPHA = 28)     pii best;     int lcs(int node, int i1, int i2, int olen) {         if (l[node] &lt;= i1 &amp;&amp; i1 &lt; r[node]) return 1;         if (l[node] &lt;= i2 &amp;&amp; i2 &lt; r[node]) return 2;         int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;         rep(c, 0, ALPHA) if (t[node][c] != -1)             mask  = lcs(t[node][c], i1, i2, len);         if (mask == 3)             best = max(best, {len, r[node] - len});         return mask;     }     static pii LCS(string s, string t) {         SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));         st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);         return st.best;     } };</pre>	aae0b8, 50 lines
--	------------------

**Hash.h**  
**Description:** Hashing for strings.  
**Time:**  $\mathcal{O}(N)$  for construction,  $\mathcal{O}(1)$  query

<pre>template &lt;ull Prime, ull Mod&gt; struct Hash {     vector&lt;ull&gt; hash;     inline static vector&lt;ull&gt; pow = {1};      Hash(string s) : hash(s.size() + 1, 0) {         for (int i = 0; i &lt; s.size(); i++) {             hash[i + 1] = (hash[i]*Prime + s[i]) % Mod;         }         if (pow.size() &lt; s.size() + 1) {             int old_sz = pow.size();             pow.resize(s.size() + 1);             for (int i = old_sz; i &lt; pow.size(); i++) {                 pow[i] = pow[i - 1] * Prime % Mod;             }         }     }     ull substr(int pos, int len) {         ull r = Mod - (hash[pos] * pow[len] % Mod);         r = hash[pos + len] + r;         if (r &gt;= Mod)             r -= Mod;         assert(r &lt; Mod);         return r;     }     static ull full_hash(string s) {         if (pow.size() &lt; s.size() + 1) {             int old_sz = pow.size();             pow.resize(s.size() + 1);             for (int i = old_sz; i &lt; pow.size(); i++) {                 pow[i] = pow[i - 1] * Prime % Mod;             }         }         ull r = 0;         for (int i = 0; i &lt; s.size(); i++) {             r = (r*Prime + s[i]) % Mod;         }         return r;     } };  typedef Hash&lt;31, 1000*1000*1000+7&gt; H; typedef Hash&lt;31, 1000*1000*1000+7&gt; H1; typedef Hash&lt;97, 1000*1000*1000+123&gt; H2;</pre>	ab3d8a, 49 lines
--	------------------

**Hash2d.h**  
**Description:** Hashing for submatrixes.  
**Time:**  $\mathcal{O}(N * M)$  for construction,  $\mathcal{O}(1)$  query

<pre>Hash2d(const vector&lt;string&gt; &amp;arr) {     int n = arr.size();     int m = arr[0].size();      hash.assign(n + 1, vector&lt;ull&gt; (m + 1, 0));     for (int i = 0; i &lt; n; i++) {         Hasher h(arr[i]);         for (int j = 0; j &lt; m; j++) {             hash[i + 1][j + 1] = (hash[i][j + 1] * PrimeQ +                                    h.hash[j + 1]) % Mod;         }     }      if (pow_col.size() &lt; n + 1) {         int old_sz = pow_col.size();         pow_col.resize(n + 1);         for (int i = old_sz; i &lt; pow_col.size(); i++) {             pow_col[i] = pow_col[i - 1] * PrimeQ % Mod;         }     }     ull substr(int y, int x, int n, int m) {         ull r = hash[y + n][x + m];         r += pow_col[n] * Hasher::pow[m] % Mod * hash[y][x] %             Mod;         r += Mod - (pow_col[n] * hash[y][x + m] % Mod);         r += Mod - (Hasher::pow[m] * hash[y + n][x] % Mod);         r %= Mod;         return r;     } };  template &lt;class H1, class H2&gt; struct DHash2d {     H1 h1;     H2 h2;      DHash2d(const vector&lt;string&gt; &amp;arr) : h1(arr), h2(arr) {}     pair&lt;ull, ull&gt; substr(int y, int x, int n, int m) {         return make_pair(h1.substr(y, x, n, m), h2.substr(y, x,             n, m));     } };  using H = DHash2d&lt;Hash2d&lt;1000*1000*1000+7, 31, 65537&gt;, Hash2d     &lt;1000*1000*1000+123, 239, 29&gt;&gt;;</pre>	9155b4, 11 lines
---	------------------

Various (10)

10.1 Misc. algorithms

**TernarySearch.h**  
**Description:** Find the smallest i in [a, b] that maximizes f(i), assuming that f(a) < ... < f(i) ≥ ... ≥ 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:**  $\mathcal{O}(\log(b - a))$

<pre>template&lt;class F&gt; int ternSearch(int a, int b, F f) {     assert(a &lt;= b);     while (b - a &gt;= 5) {         int mid = (a + b) / 2;         if (f(mid) &lt; f(mid+1)) a = mid; // (A)         else b = mid+1;     } }</pre>	
--	--



```
    rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
}
```

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) {
    if (S.empty()) return {};
    vi prev(sz(S));
    typedef pair<I, int> p;
    vector<p> res;
    rep(i,0,sz(S)) {
        // 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;
    }
    int L = sz(res), cur = res.back().second;
    vi ans(L);
    while (L-->) ans[L] = cur, cur = prev[cur];
    return ans;
}
```

FastKnapsack.h

**Description:** Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= 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) {
    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);
    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])
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
    return a;
}
```

10.2 Dynamic programming

DivideAndConquerDP.h

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

d38d2b, 18 lines

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

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

```
    rec(mid+1, R, best.second, HI);
}
void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

10.3 Debugging tricks

- signal(SIGSEGV, [](int) { \_Exit(0); }); converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). \_GLIBCXX\_DEBUG 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).

10.4 Optimization tricks

\_\_builtin\_ia32\_ldmxcsr(40896); disables denormals (which make floats 20x slower near their minimum value).

10.4.1 Bit hacks

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

10.4.2 Pragmas

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

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 (mod b) in the range [0, 2b).

751a02, 8 lines

```
typedef unsigned long long ull;
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;
    }
};
```

FastInput.h

**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.  
**Usage:** ./a.out < input.txt

**Time:** About 5x as fast as cin/scanf. 7b3c70, 17 lines

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

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

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:
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];
}
void operator delete(void*) {}
```

SmallPtr.h

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

"BumpAllocator.h" 2dd6c9, 10 lines

```
template<class T> struct ptr {
    unsigned ind;
    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
        assert(ind < sizeof buf);
    }
    T& operator*() const { return *(T*)(buf + ind); }
    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);
size_t buf_ind = sizeof buf;

template<class T> struct small {
    typedef T value_type;
    small() {}
    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);
    }
    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
#include "emmintrin.h"

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))

// 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 4xi64
// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_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
// shuffle_epi8(x, y) takes a vector instead of an mm

// 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)

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); }
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;
  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_epi16(vb, va), va);
    mi vp = _mm256_madd_epi16(va, vb);
    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
  return r;
}
```

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